Lisa Ronacher, BSc.

Machine Learning-based Location Detection of Mathematical Expressions in PDF

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Dipl.-Ing. Dr.techn. Roman Kern

Institute for Interactive Systems and Data Science
Head: Univ.-Prof. Dipl-Inf. Dr. Stefanie Lindstaedt

Graz, August 2020

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date	Signature

Abstract

Portable Document Format (PDF) is one of the most commonly used file formats. Many current PDF viewers support copy-and-paste for ordinary text, but not for mathematical expressions, which appear frequently in scientific documents. If one were able to extract a mathematical expression and convert them into another format, such as LaTeX or MathML, the information contained in this expression would become accessible for a wide array of applications, for instance screen readers. An important step to achieve this goal is finding the precise location of mathematical expressions, since this is the only unsolved step in the formula extraction pipeline. Accurately performing this crucial step is the main objective of this thesis. Unlike previous research, we use a novel whitespace analysis technique to demarcate coherent regions within a PDF page. We then use the identified regions to compute carefully selected features from two sources: the grayscale matrix of the rendered PDF file and the list of objects within the parsed PDF file. The computed features can be used as input for various classifiers based on machine learning techniques. In our experiments we contrast four different variants of our method, where each uses a different machine learning algorithm for classification. Further, we also aim to compare our approach with three state of the art formula detectors. However, the low reproducibility of these three methods combined with logical inconsistencies in their documentation greatly complicated a faithful comparison with our method, leaving the true state of the art unclear, which warrants further research.

Contents

List of Figures

1 Introduction

In present days, many Portable Document Format (PDF) documents exist and can be accessed on the web. Alongside normal text, PDF documents can contain other elements like images, tables and mathematical equations. Especially in some areas of scientific research the documents frequently contain formulas and shorter mathematical expressions. While many current PDF viewers make it possible to copy-and-paste ordinary text, this ability does not extend to mathematical expressions. When one tries to copy a mathematical symbol that can not easily be typed with a standard keyboard, the results are mixed: some symbols can be copied, some will be substituted with a similar character, some will be left out altogether. Especially fraction bars are ignored and sub- or superscripts are no longer in their correct position. For example, when copying "x_i" the i will appear on the same level as x, which results in "xi". Generally, one can say that it is not possible to copy a substantial portion of mathematical symbols or larger mathematical expressions.

The option to copy mathematical expressions and use in simple text editors or to search for them in the web does not seem likely to become available in the near future. There is simply no support for fractions or large operators like sums. However, something that is possible is the conversion from mathematical expressions in PDFs into another format like LaTeX or MathML.

LaTeX is a popular choice for creating PDF documents that contain complex mathematical expressions. It is a markup language that is commonly used to write scientific documents. However, writing mathematical expressions can be rather time consuming, especially if a person is new to the LaTeX syntax. Therefore, it would be useful to copy a formula from a PDF document, transform it into LaTeX syntax and reuse it. MathML is used to present mathematical expressions on the web and is an application of XML. It can also be parsed by screen readers. Screen readers are tools that make it possible for visually impaired people to work with a computer. Therefore, transforming mathematical expressions into the MathML

format would be a first step to make PDF documents with mathematical content more accessibly to visually impaired users. For the conversion between LaTeX and MathML a number of free tools exist[1].

To convert a formula from a PDF document into LaTeX syntax multiple approaches are possible. One possibility is to analyze an image of the formula and to recognize all symbols and their relationships. Another approach is to parse glyph names and information regarding symbol positions directly from the PDF file. Obtaining different information from both sources is also a promising path. In all these cases the mathematical expression has to be detected at some point, whether by manual clipping from an image or automatic detection through an algorithm. The detection of mathematical expressions and the recognition of its symbols can be treated as two steps in the higher-level task of formula extraction.

Early attempts for formula recognition are already about 25 years old [4, 10, 11]. At first the PDF format was not yet widely used or even developed. Many scientific documents were first printed and later scanned so that only images of the documents were available. Some PostScript documents existed, which is the predecessor of PDF format. Still, at first only the optical recognition of mathematical expressions was researched.

The reason that we are able to copy ordinary text from a PDF today, is that very good Optical Character Recognition (OCR) systems exist. The recognition of text is a quite old research topic. Berman and Fateman [4] refer to OCR systems with over 99% accuracy in 1994. However, OCR systems lose much of their accuracy when it comes to mathematical expressions, since they were not designed for them. For mathematical expressions it is often not enough to scan a line just from left to right like for plain text, because they frequently have two-dimensional elements. Such elements are factions, sum or integral signs that include the upper and lower bounds of the function, or simply sub- and super-scripts. Equation systems are another mathematical structure that have a two-dimensional design. However, the main example of two-dimensional structures in math are matrices. These mathematical elements very rarely are mentioned in papers on the topic of formula detection or recognition. Therefore it is often not clear if the proposed systems even support matrices. While it is very likely that the individual elements of a matrix can be detected by most formula detection systems, a matrix should be recognized as one

[1] For example LaTeXML https://dlmf.nist.gov/LaTeXML/, Pandoc https://pandoc.org/ or TeX4ht https://ctan.org/pkg/tex4ht

structure. In a similar fashion, equation systems should ideally be recognized as on connected block.

Apart from the two-dimensional aspect mathematical expressions and ordinary text differ in fonts and font sizes that are used. While normal text is usually written in only one font, different fonts or styles will appear quite frequently in mathematical expressions. Additionally, it is almost not possible that a mathematical expression that is longer than three symbols will be uniform in font size [33]. Small symbols in sub and super-scripts, normal sized variables and larger symbols like sum signs and large brackets for vectors and matrices alternate frequently. In the same way different styles will appear in mathematical expressions. All these aspects prevent standard OCR systems from correctly recognizing mathematical expressions, and make it inevitable that special recognition systems for math content are developed.

Since the PDF format became more and more popular, formula recognition methods gradually shifted away from solely OCR-based systems. Instead enough information could be acquired trough PDF parsers to replicate a mathematical expression without the help of a rendered representation of this expression. When it comes to recognizing mathematical symbols, a purely visual approach can have problems with similar looking symbols. For example the Greek letter ρ (rho) is similar to a lower case p in italics. In the same way a χ (chi) and $\mathcal{X}$ (\$\mathcal{X}\$ in LaTeX) look almost the same when the size is not taken into account. Special OCR systems for formula recognition need different categories for Greek letters, italic font types and other typefaces that can be used in formulas and have their own meaning. When the part of the PDF file that includes those symbols is parsed, it should be unambiguous which symbol was used.

However, working exclusively from the PDF without support from the document image also has several difficulties. One of the main ones is, that different programs produce very different PDFs with diverse amount of information. Structural information and font names can be omitted and the resulting PDF can still look identical to one that contains all this information. Omitting the additional information produces a smaller file, so saving disk space or sending smaller files are the main reason for it. Another typical problem is that sometimes mathematical expressions or parts of them are inserted as images. While mathematical expressions that are a mixture of LaTeX syntax and images will be a minority, mathematical expressions can also

appear in figures. Just with information from the PDF alone it is not possible to recognize the content of these images.

Some commercial and free tools for formula recognition are available. The INFTY system[2] [30] can convert images of PDF documents to LaTeX, HTML or XML. Mathpix[3], a mainly free software, can recognized mathematical expressions from clipped screenshot images. Like mathpix some works on formula extraction omit the detection step and assume that their input (image, PDF or both) includes only a mathematical expression that was extracted by some other application or manually [2]. In this thesis the focus is on this sometimes neglected detection step. The goal is therefore not to identify a certain mathematical symbol or expression, but to find and mark the locations of them. Ideally, a mathematical expression should be anything from a single variable, over short terms to complex formulas.

Methods for formula detection have reported good results for a number of years now. For example in 2013 Chu and Liu[8] reported 95% precision and 91% recall for the detection of displayed expressions. In one of the most recent works on formula detection from Wang et al. [33] the precision and recall values are 93.6% and 99.4% respectively. Many different approaches are used: from classification with Support Vector Machines (SVMs), over deep neural networks to unsupervised methods. Every new method compares itself to an older approach with lower detection results, to show that an improvement was achieved. Nevertheless, evaluation results from a method that was developed in 2013 are almost as high as from a very recent method.

In this thesis we want to investigate how well different machine learning algorithms are suited for the task of formula detection and how this compares to other, sometimes more complex state of the art methods. A hybrid method is constructed that relies on information gained with a PDF parser, but also uses the document image. Via horizontal and vertical whitespace analysis we detect ”blocks”, which can be either text paragraphs or displayed mathematical expressions. Features from both sources (PDF parser and document image) are selected, to profit from the different information that both can provide. The machine learning algorithms that we use to classify between formula region and text region are: Support Vector Machine, random forest, Naive Bayes and artificial neural networks.

[2] http://www.inftyproject.org/
[3] https://mathpix.com/

The structure of this thesis is as follows: The second chapter focuses on the relevant literature of formula recognition and detection. In the third chapter our method and relevant background information for it are described. This is followed by the evaluation in Chapter 4, where we compare our method with a few selected literature methods. The last chapter summarizes the most important points of the previous chapters and draws conclusions about our method.

2 Related Work

The detection of mathematical content was a topic of research even before the PDF format was developed and became a standard format for documents [11]. Instead of digital files printed pages were scanned and formula detection or recognition were performed on the scanned image. While methods have drastically changed over time, it is still a viable approach to only use Optical Character Recognition (OCR) methods for the detection of mathematical expressions. The PDF format can give additional information about the document and the content of a page. However, the amount of useful information and its structure varies depending on the application that produced the PDF document. Therefore, the majority of state of the art methods could not work accurately without rendering the document and performing at least some of the detection steps on the resulting image.

In this chapter different approaches for formula detection will be presented. Since there are not many approaches that focus purely on formula detection, methods for formula recognition are also included. For these methods we focus mostly on the detection part. A large part of existing literature consists of supervised learning methods. Some early approaches for formula recognition can not be included in this category and some recent, state of the art methods lean more toward unsupervised learning. Another way to categorize the literature is to determine whether the approach is OCR-based or uses a PDF parser.

2.1 Background

In the first part of this section the two different types of mathematical expressions are described in detail. The second part covers the importance of preprocessing before PDF documents became popular. This type of preprocessing is usually not necessary for state of the art image-based formula recognition methods. However,

b_i picks k random polynomials of the form:

$$f_{b_i,v_j}(x) = s + \alpha_1 x + ... + \alpha_{a_t} x^{a_t} \pmod p$$

Where the coefficients are uniformly randomly chosen for each polynomial. If b_i is willing to bid at price v_j, then s is set to be ID_{b_i,v_j}, (i.e., b_i's ID for price v_j). Otherwise s is set to zero. b_i sends $f_{b_i,v_j}(\alpha_i)$ to a_i for all j, $1 \leq j \leq k$, and all i, $1 \leq i \leq \ell$.

Figure 2.1: Example of a displayed expression and multiple embedded expressions. A bounding box surrounds each mathematical expression - blue for the displayed expression and red for the embedded ones. In this example every variable is classified as embedded expression.

the information is still relevant if mathematical expressions should be located or extracted from an older document with less than perfect quality.

2.1.1 Mathematical Expression Types

Mathematical expressions can appear in two types on a page. Either as displayed expressions or embedded expressions. Displayed formulae are isolated from other components. Usually there is white-space above and below them and they are centered on the page or in their column. A number can appear on the right side of the displayed formula to act as reference number. This would be an relatively easy way to distinguish displayed formulas, but unfortunately they are not always used. Another name for displayed mathematical expressions is isolated expressions.

The other type of mathematical expressions are embedded expressions, which can also be called inline expressions. They appear inside a text paragraph and have no additional white-space that separates them from the surrounding text. They also do not have a number for references. While displayed expressions are often longer formulae, inline expressions can be very short, involving just a few symbols or even just one variable. Ideally, everything that would be written inside the $ $ environment in LaTeXshould be considered an embedded expression. Figure 2.1 shows an example of displayed and embedded expressions in blue and red boxes respectively.

The detection of embedded expressions can be quite ambiguous. For example in Figure 2.1 the sequence $j, 1 \leq j \leq k$ is shown in two bounding boxes. This does

not look incorrect since all parts of the embedded expression were found, but the whole sequence could also be detected as one.

2.1.2 Preprocessing

Preprocessing is often an important step first for methods that rely on Optical Character Recognition systems. Especially when it comes to scanned documents the quality of the image is never perfect. Noise, merged or broken characters hinder accurate detection and recognition of mathematical expressions. Without preprocessing many methods that rely on the image of a document would perform substantially worse.Preprocessing is more often an issue in older works that were written before the PDF file format was commonly used.

The work of Fateman et al [11] is a good example for that. The focus is not on the detection of mathematical expressions but instead more on the recognition of mathematical symbols. This work does not mention PDF documents, since the format was relatively new, but instead works with PostScript which is practically the predecessor of PDF. In the overview of their design they mention that they do not need to scan paper, but instead can convert a TEXfile to PostScript and next into a bitmap, which is used as input. This suggests that up to that point it was more common to work with scanned images, than with originally digital files.

When scanned images are involved it is important to have good preprocessing steps, which are not always required when the document was never in printed form. One part of preprocessing is de-skewing to properly align images that were not placed perfectly in the scanner. Another important step is to remove noise from multiple sources as much as possible. Possible sources of noise are the following:

- dirt on the document, that either got on the paper sometime before the scan or was introduced through the scanning process
- artifacts on the paper that are picked up by the scanner, for example holes on the side of the paper, staples (or holes from them) or even a crease
- imperfections introduced by the printer, possibly because of too little ink or low resolution

Especially when the resolution of the printer was not high enough to produce a document of good quality, it becomes very difficult to improve the image after scanning. Sometimes this causes letters to be too close together and appear as one

symbol. For the detection of mathematical expressions this is not that much of a problem, but once it comes to recognition, such compound symbols are difficult to process.

2.2 Traditional Detectors

This section focuses on traditional formula recognition methods. A prominent part of many traditional approaches is an Optical Character Recognition (OCR) systems. The basic characteristics of OCR systems and their potential problems with mathematical expressions are described below. Additionally, some methods that do not fall into the category of data driven approaches are mentioned.

2.2.1 Optical Character Recognition

Optical Character Recognition (OCR) is the automated recognition of printed or handwritten characters [10]. Most OCR systems are developed to primary recognized letters and they perform poorly when trying to recognize a mathematical expression. This is in part due to the great amount of different fonts, Greek letters and mathematical symbols that have to be recognized in a formula. Another major problem is the often 2-dimensional structure of mathematical expressions. Subscripts, superscript, fractions and matrices (among others) pose a challenge, since the linear way to process symbols that most OCR systems use does not work here [12].

One step in the OCR process is a segmentation step [10], where connected components are extracted. These are in most cases simply characters, but there are a number of symbols that consist of more than one component. The most frequent one is the lower-case letter ”i”, which consists of the dot and the stroke. In such cases the components have to be combined to represent the correct character. Standard OCRs have ways to handle this for typical characters like ”i” or even ”=”. However, mathematical expressions can include a multitude of mathematical symbols that are not one connected component, for example $\geq, \leq, \approx$ and variables like $\hat{x}, \bar{x}$ or $\vec{x}$ to name a few.

The other problem of extracting connected components is that two symbols that are too close together can appear as one. This is often an issue for scanned images with poor quality where two letters appear to be touching because of poor lighting, dirt on the paper or simply bad printing quality. In mathematical expressions especially sub- and superscripts are in danger of being too close to other characters. The risk is even higher when they appear in embedded expressions instead of displayed ones.

While standard OCR systems are not suited for the recognition of mathematical symbols, specialized OCR systems are the basis of many formula recognition methods. A prominent one is the INFTY system by Suzuki et al. [30]. Early on in its character recognition part a commercial OCR system, developed for ordinary text, processes the document image. It gives a good recognition result for ordinary text, but fails and produces meaningless outputs when it comes to mathematical expressions. The INFTY system exploits this to gain an initial detection of mathematical content. Later a specially developed "Original Recognition Engine" handles the previously found expressions. This engine can distinguish font-types, which is especially important for mathematical content. For example the difference between "X" and "$\mathcal{X}$" or between "a" and "α" may be critical for the meaning of a formula.

2.2.2 Rule-based Methods

Rule-based methods are more often used in older approaches, before many machine learning algorithms became really popular. The main characteristic of rule-based methods is that they do not use machine learning. Instead, descriptive statistics (mean, standard deviation) are used frequently to distinguish mathematical expressions from ordinary text.

Toumit et al [31] and Garain and Chaudhuri [7, 14] search for common math symbols, like the plus sign, equal sign or a fraction bar and tag this as mathematical content. Then, the detection area is expanded to include the rest of the formula. For example if a plus is found at least one symbol on the left and right side belongs to the mathematical expression. For fraction bars the area above and below them is relevant. Similarly, if an opening parenthesis is found, everything on the right side of it can belong to the formula until a closing parenthesis is detected. Parenthesis do not always indicate a mathematical expression so additional decision steps are necessary.

This process is more often applied to look for embedded mathematical expressions. When it comes to displayed expressions it is easier to analyze the layout to find the whole formula. In the work of Garain and Chaudhuri [14] displayed expressions are detected without character recognition. Instead, they look for blocks that are surrounded by white spacing and calculate the standard deviation of the bottom left black pixel of each symbol. In a text line these pixels generally lie on a straight line. However, in a mathematical expression the pixels are more vertically scattered. A similar approach is taken by Kacem et al. [19], who observed that displayed formulas have a much lower density of black pixels than a paragraph of plain text.

2.3 Data Driven Detectors

Unlike most traditional formula detectors, contemporary methods mainly rely on databases of PDF documents. Through various machine learning techniques the appearance of mathematical expressions is learned. The majority of state of the art formula detection methods fall into the category of supervised learning methods, but there are also a few were weakly supervised or unsupervised ones.

2.3.1 Four Step Model

After looking at a number of data driven approaches for formula detection, we observed that for many methods it is possible to split them into the following four steps:

1. obtain text regions
2. split into lines
3. extract features
4. classify

In the first step regions areas that are not text blocks are removed. This mostly includes images, tables and possibly table of content pages or references. If necessary image preprocessing steps such as de-skewing or noise removal will take place at the beginning of this phase. This is only done if the input file was a scanned document that is now presented as an image or PDF file.

The next step uses different segmentation methods to split a block of text into individual lines. This can either be done from scratch or functions from existing image processing libraries are used. After a list of lines is acquired the goal is to detect mathematical expressions among them. To detect displayed expressions classification of the line is enough. However, to be able to also find embedded expressions further segmentation of a line is necessary. Methods that include the detection of embedded mathematical expressions usually first distinguish between displayed expressions and text lines and subsequently split the text lines into words. This basically means that steps two, three and four have to be repeated on the lines not classified as displayed expression to detect embedded expressions.

Step three is feature extraction. The system can only classify mathematical expressions if it has learned what normal text and mathematical content looks like. In order to learn this, some kind of features are extracted from lines or words. The specific features are often the main difference between methods. Features of a line or word can be simple characteristics like for example in [8], where the height of a line, the indent on the left side, density of black pixels and centroid fluctuation are used. Centroid fluctuation describes how much the word centers deviate from the horizontal center of the line. On the other hand features can also be something more abstract like spectral properties of the line [26], which can be obtained via fast-fourier transform.

With the help of the extracted line and word features Support Vector Machines or various types of neural networks can be trained to distinguish between normal text and mathematical expressions. The accuracy of the results highly depends on the used features but the architecture of a neural network also contributes to it.

In the work of Phong et al. [26] Fast-Fourier-Transform (FFT) and Support Vector Machines (SVM) are employed to detect mathematical expressions. The focus is on the detection of displayed expressions, while inline expressions are mostly disregarded in this approach. However, the subsequent work [27] is dedicated to finding inline variables.

Figure 2.2 shows an overview of the steps this method applies. The blue boxes represent the first phase in which preprocessing and layout analysis takes place. Features of the open source software OCRopus are used in most of these steps. In the second phase, depicted in red, the classification of displayed expressions and normal text takes place. Since this approach only aims to detect displayed expressions, segmentation into lines that are either ordinary text or mathematical expressions is

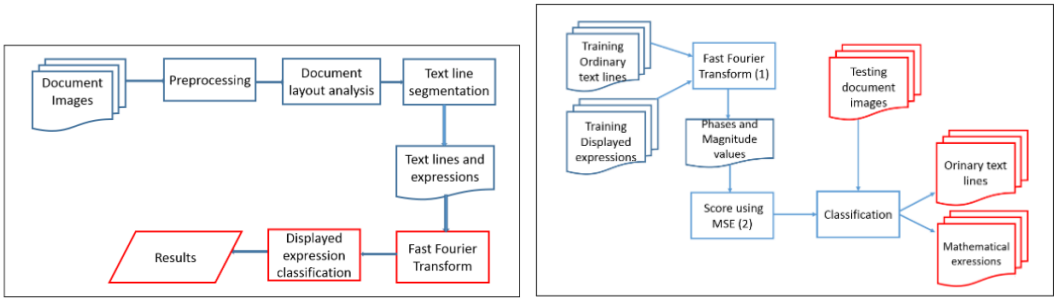

Figure 2.2: Flowcharts of the system used by Phong et al. [26]. The right image shows the process of how displayed expressions and ordinary text are classified.

sufficient for the classification. There is no further word segmentation. Phase and magnitude values are obtained for the line images with FFT. The density of black pixels is different for text lines and displayed expressions. Text lines normally have a higher density. The density can be compared through Power Spectrum which can be computed with FFT.

The phase and magnitude values are combined into F_{score} in order to have a quantitative measure for the line ($F_{score} = \alpha F_{phase} + \beta F_{magnitude}$). Then the mean square error (MSE) is calculated for two lines to determine the difference between them. Training data for the SVM is obtained by calculating the MSE between all training lines of displayed expressions and separately between all training text lines and again training lines of displayed expressions. To classify the test images the MSE between test images and training lines of displayed expressions is calculated and given as input for the SVM classification.

Tests showed that the systems can reliably extract mathematical expressions that are no longer than one line. However, once the expression exceeds one line, extraction errors appear more often.

2.3.2 Neural Networks and Deep Learning

In order to also detect inline expressions [27] builds on the previous work and adds a new module for variable detection. After the classification step that divided the lines into displayed expressions and text lines, the latter are processed further. Since the goal is to find variables among normal words the first new step splits a

line into words. In order to obtain a good segmentation the sum of black pixels in every column of the line image is utilized. This is called the vertical projection profile and the values are lower between words - this represents the white space.

For the feature extraction pre-trained Convolutional Neural Networks (CNN) are used. Specifically AlexNet and ResNet-50 were tested and the results were better with the latter one. The extracted features are then used to train a SVM, which takes over for the classification into variables and text words.

The work of Gao et al [13] also falls into the category of data driven methods, but contrary to many other learning-based approaches it does not precisely follow the four step model. Instead of line and word segmentation, a region proposal network is used to find candidate regions for formulae. This is similar to some approaches to object detection where the region proposal network locates regions in the image that are likely to contain an object [28].

A sliding-window approach is often used for region proposals, but this does not work very well for mathematical expressions because of their high variability in size. The region proposal in this case works with bottom-up and top-down layout analysis methods. Both approaches are used, since either one alone does not work very well for some scenarios. By combining them it is possible to find all formula-like regions.

For feature extraction two neural networks are trained. A convolutional neural network (CNN) extracts visual features of text and formula regions, while a recurrent neural network (RNN) is used for sequential character features. The RNN extracts features similar to manually designed character features that are used by traditional methods. After the input layer CNNs usually have alternately a number of convolutional layers and pooling layers followed by a few fully connected layers at the end. In this work max-pooling is used, however the last pooling layer is a Spatial Pooling layer [17]. This is important, because normally the input is required to have the same size. However, an important aspect of mathematical expressions is the variety of sizes in the individual symbols. The Spatial Pooling layer makes the size of input images changeable to retain this important feature of mathematical expressions. A joint layer is used to connect the features from CNN and RNN. Afterwards an additional fully connected layer transformes the features into a two-dimensional vector, which serves as input for the softmax classifier.

In tests they show that the combination of both networks achieves better results that either one network alone. The accuracy of detected formulae is high (over 94%). However, the system only detects displayed formulae, while ignoring embedded expressions.

2.3.3 Unsupervised Detectors

While most state of the art methods use supervised learning, the Font Setting based Bayesian (FSB) model by Wang and Liu [32] is an unsupervised algorithm. It first uses heuristic rules to detect a number of key mathematical expressions. In a next step new mathematical expressions are found through Bayesian inference, based on the assumption that the style of mathematical expressions does not chance within a document.

The key expressions are mostly displayed expressions. Only mathematical expressions that can be recognized with negligible error are chosen. A PDF parser extracts the Unicode values and glyph names of all symbols. Greek letters, operators, relations or big operators are math symbols and can indicate a mathematical expression. If no matched natural language word is found in the line that contains such a math symbol, the line is treated as displayed expression. Embedded expressions are slightly harder to find with high certainty. Nevertheless, mathematical function names or symbols can be processed as embedded expressions.

With the described heuristic rules the FSB model is able to estimate a highly accurate character level posterior. Since the font style of mathematical expressions does not change within a document, new mathematical expressions can be found with a likelihood ratio test. Through this method most displayed expressions can be correctly detected. An F1 score of 93.9% is reported.

In [33] Wang et al. take some ideas from the FSB model to develop another unsupervised algorithm. It also relies on the likelihood ratio test to classify symbols, in this case based on their font size. It is assumed that certain font sizes are used for mathematical expressions and others for ordinary text. The same font size might not always indicate ordinary text when it comes to PDF documents from different sources, but within one document this should not change. Therefore, instead of using ground truth to estimate the likelihood that a symbol with a certain font size is a mathematical symbol, the required information is extracted from the PDF

Figure 2.3: Examples from [33] for the successive detection of a mathematical expression. Red boxes indicate classification as mathematical expression, while green stands for ordinary text.

page. For mathematical expressions the font size changes frequently within a small number of symbols. Therefore, two groups for short and long sequences are established and character sequences with the same font size are sorted into them. The likelihood for certain font sizes to be linked to mathematical expressions or ordinary text can be estimated from the entries in the groups.

The algorithm needs several steps from detecting mathematical symbols to correctly identifying whole embedded or displayed expressions. As the example in Figure 2.3 shows, at first not all symbols in a mathematical expression are correctly detected and the rest is split into multiple fragments. In the next step characters that are closer to one another than a gap threshold are grouped together. False negatives can be eliminated when they are in the same group as math symbols. Finally mathematical expressions that lie next to each other, but with greater distances between them, are also merged.

This method can detect mathematical expression with high accuracy. The F1 scores for displayed and embedded expressions are 96.4% and 92.1% respectively. Many false positive detections from the FSB model can be corrected.

To summarize, over time multiple approaches for formula detection were developed. From methods that develop special OCR systems for mathematical content the trend slowly progressed to PDF parsers. Most methods use supervised learning, which means that a labeled data set is essential. Recently, two unsupervised methods were developed that only require labeled data for an accurate evaluation, but do not need to be trained. Most methods that were mentioned in this chapter claim to achieve a high accuracy. Since new works are still being published on the topic of formula detection the detection results are apparently still not high enough.

3 Method

In this chapter all relevant parts of our method are described. Additionally, some related concepts are explained to provide relevant background information. It starts with the internal structure of PDF files and explains the basic concepts of our chosen machine learning algorithms.

3.1 PDF File Structure

A comprehensive specification on the PDF file format was published by Adobe Systems Incorporated [18]. One of the reasons why PDF is a popular file format, is that the document looks the same on any device. To achieve this the focus is entirely on the graphical representation. What looks like a line of text in a PDF viewer, can be constructed out of several fragments which are just positioned next to each other on the page. They do not even have to appear in a consecutive order in the PDF file stream. This fact can make it challenging to parse PDF files correctly. The order of text has to be guessed from its position and white space has to be analyzed to find separate paragraphs or columns of text.

```
22 0 obj
<<
/Length 332
/Filter /FlateDecode
>>
stream
H‰T’Oof0FBÅï|
US;í@DC2SUB DC2âÐV•zØUS-]ï)˜SOi„(ÐC¿ýì¤ë´ETXõË‹Ÿüd“nöÛ½ígHßýØESp†®·-Çí¼úACKáŒ—Þ,TÐöÍ|?…w3CANBEL)™S1·iÆao»DC1ª*I?èršý
SYN;®žÂDC3¤o¥EßÛVT,ŽòóDÂáêÜ7SOhgDLEPxÐb—¤›ETBã^Î€ACKßŸx¥9EOTNAKÎò Pzlqr¦AoìENQ¡DC2ª†ªÄSUBÐ¶ÿï’Ut¤»æË ø$ViÔBÔI%»š-Û
±Ê˜ENQ}’*kETXg[æ.òŽXŸETXë5s¬Ñ\Sjb%${9EOT3é”åÞµøÍDLE#U¤¤¢ÏÄŽ%¹–’x®ƒ@LBA¬EDC4
ïÇ¹õ*6U$älÉ£³%çtE´äES±ˆKDC1+ˆcž˜€ÇÄÛ{Œ¼¹zOÛBS+SO3çi÷SYNUSESCGSS1—ŸäG€SOHNULBSßžDC3

endstream
endobj
```

Figure 3.1: Part of a PDF document when the file is opened with a text editor. It shows a stream object where the content is encoded.

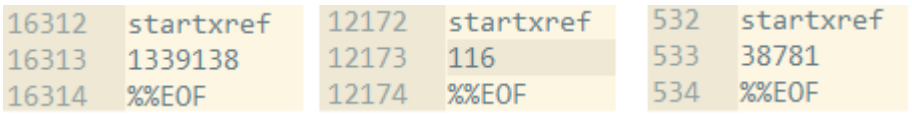

Figure 3.2: Last three lines of three different PDF files. The cross reference section is at the beginning of the file for the middle example, and near the end for the other two file.

Most of the actual content of a document is stored in so called *stream objects*. They have a length parameter (*/Length*) that specifies the number of bytes between the keywords *stream* and *endstream*. Next is the optional filter parameter (*/Filter*), which lists the names of all filters in the order that they were applied to the stream data. In the example in Figure 3.1 only one filter was applied: */FlatDecode*. If a filter has parameters they are listed afterwards under */DecodeParms*. Since the filter in the example does not have any parameters, no decode parameters are listed.

The object shown in the example is a so called *indirect object*. It has a positive integer object number (in this case 22) followed by a non-negative integer generation number (0). The generation numbers are always 0 when the PDF file is created and higher numbers can only appear when the file is updated later on. The combination of object number and generation number leads to a unique identifier for the specific object. The keywords *obj* and *endobj* mark beginning and end of the object. It is not required that indirect objects are numbered sequentially. This type of object is called indirect because indirect references that occur elsewhere in the file can refer to such an object.

Generally, it is recommended to start the parsing of a PDF document at the end of the file. The reason for this lies in the general structure of PDF files. Basic PDFs have a one-line header followed by the body section, which contains all of the objects. The cross reference table (also called xref table), that contains references to indirect objects, is most often located beneath the body. At the end of the file is the trailer, which ends with the end-of-file marker %%EOF. In the two lines before that the position of the last cross reference section is specified. The position is stated as the byte offset from the beginning of the file.

Figure 3.2 shows the last three lines of three PDF files that were opened with a text editor. For the first and third file the cross reference table is located near the end of the file. However, in the second example the table can be found nearly at the beginning of the file. This shows how the structure of a PDF file can vary. With

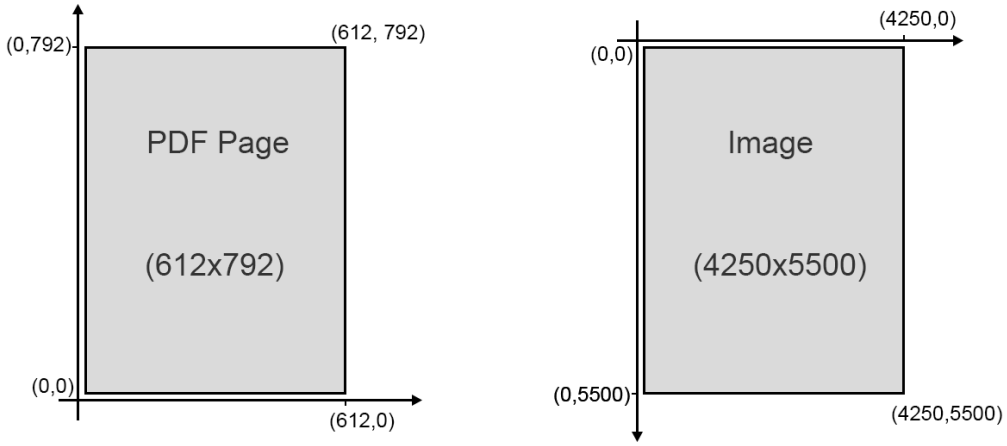

Figure 3.3: On the left hand side the PDF coordinate system is shown. The point of origin is on the lower left corner, which means that y-coordinates increase toward the top. On the right hand side is the image coordinate system. The origin is in the top left corner same as for matrices. Additionally, PDF page and image do not normally have the same dimensions. In this example the PDF is 612 x 792 and the image 4250 pixel x 5500 pixel.

the help of the cross reference table all indirect objects can be found and parsed relatively easy. Since, its location is evidently not fixed the best approach is to first parse the last three lines of a PDF file stream.

3.1.1 Coordinates

The coordinate system of a PDF file is set up differently than the coordinate system of image files, as is shown in Figure 3.3. In images the point of origin (0,0) is located in the top left corner. The x-coordinates increase to the right and y-coordinates increase toward the bottom. This is also the way indexing in matrices work. Since images are normally represented as matrices on computers, where every pixel is an entry, it is only natural that they use the same indexing system. However, in PDF files the origin is located in the bottom left corner. The x-coordinates still increase to the right, but y-coordinates represent the distance from the bottom and increase toward the top. An additional difference of PDF coordinates is that they are real numbers, while image coordinates are integers since they represent entries in the

image matrix.

Usually, the rendered PDF should have a high resolution. Therefore, the height and width of the page as image will be much higher than the dimensions of the same page as PDF file. For example the dimensions for a PDF page can be 612 x 792 points and the corresponding image has a size of 4250 x 5500 pixels. To transform PDF coordinates into image coordinates or vice versa it is necessary to know the dimensions of both files. To convert the x-coordinates, multiply the given value with the ratio of PDF width and image width. To obtain the corresponding y-coordinate, first subtract the y value from the image or PDF height before multiplying it with the ratio of heights. This is shown in Equations (3.1) and (3.2) where the PDF coordinates are computed from the image coordinates. The variables w_{pdf}, w_{img} and h_{pdf}, h_{img} describe the widths and heights of PDF and image respectively.

$$x_{\text{pdf}} = x_{\text{img}} \cdot \frac{w_{\text{pdf}}}{w_{\text{img}}} \tag{3.1}$$

$$y_{\text{pdf}} = (h_{\text{img}} - y_{\text{img}}) \cdot \frac{h_{\text{pdf}}}{h_{\text{img}}} \tag{3.2}$$

3.2 Horizontal and Vertical Whitespace Analysis

A problem that seems to occur quite often with the initial detection of displayed mathematical expressions, is that the expression is incorrectly split into multiple lines [21, 32]. The reason for this is usually a segmentation step where the text lines of a document are found. Parts of a displayed expression can be incorrectly recognized as separate lines. Sub- and superscripts of large operators, for example sum symbols, as well as numerator and denominator of fractions can be candidates for such incorrect line splits. To prevent this splitting of displayed expressions we decided to look at larger coherent blocks instead of lines. A block can for example be a text paragraph, a figure of some kind or a mathematical expression. Section titles are also seen as blocks. Basically, everything that has substantial white-space above and below it is a block. Some specific features of these blocks are later used as input for different machine learning algorithms to distinguish between displayed expressions and other blocks.

Since there is also white-space between text lines or between the sum symbol and its limits, it is necessary to make a paragraph or a mathematical expression appear as one coherent block. To this end a Gaussian filter is applied to the image. The filter should blur it just enough that each block is blurred, without obscuring the white-space between the blocks. The most important parameter of the Gaussian filter method is the standard deviation, σ. It defines the size of the Gaussian kernel and is responsible for the amount of blurring. A higher σ value leads to a more blurred image. For a multi-dimensional input (like images) σ can be a sequence of values, one for each axis. If only one number is given, the same standard deviation is used for all axes. Since we want to keep larger horizontal white-space areas the standard deviation in vertical direction has to be rather small. On the x-axis we do not have to preserve white-space, therefore σ can have a larger value here. Examples can be seen in Figure 3.5.

Not all PDF documents look the same. Documents from different sources will most likely not all have the same format, which means that among other things the font size can vary. Since different font sizes change the height of lines, the sigma parameter can not be the same for every document. Instead it has to change depending on the line height, or character height. For this reason, we compute a scale value, which is set to the median of all character heights on the page. The mean value is not used here, because the goal is to find the height of a normal text line and titles or section headings with larger font size will influence the mean too much. The character heights are calculated from the image of the PDF, since the character coordinates that can be extracted from the PDF are much larger than the actual character. Figure 3.4 shows an example of four characters and their different y-coordinates. The green lines represent the upper and lower PDF y-coordinates while the red lines show the actual start and end points of each character. As already mentioned there is a large amount of space between the upper limit of the bounding box and the first black pixel. However, the PDF bounding boxes have the advantage that the y-coordinates are the same for all characters in a line, provided that they have the same font size.

To get the actual size of a character, its PDF coordinates are transformed into image coordinates and this area is extracted from the image. Then, each horizontal pixel line is reduced to a Boolean value which depends on the existence of black pixels in that line. Lastly, the difference between the indices of the first and last black pixel line provide the size of the character. Once the actual size of all characters on a page is known, we compute the median and use this value as scale parameter. After

Figure 3.4: Four letters in the same font and font size start and end at different y-coordinates. The PDF bounding boxes around these letters have the same y-coordinates (green lines). However, the PDF coordinates are much larger than the actual characters (red lines).

the scale parameter is computed the Gaussian filter is applied to the image with $\sigma = 0.5 \cdot$ scale in y-direction and $\sigma =$ scale in x-direction. This leads to a stronger blurring in horizontal direction. Figure 3.5 shows he effect of different σ values.

The next step is to detect blocks in the blurred image. For this the color in the image is inverted, which means that everything that was white is now black and vice versa. When humans look at a document, they interpret the white background as ”nothing there” and only the black (or otherwise colored) areas as relevant. In images white pixels have the maximum value (1 or 255) while black pixels have the value 0 which is exactly reversed to how humans perceive it. For the implementation it makes little difference whether to search for blocks of low values (black) or high values (white). But the process can be more easily described if original black pixels have high values. For the inverted image the sum of each horizontal pixel line is calculated, we call this horizontal dark pixel sum. The sum of an originally white line is zero. Figure 3.6 shows an example how the horizontal dark pixel sum looks when plotted. On the left-hand side of the Figure is the corresponding document.

To find the blocks in the image a simple threshold rule was implemented. Each time the horizontal pixel sum rises above a certain threshold value a new block is started. The block ends once the value falls below the threshold again. To find a good threshold we experimented with some values and came to the conclusion that

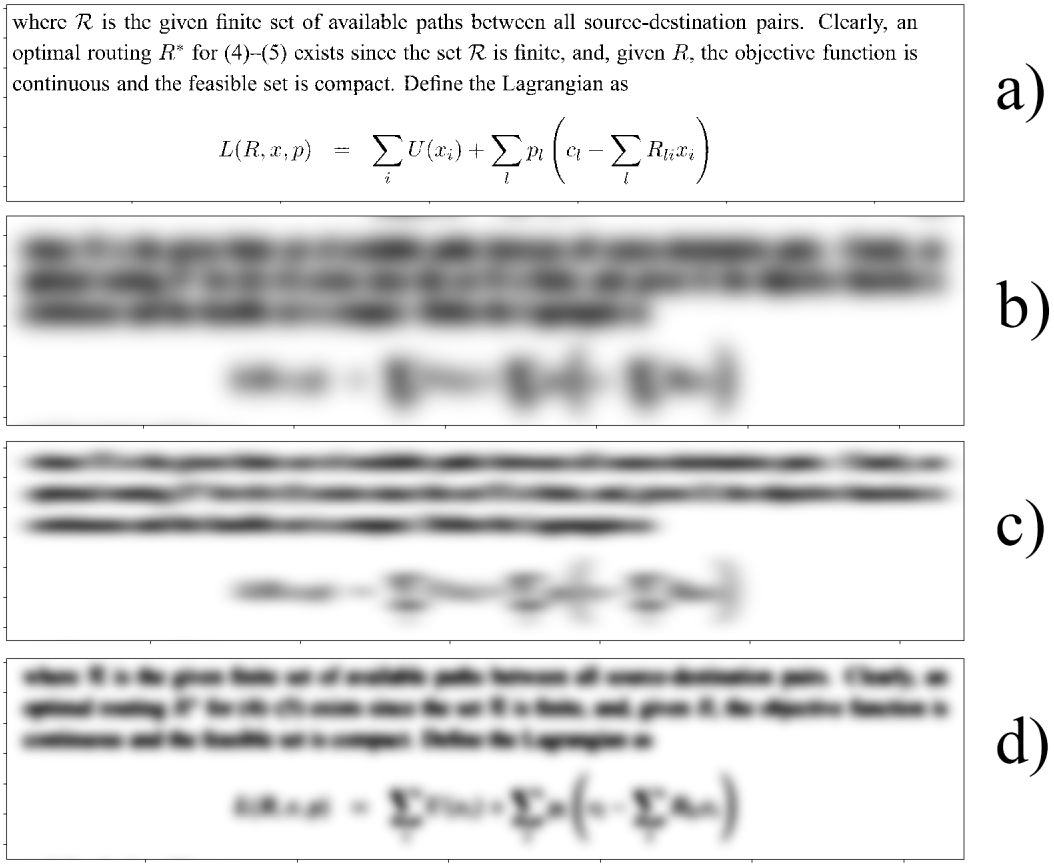

Figure 3.5: Image of a PDF document part with different Gaussian filter settings. a) original image without filter b) σ = scale for both directions c) σ = scale in x-direction and σ = $0.5 \cdot$ scale in y-direction d) σ = $0.5 \cdot$ scale for both directions. c) and d) are both acceptable results, but the blurring in b) is too strong.

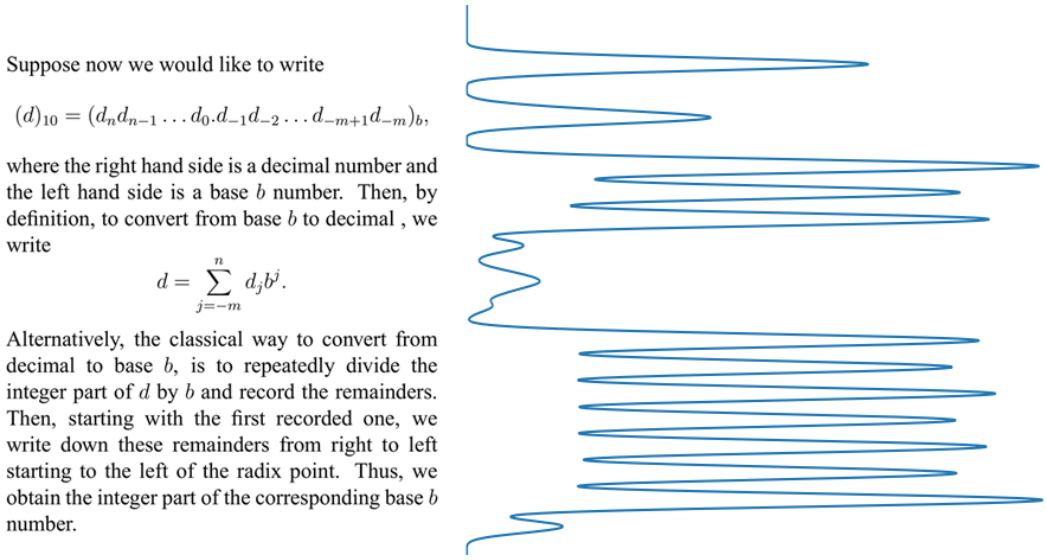

Figure 3.6: Part of a PDF document and a plot of the horizontal dark pixel sum in this area. The sum is calculated from the blurred image so that smaller horizontal whitespace areas are not detected.

a threshold between 7 and 20 gives good results. Even for those values problems occur. On one hand, sometimes two blocks merge if the threshold is too low to detect the space between the blocks. On the other hand there are cases where blocks end too soon if the threshold is too large. In the end the threshold was set to 9 which at least avoids the latter problem.

Another problem that occurs for all acceptable threshold values is that in some cases it looks like there is enough whitespace above a formula but in reality the distance between the last text line and the displayed expression is not large enough. Figure 3.7 shows such an example. The last text line has only one word in it and no words above the displayed expression. Given that there is an considerable amount of whitespace surrounding the formula, it would seem as if it is easy to detect it as new block. However, the horizontal dark pixel sum between the last text line and the beginning of the displayed formula does not fall below the threshold. In this concrete example a slightly higher threshold would solve the problem, but there are many cases where even a threshold value of 20 would not detect a new block.

To solve this problem we needed some additional checks whether a block should end. Originally once a new block started each pixel line with a horizontal pixel sum

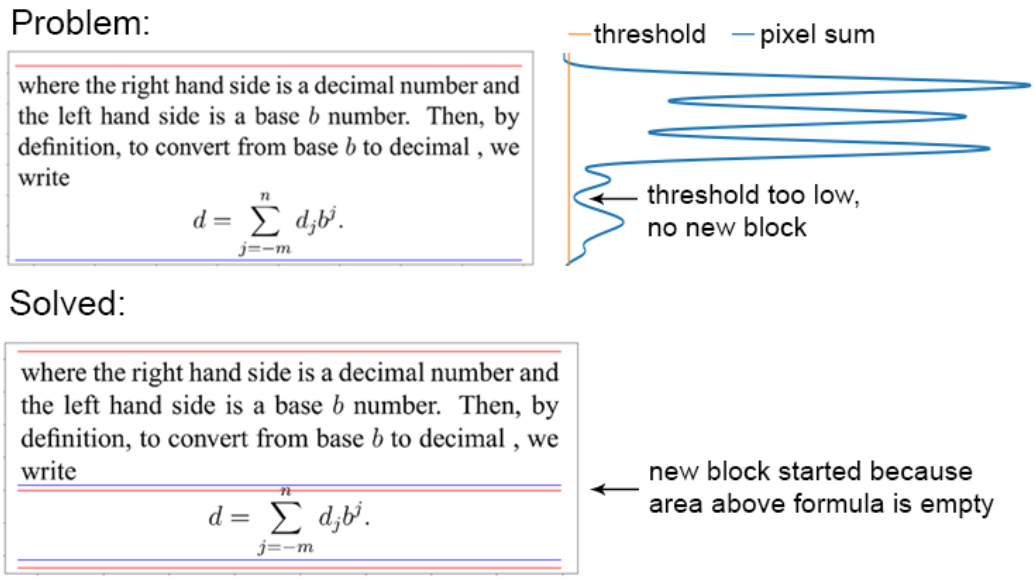

Figure 3.7: The red line shows the start of a block and the blue line its end. In the upper image the displayed expressions is not in a separate block even though it looks like there is enough distance between it and the text paragraph. However, the word ”write” in the line above is too close, so the horizontal dark pixel sum does not fall below the threshold (orange line on the right). To solve this we look at the space directly above the displayed expression. Since there is no text here a new block starts at the best y-coordinate between ”write” and the formula.

higher than the threshold was ignored. Now once we are $scale \cdot 2.5$ pixels into the block, there is an additionally check if there is text above the text of the current line. To do that we look at the line $scale \cdot 2$ pixels above and analyze where on the horizontal line the text is. If the text locations of the current line and the line above overlap for less than 20% a new block could start close to the current line. To find the best beginning point we look at the pixel sums for $3 \cdot scale$ lines above and $1 \cdot scale$ below the current line and search for the minimum value. At the position of the minimal value the current block ends, and a new one starts. However, if the minimum is found at the topmost line no new block starts. In this case it is very likely that we are still close to the beginning of the block. A distance of $scale \cdot 4$ pixels should be large enough that the minimum representing the small whitespace between the current text line and the previous text line is included. We chose all values like $scale \cdot 2$ based on the knowledge that the value of scale represents the size of an average character. From there we could estimate how much space a line

will typically need.

To summarize, the procedure to find a new block can be describes as follows :

- go over the list of horizontal dark pixel sums and set v to the current sum value
- start new block if $v > t$
- end block if $v <= t$
- for lines in block check if the text in the line above overlaps with text from current line
- if not end current block and start a new one at the best position around the current pixel line

PDF documents often have a two column layout. To find blocks here the approach from above has to be performed for each column. Therefore, before the horizontally blurring Gaussian filter, a vertical one is applied to each document image. Finding columns is easier than horizontal blocks, since they are more clearly defined. All columns that are found (even for single column layouts) are used as input for the horizontal block detection step.

3.3 Features for Displayed Expressions

Supervised machine learning algorithms need features and corresponding labels as input. In this case the features are measurable properties of blocks, which contain either text or a displayed mathematical expression. In this section the features that are used to train the machine learning algorithms are described. Additionally to the features that were useful, we also include a few ideas that did not work and the reason why these features were not used.

3.3.1 Sparsity

Displayed mathematical expressions are often shorter than the column width and do not use all the available space in one line. This leads to a great number of white pixels surrounding them. Even if the formula is especially long the gap between mathematical symbols is usually wider than the distances between characters within a text paragraph. Therefore, our hypothesis is, that displayed expression areas have

more white pixels, than text areas. In other words, mathematical expressions are more sparse than text paragraphs.

The sparse property of non-text areas was also used to detect tables in [23]. Ying et al. define a sparse line property which is fulfilled if the minimum space gap between consecutive words is larger than a threshold or if the length of the line is shorter than a threshold. In our case only single characters are obtained from the PDF instead of words. To merge characters into words the coordinates have to be analyzed so that the white-space between two words can be found. However, we decided to use the matrix definition of sparsity [15]. A matrix is called sparse if most entries are zero and dense if most entries are non-zero. To compute the sparsity of a matrix the number of elements which are zero is divided by the total number of elements. Sparsity is measured in percent.

In an image matrix 0 represents a black pixel and white pixels have the maximum value. So strictly speaking an image section with more white background than black words is very dense instead of sparse. Nevertheless, since humans perceive white on a page as ”nothing there” we will refer to sections with very little black as sparse.

For the computation a section of the document that contains either a displayed expression or text is considered. Such a section of the image will be referred to as ”block”. To get the sparsity a binary image is used, which means that all pixels are either white or black without any gray in between. In Python this is represented as a matrix with the values ”True” and ”False”. Since True can also be interpreted as 1, it is possible to compute the sum of a block (number of white pixels) and divide it by the total number of pixels in the block to obtain the sparsity value. Figure 3.8 shows a few examples of blocks and the corresponding sparsity values.

3.3.2 Horizontal Glyph Densities

While the PDF document does not really have the concept of words, it can give coordinates for every symbol in it. As described in Section 3.2 and shown in Figure 3.4 the coordinates do not describe precisely where a symbol is. There is especially much additional space above and below a character, while the left and right boundaries are more accurate. If two characters lie next to each other on the page, the right border of the leftmost character is were the bounding box of the next character

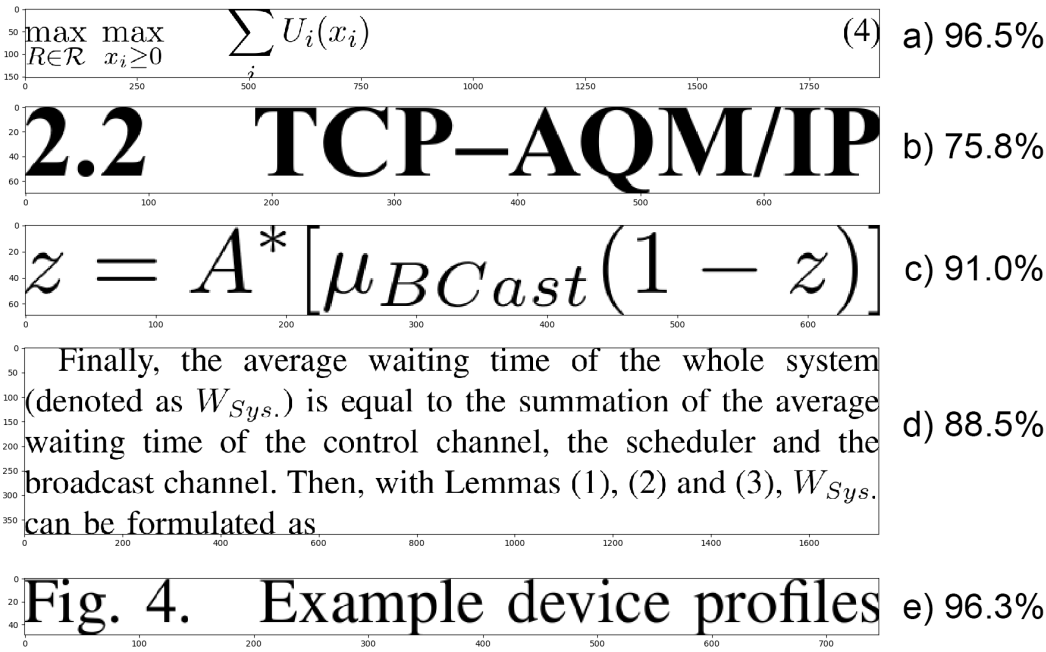

Figure 3.8: Five examples of blocks and the corresponding sparsity values. Overall the numbers are rather high. Each block can be considered sparse, since the sparsity is over 50%. Text examples have lower values than mathematical expressions, with the exception of example e).

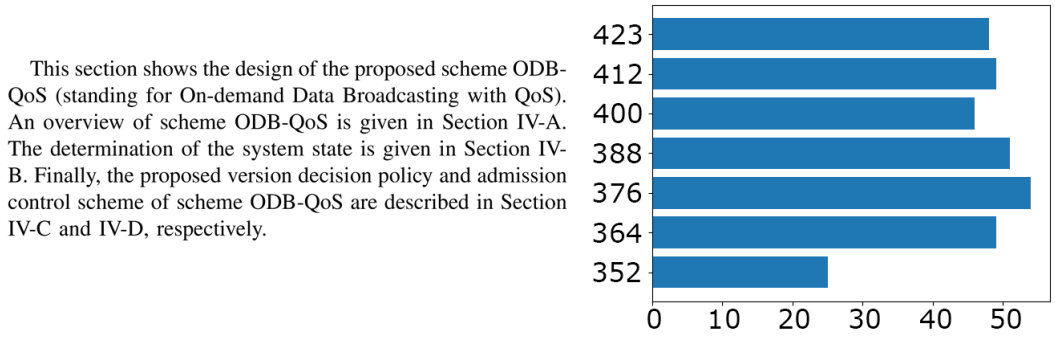

Figure 3.9: Horizontal glyph densities of a text block. Since no inline expressions are present there are the same amount of y-coordinate entries as lines in the block. The maximum is 54, the mean 46.0 and the standard deviation 8.88.

begins. The lower limits might not be accurate, however they are the same for all characters that lie on the same line. Exceptions are sub-, and super-scripts which have lower or higher y-coordinates respectively.

To utilize this we list all different y-coordinates within a block and count how many symbols lie on that specific horizontal line. This list will be referred to as the horizontal glyph densities of a block. The assumption is that a text paragraph will have multiply entries with a high number of symbols which represent the lines. Figure 3.9 shows an example text block and the corresponding horizontal glyph densities. The y-coordinates come from the PDF, therefore they decrease toward the bottom of the page. Most of the lines have a similar number of characters in them with the exception of the last line.

In a mathematical expression on the other hand, there should be more y-coordinate entries with a few symbols each, due to subscripts, superscripts and fractions. There can be a main line in a formula that contains more symbols, but the number should be substantially smaller than the number of characters on a full text line. Figure 3.10 shows an example of how the horizontal glyph densities for a formula can look. When compared to the text example from Figure 3.9 the number of characters with the same y-coordinate is significantly smaller. Both examples come from a PDF document with a two column layout, which means that the difference in character numbers for the text example could be even higher for a single column layout.

Since text paragraphs can contain embedded mathematical expressions there will

$$W_{Sys.} = W_{Ctrl.} + W_{Sche.} + W_{BCast} \quad (1)$$

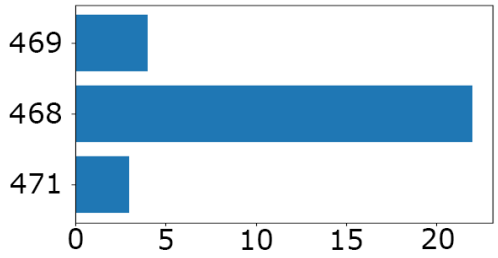

Figure 3.10: Horizontal glyph densities for a displayed expression. There is one main line in the formula and a few characters that lie slightly above or below this. The difference between the y-coordinates is much smaller than for the text example. The maximum is 22, the mean 9.67 and the standard deviation 8.73.

also be y-coordinate entries with very small symbol counts. Nevertheless, for the most part the distribution should be different enough that it is possible to distinguish text blocks and formula blocks.

The horizontal glyph densities alone can not be used as additional feature. The number of lines is different for each block and machine learning algorithms only accept a feature matrix. This means that the vector of features must have the same length for each sample. Therefore, the data has to be transformed into a single value for each block.

Maximum

One value that can be useful is the maximal number of symbols in the horizontal glyph density list from each block. This can be helpful because a full line of standard text should contain more symbols than a mathematical expression. The distance between characters is smaller in text since in mathematical expressions additional space is needed for larger mathematical operators or sub- and superscripts, which will often have separate entries. Therefore, even a displayed expression that spans the entire width of a page should have a smaller maximum value than the typical text block. Exceptions are any kind of section headings and short sentence fragments that can appear, for example between two displayed expressions. The maximum number of symbols is also lower in two column layouts. In this case the maximum value for displayed expressions should still be lower than the maximum for text

blocks. However displayed expressions blocks from single column layouts could have higher maximum values, than text blocks from two column layouts.

Mean and Standard Deviation

Mean and standard deviation of the horizontal glyph densities could potentially also be useful features. They give information about the distribution of the horizontal glyph densities. The maximum number of symbols on the same y-coordinate should be smaller for displayed expressions. This, together with many entries with low values should lead to a smaller mean value. Text blocks, especially if they are long, will have multiple entries with values near the maximum. Additionally to fewer low valued entries this should lead to a higher mean value. Since the different between the high and low values is high, the standard deviation can also be expected to be high. Mean $\bar{x}$ and standard deviation s are calculated as follows:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{3.3}$$

$$s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{3.4}$$

where n is the number of entries and x_i is the number of characters on a y-coordinate.

3.3.3 Permutation Entropy

Permutation Entropy belongs to the area of time series analysis. It captures permutation patterns in the time series, i.e. the order relations between individual values [3]. In our case the time series is an array of y-coordinates, which were sorted by the corresponding x-coordinate.

$$PE_{D,norm} = -\frac{1}{\log_2 D!}\sum_{i=1}^{D!} p_i \log_2 p_i \tag{3.5}$$

Equation (3.5) shows how to calculate the final value of the Permutation Entropy. However, there is a number of steps before, to get the values for p_i. D is called the *embedding dimension* and is used in the first step of calculating the Permutation Entropy. There is another parameter that is not present in the equation: τ, the *embedding time delay*. The time series is transformed into a matrix with D rows. The columns of the matrix represent the effect of a sliding window with size D moving over the time series. τ gives the step size and is often set to one, which means that the window just moves one value to the left in each step. The value for D is typically chosen to be between 3 and 7. The next step is to link the column vector to ordinal patterns. An ordinal pattern describes the relation between the values of the vector. For example, if $D = 3$ one such pattern would be $\pi_1 = \{0, 1, 2\}$. Each vector $[x, y, z]^T$ where $x < y < z$ would be mapped to this pattern. There are always $D!$ possible ordinal patterns. In the example where $D = 3$ there exist $3! = 6$ permutations. To get the relative frequencies p_i the number of times each ordinal pattern appears is counted and divided by the total number of vectors. Finally, Equation (3.5) can be used to compute the normalized Permutation Entropy.

The normalized PE value is restricted between 0 and 1. A smaller $PE_{D,norm}$ value indicates a more regular pattern in the time series. If $\tau = 1$ a result of 0 can only be achieved by a monotonically increasing or decreasing time series. A value close to 1 means that the time series is more noisy or random. The idea is to use Permutation Entropy to look for regular patterns in the y-coordinates that were sorted by their corresponding x-coordinates. For example, in a paragraph with four lines the first four values should be the y-coordinates of the first character in each lines. The next four values should be the coordinates of the second characters and so on. If embedded expressions are present this will distort the pattern somewhat. It should be more difficult to find some kind of regular pattern in displayed expressions. Therefore, the Permutation Entropy should be closer to 1 than for text examples.

To make this possible τ has to be chosen separately for every block. It should ideally represent the number of lines in a text block. To get this number the line values are split into two groups. Smaller values that most likely represent part of a mathematical expression, and large values that should represent whole lines of text. To find the ideal splitting point the values are sorted in ascending order. The largest difference between two consecutive values is chosen as splitting point. Then, the number of entries in the part with higher values represents the number of lines.

3.3.4 Font Information

Additionally to the position of each symbol in the PDF it is also possible to extract the font name of this glyph [18]. The name alone does not always provide much information. The font name can for example be 'Times-Roman', or 'Times-Bold', which indicates that the font *Times New Roman* is used. Sometimes a symbol has a font name like 'TXCMVS+CMMI8'. In this case 'cmmi8' is the name of the font and the six letters before it followed by the plus sign, indicate that this is a subset embedded font. All subset embedded fonts can be identified by six random characters and a plus sign that will precede the font name. This means that not the whole font was embedded in the PDF document. Instead only the information about the characters that were actually used is embedded to save space. There are certain fonts that are more often used in mathematical expressions. However, it is not guaranteed that each mathematical expression contains symbols in these fonts. Nevertheless, the fonts used in mathematical expressions are often not the same as the font of the majority of the document text. Since the font names can not be used as feature, a suitable value that is somehow linked to the font name has to be found.

One possible value is the relative frequency of the font. The majority of each document page will be written in a font like for example *Times New Roman*. Therefore, the relative frequencies of the other fonts are rather low. In each block the absolute frequencies for all font names are counted. Then, the font with the maximum number of appearances is chosen. For a text block this should be main font. For a displayed expression on the other hand, this will in most cases be a different one. The corresponding relative frequency from the whole document is used as feature for this block.

3.3.5 Unsuccesful Features

During our search for appropriate features for the blocks there were some ideas that did not work as expected. This section describes the ones that were more interesting, or where more time was invested.

Character Information

When humans look at an extracted block it is clear on first glance if this block is an displayed expression or not. There is an obvious difference between an ordinary sentence and a mathematical formula. One of our first ideas was to use the characters and symbols in a block in some way. In a paragraph of text there are mostly alphabetical characters, whereas a displayed expression should have mostly mathematical symbols. The approach to count the number of mathematical symbols was discarded very quickly, since it is difficult to say which symbols should count as mathematical. Many mathematical expressions include parenthesis or square brackets, but they can also appear in text. Plus and minus signs are less common in text, but firstly they are not necessary in every mathematical expression and secondly they can appear in embedded expressions in text blocks. The same is true for numbers.

A list of characters that appear in a certain block alone can unfortunately not be used as feature. Features in machine learning have to be numeric values since the underlying math will not work with anything else. It is however possible to convert characters into numbers and use them that way. Since there have to be the same number of features for each block, it is easiest to count the frequency of each character in a block. It is still important that the frequency numbers always correspond to the same characters. The list can not change between blocks, but has to include all characters that will ever appear from the beginning. A list of possible characters like ASCII (American Standard Code for Information Interchange) is an easy solution for this. ASCII includes Latin uppercase and lowercase letters, numbers, the most common special characters and some white space characters. It does not include all possible special characters, or letters from different languages. The absolute frequency of ASCII characters in a block can be used as feature.

However, this has several problems. First, not all symbols that appear in the PDF documents are part of ASCII. One example are Greek letters, which are used quite frequently in mathematical context. Unicode and its encoding schemes, like UTF-8, would include Greek letters among many other relevant characters. The newest version of Unicode[1] consists of 143,859 characters. As a result of this large number the list of absolute characters frequencies will be sparse for each block. If the feature vectors that represents blocks have over 140,000 entries but a substantially

[1] https://unicode.org/versions/Unicode13.0.0/

smaller number of data samples is available no good solution for the classification problem will be found. The second problem is that too much context is lost by only looking at the absolute frequency. Displayed expressions often contain words like "max", "sin" and other mathematical function names. However, this information can not be harnessed this way. Additionally, every symbol that appears in displayed expressions can also appear in an embedded expressions and therefore in a text block. A list of symbol appearances is therefore not the right kind of feature for our method.

Gini Coefficient

The Gini coefficient comes from economics and is usually used to represent the wealth distribution of a country. Generally, it is a measure of statistical dispersion and its definition is shown in Equation (3.6). The values can range between zero and one and are sometimes expressed as a percentage. A Gini coefficient of zero means an equal distribution is present, while a value close to one means that in a number of data samples one has a very high value while the others are nearly zero.

$$G = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} |x_i - x_j|}{2n \sum_{i=1}^{n} x_i} \tag{3.6}$$

The idea was to compute the Gini coefficient for the horizontal glyph densities of every block. Text blocks should have more entries with roughly equal values, which represent the lines in the paragraph. Displayed expressions should have a greater number of entries with rather low values, since mathematical equations often have a more two-dimensional structure. However, we found that there is not enough difference between the results for text blocks and displayed expression blocks. Too many text blocks include embedded formulas, which leads to too many entries with small values. Additionally, it does not make much of a difference that there are typically a higher number of symbols in a text line. In both cases a few values are much higher than the majority, which leads to similar Gini coefficients. Combined with outliers from both cases this feature would have complicated the classification and was therefore discarded.

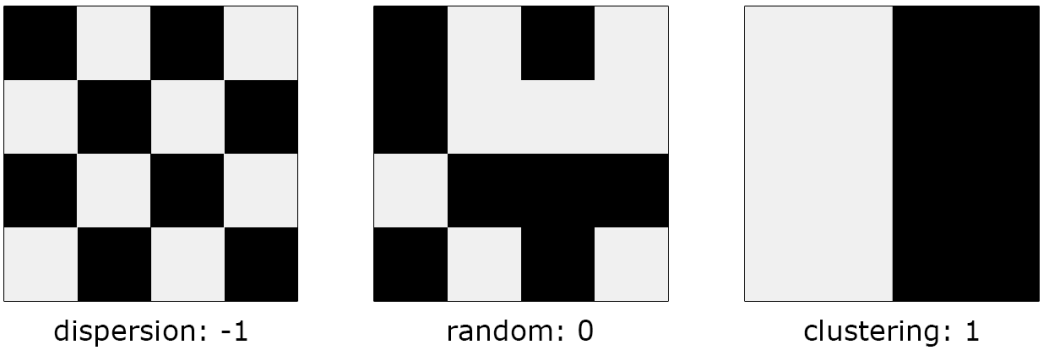

Figure 3.11: Moran's I examples: from total dispersion (-1) to perfect clustering(+1).

Moran's I

Moran's I [25] is a measure for spatial autocorrelation and measures the similarity between one object and its neighbors. Its values range from -1 to 1. A value near 1 means that in the data similar objects lie near each other (clustering). The opposite is true for -1, where dissimilar objects are next to each other (dispersion). If the Moran's I value is 0 there is no autocorrelation in the data (perfect randomness) which also means that the data might be independent. Independence of data is an assumption made by many machine learning algorithms. Figure 3.11 shows some examples how the values -1, 0 and 1 can be reached. Moran's I is calculated as follows:

$$I = \frac{N}{W} \frac{\sum_i \sum_j w_{ij}(x_i - \bar{x})(x_j - \bar{x})}{\sum_i (x_i - \bar{x})^2} \tag{3.7}$$

where x is the variable of interest, $\bar{x}$ is the mean of x, N is the size of x, w_{ij} is a spatial weight matrix with zeros in the diagonal and W is sum of all w_{ij}. The weight matrix should reflect assumptions about the data.

We experimented with different approaches for the weight matrix: setting it to 1 everywhere except on the diagonal, using automatic weight matrices from a library that includes an implementation for Moran's I and we also tried to reflect lines of text in the weight matrix. However, no significant difference between Moran's I values for text and formula blocks could be found. In most settings the results for

both text and formula block where between 0.8 and 0.9. Overall, the white pixels are rather clustered together in both cases, which makes Moran's I not a good feature to discriminate between text and displayed expressions.

To summarize, our method uses the following features:

- sparsity of the image of a block
- maximum, mean and standard deviation of the horizontal glyph density list
- permutation entropy
- the relative frequency of the most used font in a block

Figure 3.12 shows an example of two blocks and the corresponding features. The presented values are exactly how they are first calculated. After collecting the features for all PDF documents in the training set the data will be normalized to have zero mean and unit variance. This helps for example an SVM, which is only successful when the values of all features lie in a certain range. If one feature ranges between 10 and 100, and another has values between 0 and 1 many machine learning algorithms will not work as expected. However, since the normalized features are not easy to interpret for humans, the example shows the features before normalization.

where $\mathcal{R}$ is the given finite set of available paths between all source-destination pairs. Clearly, an optimal routing R^* for (4)–(5) exists since the set $\mathcal{R}$ is finite, and, given R, the objective function is continuous and the feasible set is compact. Define the Lagrangian as

$$L(R, x, p) = \sum_i U(x_i) + \sum_l p_l \left(c_l - \sum_l R_{li} x_i \right)$$

Sparsity	Max	Mean	StD	Perm. Entropy	Font	Label
0.87	83	37.67	37.68	0.81	0.90	0 (text)
0.95	18	5.5	5.80	0.74	0.03	1 (formula)

Figure 3.12: Examples for two blocks and the corresponding feature vector. The red box contains a text block while the green one contains a displayed expression. The table below shows the feature values for both blocks. The values will be normalized to have zero mean and unit variance before the data is used as input for a machine learning algorithm.

3.4 Machine Learning Algorithms

This section describes the basics of some machine learning algorithms. For our method we experiment with Support Vector Machines, Random Forests, Naive Bayes classifiers and Artificial Neural Networks. For the latter we do not dive into the topic of Deep Neural Networks since the time needed to train one is not comparable with the other machine learning algorithms.

3.4.1 Support Vector Machine

Support Vector Machines (SVMs) are machine-learning methods that can be used for regression, binary and multi-class classification [9]. They are a supervised learning method, which means that labeled training data is used. The goal of SVMs is to find the optimal hyperplane that separates the classes. In the most simplistic case this means finding the optimal line between two classes (see Figure 3.13). A line is optimal if it separates the classes with the highest possible margin.

In the following we formalize the concept of SVMs (cf.[9]): Let $x \in \mathbb{R}^{n \times d}$ be a dataset consisting of n vectors with d features and let $y \in \{-\ell, \ell\}^n$ be a set of n labels, where ℓ and $-\ell$ represent positive and negative classes respectively. Commonly in literature the labels are set as $\ell = 1$. The goal of an SVM is to construct a $(d-1)$-dimensional hyperplane that splits x into two groups x_+, x_- such that $\forall x_i \in x_+ : y_i = \ell$ and $\forall x_j \in x_- : y_j = -\ell$.

It is not always possible to separate two classes as clearly as in Figure 3.13, i.e. without errors. In this case it is inevitable to tolerate some misclassified samples. The separating hyperplane should then maximize the margins while the number of misclassified samples is minimized.

Such a hyperplane can be constructed by solving the following primal optimization problem

$$\textbf{minimize} \qquad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \tag{3.8}$$

$$\textbf{subject to} \qquad y_i(w^T\phi(x_i) + \beta) \geq 1 - \xi_i \tag{3.9}$$

$$\xi_i \geq 0 \quad \forall i \in 1, \ldots, n \tag{3.10}$$

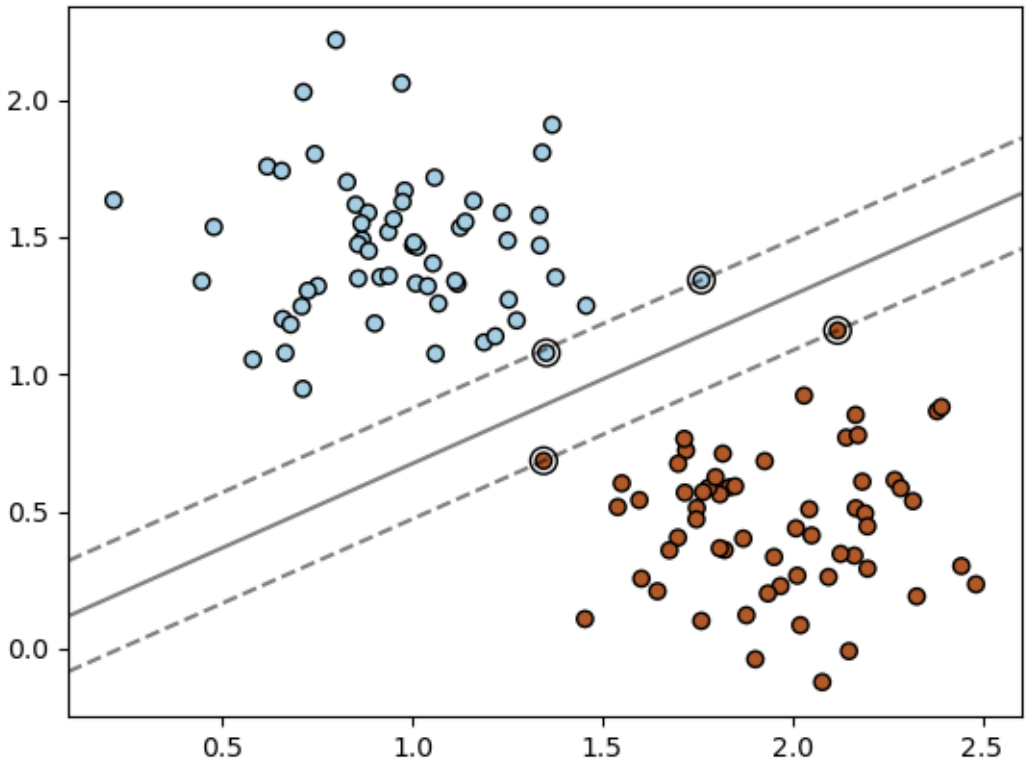

Figure 3.13: Binary classification with SVM. The solid line represents the optimal hyperplane between the two classes. The dots with additional circles around them are the support vectors. The dashed lines represent the margin.

where w are the weights, C is a penalty term, $\phi(\cdot)$ is a kernel function and β is a bias term. The non-negative variables ξ_i represent the distance of a sample from its correct margin boundary. With this hyperplane the SVM prediction

$$y' = \text{sign}(w^T \phi(x) + b) \tag{3.11}$$

should be correct in most cases. With a high value for C the penalty for misclassified samples is higher, which means that more training samples will be classified correctly. A low C will lead to more generalization in the decision function.

Since data is not always linearly separable, for example in an XOR-case, it is common to use kernel SVMs. A kernel is basically a mapping function that maps the data to another space. In this space it should be possible to separate the classes. The following tabel shows some common SVM kernels, the mathematical explanation and a list of parameters. The basic linear SVM is also included.

Kernel	**Formula**	**Parameters**
linear	$K(x, y) = x^T y + C$	C
polynomial	$K(x, y) = (\gamma x^T y + r)^d$	C, γ, degree d, offset r
RBF	$K(x, y) = \exp(-\gamma \|x - y\|^2)$	C, γ
sigmoid	$K(x, y) = \tanh(\gamma x^T y + r)$	C, γ, offset r

When it comes to imbalanced data, where one class has substantially more samples than the other, the success of a classical SVM may be limited [1]. Formula detection is typically an imbalanced classification problem since most PDF documents have more text than mathematical expressions. Usually, the negative samples are the majority class and positive samples the minority. Figure 3.14 shows that more samples are classified as negative if both classes are treated equally. As a result the accuracy is high due to the high number of correctly classified negative samples. However, many misclassifications happen for the positive class. To improve the results of the positive class, class weights can be used. Basically, class weights make the positive class more important and the penalty for misclassification in this class higher.

The results of the class weights 1:10, which means that misclassification penalty for the positive class is ten times higher than for the negative class, can be seen in the orange line in Figure 3.14. The decision boundary has moved closer toward the center of the negative class and almost all orange dots lie on the correct side. The

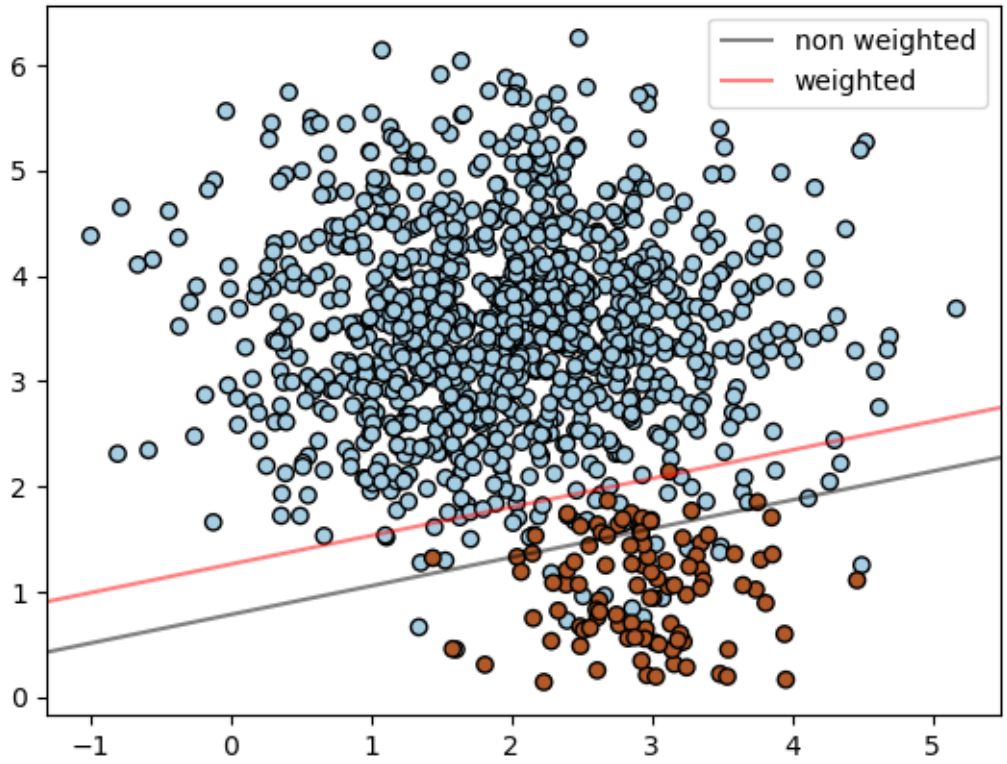

Figure 3.14: Imbalanced data since there are ten times more negative samples (blue) than positive ones (orange). The classification in the non weighted case is biased towards the majority class, which means that the gray decision boundary is further from the center of the majority class. In the weighted case the misclassification penalty for a positive sample is ten times as high as for a negative sample. Through this the orange decision boundary moves towards the majority class and more positive samples are correctly classified.

class weights can be chosen as an inverse of the class imbalance ration [1]. For the example, where the negative class has ten times more samples, this means setting the weights 1 to 10 for the positive class.

3.4.2 Random Forests

A random forest can be used for classification or regression. Random forests originated from decision-tree classifiers, which were extremely fast at classification but could not be grown to arbitrary complexity without overfitting on the training

data. Tin Kam Ho [20] was the first to use multiple trees (a forest) to overcome the generalization bias of single trees. A random forest classifier consists of an ensemble of trees, that vote for the most popular class. The trees are all slightly different from one another and should be relatively uncorrelated. The idea behind random forest is that even if a few trees make the wrong decision, the majority will vote for the right class. Since the Strong Law of Large Numbers holds [6] even a large number of trees can be used. If other hyperparameters of the random forest are tuned accordingly the large number of trees does not lead to overfitting.

The term 'random' in random forest comes from the fact that each tree gets a random subset of features. This influences the trees at each node, when the best feature to split the node is chosen. A classical decision tree that is not part of a random forest has all features available, but the random forest tree only has a subset. It cannot select the overall best splitting criterion and instead needs to rely on the fewer features it has access to. This ensures that multiple trees are different from each other and will ideally not produce the same outcome. Depending on the random forest implementation, sometimes the nodes also only get a random subset of all features of the tree. This adds a second source of randomness.

We use the scikit learn implementation for random forests [2]. In this implementation it is possible to specify the number of trees, the maximal depth of trees and the class weights among other parameters. The outcome is the average of all predictions given by the individual trees. This implementation also offers the option that each node has only access to a subset of the features. It is possible to use for example a specified fixed number or fraction of features, the square root of the total number of available features or all features of a tree.

An example for a single decision tree from a random forest is shown in Figure 3.15. At each node the value of a feature decides which path to take. The leafs give the resulting class. In this examples colors are used to visualize the class (here orange for text and blue for formula) and the certainty behind the decision. In a node the first line describes the split criterion, for example $mean \leq -0.5$. The word "gini" in the second line refers to the Gini impurity measure, which is one method to choose the best split criterion for a node. A lower Gini impurity value corresponds to a lower probability for misclassification. The number of samples show how many samples passed trough that node. The root node has the total number of samples.

[2] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

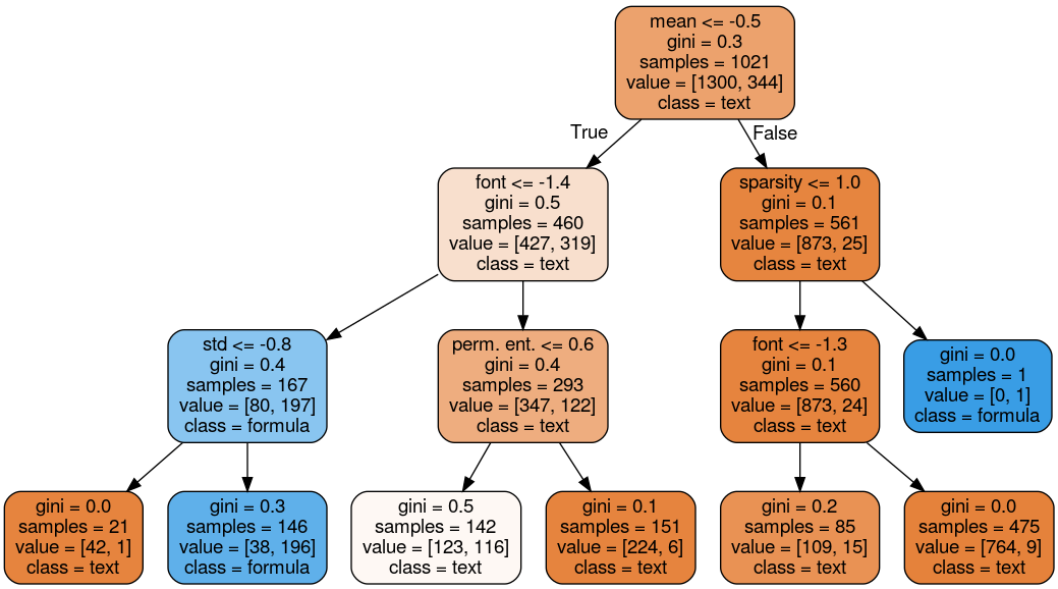

Figure 3.15: Example of small decision tree. The maximal depth of the tree was set to 3. Orange nodes are classified as text, blue ones as displayed expression. The one white node is classified as text but there is not much certainty behind it.

3.4.3 Naive Bayes Classifier

For a Naive Bayes classifier Bayes' theorem is applied while a strong independence between the features is assumed. Since features rarely are all independent this method is called "naive". Nevertheless, Naive Bayes often performs well when compared with other machine learning algorithms. Bayes's theorem can be seen in equation (3.12)

$$p(C_k|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|C_k)p(C_k)}{p(\boldsymbol{x})} \tag{3.12}$$

where $\boldsymbol{x} = (x_1, ..., x_n)$ is a feature vector and C_k is one of K classes. Since the features are assumed to be independent it follows that

$$p(x|C_k) = \prod_{i=1}^{n} p(x_i|C_k) \tag{3.13}$$

According to the work of Rish [29] a Naive Bayes classifier also performs well for functionally dependent features, where the mutual information $I(x; y|C_k)$ is at its maximum. A Naive Bayes classifier can easily be constructed from the probability model above and looks as follows:

$$y = \text{argmax}_{k\in\{1...K\}} p(C_k) \prod_{i=1}^{n} p(x_i|C_k) \tag{3.14}$$

where y is the assigned class label. The most commonly used variants of Naive Bayes are: Multinomial Naive Bayes, Bernoulli Naive Bayes and Gaussian Naive Bayes. Multinomial Naive Bayes is most often used for text classification. Therefore, the features should usually be discrete values that represent word occurrences in text, but term frequency-inverse document frequency (tf-idf) vectors are also acceptable. The likelihood $p(x|C_k)$ is given by

$$p(x|C_k) = \frac{(\sum_i x_i)}{\prod_i x_i!} \prod_i p_{ki}^{x_i} \tag{3.15}$$

A Bernoulli Naive Bayes classifier is similar to the multinomial variant and is also suited for text classification. Instead of discrete value it takes binary values as input. This changes $p(x|C_k)$ as follows

$$p(x|C_k) = \prod_{i=1}^{n} p_{ki}^{x_i}(1 - p_{ki})^{(1-x_i)} \tag{3.16}$$

Another popular variant, that is not so limited in its classification task is the Gaussian Naive Bayes classifier. This classifier assumes that the likelihood of features for each class is Gaussian.

$$p(x = v|C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp(-\frac{(v - \mu_k)^2}{2\sigma_k^2}) \tag{3.17}$$

The variables μ_k and σ_k^2 are parameters of the Gaussian distribution. Although our features are most likely not independent, experimenting with Naive Bayes classifiers is an interesting endeavor, since there are many reported cases with dependent features where it produced good results.

3.4.4 Artificial Neural Networks

Artificial Neural Networks are another commonly used type of supervised machine learning. There are many variants and special areas of neural networks that can be described in detail. However, this section will only outline the basic aspects of these machine learning algorithms. Neural networks are networks with an input layer, an output layer and an arbitrary number of hidden layers between them. The units within a layer are called neurons. They take a number of values as input, perform a mathematical function on it and produce one value as output. The output of a single neuron can be described as

$$y = f(wx + b) \tag{3.18}$$

where x is the input vector, w a weight vector and b the bias. An activation function f is applied to the sum of these vectors to produce the output y. Options for the

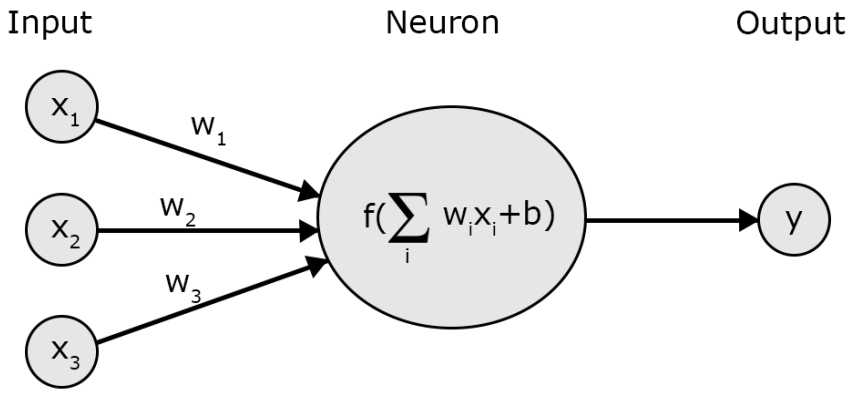

Figure 3.16: A single neuron in a hidden layer. It receives some inputs (x_i) with some weights (w_i) and applies the activation function f.

activation function are for example Rectified Linear Units (ReLU) or sigmoid. ReLu in its simplest form sets values smaller than 0 to 0 and leaves positive values as they are. Figure 3.16 shows an illustration of a neuron.

An important step for neural networks is backpropagation. After the information has flown from the input layer through the whole neural network to the output layer the error between actual output and desired output is calculated. This error is then backpropagated through the network to adjust the weights [34]. Through this the neural network learns. The process is usually repeated until the change in weights converges.

3.5 Method Overview

In this section we will recap the most important steps and aspects of our method. First a PDF document and its rendered image are processed. The coordinates, content and font name are extracted for each character object. From the PDF coordinates and the image, which provides the actual character size, the scale parameter is computed. Then the horizontal and vertical whitespace is analyzed to find blocks. For this a vertical Gaussian filter is applied to the image to find all columns. Afterwards, a horizontal Gaussian filter with scale as σ blurs the original image and in each

column the horizontal dark pixel sum is calculated. A fixed threshold helps to find blocks, which are defined by larger whitespace areas.

For each block the following features are computed: sparsity, maximum, mean, standard deviation, permutation entropy and relative frequency of most used font. Maximum, mean and standard deviation are calculated from the horizontal glyph density lists which count how many characters lie on a certain y-coordinate. A similar approach is used for the permutation entropy which is calculated on a list of y-coordinates that was sorted by x-coordinate. In other words the list holds the y-coordinates of each character from top to bottom, left to right. For the font feature the relative frequencies of all fonts are calculated for the whole PDF. Then the frequency value that corresponds to the font which was most used in a block is selected.

Before evaluation can begin the blocks need correct labels. From XML ground truth files the coordinates of all displayed expressions are extracted and compared against the coordinates of our blocks. If a block and a ground truth area overlap for at least 80%, the block gets the label 1. All other blocks are labeled as 0, which means they are text blocks. The features from all PDFs in the training set are normalized before giving them to machine learning algorithms. A smaller subset is used as validation set. We used SVMs, random forests, Naive Bayes classifiers and Neural Networks for our experiments.

4 Evaluation

In this chapter the evaluation of our formula detection method is described. We first present the concrete setup of our evaluation by discussing the dataset, ground truth handling, performance measures and baselines. Then, the results of a preliminary hyperparameter analysis on a validation dataset are discussed, before finally the actual test results are presented and discussed.

4.1 Setup

To evaluate the developed method the results of specific measures is compared against the results from literature methods. To be able to compare the results it would be ideal to find available code of different methods. Then, the measure results can be verified and a detailed evaluation is possible. When there is no source code available it is preferable that the method is described in enough detail that it is possible to re-implement it. Both of these scenarios are considered as reproducible. For reproducible methods it is possible to verify the results and calculate additional measures for evaluation. The next best circumstance would be methods that were evaluated on the same test data that we used. If a work used a different dataset but one that is publicly available it is possible to run our method on this data and thereafter compare the new results. In the case that unknown data was used it is not meaningful to compare results.

In Table 4.1 a number of state of the art methods are listed. Since the methods themselves do not all have names that were mentioned in the works and the titles are too long, only the author names are listed. The corresponding work is referenced. The column ”Category” describes in one or two keywords how the method works. ”Dataset” gives the name of the PDF document collection that was used to gain the

Name	Category	Dataset	Reproducible
Xing Wang, Jyh-Charn Liu [32]	unsupervised	Marmot	no
Gao et al.[13]	Deep Neural-Network	Marmot + other	no
Wang et al. [33]	unsupervised	Marmot	no
Phong et al. [26]	SVM	random from Harvard and Infty database	no
Wei-Ta Chu, Fan Liu [8]	SVM	custom	no

Table 4.1: Potential methods for comparison

presented results. The last column, "Reproducible" states whether we consider the method as reproducible or not.

Out of all listed methods, the majority use the Marmot dataset. This is fortunate, because it is the dataset that we intended to use for the evaluation of our method. Gao et al. [13] used the Marmot dataset as test set, but the training was done on a custom dataset. Since they used a deep neural network, the Marmot dataset would have been too small for training. None of the works that were found are easily reproducible. While some [32, 33] are described in great detail, some crucial information is always missing. This prevented a re-implementation of the methods.

4.1.1 Dataset

For the evaluation the Marmot dataset [1] was used. It consist of 400 PDF documents with corresponding image and ground truth files. The document pages were selected from a variety of sources. This includes journals, books, conference proceedings and technical reports. Further, the documents where created in different years, with different PDF versions, come from different domains (including mathematics, computer science and physics) and have different page layouts. In total, 1575 displayed expressions (which are called isolated formulae here) and 7907 embedded expressions appear in the dataset. Each document page is available as PDF file and as rendered image. The ground truth is represented in XML files. Position and other relevant information about an isolated or embedded expression are stored in appropriate tags, as shown in Figure 4.1. The figure shows the detailed attributes only for an isolated formula, since an embedded formula has the same ones.

[1] http://www.icst.pku.edu.cn/cpdp/sjzy/index.htm

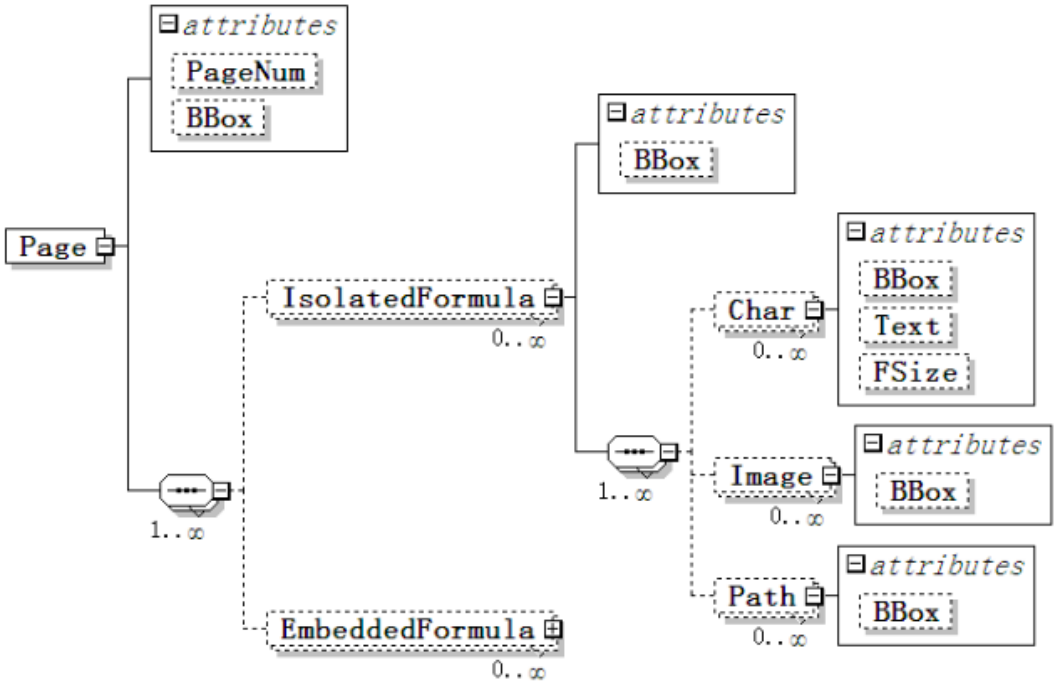

Figure 4.1: Visualization of the XML schema of ground truth for the Marmot Dataset [22]

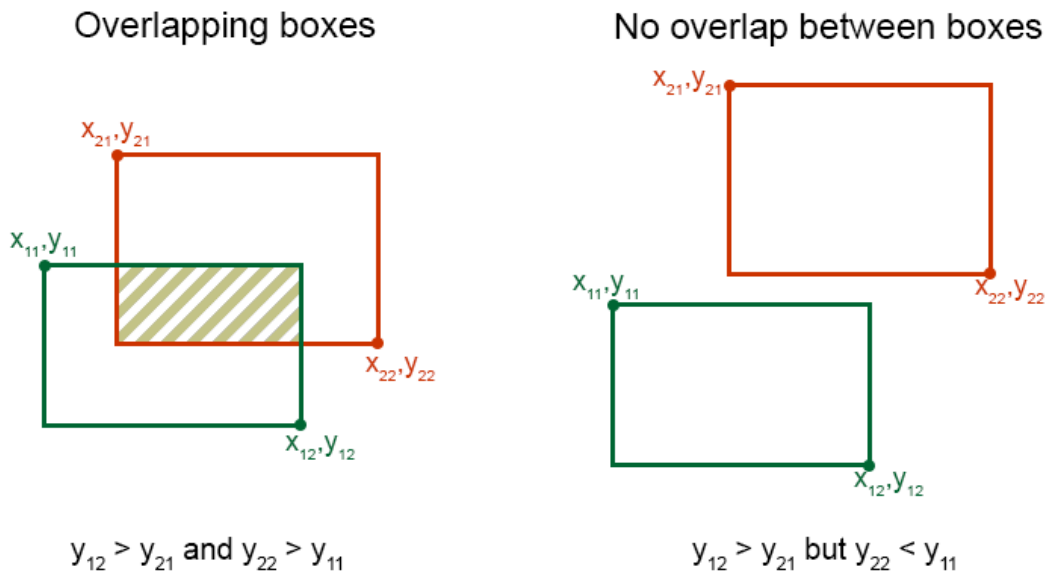

Figure 4.2: In the example on the left the two boxes overlap, therefore it is possible to compute the overlap percentage. On the right side the bottom border of the red box is above the top border of the green one and they do not overlap. This can be tested by checking if $y_{22} < y_{11}$. Since the the order of the boxes could be reversed (green box above the red one), also test if $y_{12} < y_{21}$. If either of these conditions are true, the boxes do not overlap. In other scenarios it is necessary to also check if one box is to the left of the left edge of the other box by comparing the x coordinates.

4.1.2 Ground Truth

The evaluation data set includes a corresponding XML file for each PDF document. Every mathematical expression is listed as isolated or embedded formula and has an attribute ”BBox” (short for bounding box) which stores the PDF coordinates of the bounding box around the mathematical expression. The coordinates are written as hexadecimal values and first have to be converted to double values. Next, the PDF coordinates are transformed into image coordinates so that they can be compared against the blocks of the image. As a reminder, in an image the point of origin is at the top left corner. Consequently the y-coordinates start with 0 at the top of the image, and increase toward the bottom edge.

The image coordinates x_1, y_1 give the top left corner of a bounding box and x_2, y_2

give the bottom right corner. For two boxes x_{11}, y_{11} and x_{12}, y_{12} are the coordinates of the first box and x_{21}, y_{21} and x_{22}, y_{22} the coordinates for the second box, the ground truth box. To check if two boxes can overlap, first the y-coordinates are compared. If y_{12}, the bottom edge of the first box, is smaller than y_{21}, the upper line of the second box, the boxes can not overlap since the first one lies completely above the second one. The same is true if y_{22} is smaller than y_{11}. Figure 4.2 shows this example. We also need to compare the x-coordinates to check if the right edge of the leftmost box lies to the left of the other box. If the boxes overlap to some degree, the size of the overlapping area is calculated. The equation to compute the overlapping area of two boxes is given by (4.1).

$$\text{overlap} = (\min(x_{12}, x_{22}) - \max(x_{11}, x_{21})) * (\min(y_{12}, y_{22}) - \max(y_{11}, y_{21})) \quad (4.1)$$

Basically, it is just the equation to compute the area of a rectangle: multiply the width of the rectangle with its height. Since we are working with coordinates instead of lengths, some additional calculations are necessary to obtain the dimensions of the overlap rectangle. To get the width, subtract the x-coordinate of the right edge from the x-coordinate of the left one. To get the height, to the same with the y-coordinates. The left side of Figure 4.2 shows two overlapping boxes. In this example we would compute $x_{12} - x_{21}$ for the width and multiply it with $y_{22} - y_{11}$. However, if the green box would lie more to the right, computing $x_{12} - x_{21}$ would not always give us the overlapping area. Therefore, Equation (4.1) takes the leftmost right edge of both boxes ($\min(x_{12}, x_{22})$) and the rightmost left edge ($\max(x_{11}, x_{21})$) to compute the width of the area. It is important to verify that the boxes have an overlap area before computing it, because the equation gives a result regardless. If the boxes do not overlap the result will be some area between them.

Since we can not gain much information from the absolute overlapping area alone, we further compute the overlap percentage by dividing the overlapping area by the size of the block area. The last step is to assign labels to each block in the image. If a block overlaps to 80% with one of the ground truth bounding boxes, it is labeled as displayed expression. It is not practical to try to achieve 100% accurate box coordinates. First of all, the ground truth coordinates refer to the location in the PDF file. After transforming the coordinates into image coordinates most of them will not be integer values, which means they have to be adjusted. This makes

a perfect overlap almost impossible. Secondly, if the detected box is a bit larger than the displayed expression the additional area will most likely just be white.

In some cases our algorithm found multiple blocks where the ground truth lists only one displayed expression. This can happen when the distance between parts of the displayed expression is especially large. Since our blocks contain only mathematical content and lie within the ground truth area, this is not a problem. The overlap percentage will be 100% for every block that is enclosed in the ground truth area and therefore all such blocks are labeled as displayed expression.

The reverse, where one of our blocks contains multiple ground truth areas, can also occur and needs to be handled differently. If two or more displayed expressions are very close together they are not always detected individually. The overlap percentage will be lower than 80%, but if only mathematical content is in our block, it should still be labeled as displayed expression. Therefore we search for all ground truth areas that lie inside our block and then check if the distances between them are small. If there is a text line in the block, the distance between two ground truth areas should be bigger than $scale * 1.5$, else the block can be labeled as displayed expression.

Unfortunately it can still happen that a box will contain a text block and a displayed expression because the gap between the two was not large enough. In this case the box is substantially larger than the displayed expression and the overlap with the corresponding ground truth coordinates will be smaller than 80% of the block area. Therefore the box will not be labeled as displayed expression. This is not ideal since the mathematical expression was not found. Nevertheless, it is better than treating the larger block like a displayed expression anyway. Our goal is to find the bounding boxes of mathematical expressions and if a text paragraph is included in this bounding box this defeats the purpose.

4.1.3 Measures

In works on formula detection, the most commonly used measures are *precision* and *recall*. They are used instead of *accuracy*, which is defined as all correctly predicted samples divided by the total number of samples. The accuracy measure is unreliable for imbalanced data sets because is does not distinguish between the numbers of correct labels of different classes [16]. It could be high when almost no positive

Actual \Predicted	**Text**	**ME**
Text	True Negative	False Positive
ME	False Negative	True Positive

Table 4.2: Confusion Matrix for binary classification. The two classes are *Text* and *Mathematical Expression (ME)*. Since the goal is to find mathematical expressions, ME is the positive class.

samples were correctly predicted. For example, if the test data consists out of 80% negative samples and a method classifies everything as negative, the accuracy would be 80%.

The precision metric describes how precise or accurate the method is in identifying positive samples. Precision is defined as

$$\text{precision} = \frac{tp}{tp + fp} \tag{4.2}$$

where ”tp” stands for true positive and ”fp” for false positive. In the formula detection scenario true positive means that a mathematical expression was correctly recognized. False positive means that instead of a formula, normal text was detected. This is also shown in Table 4.1.3. Precision is therefore the fraction of correctly predicted positive samples out of all predicted positive samples.

Precision only considers true positive and false positive samples. It does not take missed positive samples into account. Therefore, a second measure is needed: recall, which is defined as

$$\text{recall} = \frac{tp}{tp + fn} \tag{4.3}$$

where ”fn” stands for false negative. For formula detection a false negative is a mathematical expression that was missed. Therefore, the denominator gives the number of actual positive samples. Recall then calculates what fraction of actual positives were found by the method.

Depending on the scenario it can be more important to achieve high precision or high recall. In formula detection one can argue that it is more important to find all mathematical expressions instead of not misclassifying any normal text. Mathematical expressions should be located so that they can later be extracted

by another method. If a text paragraph is labeled as mathematical expression it will also be extracted, but the other method can most likely handle normal text. If instead a formula is missed altogether it can not be extracted and the information that this part of the PDF holds is lost. This reasoning would mean that a high recall value is more important. However, it is always ideal to have both, high precision and high recall.

Another measure that is almost always used is the F1 score. It combines precision and recall to give an overall score of the methods classification performance.

$$\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{4.4}$$

The F1 score is the harmonic mean between precision and recall. Like accuracy, the F1 score is a measure that can be used alone, while precision and recall should generally not be mentioned without the other. However, accuracy can be misleading when the classes have an uneven distribution. In this case the F1 score gives a more meaningful result. Since an average PDF document will have more text than mathematical expressions, formula detection definitely has an uneven class distribution.

All measures that were described so far do not include the number of true negatives. The Matthews Correlation Coefficient (MCC) [24] takes into account all classification results from Table 4.1.3. While precision, recall and F1 score return values between 0 and 1, the MCC returns a value between -1 and 1. Perfect prediction is still indicated by a the value 1 and 0 means random prediction. If the result is -1 the prediction is always wrong, but in this case the labels could just be inverted to be correct. The coefficient is defined as,

$$\text{MCC} = \frac{tp * tn - fp * fn}{\sqrt{(tp + fp)(tp + fn)(tn + fp)(tn + fn)}} \tag{4.5}$$

MCC is especially useful in scenarios were the two classes are imbalanced (one class has substantially more samples than the other) [5], but both are equally important. Precision, recall and consequently the F1 score all focus on the positive class. It can happen that all three measures give nearly perfect values, but if the classes are switched the results become much lower. For example, the classification results

can look like this: $tp = 50, fp = 4, fn = 3, tn = 1$. In this case precision is $50/(50 + 4) = 0.926$ and recall is $50/(50 + 3) = 0.943$. These are good results. If the positive and negative class are switched, this leads to $tp = 1, fp = 3, fn = 4, tn = 50$. Now, precision has a value of $1/(1+3) = 0.25$ and recall $1/(1+4) = 0.2$ which are clearly not good results.

Matthews correlation coefficient gives the same result regardless of which class is labeled as positive class. A high MCC value shows that both classes are predicted well. For the example above the result is 0.159, because the classifier does not perform well for one of the classes. MCC is a useful measure for imbalanced classes or if both classes should be treated as equally relevant.

In [22] Lin et al. introduce additional evaluation measures for formula detection. To better understand strengths and weaknesses of different algorithms additional recognition results are used. In total eight possible outcomes are defined, as shown in Figure 4.3.

- **Correct** The detected region is the same as exactly one region in the ground truth set. N^R_{cor} is the number of correct results.
- **Missed** A formula region from the ground truth set does not match any detection result. N^G_{mis} is the number of missed formula regions.
- **False** The detection result does not match any region of the ground truths. N^R_{fal} is the number of false results.
- **Partial** The detected region is part of exactly one ground truth region and does not include any non-formula text. N^R_{par} is the number of partial results.
- **Expanded** At least one formula region from the ground truth set is covered entirely by the detected region, but it also includes non-formula regions or parts of other formula regions. N^R_{exp} is the number of expanded results.
- **Partial & Expanded** Combination of partial and expanded - the detected region covers part of one formula but also non-formula regions or part of another formula region. $N^R_{p\&e}$ is the number of such results.
- **Merged** The detection result covers two or more regions of the ground truth set completely. No non-formula region is involved. N^R_{mer} is the number of merged results.
- **Split** There is more than one detected region overlapping with the ground truth but the formula is covered entirely. N^G_{spl} is the number of formulas that are split.

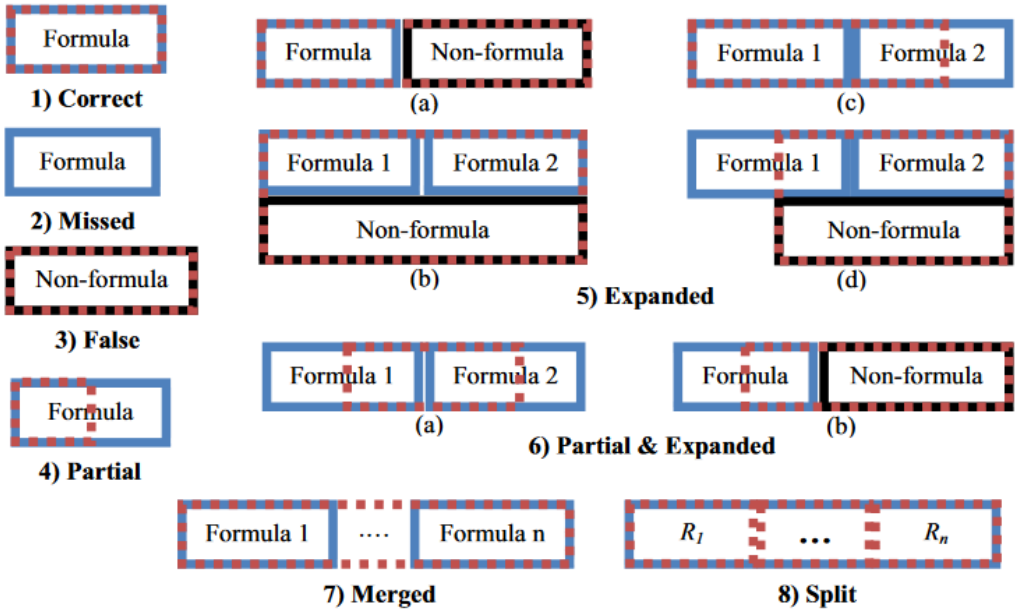

Figure 4.3: The eight possible detection results (cf. [22]). One, two and three are the classical results: either a formula is found, missed or text is wrongly classified as formula. In the other five cases the mathematical expression was found to some degree. Either not all of it was found, or the bounding box includes some additional areas.

From these different detection cases an overall score is computed.

$$\text{score} = \frac{1}{WN} \cdot \begin{pmatrix} W_{cor}N^R_{cor} - W_{mis}N^G_{mis} - W_{fal}N^R_{fal} + W_{par}S_{par} + \\ W_{exp}S_{exp} + W_{p\&e}S_{p\&e} + W_{mer}S_{mer} + W_{spl}S_{spl} \end{pmatrix} \tag{4.6}$$

$$S^A_{par} = \sum_{i=1}^{N^R_{par}} \frac{Area(R_i \cap G_j)}{Area(G_j)} \tag{4.7}$$

$$S^S_{par} = \sum_{i=1}^{N^R_{par}} \frac{Symbol(R_i \cap G_j)}{Symbol(G_j)} \tag{4.8}$$

$$S^A_{exp} = \sum_{i=1}^{N^R_{exp}} \frac{Area(R_i \cap G_j)}{Area(R_i)} \tag{4.9}$$

$$S^S_{exp} = \sum_{i=1}^{N^R_{exp}} \frac{Symbol(R_i \cap G_j)}{Symbol(R_i)} \tag{4.10}$$

$$S^A_{p\&e} = \sum_{i=1}^{N^R_{p\&e}} \frac{Area(R_i \cap G_j)}{Area(R_i)} \tag{4.11}$$

$$S^S_{p\&e} = \sum_{i=1}^{N^R_{p\&e}} \frac{Symbol(R_i \cap G_j)}{Symbol(R_i)} \tag{4.12}$$

$$S_{mer} = \sum_{i=1}^{N^R_{mer}} \frac{1}{N_{R_i \cap G}} \tag{4.13}$$

$$S_{spl} = \sum_{j=1}^{N^R_{spl}} \frac{1}{N_{G_j \cap R}} \tag{4.14}$$

4.1.4 Baselines

In addition to comparing with the results of the above literature methods, we compare our method with trivial baselines. Any reasonable formula detection method should outperform these baselines. The baselines that our method will be compared against are:

- **negative** always predicts the majority class for every sample. This means that the classification result will always be the negative class.
- **stratified** random prediction that respects the class distribution of the training set.
- **uniform** the predictions are uniformly random.

The ”negative” baseline is a very simple one that lends itself to binary classification tasks. However, if every sample is assigned to the negative class the number of true positives and false positives will be zero. Therefore, none of the measures that were previously described can be calculated. The only available measure is accuracy. To be able to compare other measures like the F1 score, the additional baselines ”stratified” and ”uniform” are included. Both randomly label the samples and therefore it should be possible to calculate precision, recall and F1 score. Both baselines are initialized with a random seed.

4.2 Hyperparameter Analysis

The validation set was used to adjust the hyperparameters of the proposed method. Each machine learning algorithm has different hyperparameters that need to be adjusted to get the best results possible.

4.2.1 Support Vector Machine

Depending on the SVM kernel function there are different hyperparameters. For example with a polynomial kernel, the polynomial degree can be adjusted. For the kernel parameters grid searches were performed. This means that for each hyperparameter a list of values is chosen and the result of each combination is computed. To get a reasonable list of values we take the default value for the specific

parameters and choose additional values that are one magnitude smaller and larger. After the initial search we might perform a finer grid search with values closer to the best value from the first search. For example if 1 performed best in the initial grid search, we can try again with $0.5, 0.8, 1, 1.2, 1.5$. The grid searches will not be performed indefinitely long, after one or two finer searches the best parameter value will be chosen. Since the goal is to avoid overfitting it is not necessary to test every possible value and choose the absolute best. To make the results more manageable, only the F1 scores are reported.

The linear SVM has only one parameter: C. The default value of C is 1 and a lower C leads to more generalization. A higher value for C means that the penalty for misclassification increases. If misclassification is avoided the predicted results get more precise but this can also lead to overfitting. However, a linear kernel can not adapt very much so increasing C becomes pointless above a certain threshold.

C	0.1	0.3	1	10	100
F1	74.11	**74.7**	74.52	74.16	74.25

Table 4.3: Hyperparameter for linear SVM kernel: C. The table shows F1 scores in percent for different C values. Best result for $C = 0.3$, which was found after a finer search between 0.1 and 1.

Table 4.3 shows the F1 results for $C = \{0.1, 0.3, 1, 10, 100\}$. The difference between the results is not very large with only a 0.6% gap between best and worst result. The best F1 score is achieved with $C = 0.3$, which was found after a finer grid search. It is only slightly better than for the default value 1. Therefore $C = 1$ appears to be also a good choice.

The Radial Basis Funcion (RBF) kernel has two parameters: C and γ. C has the same function for each kernel, therefore the same list of values was used. The parameter γ represents the amount of influence that a single training sample has. Low γ values mean that the influence reaches far, while high values mean other samples are only affected if they are very close. With a too small γ value a model becomes too constrained, to the point where it behaves like a linear model. On the other hand too large γ values will lead to overfitting. The default value for γ is "scale" which is defined as $1/(n\text{Var}[X])$, where n is the number of features and $\text{Var}[X]$ the variance of the training data. For our data it is 0.16.

The results for different parameter configurations are shown in Table 4.4. There are

C	0.1	1	1.2	10	100
$\gamma = 0.01$	0	73.87	73.55	75.34	79.03
$\gamma = \text{scale}$	73.85	80.48	**80.63**	79.85	79.27
$\gamma = 1$	79.78	79.66	79.18	79.56	75.47
$\gamma = 10$	41.61	75.97	76.12	73.23	71.98

Table 4.4: Hyperparameters for RBF kernel: C and γ. The table shows F1 scores in percent for different configurations. $C = 1.2, \gamma = \text{scale}$ gives the best result.

two parameter value pairs that achieve rather high results. The highest F1 score of 80.63% is the result of $C = 1.2, \gamma = \text{scale}$. The two default values $C = 1, \gamma = \text{scale}$ performed only slightly worse, while no other configuration reaches 80% F1 score.

A polynomial kernel has 4 hyperparameters that can be adjusted. These parameters are C, γ, degree and the coefficient r. To keep the parameter space small, only the two most significant parameters were selected: γ and degree. For the polynomial degree the values 2 to 5 were chosen, which also includes the default value 3. A degree of 1 is redundant since this would lead to a linear solution. For C and r the default values are used, which are 1 and 0 respectively. $C = 1$ has achieved good results for linear and RBF kernels so it seem reasonable to choose it here.

degree	2	3	4	5
$\gamma = \text{scale}$	73.14	68.65	66.39	66.77
$\gamma = 1$	78.23	78.99	76.89	76.49
$\gamma = 2$	**81.21**	78.86	76.21	72.08

Table 4.5: Two out of four hyperparameters for polynomial kernel. C is set to 1 since this seems to be the best setting for other kernels. r is set to zero. The table shows the F1 scores in percent for different degree and γ configurations. The best result is achieved for $\gamma = 2$, degree $= 2$.

The resulting F1 scores are shown in Table 4.5. The best value 81.21% is the result of degree $= 2$ and $\gamma = 2$. The results for degree 4 and 5 are always smaller than for degree 2 and 3. Surprisingly, in this case the default value for γ performed substantially worse than higher values. In the best case (degree 5) there is still a 5% difference between the F1 score for $\gamma = \text{scale}$ and $\gamma = 2$.

The sigmoid kernel has three hyperparameters: C, γ and the coefficient r. After a few preliminary tests with different values for r, the default value 0 appeared to be a good choice. Therefore, the grid search was only necessary for C and γ.

C	0.05	0.09	0.1	1	10
$\gamma = 0.05$	0	60.83	66.77	74.0	65.42
$\gamma = 0.1$	59.44	73.93	74.37	66.5	64.02
$\gamma = \text{scale}$	72.94	**74.64**	74.57	64.72	61.93
$\gamma = 0.5$	25.51	26.26	26.62	27.14	27.28

Table 4.6: C and γ hyperparameter for sigmoid kernel. The coefficient r is set to 0. The table shows the F1 scores in percent for different configurations. $C = 0.09$ and $\gamma = \text{scale}$ give the best result.

Table 4.6 shows the results from different configurations. The results for all entries where $\gamma = 0.5$ are especially low. The best F1 score is achieved with $C = 0.09$ and $\gamma = \text{scale}$, but also $(0.1, \text{scale})$ and $(0.1, 0.1)$ give reasonably good values. Instead of $C = 0.01$ we listed results for $C = 0.05$ to have a comparison with the best C. The former value only produced 0 as result.

Validation Results

In table 4.7 we report the measured results with different SVM kernels and varying class weights. The four available kernels (linear, RBF, polynomial and sigmoid) were each used with four different class weights. In most documents from the dataset the ratio between text and formula blocks seems to be between 1:3 and 1:5. Therefore, the four different weights that were chosen are: no weights, 1:3, 1:4 and 1:5. Precision, recall, F1 score and MCC were all listed as percent values with two decimal places. The hyperparameters were chosen as follows:

- **linear** $C = 0.3$
- **RBF** $C = 1.2, \gamma = \text{scale}$
- **polynomial** $C = 1, \gamma = 2, \text{degree} = 2, r = 0$
- **sigmoid** $C = 0.09, \gamma = \text{scale}, r = 0$

The polynomial kernel without weights achieved the highest precision, F1 score and MCC values. The highest recall value of 94.63% is the result of a sigmoid kernel with the class weights 1:5. The RBF kernel with no weights achieved the second highest F1 score and MCC value. The sigmoid and linear kernel give overall the lowest results, even though the highest recall value was reached with a sigmoid kernel. The F1 score is more meaningful than just a recall value and all F1 scores for

SVM Kernel	Class Weights	Precision	Recall	F1	MCC
linear	1:1	74.6	77.85	76.19	69.05
linear	1:3	68.22	88.59	77.08	70.28
linear	1:4	66.26	90.27	76.42	69.6
linear	1:5	65.38	91.28	76.19	69.43
RBF	1:1	81.51	82.89	82.2	76.93
RBF	1:3	74.64	86.91	80.31	74.34
RBF	1:4	73.6	87.92	80.12	74.13
RBF	1:5	73.26	88.26	80.06	74.07
polynomial	1:1	**81.82**	84.56	**83.56**	**78.15**
polynomial	1:3	74.01	87.92	80.37	74.45
polynomial	1:4	72.38	87.92	79.39	73.2
polynomial	1:5	71.58	87.92	78.92	72.58
sigmoid	1:1	69.69	82.55	75.58	68.05
sigmoid	1:3	65.36	89.26	75.46	68.29
sigmoid	1:4	63.16	92.62	75.1	68.25
sigmoid	1:5	62.67	**94.63**	75.4	68.93

Table 4.7: Validation results for different kernel and class weight configurations. All values are given as percent.

linear and sigmoid kernel are lower than the lowest F1 score of RBF and polynomial kernel. The recall usually improves when class weights are used while the precision results decrease. For the polynomial kernel with class weights the recall value does not change when the weight ratio changes. With exception of the linear kernel the F1 score always decreases when class weights are used. For the sigmoid kernel there is a slight increase with 1:5 weights, but the F1 score is still lower than without class weights.

Baseline	Precision	Recall	F1 score	Accuracy
negative	-	-	-	77.4
stratified	27.2	22.5	24.6	68.8
uniform	22.1	48.0	30.3	49.9

Table 4.8: Baseline results. All values are given as percent.

When compared with the baseline results from Table 4.8 it is clear that the validation results are substantially higher. The negative scenario does not have any values for precision, recall and F1 score. Since the prediction result is always negative, there are no true positives and false positives, which leads to a division by zero for precision and a zero in the denominator for recall. The F1 score as combination of precision and recall can also not be calculated. The only available measure is accuracy, which represents with 77.4% the percentage of negative samples in the validation set. We do not want to report accuracy scores for each SVM configuration, because the high values can be misleading. Nevertheless, to compare the accuracy of the negative baseline with something, the accuracy of the SVM configuration with lowest F1 score (sigmoid kernel with 1:4 weights) was calculated. The results are 1133 correct predictions and 183 incorrect ones. This leads to an accuracy of 86.1%. The scenario with the lowest precision (sigmoid with class weights 1:5) has 1132 correct and 184 incorrect predictions, which leads to 86.0% accuracy. This is still 8.6% higher than the best accuracy of the baselines.

If a linear kernel is chosen, it is possible to gain some information about the amount of influence that individual features have on the result. Figure 4.4 shows the size of the corresponding coefficients for all 6 features. The negative values are displayed in green, while positive values are blue. This shows whether a feature is linked to the negative class (text) or the positive class (formula). The size indicates how important the feature was for the class separation.

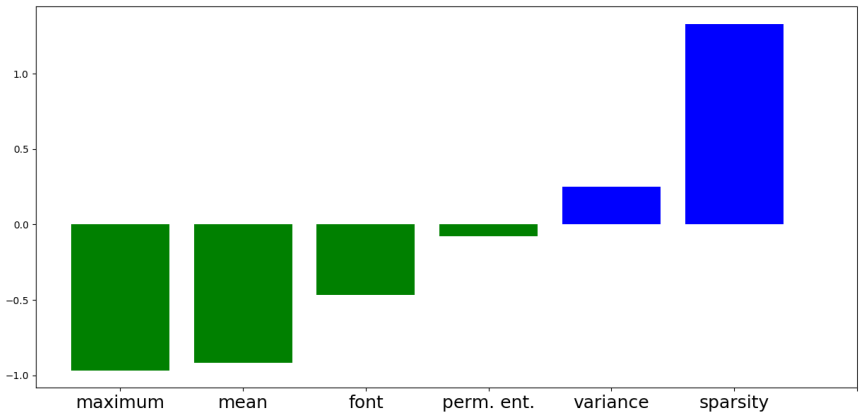

Figure 4.4: Amount of influence that the individual features have on the results of a linear SVM. Green means the feature is linked to the negative class (text), while blue is linked to the positive class. Maximum and sparsity have the most influence on the decision for a linear SVM.

This can be useful for feature selection if the number of features is especially large. If this is the case the computational cost of training a model is often high and the risk of overfitting the training data also becomes more likely. For our method it is not necessary to reduce the number of features. Still, we can see that the features maximum and sparsity are the most important ones for the linear SVM.

4.2.2 Random Forest

The random forest classifier that we used[2] had a large number of hyperparameters available. Many of them influence how the decision trees are build. The most important ones however, are the number of trees for the forest and the maximal depth of the trees. For these two hyperparameters a grid search was performed. The random forest classifier uses randomly selected feature subsets, that influence the outcome. When a classifier with the same settings is trained and tested on a certain dataset multiple times, the results are different each time. Even after creating the same random forest 50 times and taking the mean F1 score, there are differences

[2]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

# trees	1	10	100	1000
max depth = 1	56.12	0	0	0
max depth = 3	66.79	80.12	79.94	79.75
max depth = 5	77.1	82.8	82.98	82.78
max depth = 10	72.97	82.93	84.11	83.81
max depth = 15	71.49	82.39	**84.27**	83.64
max depth = None	71.49	81.89	83.71	83.64

Table 4.9: F1 score results for random forest classifier with different number of trees and maximal tree depth. 100 trees with a depth of 15 had the best F1 score of 84.27%.

of up to 0.05%. Since the differences between using a higher or lower number of trees also appear to be rather small, a fixed random seed (15) was used for the grid search.

Table 4.9 shows the results of different tree count and depth configurations. If the maximal depth is set to one it heavily depends on the random seed if there will be useful results. For the random seed 15 we got a result with one tree, but for more trees the classifier classified everything as negative. With different random seeds the predictions for more trees were sometimes better. Nevertheless, this shows that the maximal depth should be greater than 1 for the classifier to be reliable. The results for only one tree are mostly above 70% F1 score. However, a random forest with a single tree is more likely to have problems with overfitting than random forests with more trees. All results with at least 10 trees and a maximal depth of at least 5 are above 80%. The best F1 score of 84.27% was achieved with 100 trees and a maximal tree depth of 15. The result with 10 as maximal depth is only slightly smaller. All results with 100 trees are better than with 1000 trees.

Random forest classifiers also have a class weight option. The default is to use no weights and there is a ”balanced” option available, which calculates the weights inversely proportional to class frequency in the dataset. However, as Table 4.2.2 shows, this does not improve the results. Contrary to weighted SVM, where at least the recall values are higher, for random forest all results are lower when the ’balanced’ option is used.

Table 4.11 shows a comparison of all measures between a random forest with 100 trees and a maximal depth of 10 and another random forest with 15 as maximal depth. The first one has a higher recall value, but the random forest with more

Class weights	Precision	Recall	F1	MCC
None	85.45	83.13	84.27	80.38
balanced	83.49	82.23	82.85	78.56

Table 4.10: Random forest with 100 trees and a maximal depth of 15 with and without weights. All values decrease when weights are used.

max depth	Precision	Recall	F1	MCC
10	84.5	83.73	84.11	80.12
15	85.45	83.13	84.27	80.38

Table 4.11: Validation results with fixed seed (15) and 100 trees for direct comparison

depth achieved almost 1% more precision. This made F1 score and MCC slightly higher than for the shallower random forest.

The random forest classifier has a property which gives some clue about the importance of the features. In Figure 4.5 the values were plotted, and according to this measure sparsity is the most important feature. Mean seems to be the second most important feature, while the rest have approximately the same value.

4.2.3 Naive Bayes

The Naive Bayes classifier applies Bayes' theorem with strong feature independence assumptions. Since our features are most probably not independent, we expect this machine learning algorithm to perform worse than the others. There are different types of Naive Bayes classifiers but not all can be used with our type of features. For example there is the Categorical Naive Bayes, which assumes that the features have a categorical distribution. Basically, this means that the features are expected to be categories, which our features are not. Other variants are classifiers that assume a multinomial or Bernoulli distribution in the data and require discrete and binary features, respectively. Apart from the different variants, there are no hyperparameters for the Naive Bayes classifier.

Table 4.12 shows the validation results for a Gaussian, a multinomial and a Complement Naive Bayes classifier. The Gaussian Naive Bayes assumes that the likelihood

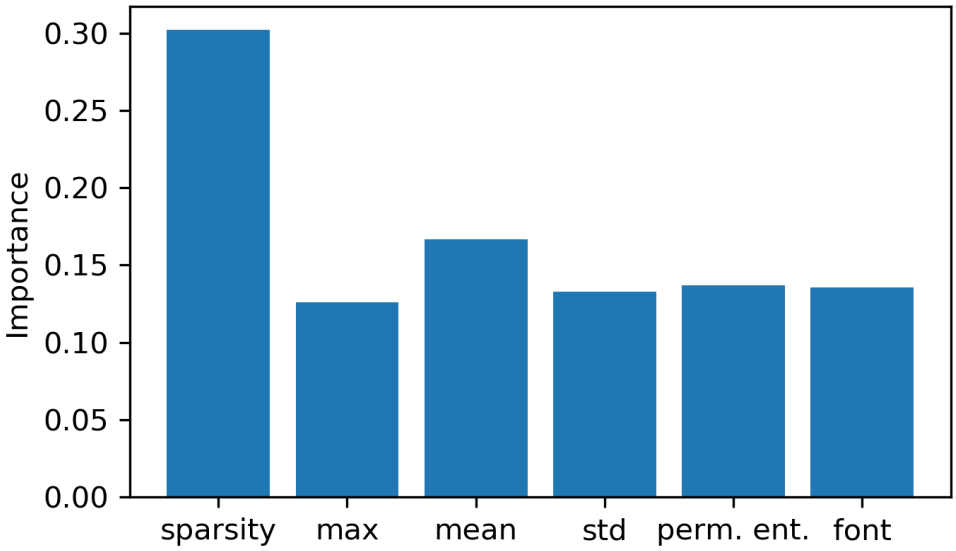

Figure 4.5: Feature importance in a random forest with 100 trees and a maximal depth of 15.

Type	Precision	Recall	F1	MCC
Gaussian	**67.95**	90.06	**77.46**	**71.92**
Multinomial	46.25	90.96	61.32	52.69
Complement	38.01	**97.89**	54.76	46.21

Table 4.12: Naive Bayes validation results. Values are given in percent. Gaussian Naive Bayes is clearly the best out of the three.

of the features is Gaussian. Since this Naive Bayes variant does not have any obvious feature limitation, we expected the best results from it. A multinomial Naive Bayes classifier is indented for text classification and typically used with discrete features. Nevertheless, we decided to use it, to have a comparison for the results of the Gaussian classifier. The Complement Naive Bayes classifier is similar to the multinomial classifier, but should be more suited for imbalanced datasets. As the table shows,the precision is much lower than the recall value for all variants. Recall is specially high for the Complement Naive Bayes, but since the precision is especially low this classifier gives the worst results. With accuracy values of 76.84% and 67.36% the multinomial and Complement classifier perform worse than our ”negative” baseline. The Gaussian Naive Bayes classifier reached an F1 score of 77.46% and is therefore about as good as a linear SVM.

4.2.4 Neural Network

From all machine learning algorithms that we used, neural networks have the most configuration possibilities. The combinations of number of hidden layers and neurons in them alone are practically endless. Additionally there are different options for activation functions in the neurons, different ways to initialize weights and bias, and different functions to calculate the loss. Listing different combinations of all these settings would be rather pointless, especially since not all available options can be expected to work well for a classification task. Therefore, we will only describe our settings and then discuss validation results for different hidden layer and neuron combinations.

The weights are initialized with a normal distribution to have zero mean and unit variance. The biases are initialized with zeros. In the hidden layers we use Rectified Linear Units (ReLU) as activation function. The l1-norm is applied as regularizer to help avoid overfitting. In the output layer with a single neuron the sigmoid activation function produces an output between 0 and 1. This is then rounded to 0 or 1 depending on whether the value is smaller than 0.5 or not. When training the model we use the Adam optimization algorithm and compute the binary cross-entropy loss. For the experiments with different layer structures we always trained for 100 epochs.

Like random forests neural networks also rely on randomness, for example for initialization of weights. Therefore, even with the same configurations and the same

Neurons in Layers	Precision	Recall	F1	MCC
6	81.0	75.78	78.29	73.1
6 - 6	80.33	78.31	79.3	74.17
6 - 3	80.14	76.45	78.23	72.94
2	78.35	71.51	74.77	68.89
9	80.05	76.57	78.26	72.95
18	**82.26**	**78.92**	**80.55**	**75.79**
10 - 5	78.67	75.9	77.2	71.63
24	81.13	78.73	79.91	74.94
12 - 12	76.63	78.37	77.39	71.67
6 - 4 - 2	64.39	62.44	63.37	59.28

Table 4.13: Neural network validation results for different hidden layer combinations. One hidden layer with 18 neurons achieved the best result.

data, the results from the neural network will be different in each run. Contrary to random forest it is not easily possible to set a random seed and always get the exact same result. For this reason each experiment was repeated for 10 times and the mean of the results was calculated. This is still not enough to get the same results after each run, but the differences should not be higher than 1%.

In Table 4.13 the validation results are presented. We started with one hidden layer and 6 neurons, which matches the number of features. From there we tried a number of different combinations, though the number of layers and neurons in them never became very large. One hidden layer with 18 neurons achieved the best results for each measure. One layer with 24 neurons, and two layers with 6 neurons each have the second and third highest results.

4.2.5 Experimental Hypotheses

For the Support Vector Machine both, RBF kernel and polynomial kernel can be expected to produce good results. Both perform best without class weighs. Through the introduction of class weights the minority class becomes more important and the decision boundary shifts towards the majority class. As a consequence more positive samples are correctly classified, increasing the recall value. However, also the number of false positives increases, which leads to a decrease in precision.

Except in the linear case with 1:3 weights, F1 score and MCC always decrease which makes class weights not a good decision for our data.

A random forest with 100 trees performs best. A maximal depth of 10 or 15 seems to be a good decision. If the depth is not limited the F1 score decreased again. As with SVM, random forest also performs better without class weights. On the validation set random forest achieved a higher F1 score than the best SVM result.

Only the Gaussian Naive Bayes Classifier can be expected to achieve good results, since the other two do not outperform one of the baselines. We expect that the Gaussian classifier will be about as good as a linear SVM.

It is hard to say if we found an adequate structure for the hidden layer of our neural network. Three different settings achieved acceptable results: 18, 24 and two times 6 neurons. Since there is some variance in the results we do not expect the setting with 18 neurons to be clearly better than the other two on the test set.

To summarize, from the validation results we form the following hypotheses about the test results:

1. **SVM:** RBF and polynomial kernel without weights will achieve good results. This means that the F1 score and MCC should be higher than the results from other kernels.
2. **SVM:** Class weights are not suited for our dataset.
3. **SVM:** When class weights are used the recall will increase while the precision will decrease.
4. **Random forest:** Not limiting the tree depth leads to lower results.
5. **Naive Bayes:** Gaussian Naive Bayes will have similar results to the linear SVM.

4.3 Results

In this section the result from the test set are presented. To examine whether our hypotheses about the SVM test results were correct all combinations of kernel type and class weights were again reported. The results of a random forest with three different maximal depth settings are shown. We expect only one Naive Bayes variant to work for our data, however to have a comparison we will also test on

		Test				Validation	
SVM Kernel	**Weights**	**Precision**	**Recall**	**F1**	**MCC**	**F1**	**MCC**
linear	1:1	80.35	79.4	79.87	73.04	76.19	69.05
linear	1:3	77.66	90.99	83.79	78.11	77.08	70.28
linear	1:4	75.7	92.6	83.3	77.55	76.42	69.6
linear	1:5	74.74	**93.56**	83.09	77.35	76.19	69.43
RBF	1:1	**92.22**	82.73	87.22	83.37	82.2	76.93
RBF	1:3	87.45	88.95	88.19	**84.11**	80.31	74.34
RBF	1:4	86.11	90.45	**88.23**	**84.11**	80.12	74.07
RBF	1:5	85.24	91.09	88.07	83.89	80.06	74.07
polynomial	1:1	89.52	81.55	85.35	80.79	**83.56**	**78.15**
polynomial	1:3	80.7	89.27	84.77	79.39	80.37	74.45
polynomial	1:4	76.65	89.81	82.71	76.61	79.39	73.2
polynomial	1:5	75.29	90.24	82.09	75.79	78.92	72.58
sigmoid	1:1	75.82	84.44	79.9	72.71	75.58	68.05
sigmoid	1:3	72.51	92.27	81.21	74.76	75.46	68.29
sigmoid	1:4	73.28	90.34	80.92	74.24	75.1	68.25
sigmoid	1:5	71.43	91.2	80.11	73.22	75.4	68.93

Table 4.14: SVM result for the test set. All values are given in percent. The two right most columns are the results of the validation set, which are inserted here for an easier comparison. The results from all kernel and weight pairings are listed to examine whether our hypotheses about the test results were correct or not.

a multinomial Naive Bayes classifier. For the neural network we chose to test the three settings with the best results, since it was not clear which one was the absolute best.

Table 4.14 shows the results from all kernel and class weight configurations for SVM. The highest precision of 92.22% is achieved with the RBF kernel and no weights. The linear kernel with weights 1:5 obtains the highest recall value of 93.56%. The best F1 score and MCC value are both results of the RBF kernel. The weights for the best F1 score are 1:4 while both 1:3 and 1:4 achieve the best MCC results. No other kernel achieves F1 and MCC values that are higher than any of the RBF results. For the polynomial kernel F1 and MCC results steadily decreased when higher class weights were used. For all other SVM kernels the setting without weights has the lowest results. When compared against the F1 and MCC values from the validation

	Test				Validation	
max depth	**Precision**	**Recall**	**F1**	**MCC**	**F1**	**MCC**
10	91.59	75.97	83.05	78.54	84.11	80.12
15	90.01	73.5	80.92	75.91	84.27	80.38
None	89.38	72.21	79.88	74.66	83.71	79.66

Table 4.15: Random forest test results for three different depth limits and 100 trees. All values are given in percent.

set, the test results are higher for all scenarios.

The results for the random forest are shown in Table 4.15. Random forests with 100 trees and three different depth limits were tested. To be able to compare the results directly the same random seed was used for validation and tests. The random forest with a maximal depth of 10 achieved the highest results for each measure. On the validation set a maximal depth of 15 was slightly better, but on the test set its F1 score is over 2% lower than for a depth limit of 10. If no maximal depth is used the values are lower than in the other settings.

	Test				Validation	
Type	**Precision**	**Recall**	**F1**	**MCC**	**F1**	**MCC**
Gaussian	58.87	91.64	71.69	65.14	77.46	71.92
Multinomial	44.51	94.33	60.48	52.24	61.32	52.69

Table 4.16: Naive Bayes test results for two Naive Bayes variants. All values are given in percent.

Table 4.16 shows the results for two Naive Bayes variants. As expected the Gaussian Naive Bayes classifier performed better than the multinomial one. However, there is a quite substantial difference between the F1 score on the validation set (77.46%) and the test set (71.69). For the multinomial classifier the difference is much smaller. The precision for the Gaussian setting does not even reach 60%.

In Table 4.17 the test results of the Neural Network are presented. One layer with 18 neurons reached the highest values for precision, F1 and MCC. The recall is slightly higher when the layer has 24 neurons. The F1 and MCC values from validation and test are almost the same for all settings. However, precision and recall are very different.

	Test				Validation	
Neurons	**Precision**	**Recall**	**F1**	**MCC**	**F1**	**MCC**
6 - 6	73.93	87.03	79.93	74.68	79.3	74.17
18	**74.59**	87.33	**80.46**	**75.34**	80.55	75.79
24	74.3	**87.36**	80.3	75.15	79.91	74.94

Table 4.17: Neural Network results for three different architectures. All values are given in percent.

SVM Kernel	**1:1**	**1:3**	**1:4**	**1:5**
linear	+4.83%	+8.71%	+9.00%	+9.06%
RBF	+6.11%	+9.81%	**+10.12%**	+10.00%
polynomial	+2.14%	+5.47%	+4.18%	+4.02%
sigmoid	+5.72%	+7.62%	+7.75%	+6.25%

Table 4.18: F1 score change between validation and test set for all SVM configurations. SVM with RBF kernel and 1:4 weights has the highest change.

Type	**F1 change**
Random Forest (10)	-1.26%
Random Forest (15)	-3.98%
Random Forest (None)	-4.58%
Gaussian NB	-7.45%
Multinomial NB	-1.37%
Neural Network (6-6)	+0.79%
Neural Network (18)	-0.11%
Neural Network (24)	+0.49%

Table 4.19: F1 score change between validation and test set for random forest, Naive Bayes and neural networks. The maximal depth of the random forest and the number of neurons for neural networks are shown in the parentheses. The F1 score decreases in all but two cases. The increases for the neural network are both less than 1%.

Table 4.18 and 4.19 show the relative difference between F1 scores for the validation set and test F1 scores. In Table 4.18 the results for all SVM configurations are listed. They are all positive, since the F1 results from the test set were always higher than for the validation set. SVM with RBF kernel and 1:4 class weights has the highest increase with +10.12%. Polynomial kernel has overall the smallest increases. The F1 score changes for all other machine learning techniques can be seen in Table 4.19. Most values here are negative, since the validation F1 scores were mostly higher than the test results. Only two neural network variations have a slight increase in F1 score. In both cases the relative difference is less than 1%.

Method	Precision	Recall	F1
Deep Neural Network [13]	-	-	93.4
Font Setting Baysian (FSB) [32]	**99.4**	88.9	93.9
Unsupervised Font Modeling [33]	93.6	**99.4**	**96.4**
SVM	86.11	90.45	88.23
Random Forest	91.59	75.97	83.05
Naive Bayes	58.87	91.64	71.69
Neural Network	74.59	87.33	80.46

Table 4.20: Results from different methods. The first three are literature methods. For the first method no precision and recall values were reported, which makes the comparison more difficult. All values are given in percent.

The comparison of our method with other methods can be found in Table 4.20. For each machine learning algorithm the setting with the best results was chosen and is listed in the table. The deep neural network method by Gao et al. [13] does not report any precision and recall values. However, the F1 score is given, therefore it is still possible to compare results from different methods with this one. The unsupervised font modeling method by Wang et al. [33] achieves the highest scores overall. Its F1 score is about 8% higher than the F1 score of our best result: the RBF SVM. It was not possible to compute MCCs for the literature methods since no information about the number of true negatives is given.

4.4 Discussion

The purpose of this paper was to explore different machine learning algorithms for formula detection combined with simple yet reasonable feature engineering techniques. The main motivation for this were the high complexity and low reproducibility of existing approaches. To complete this exploration, we now evaluate our formulated hypotheses and contrast our results with the performance of others.

Of the five hypotheses that were formed during the evaluation of the validation set only three held entirely. The first hypothesis, that RBF and polynomial kernel without weights would achieve better results on the test set than linear and sigmoid kernel, was correct. However, the results from the RBF kernel with weights are even higher, which makes the second hypothesis incorrect. On the test set most configurations with class weights produced better results than without weights. The polynomial kernel is the only scenario where the F1 score decreases when class weights are used. The results for the RBF kernel increased much more than for the polynomial case. This can also be seen in Table 4.18, where the relative differences between validation and test results were calculated.

We consider the third hypothesis, about the increase in recall and decrease in precision when class weights are used, to hold. The precision decreases steadily for all kernels. In case of the sigmoid kernel the recall value with 1:3 weights is higher than without weights, but the results for 1:4 and 1:5 weights are lower than for 1:3. Since both results (90.34% and 91.2%) are still higher than the recall without class weights (84.44%), the third hypothesis is correct.

The fourth hypothesis says that a random forest without depth limit will have lower results than with a limit. As Table 4.15 shows, this is correct. The F1 score without a maximal depth is 79.88%, which is 1% less than with a depth restriction. A reason for this could be that a too big decision trees leads to less generalization. Since the random forest without maximal depth still achieves almost 80% F1 score, this is not a real case of overfitting.

In the fifth hypothesis we speculated that Gaussian Naive Bayes classifier and linear SVM would have similar results. This hypothesis did not hold. The linear SVM without weights has an F1 score of 79.87% , while the Gaussian Naive Bayes classifier has only reached 71.69%. All other results of a linear SVM are even higher.

SVM with RBF kernel and 1:4 class weights achieved the best results overall. This SVM configuration has also the highest relative difference between validation result and test result. Only the results of SVM increased substantially between validation and testing. Random forest, which had the best results on the validation set, is now 5% behind SVM when the F1 scores are compared.

When our method is compared to literature methods the results are visibly lower. While all other methods have F1 scores above 90%, the best result from our method is 88.23%. The recall from SVM (90.45%) is higher than for the Font Setting Baysian method, which only reaches 88.9%. No precision or recall results could be found for the deep learning method, but since the F1 score is only sightly lower, it is possible that they would have been similar to the results from the font setting Bayesian (FSB) model.

While the literature method results suggest a very good detection of mathematical expressions, we found some inconsistencies and potential problems in the methods. The deep neural network method [13] uses a custom training set which consists of 1000 scientific papers from CiteSeerX[3]. The marmot dataset, which is used for testing, is also composed out of PDF documents from CiteSeerX. It is not clear if this was taken into account or if the training and test set have overlapping documents.

Another inconsistency can be found in the performance results of the FSB model. The unsupervised font modeling method uses the results from FSB as a baseline. Since the same author contributed to both methods we can assume that an accurate implementation was used. However, the precision, recall and F1 values with which the unsupervised font modeling method compares are not the same as the results reported in the original work. The values that are used as baseline are: precision 80.3%, recall 90.3% and F1 85.0%. The precision and F1 results are substantially lower than in Table 4.20 (precision 99.5%, recall 88.9%, F1 93.9 %). These values are also lower than the best results from our method (precision 86.11%, recall 90.45%, F1 88.23%). A potential explanation for this discrepancy could be a documentation error, a bug in their code or a limitation of the method, that was not mentioned in the text.

To summarize, at first it seems as if our method is clearly inferior to all selected literature methods. However, as described above the results of the FSB model are questionable. When our method is compared with the FSB results as they are

[3] https://citeseerx.ist.psu.edu/

described in its successor work, our method achieved clearly higher results. We were only able to compare with the *reported* results, since their documentation provided insufficient details for an accurate replication of their systems. Therefore, the real performance of state of the art formula detectors remains unclear.

5 Conclusion

The goal of this thesis was to find methods for the precise detection of mathematical expressions in PDF documents an therefore close the remaining step in the formula extraction pipeline. Most literature formula detectors focus on text lines for their classification. Contrary to that, we analyzed the horizontal and vertical whitespace in a document to find larger coherent regions. These regions are either displayed expressions or blocks of ordinary text. The features for classifying these regions come from two sources: the grayscale matrix of the rendered PDF file and the list of character objects that a PDF parser has extracted from the original file. These features were used as input for four machine learning algorithms: Support Vector Machine, random forest, Naive Bayes classifier and Neural Network. In our experiments the Support Vector Machine with a radial basis function kernel was superior to the other classifiers.

Finally, we compared our results with three state of the art formula detectors. While at first glance the results cannot quite keep up with other methods, we discovered some inconsistencies that question the reliability of the state of the art methods. Due to this, it is hard to faithfully assess the performance of our method. To resolve these observed inconsistencies, we propose to conduct a survey, where the most promising state of the art formula detection methods are analyzed and reimplemented. Through testing on a carefully selected benchmark dataset it will be possible to achieve a more accurate comparison.

Bibliography

[1] R. Akbani, S. Kwek, and N. Japkowicz. Applying support vector machines to imbalanced datasets. In *European conference on machine learning*, pages 39–50. Springer, 2004.

[2] J. B. Baker, A. P. Sexton, and V. Sorge. A linear grammar approach to mathematical formula recognition from pdf. In *International Conference on Intelligent Computer Mathematics*, pages 201–216. Springer, 2009.

[3] C. Bandt and B. Pompe. Permutation entropy: A natural complexity measure for time series. *Phys. Rev. Lett.*, 88:174102, Apr 2002.

[4] B. P. Berman and R. J. Fateman. Optical character recognition for typeset mathematics. In *Proceedings of the international symposium on Symbolic and algebraic computation*, pages 348–353, 1994.

[5] S. Boughorbel, F. Jarray, and M. El-Anbari. Optimal classifier for imbalanced data using matthews correlation coefficient metric. *PloS one*, 12(6), 2017.

[6] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[7] B. Chaudhuri and U. Garain. An approach for processing mathematical expressions in printed document. In *International Workshop on Document Analysis Systems*, pages 310–321. Springer, 1998.

[8] W. Chu and F. Liu. Mathematical formula detection in heterogeneous document images. In *2013 Conference on Technologies and Applications of Artificial Intelligence*, pages 140–145, Dec 2013.

[9] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[10] L. Eikvil. Optical character recognition. *citeseer. ist. psu. edu/142042. html*, 1993.

[11] R. J. Fateman, T. Tokuyasu, B. P. Berman, and N. Mitchell. Optical character recognition and parsing of typeset mathematics1. *Journal of Visual Communication and Image Representation*, 7(1):2–15, 1996.

[12] R. J. Fateman and T. A. Tokuyasu. Progress in recognizing typeset mathematics. In *Document Recognition III*, volume 2660, pages 37–50. International Society for Optics and Photonics, 1996.

[13] L. Gao, X. Yi, Y. Liao, Z. Jiang, Z. Yan, and Z. Tang. A deep learning-based formula detection method for pdf documents. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 553–558. IEEE, 2017.

[14] U. Garain and B. Chaudhuri. A syntactic approach for processing mathematical expressions in printed documents. In *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, volume 4, pages 523–526. IEEE, 2000.

[15] G. H. Golub and C. F. Van Loan. *Matrix computations*, volume 3. JHU press, 2012.

[16] Q. Gu, L. Zhu, and Z. Cai. Evaluation measures of the classification performance of imbalanced data sets. In *International symposium on intelligence computation and applications*, pages 461–471. Springer, 2009.

[17] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[18] Document management — portable document format — part 1: Pdf 1.7. Standard, Adobe Systems Incorporated, July 2008.

[19] A. Kacem, A. Belaïd, and M. Ben Ahmed. Automatic extraction of printed mathematical formulas using fuzzy logic and propagation of context. *International Journal on Document Analysis and Recognition*, 4(2):97–108, Dec 2001.

[20] H. T. Kam et al. Random decision forest. In *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, volume 1416, page 278282. Montreal, Canada, August, 1995.

[21] X. Lin, L. Gao, Z. Tang, J. Baker, and V. Sorge. Mathematical formula identification and performance evaluation in pdf documents. *International Journal on Document Analysis and Recognition (IJDAR)*, 17(3):239–255, 2014.

[22] X. Lin, L. Gao, Z. Tang, X. Lin, and X. Hu. Performance evaluation of mathematical formula identification. In *2012 10th IAPR International Workshop on Document Analysis Systems*, pages 287–291, March 2012.

[23] Y. Liu, K. Bai, and L. Gao. An efficient pre-processing method to identify logical components from pdf documents. In J. Z. Huang, L. Cao, and J. Srivastava, editors, *Advances in Knowledge Discovery and Data Mining*, pages 500–511, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[24] B. W. Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451, 1975.

[25] P. A. Moran. Notes on continuous stochastic phenomena. *Biometrika*, 37(1/2):17–23, 1950.

[26] B. H. Phong, T. M. Hoang, and T. Le. A new method for displayed mathematical expression detection based on fft and svm. In *2017 4th NAFOSTED Conference on Information and Computer Science*, pages 90–95, Nov 2017.

[27] B. H. Phong, T. M. Hoang, and T. Le. Mathematical variable detection based on convolutional neural network and support vector machine. In *2019 International Conference on Multimedia Analysis and Pattern Recognition (MAPR)*, pages 1–5, May 2019.

[28] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[29] I. Rish et al. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001.

[30] M. Suzuki, F. Tamari, R. Fukuda, S. Uchida, and T. Kanahori. Infty: an integrated ocr system for mathematical documents. In *Proceedings of the 2003 ACM symposium on Document engineering*, pages 95–104, 2003.

[31] J.-Y. Toumit, S. Garcia-Salicetti, and H. Emptoz. A hierarchical and recursive model of mathematical expressions for automatic reading of mathematical documents. In *Proceedings of the Fifth International Conference on Document Analysis and Recognition. ICDAR'99 (Cat. No. PR00318)*, pages 119–122. IEEE, 1999.

[32] X. Wang and J.-C. Liu. A font setting based bayesian model to extract mathematical expression in pdf files. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 759–764. IEEE, 2017.

[33] Z. Wang, D. Beyette, J. Lin, and J.-C. Liu. Extraction of math expressions from pdf documents based on unsupervised modeling of fonts. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 381–386. IEEE, 2019.

[34] B. J. Wythoff. Backpropagation neural networks: a tutorial. *Chemometrics and Intelligent Laboratory Systems*, 18(2):115–155, 1993.