CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

OCR post-processing of historical Swedish text using machine learning techniques

Master's thesis in Computer science and engineering

Simon Persson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

Master's thesis 2019

OCR post-processing of historical Swedish text using machine learning techniques

Simon Persson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2019

OCR post-processing of historical Swedish text using machine learning techniques
Simon Persson

Supervisor: Dana Dannélls, Språkbanken
Examiner: Krasimir Angelov, University of Gothenburg

Master's Thesis 2019
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2019

OCR post-processing of historical Swedish text using machine learning techniques
Simon Persson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

We present an OCR post-processing method that utilizes machine learning techniques and is targeted at historical Swedish texts. The method is developed completely independent of any OCR-tool with the aim is to avoid bias towards any single OCR-tool. The purpose of the method is to solve the two main problems of OCR post-processing, i.e. detecting and correcting errors caused by the OCR-tool. Our method is divided into two main parts, each solves one of these problems. Error detection is solved by a Support Vector Machine (SVM) that classifies each word to be either valid or erroneous. In order for the SVM to classify the words, each word is converted into a feature vector that contains several word features for indicating the validity of the word. The error correction part of the method takes the words that have been classified as erroneous and tries to replace them with the correct word. The error correction algorithm is based upon Levenshtein edit distance combined with the frequency wordlists. We OCR processed a collection of 400 documents from the 19th century using three OCR-tools: Ocropus, Tesseract and ABBYY, and used the output result from each tool to develop our method. Experiments and evaluations were carried out against the ground truth of the documents. The method is built in a modular fashion and evaluation was performed on each module. We report quantitative and qualitative results showing varying degrees of OCR post-processing complexity.

Keywords: machine-learning, NLP, OCR, post-processing

Acknowledgements

I would like to express my deepest gratitude towards my supervisor at Språkbanken, Dana Dannélls for her excellent help and patience. I would also like to thank my examiner Krasimir Angelov at University of Gothenburg for the support and feedback. Lastly, I would like to thank my family and close friend for the support and encouragement throughout this project.

Simon Persson, Gothenburg, June 2019

Contents

List of Figures

List of Tables

1
Introduction

In many libraries and research institutions, there exist a vast amount of digitized records and historical documents containing an enormous amount of information. To access this information in any useful way by Natural Language Processing (NLP) applications, there must be a procedure to search and access it. Since it is very time consuming to do it manually, a technique called Optical Character Recognition (OCR) is used to enable computers to convert the physical documents into text files. Text files are more easily read and most importantly allow users to search for specific content using NLP tools. Current OCR algorithms perform well on modern and high-quality print. Performance of state of the art systems shows a character recognition rate of about 99% [35], depending on both the algorithm and input quality. The quality decreases however greatly when applied on historical prints. Depending on the test considered the character recognition rate varies significantly. An OCR result is considered poor if the accuracy drops below 90% according to Holley [21] but for historical texts, the accuracy often yields a result that is lower than that. Some implementations produce character recognition rates between 70% and 80% [23] but it is not uncommon to have even poorer results [13]. A number of challenges can be attributed to the poor OCR performance including bad image quality and linguistic factors. The bad image quality is often a result from paper deterioration and bad inking [13]. Some linguistic factors might include obsolete characters (e.g. long-s) and spelling variation that are not present in any dictionaries [13]. Considering the aforementioned factors, achieving good OCR result for historical texts is a great challenge. However, it is important to overcome this challenge since with a better OCR process our ability to take in, analyze and preserve our culture legacy will increase.

One approach for increasing the quality of the OCR is to apply *post-processing*. Post-processing is a step that is applied as the last part in the whole OCR process. Correction of errors made in previous steps is conducted in this step. Previous works on post-processing include different approaches to texts from various languages and time periods. In 2017 Kirbat [26] developed a method that utilizes machine learning techniques with supervised learning. The work was conducted as a contribution to the shared task ALTA 2017 [31] which tries to post-process Australian publications from the *Trove* database [11]. Other approaches apply a more rule-based approach that driven by grammar rules, and corpus look-up. Lehal and Singh [27] constructed a post-processor for the Punjabi language using a rule-based method and achieved a recognition rate improvement of 4%. However, no previous work has applied a machine learning approach towards Swedish historical texts. Therefore,

this project aims to develop a post-processing method using machine learning techniques. Furthermore, the method will be completely separated from the rest of the OCR process. The performance of the chosen approach, could lay the foundation for future projects aiming to improve the OCR process for other historical texts.

1.1 Aim

This project aims to develop a method for OCR post-processing of historical Swedish text using machine learning techniques. The method tries to solve the two main problems of post-processing, i.e., detecting and correcting errors that have been caused by the OCR tool. Furthermore, the project tries to answer the question of whether machine learning methods are appropriate for decreasing the amount of OCR errors in historical Swedish texts. The evaluation of the method will decide if it can successfully improve the performance of existing OCR tools regarding Swedish historical texts. Another part of the aim is to develop a method that is not biased towards any single OCR tool.

1.2 Delimitations

The project will not improve the internal process of any existing OCR tool but rather view it as "black-box" and consider the output of the tool as input to the system of this project. This delimitation is well motivated since the aim of the project is to develop a method for post-processing that is not biased towards any single OCR tool. Furthermore, the project target Swedish historical text from the 19th century, and no other languages or time periods.

2
Background

This chapter provides the necessary theoretical background required to fully understand the problem this project is attempting to solve and the proposed method in chapter 4. The chapter starts with an explanation of the challenges when conducting OCR on historical texts. Secondly, a description of a generic OCR-process followed by a presentation of the OCR-tools used throughout the project. Finally, some theoretical background about Support Vector Machine (SVM) followed by an explanation the evaluation process of the different subsystems.

2.1 OCR of historical texts

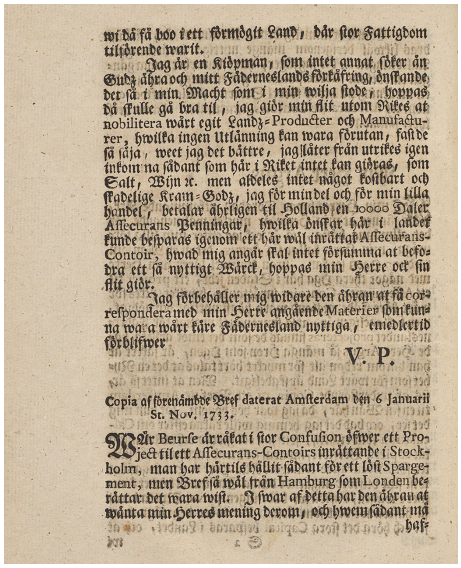
OCR of historical texts is an essential task since it increases our ability to take in and analyze our legacy. Unfortunately, there are many problems caused by OCR recognition of historical text. During the last decade, there has been several European initiative aiming to improve the quality of OCR-output of historical text. One example is the *Improved Access to Text* (IMPACT) project which was founded in 2005 by the European Commission specifically to improve the quality of the OCR-output for historical texts. The project brings twenty-six national and regional libraries, research institutions, and commercial suppliers across Europe to share knowledge and experience concerning improved access to text [3].

Generally, OCR-tools perform well on modern prints, but the quality decreases on historical prints [14]. According to Holley [21], a sufficient OCR accuracy is between $98\% - 99\%$ while an average accuracy is between $90\% - 98\%$. The OCR accuracy is, however, considered weak if it is below 90%. OCR of historical texts have accuracy ranging from up to 80% [23] down to about 30% [14] depending greatly on the selection of texts. The reasons for the further reduced quality can be divided into two main categories. The first one is the poor quality of the physical material, and the other is limited knowledge about the targeted historical language.

One reason for poor physical quality could be that the paper has suffered some damage due to deterioration, making it harder to recognize the characters. Other challenges include irregularities in character, word, and column spacing. Some texts are presented in unusual fonts that are not used today, which cause the OCR-tools some troubles [13]. The use of the typeface *Fraktur* was widespread during the 19th century, and an example of such text is displayed in Figure 2.1. Figure 2.1 presents an example of deterioration where the words from nearby pages are visual; this is

called bleed-through. Also, the scanning process could have been done poorly, creating visual errors in the images.

When considering the second category, historical texts tend to have spelling variations, acronyms, abbreviations, and morphological inflections that are not used today. These variations are not represented in any dictionaries and thus making the available dictionaries incomplete. Incomplete dictionaries may cause OCR-tools to recognize a word wrongly since the correct version was not present in any built-in wordlist. Depending on the language and time period the text may also contain obsolete characters (e.g., long-s) which the OCR-tool might be unable to recognize and thus cause character errors.



wi då få boo i ett förmögit Land, där stor Fattigdom tilstörende warit.

Jag är en Kiöpman, som intet annat söker än Gudz ähra och mitt Fäderneslands förkåfring, dnstande der så i min Macht stont i min wisa stode, hoppas, då skulle gå bra til, jag gör min flit utom Rikes at nobilitera wärt egit Landz-Producter och Manufacturer, hwilka ingen Utländning kan wara förutan, fast de så säja, weet jag det bättre, jag låter från utrikes igen inkom na sådant som här i Riket intet kan göras, som Salt, Wijn ꝛc. men aldeles intet något kostbart och skadelige Kram-Godz, jag för min del och för min lilla handel, betalar åhrligen til Holland en 10000 Daler Assecurans Penningar, hwilka dnssar här i landet kunde besparas igenom ett här wäl inrättat Assecurans-Contoir, hwad mig angår skal intet försumma at befodra ett så nyttigt Wärck, hoppas min Herre och sin flit gör.

Jag förbehåller mig widare den åhran at få correspondera med min Herre angående Materier som kunna wa a wärt kåre Fäderneland nyttiga, emedlertid förblifwer

V. P.

Copia af förenämbde Bref daterat Amsterdam den 6 Januarii St. Nov. 1733.

Wår Beurse är rakat i stor Confusion öfwer ett Project til ett Assecurans-Contoirs inrättande i Stockholm, man har härtils hållit sådant för ett löst Sparge-ment, men Bref så wäl från Hamburg som Londen bekräftar det wara wist. I swar af detta har den ähran at wänta min Herres mening derom, och hwem sådant må haf-

Figure 2.1: An example of deteriorated page. The prints from nearby pages has left marks on the current page.

2.2 Optical Character Recognition

Optical Character Recognition (OCR) is the process of translating digital images of physical media into machine-readable, searchable, and editable texts. The concept was introduced in the 1950s and has since been used in much different application ranging from banking, optical music recognition, and library digitalization [35]. Today OCR functionality is even integrated into many smartphone applications which allow the user to take a picture of a piece of text and convert it into a text file [18]. OCR also plays a vital part for many digitalization projects conducted by institutions such as libraries, archives, and museums. Some of these projects focus on historical texts which have shown to be more challenging for most OCR-tools [14].

The OCR-process is not flawless but does sometimes create *OCR-errors*. Examples of OCR-errors are when a character or a word from the original image is not recognized correctly and thus misrepresented in the OCR-output. These errors

are classified into two types, non-word, and real-word errors. Non-word errors are "words" generated by the OCR-process these are not actual words, i.e., they are not represented in any dictionary. To illustrate an example of a non-word error lets consider the plain text "Mum" and the OCR-tool output "Wum". Since "Wum" is not represented in the dictionary, it is a non-word error. The real-word errors are when the OCR-tool produces an incorrect word, but it is presented in the dictionary. Let us consider the plain text "dwell" which incorrectly is interpreted as "well" by the OCR-tool. Then "well" is a real-word error since it is represented in a dictionary.

2.2.1 OCR-process

The OCR-process is often divided into three main blocks: pre-processing, character recognition, and post-processing. The blocks are located as a chain, so the output of the first block is feed as input to the second block and so on. The position of each block in the chain is visualized by a simple diagram in Figure 2.2.

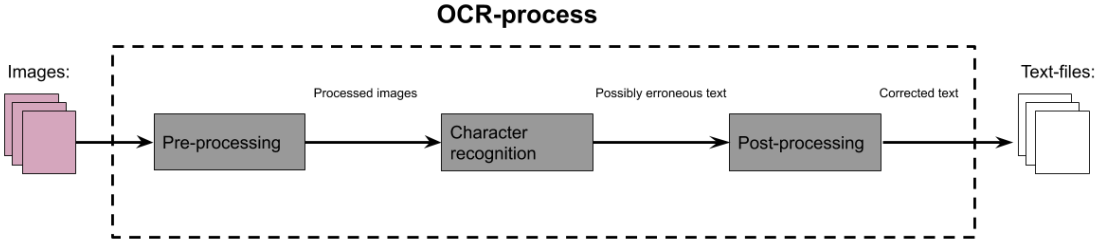


Figure 2.2: A diagram of the different blocks in the OCR-process

2.2.1.1 Pre-processing

The quality of the input highly influences the quality of an OCR-tool. To mitigate the impact of possible poor input quality, the images are often pre-processed. The actions performed in the pre-processing stage depend on the OCR-tool. Examples of pre-processing actions that can be executed are *binarization* or *layout analysis*.

Binarization converts a colored or grayscaled image into a black and white image. The goal is to more easily separate the text from the background and remove unwanted features like smudges or stains [37]. An example of the effects of binarization is displayed in Figure 2.4, notice how some unwanted information from the left image is removed and is no longer present in the right image.

Layout analysis is a common pre-processing action that many OCR-tools utilize in some form. The layout analysis first categorizes the image into text and non-text regions. The text region is further divided into text lines and then into individual words and finally characters [17]. Figure 2.3 illustrates an optical visualization of layout analysis where the regions are squared off with different colors.

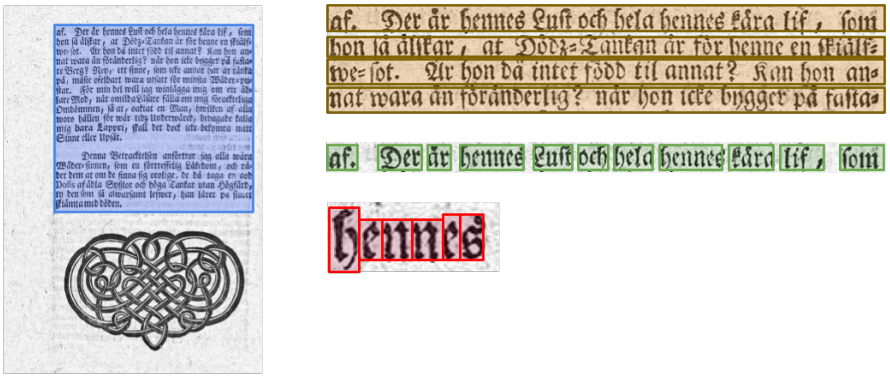


Figure 2.3: Illustration of a text getting segmented by the layout analysis during the pre-processing step in the OCR-process.

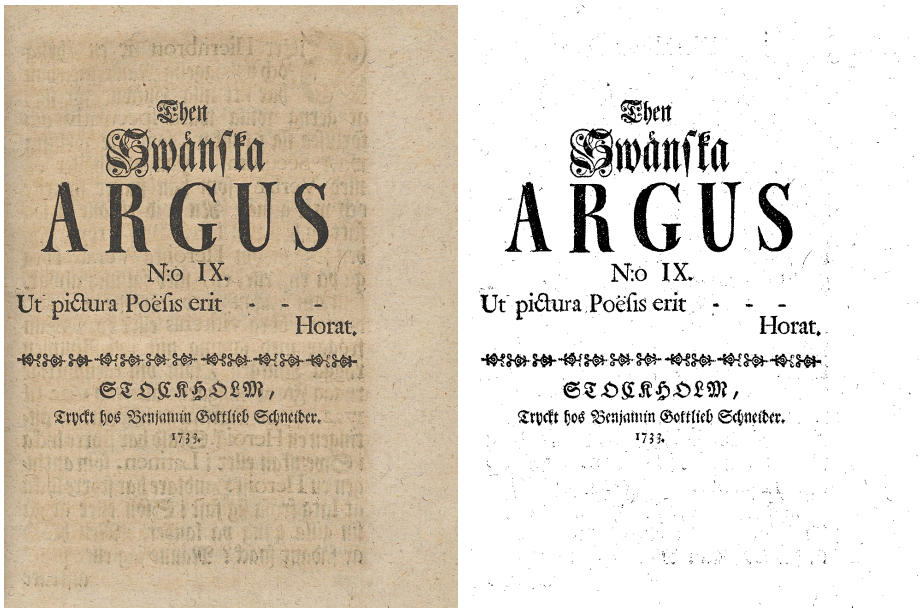


Figure 2.4: Depiction of an image before (left) and after (right) binarisation.

2.2.1.2 Character recognition

The next part of the OCR-process is the core element called character recognition. This step preforms the extraction of characters from the image and produces a text file. There exist many approaches to character recognition including *matrix matching*, *feature extraction* and *neural network*.

The matrix matching approach converts the character into a pattern in a matrix and then tries to match it against other stored patterns. The stored matrix that is most alike is chosen. This approach works best on single column pages were all words have the same font [35].

Feature extraction breaks down the characters into features like lines, loops, or line directions. The features or absence of features are matched against stored patterns

of features, and a candidate is chosen. This approach is generally suited for high-quality images like laser prints and magazines [35].

The neural network approach trains a neural network to recognize pixel patterns and matches them to known indices. This approach has shown good performance for faxes and damaged text [35].

2.2.1.3 Post-processing

The final part of the OCR process is the post-processing, which tries to correct the OCR-errors caused by previous steps. OCR post-processing is sometimes done in two stages, first a detection stage which marks all erroneous words and second a correction stage which tries to correct the erroneous words.

One approach for detecting errors can be to check each word against a dictionary and list non-word errors. This approach has the same functionality as a spell-checker. One challenge with this approach is detecting real-word errors because some texts might have spelling variations which are not represented in dictionaries. Incomplete dictionaries is a problem often encountered when the approach is applied to historical texts. As a result, spell checker based algorithms have shown to perform poorly [25]. Other detection algorithms are based upon statistic analysis by collecting n-grams [38]. Given a word and some language resources, it calculates the probability that the word is indeed part of the language. Kettunen [25] has used this to calculate the "Finnishness" of a word. An example that is given in the paper is the incorrectly spelled word "ytsimicliscsti" which got a score of $7.6 * 10^{-7}$ in contrast to the correct form "yksimielisesti" that got 21.4. Another possible approach is to utilize machine learning to classify all words to be either erroneous or valid [26].

Correction algorithms generally start by finding a set of candidates most suitable to replace the incorrect word which is identified in the detection stage. The selection for the candidate set is based upon how much they differ from the incorrect word. When a set of candidates have been collected, they are ranked based on the likelihood that they are the correct word. This ranking differs depending on the correction algorithm but can be done on how common the words are, i.e., the most common word gets the highest rank. Finally, the most promising word is selected, and the erroneous word is replaced.

2.3 OCR-tools

Today there exist many OCR-tools, both open source, and commercial. One well-known commercial tool is Abbyy Finereader [1] which generally performs with high quality.
Previous approaches toward improving the post-process have integrated it into the OCR-tool and therefore could not consider Abbyy since it operates as a "black box". However, this project aims to develop a post-processing method that is completely separated from any OCR-tool and therefore considers all OCR-tools as black boxes.

Thus Abbyy is considered during this project alongside the two leading open-source OCR-tools *Ocropus* [5] and *Tesseract* [10]. The reason for considering multiple tools is that part of the project aims is to develop a method for post-processing that is not biased towards one single OCR-tool but rather as general as possible.

2.3.1 Abbyy Finereader

One of the leading commercial OCR-tools today is Abbyy Finereader [1]. Since it is a commercial product, it operates as a "black box" causing the inner works to be unknown. The performance on modern text is considered very high and is therefore used by many users. It was initially released in 1993, the latest version as of May 2, 2019, is 14.

2.3.2 Ocropus

Ocropus is an open-source OCR-tool initially developed in 2008. It is aimed towards both the research community and large scale commercial applications [17]. Ocropus is modular, easy to develop further, and reuse. The program is divided into building blocks responsible for separate tasks. The chain of blocks is displayed in Figure 2.5. The first block starts with a program that performs *binarization* on the input. The next step in the chain is *physical layout analysis* which divides the image into text- and non-text regions. The text-regions are further divided into lines of text, and the correct reading order is decided for all lines. Next up is the *text line recognition* step which takes the images of individual text-lines from the layout analysis and converts them into text. Ocropus offers several different text line recognizers, but the default one utilizes a Long short term memory (LSTM) Recurrent Neural Net. The final part of the Ocropus chain is the *language model*, which corrects characters and resolves ambiguities. The model that is used by Ocropus is a statistical language model which contains dictionaries, character- and word-level n-grams and stochastic grammar. After this step, the OCR-process is complete, and the text files are produced.

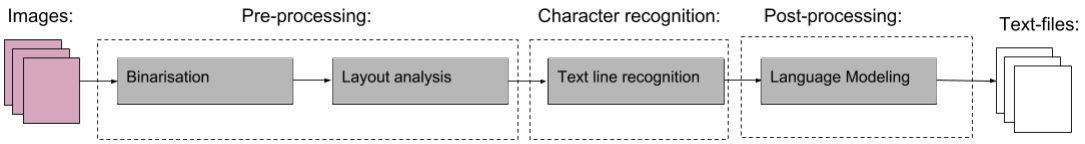


Figure 2.5: A diagram of the different parts of Ocropus.

2.3.3 Tesseract

Tesseract is an open source OCR-tool originally developed by HP as Research Prototype in the UNLV Fourth Annual Test of OCR Accuracy [36] in 1995. In late 2005 Tesseract was however released as open source. Tesseract utilizes a step by step pipeline often seen in OCR-tools. The steps are visualized in Figure 2.6. The first step is *Adaptive Thresholding*, which is a type of binarization which tries to classify

each pixel as either dark or light [16]. The second step of the Tesseract process is the *Page Layout Anlaysis*. Just like the layout analysis in Ocropus, this step tries to classify the page into text- and non-text regions. The text regions are then divided further into regions containing only one line of text, then only one word and lastly each character is divided into individual regions. The character recognition part of Tesseract uses a feature extraction approach, and consist of two passes through the *Recognize word* block. The two passes allow for an adaptive classifier to learn on the first pass and revisit unsatisfactory classified words on the second pass. The last three *fixing* phases are part of the post-processing and try to strengthen or fix some of the decisions made in earlier steps. The things that are typically fixed are spacing, X-heights, and words that need multi-word context in order to be resolved properly. The last step outputs several text files, and the OCR-process is over.

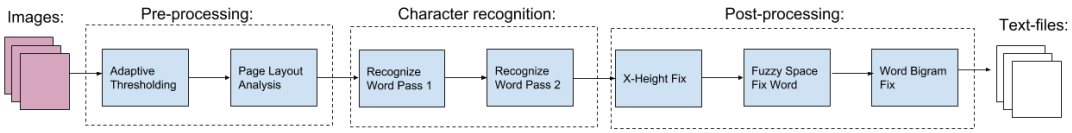


Figure 2.6: A diagram of the Tesseract structure.

2.4 Support Vector Machine

Support Vector Machine (SVM) is a technique to solve classification problems. A classification problem is solved by successfully label a data point to one or more predefined classes. A simple classification problem is displayed in Figure 2.7 where the task is to separate two classes (triangles and circles). The diagonal full line separates the two classes and provides a solution to this binary classification problem. An example of a real-world application of classification problem is text categorization which tries to label a text with one or more classes, e.g., news, reviews [24]. There exist two types of classification problems, i.e., linear and non-linear. Linear classification problems can easily be solved by separating the classes in low dimensions such as in Figure 2.7. Non-linear classification problems are harder to solve and often require further manipulation in order to separate the classes [34] easily.

SVM is a supervised machine learning algorithm. Given a set of training data, it can learn to predict the class of new unlabeled data points. Considering the simple two-dimensional example in Figure 2.7, the SVM takes a sample of data points (triangles and circles) and builds a model (diagonal line) that can predict the future data. A central part in optimizing the solution, the SVM tries to maximize the distance to the closest data point (margins) in either class. These margins are visualized in Figure 2.7 as dotted lines. Often the classification problem is in higher dimensions than the example discussed, and thus the solution is not a linear line but a linear hyperplane. When the SVM solves a non-linear classification problem, the data points (x) are mapped to a higher dimensional space where the problem becomes linear separable by a linear hyperplane. The mapping to the higher dimension is

achieved using *kernel function*, $\phi(x)$ [34].

$$\text{i.e. } x \longrightarrow \phi(x)$$

There exist some different kernels, each of which maps the points differently. Selecting the appropriate kernel function is a vital part for solving a classification problem using an SVM. The most common kernel-functions are displayed below where $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ and γ, r and d are kernel parameters [22].

- Linear function: $K(x_i, x_j) = x_i^T x_j$
- Polynomial function: $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0$
- Radial based function: $K(x_i, x_j) = exp(-\gamma ||x_i - x_j||^2), \gamma > 0$
- Sigmoid function: $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r)$

Apart from the choice of the kernel function, another important parameter in an SVM is the *c*-value. It is sometimes called the complexity parameter and is the sum of distances for all the points that are on the wrong side of the hyperplane. More concrete, the *c*-value defines how much of an error margin is allowed. It is crucial to find an optimal value for c since a too high or too low value will yield bad performance. A trial and error approach is often used to find a good c-value. For some choices of kernel function, it is also required to find appropriate kernel-parameters, which much like *c*-value is an iterative process that requires trial and error [22].

In order to find the hyperplane that solves the classification problem, the SVM needs to find the solution to the optimization problem displayed below [22].

$$\begin{aligned} \text{min: } & \frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi_i \\ \text{Subject to: } & y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0 \end{aligned}$$

2.5 Word feature

There exist several different methods for deciding whenever a word is erroneous or valid. The most obvious approach is to perform a lexicon lookup and check whether the word is present or not. This approach might be unavailable since the targeted text might not have any representative lexicons. For historical texts, this is a common problem and is further discussed in Section 2.1. Another approach is to use word features to indicate if a word is indeed erroneous or valid. Word features can be divided into two classes, depending on what they require to compute them. One way to distinguish between these classes is to classify them into word-based and statistically-based. Khirbat [26] presents one word-based feature which returns the number of non-alphanumeric characters in the given word. Non-alphanumeric comprises symbols that are neither characters or numbers such as '+', '#', '!', '?'. This feature is word based since it only requires the word and no other external

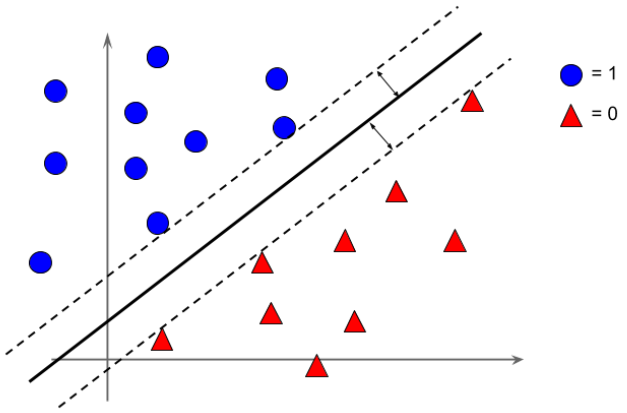


Figure 2.7: Classification problem with two classes (triangles and circles) labeled with values one and zero. The diagonal full line is a possible solution to the problem. By maximizing the distance to the margins (dotted lines) the solution is optimized.

resources to compute. An example of a statistically based feature is presented by Mei et al. [29] and is the frequency of the word in the given language. In order to extract the frequency of a word, it is required to have a dataset that contains, the word with its corresponding frequencies ideally the dataset should be representative for the given language and time period.

2.6 Levenshtein distance

Levenshtein distance is a metric that is used when describing the similarity between two strings [28]. The metric gives the minimum amount of edits between two strings. The possible actions that constitute an edit are either inserting, removing, or substituting a character [30]. Figure 2.8 displays an example of three pairs of strings and their corresponding Levenshtein distance.

$$
\begin{aligned}
&LD(\text{word, wurd}) = 1 \\
&LD(\text{Levenshtein, Lewemshteine}) = 3 \\
&LD(\text{Levenshtein, Levenshtein}) = 0
\end{aligned}
$$

Figure 2.8: The Levenshtein distance function *(LD)* applied to three pairs of strings.

2.7 Evaluation

In order to evaluate the performance of the system, appropriate methods and metrics are required. In this Section, the metrics and methods used throughout the evaluation of the project are explained.

2.7.1 Precision, Recall and F1-score

When evaluating a system that solves a classification problem three metrics are often used, namely precision, recall and F1-score. In order to define these metrics we first need to describe in which ways data points can be classified. Consider a set of data points x_i, which have a binary label ℓ that corresponds to the class the point belongs to. The system assigns another binary value z to each point that corresponds to which class the system thinks the point belongs to. By following this definition True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) could be extracted, these are displayed in Table 2.1 [19].

	System assignment, z		
Class label, ℓ		0	1
	0	TN	FP
	1	FN	TP

Table 2.1: Definition of True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN).

From this definition we can define the three different metrics as follows, where F1-score is the harmonic average between precision and recall [19].

$$\text{Precision, } p = \frac{\#\text{TP}}{\#\text{TP} + \#\text{FP}}$$

$$\text{Recall, } r = \frac{\#\text{TP}}{\#\text{TP} + \#\text{FN}}$$

$$\text{F1-score, } F_\beta = \frac{2pr}{r + p}$$

To get a better grasp on what these metrics mean in real world applications, an example is provided. Lets consider a data set with 20 points which are labeled into two classes (circles and triangles) as shown in Figure 2.9. The classification system sets a decision boundary, visualized by the diagonal line, so that all points above are believed to be circles and all below triangles. These metrics are calculated for one class at the time and later averaged for an overall measurement. The following examples will therefore only consider just one class (circles).

The precision metric corresponds to the number of circles above the decision boundary divided by the total amount of points above the decision boundary (#TP+#FP). In this example, the precision would be $\frac{8}{10}$. The recall metric is the number of circles above the decision boundary divided by the total amount of circles which corresponds to $\frac{8}{12}$. The F1-score has no visual representation since it is the harmonic average between precision and recall. In this example the F1-score is $\frac{2*\frac{8}{10}*\frac{8}{12}}{\frac{8}{10}+\frac{8}{12}} = \frac{8}{11}$ [19].

2.7.2 OCR evaluation

The evaluation of an OCR-process is usually done by measuring two metrics, *Character error rate* (CER) and *Word error rate* (WER). Both of the metrics are calculated

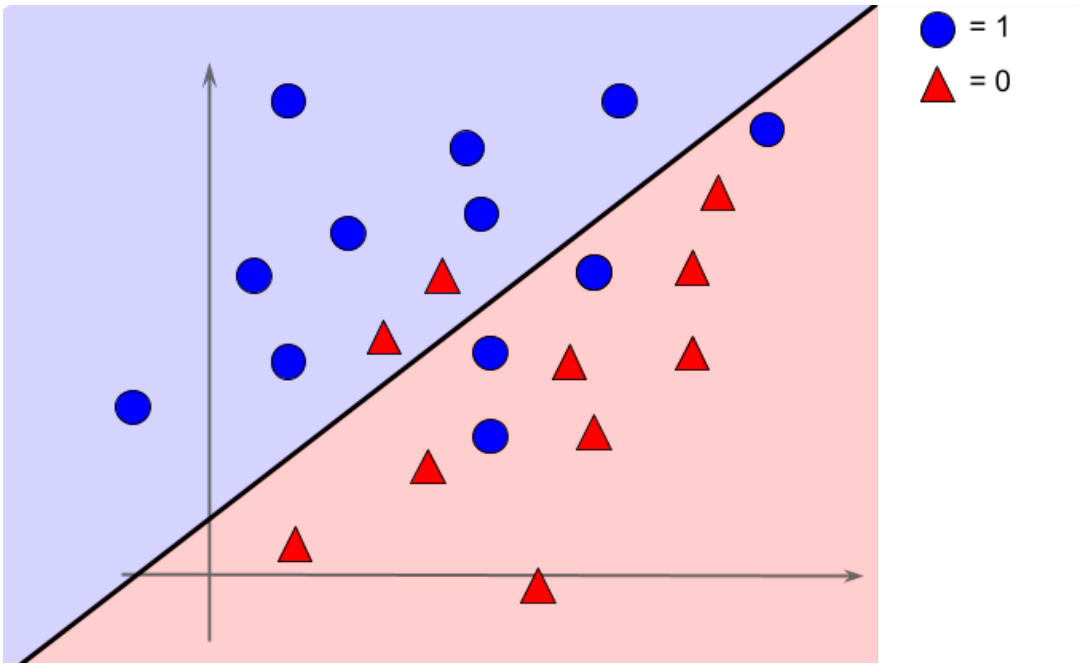


Figure 2.9: Classification problem with two classes (triangles and circles) labeled with values one and zero. The diagonal full line is a possible solution to the problem.

the same way, by figuring out how many character/word errors there are compared to the total number of characters/words. In order to detect character or word errors in a text, it is required to have the corresponding ground truth. This is a drawback when developing OCR-tools because ground truth creation requires manual transcription something that is a slow and expensive process.

Some evaluation tools do not use CER and WER as metrics but rather accuracy. The accuracy is also calculated separately for both characters and words. The character accuracy (CACC) is the result of the comparison of two strings, i.e., the OCR-output and the ground truth. The Levenshtein distance, d is calculated, and the CACC is defined as follows, where m is the length of the ground truth string.

$$\text{CACC} = \max(0, \frac{m-d}{m})$$

The word accuracy (WACC) is defined similarly by comparing two lists of words, one from the OCR-output and another from the ground-truth. Two words need to match 100% to be considered equal. Based on the Levenshtein distance, d the WACC is defined as follows, where m is the number of ground truth words.

$$\text{WACC} = \max(0, \frac{m-d}{m})$$

There are some evaluation tools for OCR that currently are available [6, 8, 12]. All require access to the ground truth in order to produce evaluation results. However, it is often required to use multiple tools to achieve reliable evaluation results since the result can vary significantly from tool to tool. As stated by Kettunen et al. [25] "there is no single software that could give us 'the truth' of the quality of the material".

2.7.3 Word metrics evaluation

In order to evaluate the word features, some statistical analysis needs to be conducted. Consider two sets of words, one with valid words and the other with erroneous words. By computing the average of a word feature for both sets and comparing the difference, the feature can be evaluated. A constructed example of this is displayed in Figure 2.10 was the difference of average is 15 for feature 1 and 40 for feature 2. It would be misleading to conclude that feature 2 gives a better

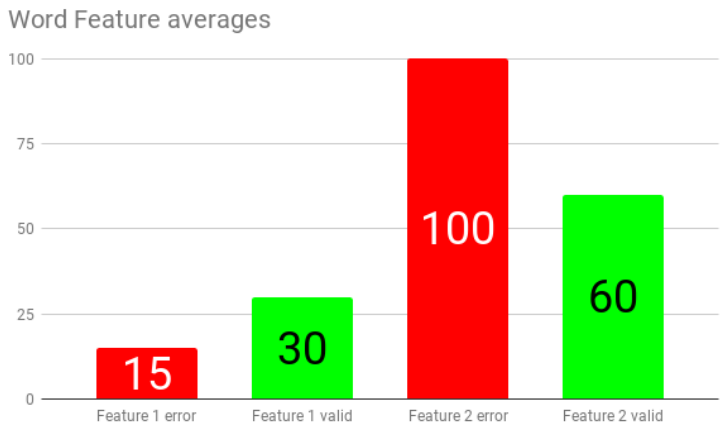


Figure 2.10: Example of averages of two word features divided by the validity class.

indication of the words validity just because it has a higher numerical difference since the change in percentage is greater for feature 1. Considering the greatest percental difference, the following equation is presented;

$$1 - \frac{\text{avg}_{low}}{\text{avg}_{high}}$$

where avg_{high} is the numerical greatest of the two averages. In the example from Figure 2.10 the avg_{high} would be the right bar (valid) for feature 1 and the left bar (error) for feature 2. The avg_{low} is the contrary lowest of the two averages. This metric will yield a maximum value of 1 if the two averages have maximum difference, i.e. avg_{low} is equal to zero and avg_{high} have an arbitrary positive value. The minimum value is 0 and occur when the avg_{low} and avg_{high} is equal. This metric will be called *Percent difference metric* throughout this project and will be calculated for the example from Figure 2.10 as displayed in Figure 2.11. Observe that feature 1 indeed yields a greater value in this metric.

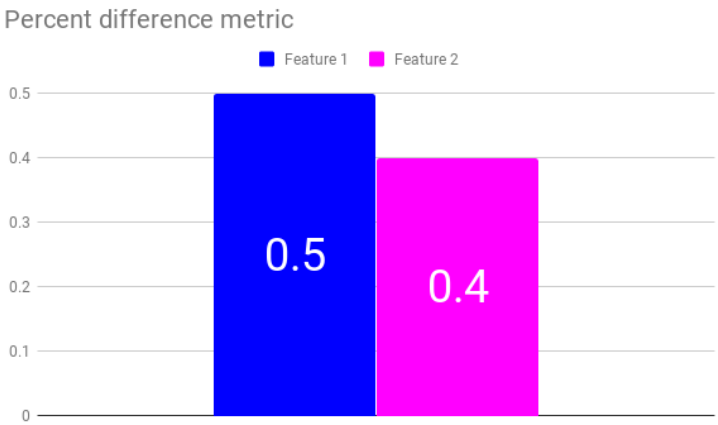


Figure 2.11: Example of the percent difference metric for two word features.

3
Data

The performance of an OCR system is highly dependent on the targeted text. As explained in Section 2.1, it is challenging to apply OCR on historical texts. The targeted texts that we experiment with in this project are extracted from *Swänska Argus* and *Grepect*. The texts were provided by Språkbanken and are freely available for download under Creative Commons Attribution License (CC-BY) [9]. Part of this project method relies on statical analysis of historical Swedish text. In order to increase the accuracy of this analysis, a large data set of historical Swedish text were provided by the *Project Runeberg* [7].

3.1 Targeted text

The text that is the subject for this project is historical Swedish from the time period between the 17th and 19th century. The pages were old at the time of scanning and have suffered some damage, most notable is the ink "bleed-through". An example of two pages from both *Then Swänska Argus* and the text collection *Grepect*, is presented in Figure 3.2. In order to evaluate the OCR-process, the ground-truth for each text is required. Producing the ground-truth is an expensive and time-consuming process since the images need to be manually transcribed. Therefore, a rather small amount of about 400 pages from the targeted text were randomly selected and manually transcribed. The word length was inspected, and the distribution is displayed as a histogram in Figure 3.1. This distribution will be compared with the distributions of the output from the OCR-tools to highlight potential deviations.

3.1.1 Then Swänska Argus

Then Swänska Argus is a work that was published in Stockholm in 1732–1734 and written by Olof von Dalin. The language in this text is interesting since it is written in a free and entertaining way, which is very different from contemporary texts. This text marks a significant change in Swedish language and is considered the point were Swedish transformed from the historical *gammel-svenska* to the more modern version *ny-svenskan* [4]. This was a period where changes in spelling and morphology took place [15].

Since the texts are old, the pages have suffered some deterioration. The most visual deterioration is the ink "bleed-through" which can be seen in Figure 3.2. Another

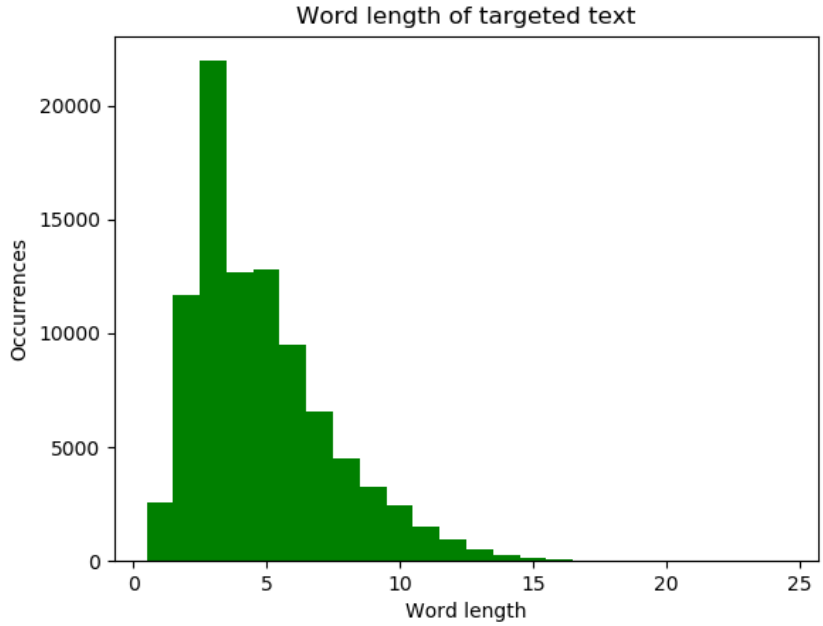


Figure 3.1: Histogram of word length distribution for the targeted text.

factor that causes these texts to be more challenging for an OCR-tool is the fact that many different typefaces are mixed throughout the text. A typical typeface configuration in this kind of texts is to write the body text in Fraktur, highlights in Schwabacher, and names as well as loan words in Roman type [14]. Furthermore, the old language poses some difficulties because many of the spelling variations that are observed are not represented in any electronic lexicons.

3.1.2 Grepect

Grepect is a collection of randomly selected pages from *Gothenburg University Library* from the period 17th to 19th century. The scans were manually transcribed by an external compainy called Grepect [2] and therefore this text collection will be referred to as *Grepect* throughout this project. An example page is presented in Figure 3.2. Similar to *Then Swänska Argus*, these pages suffer from deterioration and also have a mixture of typefaces in old Swedish. Also in these texts we can observe spelling variations, mixture of fonts (Fraktur and Schwabacher) and difference in font size.

3.2 Project Runeberg

Some of the methods of the project rely on statical analysis of old Swedish language. Conducting this analysis on a large data set will make it more reliable and possibly increase the performance of the entire post-processing. Project Runeberg is an OCR-project that started in 1992 at *Linköping University* intending to scan books from mainly Sweden but also other Nordic countries [7]. After the books are scanned, the

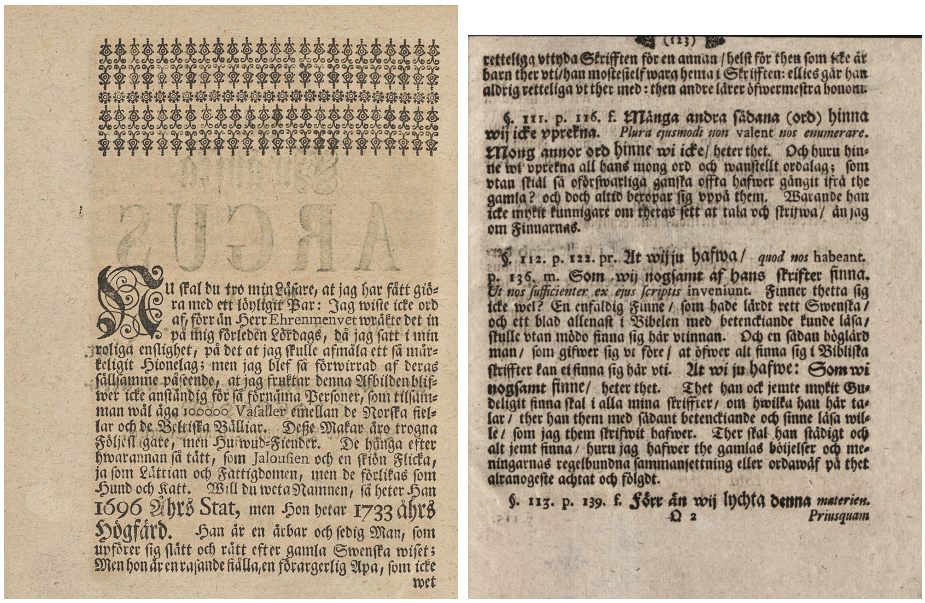


Figure 3.2: Example of pages from both *Then Swänska Argus* (left) and *Grepect* (right). Notice the ink "bleed-through" and mixture of typefaces.

text is sometimes also proofread by volunteers and errors are corrected. Therefore can these texts be considered correct and can reliably be used in any statical analysis. At the time of this project, the Project Runebergs database contained about 384.000 pages that were all proofread [7]. The pages with text from the 17th and 19th century were then extracted and used throughout this project. The resulting dataset extracted from Project Runeberg is 2.730.042 words.

4

Methods

This project develops a post-processing method that is suitable for correcting OCR errors of historical Swedish texts. The post-processing system is completely separated from the chosen OCR-tool. Therefore the output from the OCR-tool is the input into the post-processing system. An overview of the system architecture is visualized in Figure 4.1.

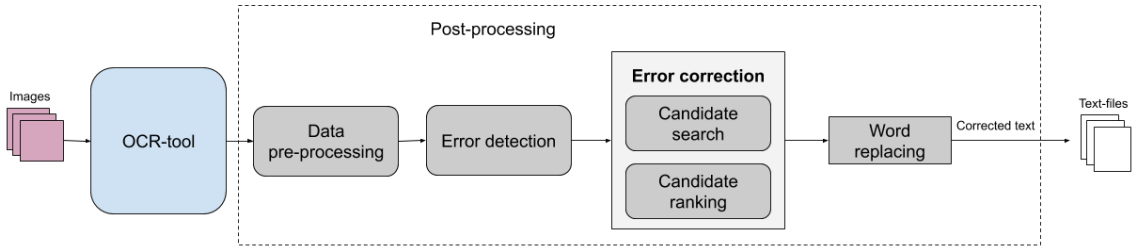


Figure 4.1: Simplified diagram of the complete OCR process including a more detailed version of the *post-processing* step.

4.1 Data pre-processing

Before the data enters the main post-processing system, it is pre-processed to remove some irregularities. These pre-processing measures are applied to both the ground truth and the output from the OCR-engine in order to establish a common ground. The first measure is to remove any *tags*, e.g. '<h1>', '<fr>', '<aq>' which have been inserted into the text to indicate a visual change such as font-change or headings. The next measure is to remove any unnecessary white-spaces. This consists of first removing any duplicate white-spaces and changing all non-space white-space such as *tab* to space. Figure 4.2 below illustrates the pre-processing by applying the two measures to the first line, and the result is the second line.

```
This␣<h1>text</h1>␣visualizes␣␣␣␣the␣␣pre-processing
This␣text␣visualizes␣the␣pre-processing
```

Figure 4.2: Visualization of the data pre-processing. The second line is the processed version of the first line.

4.2 Post-processing

The post-processing is divided into two blocks. Both blocks try to solve the two main problems of post-processing, detecting, and correcting OCR-errors. The detection stage utilizes SVM machine learning techniques. The error correction uses frequency analyses in order to find suitable replacements for the erroneous words.

4.2.1 Error detection

In order to detect the erroneous words, each word is classified by the SVM to be either valid (1) or erroneous (0). Additional information is provided by computing *word metrics* for each word and combining them into input vectors. In order for the SVM to provide a satisfactory solution to the given classification problem, it is trained on a large number of input vectors.

4.2.1.1 Training data generation

When generating the training data, it is required to have both valid and erroneous words to provide a complete picture of the problem. Valid words are easily extracted from the manually transcribed versions of the targeted texts. The extraction of erroneous is more challenging since they need to be extracted from the OCR-output, and the output does contain not only erroneous words but also valid words. This extraction is conducted by slightly modifying the *Språkbanken evaluation-script* [8] to output all the erroneous words. The modifications changed the format of the erroneous words when outputted to better suit our needs. One advantage of this process is that it is fully automatic and requires no manual work. That makes the training data generation method scalable and applicable in more contexts. However, since no manual inspections of the training data are conducted, the possibility of some misclassification is possible. Misclassification could be caused by either the *Språkbanken evaluation-script* [8] not operating correctly or one of the erroneous word is actually part of the lexicon and thus constitute a real-word error. A real-world error will potentially cause the same word to appear both as valid and erroneous in the training data. Misclassified examples in the training data hinder the SVM from conducting correct classification of words with high accuracy.

Lastly, when all the words are extracted, two kinds of filtering are applied before the *word features* are computed for each word, and the resulting feature vector is saved to the training data set. Filtering is constructed to remove as much information that is not strictly part of the word. The first filter is called *last character filter* and removes the last character of the word if it is punctuation, e.g., “example,” becomes “example”. The next filter is called *non-alfa filter* and removes all words that are only symbols or numbers, i.e., ‘?” or “1888”. A possible negative consequence of these filters is that they remove possible important information. Therefore the effect of these filters will be evaluated by comparing the quality of word metrics with and without them, see Section 5.1.1.1.

4.2.1.2 Feature vector

There exist two versions of feature vectors depending on whether the vector is actual input or training data. The training data contains an additional value named *valid*, which indicates the validity of the word. Obviously, the actual input to the system can not have this value. The rest of the values in the feature vectors are different word features. The features *Number of non-alphanumeric characters in a word* and *Frequency of word* are taken from previous works and can, therefore, be used as a baseline since their performance has shown to be high on historical text [26, 29]. The rest of the word features that are presented below will be tested and evaluated using the *percent difference metric* that is described in Section 2.5. The actual input to the system will also be filtered through both the aforementioned filters (*non-alfra filter* and *last character filter*).

Below a description of word features that make up the input vectors.

- *Number of non-alphanumeric characters in a word.* Non alphanumeric character include special characters like '+', '#', '&', '%' and punctuation like '!', '?', ':', ';'. Words with a high number of non-alphanumeric character are more likely to be erroneous [26].
- *Number of vowels in a word.* Since there exist very few words without any vowels, this feature might indicate the validity of a word.
- *Frequency of word.* If the same word occurs many times, it is less likely to be erroneous since there often exist many wrong versions of a correct word, and it is not likely that it will be misrecognized the same way multiple times [25].
- *"Swedishness" of a word.* This is a modification of a common language detecting technique based on collecting statistics of character n-grams [38]. A prerequisite before being able to calculate the "Swedishness" of a word is to construct a database with the most common tri-grams of the targeted language. Calculating the feature is done by multiplying the frequencies of each tri-gram in the word. Since the logarithmic value of zero is undefined a small value (0.1) is chosen for tri-grams not represented in the database [25].
- *Word length over 13 characters.* Since there are very few words above 13 characters, it is likely to be erroneous. These long words can be created when a space is missed causing two or more words to merge.
- *Number of uppercase characters.* If a word has more then one uppercase character, it is likely to be erroneous.
- *Contains numbers.* If a word contains numbers, it is an indication that the word is erroneous and maybe a character has been falsely recognized as a number.
- *Validity of word.* Returns 1 if the word is valid and 0 otherwise. This value is only present in the training data.

4.2.1.3 Support Vector Machine

The motivation for the choice of SVM as the machine learning technique for this project is that SVM has been shown to be efficient for solving classification prob-

lems [34]. Previous work by Khirbat et al. [26] has shown promising result in classifying words using an SVM for similar task. The performance of Khirbat et al. [26] SVM is displayed in Table 4.1. The SVM for this project is implemented using python and the Scikit-learn library [32], it is a widely used library containing useful tools for easier implementation [33]. After the SVM is trained with the training data, the model is saved so that future runs of the program will not have to perform unnecessary work. The Scikit-learn library provides useful evaluation tools that outputs an matrix with *precision, recall* and *F1-score* for both classes namely valid word (1) and erroneous word (0).

Precision:	**Recall:**	**F1-score:**
0.69	0.44	0.54

Table 4.1: The performance of the SVM that was implemented by Khirbat et al. [26].

The performance of an SVM is highly dependent on the selection of parameters. The parameters that are available for selection is *kernal, c-value* and for certain kernel choices there is an additional *gamma-value*. Since the other parameters depend on the choice of *kernel* that is selected first. The method for selecting *kernel* is rather simple and consists of testing all available *kernels* and comparing performance. When the *kernel* is chosen, the other parameters need to be determined, and this is done with a grid search algorithm. The grid search algorithm is part of the Sci-learn library [32] which conducts an exhaustive search over predefined lists of possible values for parameters. Since it is an exhaustive search, the time increases drastically with each additional value in the lists. Therefore the grid search is done iteratively by first selecting few values in a big range and then narrowing the range in stages until a satisfactory value is found.

After the training and parameter selection, the model is ready to predict the validity of words. The prediction starts by entering some words and their corresponding input vector to the SVM. The classification is conducted based on the input vectors, and the SVM finally returns the words and their predicted classes.

4.2.2 Error correction

The next step of the post-processing is to correct the OCR-errors found by the previous step. This is done in two stages, first we search for candidate words to replace the erroneous one, and second, we rank the candidate and pick the most promising one. A fundamental part of this step is to have a large dataset of potential candidate words. This dataset is constructed by extracting words from books from the Runeberg project [7] but also some wordlists from the targeted time period. The dataset does not contain any duplicates, but the frequency of the word is calculated and stored. The benefit of having a large dataset is that there exist many potential candidates, and the chance of the correct word being present is higher than with a smaller dataset. To understand the potential downside of an extensive dataset lets consider the following example. The word “jag” is misrecognized by the OCR-tool

as “ag” and the post-process tries to correct this error. With a vast database, the post-process might correct it to “og”, which is a quite uncommon word compared to “jag”. Therefore an evaluation of different dataset sizes will be conducted to observe how it affect the performance of the overall system.

Two error correction algorithms are designed. The first algorithm is called *no-split* algorithm throughout this project and tries to find a replacement word for each erroneous word [26]. The next algorithm is called *split* algorithm and is a upgraded version of the *nosplit* version. The *split* algorithm tries to find suitable ways to split a word into two and correct each split. The motivation for this functionality is that OCR-tools tend to merge adjacent words wrongfully. Figure 4.3 displays an illustration on how the *split* algorithm processes erroneous words.

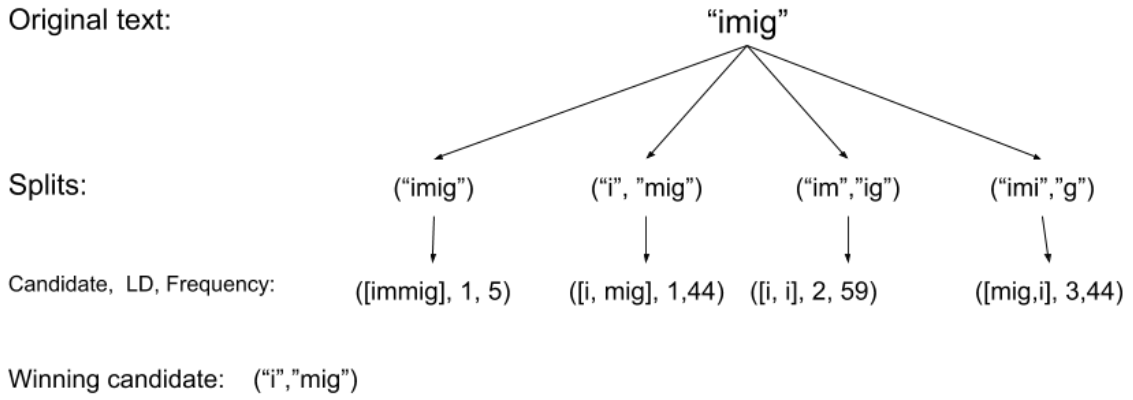


Figure 4.3: Example of how a word is split and some candidates for each split. The winning candidate is chosen by ranking the candidates by Levenshtein distance (LD) and the frequency of the word.

4.2.2.1 Candidate search

The search for candidates is done for each split for the given word. The search starts by calculating the Levenshtein distance (LD) between all the words in the dataset and the splits. The word from the dataset with the lowest LD is chosen as the candidate. If the split consists of two words, an additional edit-distance is added to accommodate for the action of splitting the word. This is illustrated in Figure 4.3 where the candidate to the split (“i”, “mig”) is the same, but the LD is still 1. The list of candidates and the corresponding LD values are then handed off to the candidate ranking part.

When considering this candidate search, it is crucial to specify the minimum and maximum LD for indicating whether a candidate is the original word. If the minimum LD is set to 0 the system can not correct real-word errors since there will

always be a candidate with LD= 0 and that will be the winning candidate. However, since the error detection stage does not yield a perfect result, there will exist several false negatives and a minimum LD of 0 would potentially not "correct" those and thus improve the performance. Contrary, a minimum LD of 1 would potentially correct some real-word errors but would wrongfully correct all false negatives. Different minimum LD will be tested, and the effect on the performance of the complete system will be analyzed. When considering a maximum LD, the question becomes about computational time rather than correctness. It is improbable that candidates with significant LD from the considered word will be the winning candidate or correct word. Therefore a reasonable high number is needed as a threshold when the candidate should not be considered. The maximum LD between the considered word and a candidate is set to 8.

4.2.2.2 Candidate ranking

The ranking of candidates is done primarily by the LD, but further distinction is needed when two or more candidates have the same LD. This distinction is made by comparing the frequency of the candidates. The frequencies are taken from the aforementioned dataset. If the candidate is two words, an average of the two is used. Considering the example in Figure 4.3, there are two candidates with LD equal to one. Since "i" and "mig" are very common words in the Swedish language and "immig" is quite unusual the winning candidate is ("i","mig"). The original word (*imig*) is replaced with the winning candidate, and the algorithm continues with the next erroneous word.

4.3 OCR evaluation

In order to get a more reliable evaluation of post-processing, multiple evaluation-tools are used as discussed in Section 2.7.2. The evaluation-tools that will be used in this project are presented in the list below. The first two tools use CACC and WACC, but Språkbanken Evaluation script uses CER and WER. The decision to separate the evaluation on the OCR-tools (Ocropus, Tesseract, and ABBYY) was made in order to notice any biases. This is crucial to notice since one of the aims of the project is not to be biased towards any single OCR-tool.

- OCR Frontiers Toolkit 1.0 [12]
- PrimA Text Evaluation [6]
- Språkbanken Evaluation script [8]

4.4 Project baseline

A good baseline to start from and evaluate the result towards is a necessity in any project. To achieve this baseline, the images of the targeted texts were processed by two OCR-tools, namely Ocropus and Tesseract. To achieve the best possible

output, the appropriate configurations for each OCR-tools need to be used. Tesseract offers a language model that is optimized towards Swedish in Fraktur typeface, which works well since most of the targeted text are in Fraktur. A modified version of Ocropus that switches between two language models that were developed by *Språkbanken* in their project *A free cloud service* [14] is used instead of the original version. The original version of Ocropus produced an output with so poor quality that it was considered unusable for this application. The two language models used by the modified Ocropus is Swedish in Fraktur and Antiqua.

The output of these OCR-tools is the input for our method and therefore also the baseline we aim to improve upon. The quality of the baseline is displayed in Table 4.2, notice that *Språkbanken Evaluation script* provides performance in CER/WER rather than CACC/WACC.

	OCR Frontier Toolkit		Språkbanken evaluation script		PrimA Text Evaluation	
Engine	**CACC**	**WACC**	**CER(%)**	**WER (%)**	**CACC**	**WACC**
Ocropus	0.80	0.58	17.68	59.60	0.80	0.41
Tesseract	0.64	0.35	34.08	76.37	0.64	0.25
ABBYY	0.58	0.24	39.30	86.59	0.62	0.2

Table 4.2: The performance of the projects baseline.

5
Results

The performance of the post-processing method when applied on the targeted text is described below. The evaluation is divided into the aforementioned problems of OCR, detection and correction of OCR errors. Since the system is built as a chain where the output of one subsystem is the input of the next subsystem, the evaluation needs to be conducted in the same order to always have optimal configurations for previous subsystems. The evaluation order of each subsystem is presented in Figure 5.1.

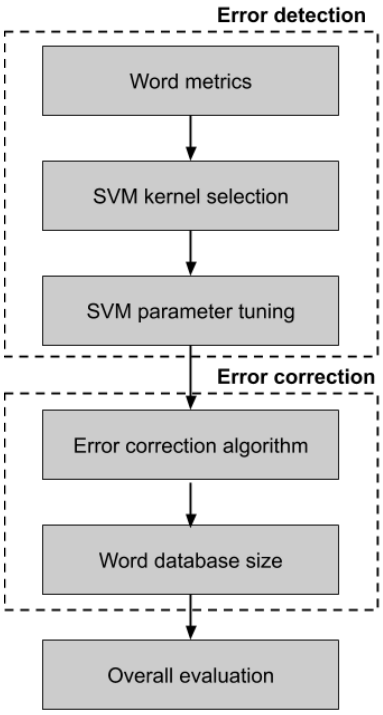


Figure 5.1: The order in which each subsystem will be evaluated. Each subsystem will be evaluated with optimal configurations for all previous subsystems. The dashed boxes indicate which problem each subsytem contribute to solving.

5.1 Error detection

The first problem of OCR post-processing is the detection of OCR errors, which is done by utilizing an SVM. Like all machine learning, the SVM is highly influenced

by the quality of the training data. Word features construct the training data in this project. Therefore the evaluation of the error detection is split into two separate problems, word features and SVM. As previously stated, the word features are evaluated first in order to utilize the optimal configuration when evaluating the SVM.

5.1.1 Word features

The word features are used for both the training data and the input to the system. The quality of word feature is therefore crucial for the performance of the entire system. The performance of the word feature will be described with the *percent difference metric*, which is described in Section 2.5. Different configurations that in some way affect the performance of the word metrics are tested and evaluated in this Section. Lastly, in Section 5.1.1.4 the overall performance of the word features will be presented along with their effect on the performance of the SVM.

5.1.1.1 Training data filtering

When generating the training data, each word is processed by two filters called *last character* filter and *only non-alpha* filter before the feature vector is generated. To evaluate the effects these filters have on the training data, four different tests were conducted. First, the filters were removed and the effects on the word features performance were measured. Each filter was then tested individually and lastly; both filters were used together. Since no optimal configuration is found for the size of the tri-gram and word database a default value of 10000 is used throughout these tests.

Figure 5.2 shows the quality of the word features without any filters. The quality is presented in the aforementioned *percent difference mean* metric.

The performance when introducing the *last character* filter can be observed in Figure 5.3. The performance of most of the features is unchanged except for the *#Alphanumeric* feature which was further increased.

Figure 5.4 visualizes the performance when only using the *only-alpha* filter. Compared to the initial results in Figure 5.2 most features are unchanged. The exception being is the *#Numbers* feature which has improved.

In Figure 5.5, the result when using both filters is presented. We can observe the advantages of both filters since both *#Alphanumeric* and *#Numbers* are improved compared to the initial results without any filters in Figure 5.2. The optimal configuration for filtering of the training data is to use both the *last character* filter and *only non-alpha* filter.

5.1.1.2 Tri-gram database size

The generation of the *Swedishness* feature utilizes a tri-gram database containing the most common character tri-grams accompanied by their frequencies. The influence

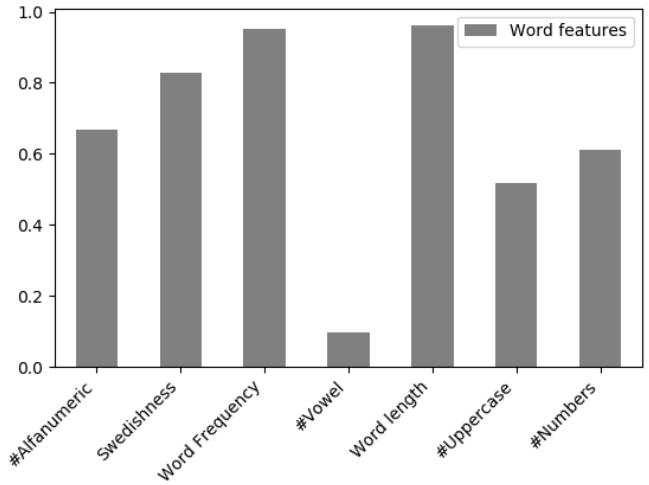


Figure 5.2: Performance of the word features without any of the two filters, *last char* filter and *only non-alpha* filter. The performance is displayed in the *percent difference mean* metric.

of the database size on the quality of the *Swedishness* feature is evaluated in order to find the optimal configuration. A range of database sizes between $0 - 35000$ tri-grams were teseted and the performance of the *Swedishness* feature was calculated. The result is plotted in Figure 5.6, as this Figure shows the feature seems to perform best when the database contains around roughly 15000 tri-grams. The performance is low for small sizes, but increases as the database grow.

5.1.1.3 Word frequencies database size

Similar to the *Swedishness* feature the size of the word database needs to evaluated in order to maximize the performance of the *Word Frequency* feature. A range of sizes from 0 up to 200000 with increments of 500 were tested and the performance of the *Word Frequency* feature was calculated at each step. The result is presented in Figure 5.7. The performance is unchanged after the size of 500, and therefore the optimal configuration is a database size of 500.

5.1.1.4 Overall performance

The performance of the word feature when applying the aforementioned optimal configurations considering training data filtering, tri-gram database size and word frequency database size is presented in Figure 5.8.

Since the features have different performance, their impact on the performance on the SVM are also evaluated. The SVM configuration that is used throughout this test is a linear function kernal with default c-value ($c = 1$). Four different word feature configurations are considered in this test. The first configuration is just to use the *#Alphanumeric* and *Word frequency* since they are taken from previous work

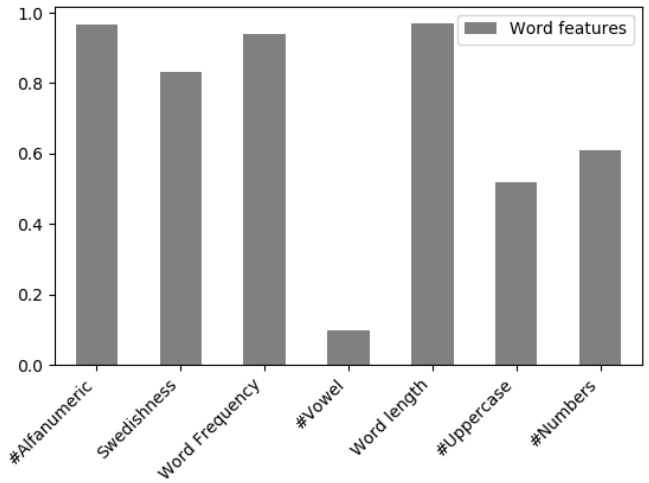


Figure 5.3: Performance of the word feature with only *last char* filter. The performance is displayed in the *percent difference mean* metric.

and is used as a baseline during this project [29, 26]. The next test will exclude the two word features with the poorest performance, *#Vowel* and *#Uppercase*. Thirdly, only the highest performing features is used, i.e. *#Alphanumeric*, *Word Frequency*, *Word length* and *#Numbers*. Lastly, all features are included and the result of all tests is presented in Table 5.1. The result indicates that more features give better results since the best performance occurred when all features were used.

Word metric configuration:	**Precision:**	**Recall:**	**F1-score:**
Only #alphanumeric and Word frequency	0.67	0.62	0.56
All features except #Vovel and #Uppercase	0.77	0.65	0.63
Only high performing features	0.77	0.65	0.61
All features	0.77	0.67	0.63

Table 5.1: The impact of four different configurations of word features on the performance of the SVM.

5.1.2 Support Vector Machine

The evaluation of the SVM is done in two steps, first the choice of kernel is made and then the search for optimal parameters is conducted. Both of these steps will be evaluated using the three metrics *precision, recall* and *F1-score*.

5.1.2.1 Kernel selection

The kernel selection is done by an exhaustive search of all kernels provided by the python library scikit-learn [33]. Any parameters that are offered by the different kernels are set to default values in order to isolate the choice to be only about

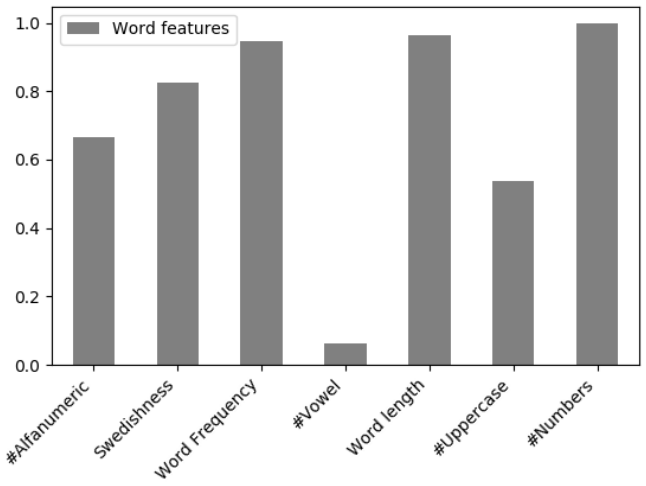


Figure 5.4: Performance of the word feature with only *only non-alpha* filter. The performance is displayed in the *percent difference mean* metric.

kernel selection. The training data is set to be 10000 feature vectors with the optimal configurations for word metrics as presented in Section 5.1.1. The size of the training data is set to a relative low value in order to save some computational time. The performance of each kernel-function is presented in Table 5.2. We can observe that the radial basis kernel function yields the overall best result despite the polynominal function producing better precision.

Kernel function:	**Precision:**	**Recall:**	**F1-score:**
Linear	0.77	0.67	0.63
Polynominal	0.79	0.67	0.64
Radial basis	0.77	0.70	0.67
Sigmoid	0.66	0.64	0.62

Table 5.2: Performance of SVM with different kernel configurations. Training size was 10 000 feature vectors and default parameters was used.

5.1.2.2 Parameter selection

During the parameter selection only the optimal kernel function is considered and as presented in Section 5.1.2.1 the radial basis kernel function performed the best. The radial basis kernel has two parameters, c-value and γ-value and the optimal configuration was found by performing grid-search. The range of values that was be tested is displayed below:

$$c = \{0.0001, 0.01, 1, 100, 1000, 10000, 100000\}$$

$$\gamma = \{0.0001, 0.01, 1, 100, 1000, 10000, 100000\}$$

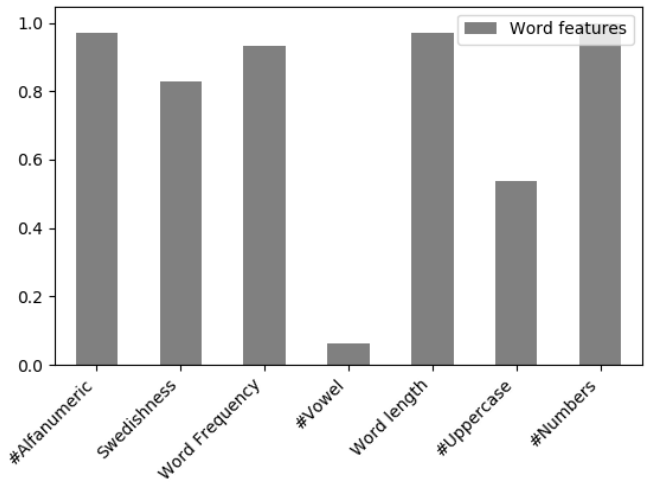


Figure 5.5: Performance of the word features with both *only non-alpha* filter and *last character* filter. The performance is displayed in the *percent difference mean* metric.

The optimal parameters were $c = 1000$ and $\gamma = 1000$ and the resulting performance are displayed in Table 5.3. Notice that these values were produced with a training data size of 100 000 feature vectors in order to get a more optimal performance of the SVM that can be used for evaluation.

Precision:	**Recall:**	**F1-score:**
0.75	0.75	0.75

Table 5.3: Performance of SVM with radial based kernel function configuration. Training size was 100 000 feature vectors and optimal kernel parameters were used ($c = 1000$, $\gamma = 1000$).

5.2 Error correction

The resulting performance of the error correction is evaluated using the optimal configuration for all previous parts of the system. To save some computational time, only a sample of all the available texts is considered in the different tests. There are two targeted texts (Argus and Grepect) and three OCR-tools (Ocropus, Tesseract, ABBYY) and 10 pages of all text-tool combinations were tested. Since there are six possible text-tool combinations, the sample size is 60 pages. The average for each evaluation-tool is then presented.

5.2.1 Input filtering

Similar to the training data the input to the system is filtered through two filter: *last character* filter and *only non-alpha* filter. The effect of these filters on the

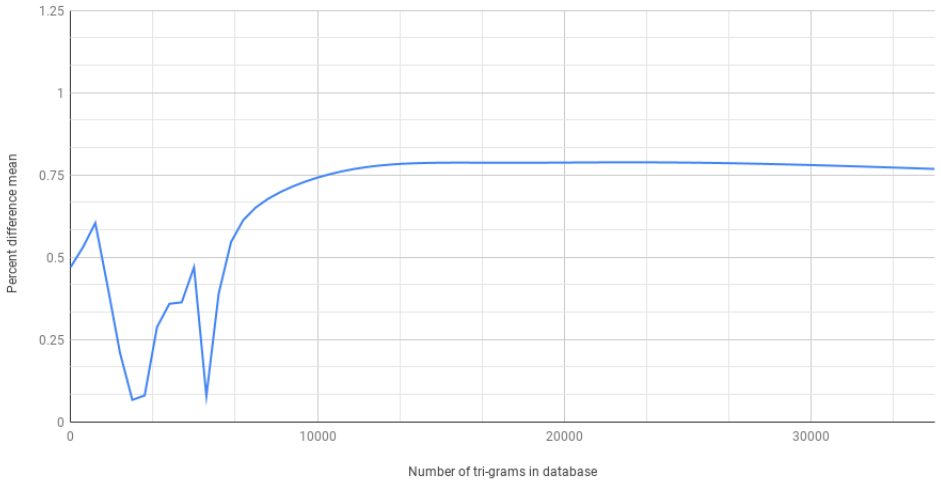


Figure 5.6: Performance of the *Swedishness* metric with different sizes on the tri-gram database. The performance is displayed in the *percent difference mean* metric.

performance of the complete system is evaluated and displayed in Table 5.4. During these tests the *split* algorithm with minimum edit distance zero was used as the default configuration. The two latter configurations, *only non-alpha* filter and both filters perform very similar but "both filter" configuration is selected as the optimal configuration since the difference in word accuracy from PrimA Text Evaluation is substantial.

	OCR Frontier Toolkit		Språkbanken evaluation script		PrimA Text Evaluation	
Filter:	**CACC**	**WACC**	**CER(%)**	**WER (%)**	**CACC**	**WACC**
No filter	0.63	0.36	36.63	84.10	0.61	0.18
Last character filter	0.65	0.39	33.83	82.38	0.65	0.20
Only non-alpha filter	0.66	0.39	32.89	78.90	0.65	0.23
Both filter	0.65	0.39	33.72	77.68	0.65	0.26

Table 5.4: The effect of the input filtering on the performance of the whole system.

5.2.2 Error correction algorithm

There exist two versions of the error correction algorithm. The first and most simple algorithm does not consider any splits of the erroneous words. The second and upgraded version considers all splits of the erroneous word and can therefore correct words that are wrongfully merged. The size of the database containing potential candidates for replacement affects the performance of the error correction algorithm. Therefore the size is fixed to 10000 during the evaluation of the error correction algorithm.

Furthermore, both of the algorithms are tested with different minimum LD threshold. The only thresholds that are reasonable to test are 0 and 1, so those are the only values considered. The result of the different configurations is displayed in Table 5.5.

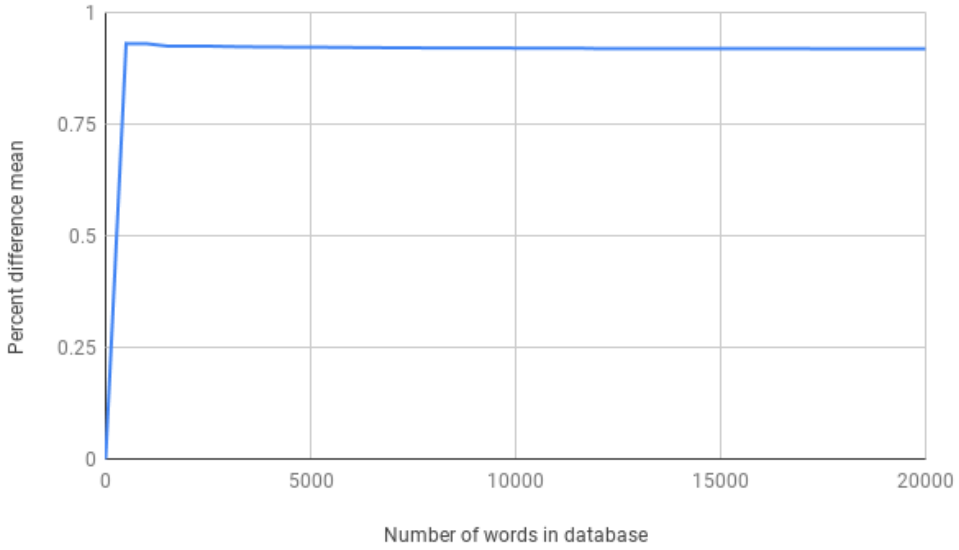


Figure 5.7: Performance of the *Word Frequency* feature with different sizes on the word database. The performance is displayed in the *percent difference mean* metric.

According to Språkbanken evaluation script and PrimA text evaluation, the no split algorithm yields better result on word level but worse on character level. However, according to the OCR frontier toolkit, the split performs better on both word and character levels. Since the split algorithm performs better on character level according to all evaluation-tools and also better on word level according to OCR frontier toolkit, it is the optimal algorithm. The optimal minimum edit-distance is 0 since it yields a better result.

	OCR Frontier Toolkit		Språkbanken evaluation script		PrimA Text Evaluation	
Algorithm:	**CACC**	**WACC**	**CER(%)**	**WER (%)**	**CACC**	**WACC**
No split, minimum edit-distance=0	0.64	0.37	34.96	74.84	0.63	0.29
No split, minimum edit-distance=1	0.64	0.37	35.19	75.51	0.63	0.29
Split, minimum edit-distance=0	0.65	0.39	33.72	77.68	0.64	0.27
Split, minimum edit-distance=1	0.65	0.38	33.90	77.56	0.64	0.27

Table 5.5: Evaluation of different error correction algorithms and minimum edit distances.

5.2.3 Word database size

The error correction stage replaces erroneous words with potentially correct words. These words are extracted from a word database so the size of the database therefore determines which words are potential candidates to replace the erroneous word. The maximum amount of words that is available is 1174679 but it is desirable to use the smallest database possible given that it produces satisfactory results since the computational time rises with the size of the database. The sizes that were tested start off at 10 and increase exponentially up to 1000000 which is close to the maximum value. The result of these tests is displayed in Table 5.6. Considering that

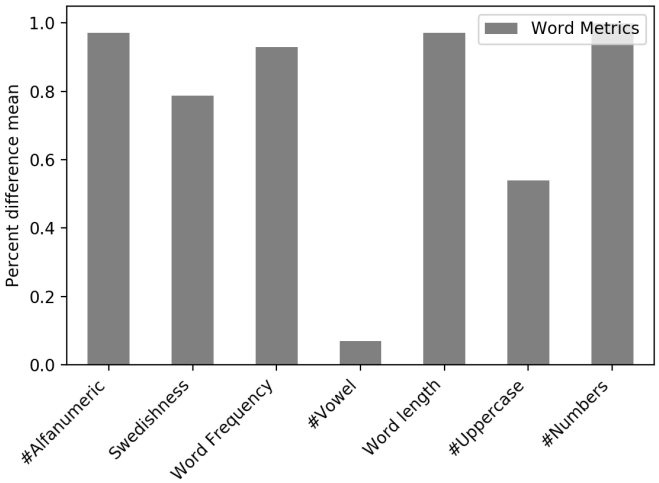


Figure 5.8: Performance of the word features with optimal configurations. The performance is displayed in the *percent difference mean* metric.

the database size increases exponentially the performance of the system increases marginally. The computational time is proportional to the amount of words in the database since every potential replacement words needs to be considered. Considering this the size of 10000 is the most optimal configuration since the largest size of 1000000 only yields minor improvement that is deemed not to justify the much increased computational time.

	OCR Frontier Toolkit		Språkbanken evaluation script		PrimA Text Evaluation	
Word dataset size	**CACC**	**WACC**	**CER(%)**	**WER (%)**	**CACC**	**WACC**
100	0.67	0.38	36.33	77.62	0.63	0.24
1000	0.66	0.39	33.82	77.19	0.65	0.25
10000	0.67	0.40	32.36	76.89	0.67	0.26
100000	0.64	0.38	31.67	77.08	0.66	0.26
1000000	0.68	0.40	31.43	78.37	0.68	0.25

Table 5.6: The performance of the complete system with a number of different word dataset sizes.

5.3 Finalized performance

Below we present the finalized performance of the post-processing system with the optimal configurations for each subsystem. In Table 5.7 the result is displayed and separated by each OCR-tool. Contrary to the previous tests this test been conducted on all available pages in both targeted text. The value that is displayed to the right in either green or red is how much higher or lower, the result is compared to the project baseline presented in Section 4.4. Of the eighteen different values in Table 5.7 seven of them have been managed to be improved by the post-processing system. Unfortunately, eleven of the values have not been improved, but the performance

has rather been decreased. Six of the improvements are made on data generated from the Tesseract engine.

	OCR Frontier Toolkit		Språkbanken evaluation script		PrimA Text Evaluation	
Engine	**CACC**	**WACC**	**CER(%)**	**WER (%)**	**CACC**	**WACC**
Ocropus	0.75 (-0.05)	0.56 (-0.02)	66.49 (+6.89)	24.37 (+6.69)	0.75 (-0.05)	0.38 (-0.03)
Tesseract	0.65 (+0.01)	0.36 (+0.01)	70.86 (-5.51)	33.98 (-0.10)	0.65 (+0.01)	0.27 (+0.02)
ABBYY	0.56 (-0.02)	0.26 (+0.02)	42.79 (+3.49)	92.34 (+5.75)	0.55 (-0.06)	0.09 (-0.1)

Table 5.7: The finalized performance of post-processing method of this project.

A statistical overview of the results is presented in Figures 5.9, 5.10 and 5.11 in the form of histograms of the word length distribution. The figures are divided into two images where the left one is generated from the unprocessed text from the OCR-tool. The right one is generated on the text processed by this projects post-processing method and the red part indicates the amount of words that are represented in our word database. The raw data for these histograms is presented in Appendix A.

The results in Figures 5.9, 5.10 and 5.11 show that the method manages to break down longer strings into smaller units. The longest strings in the different data inputs are: ABBYY 55 characters, Ocropus 66 characters, and Tesseract 63 characters. Compared to the post-processed results where the longest strings are: ABBYY 19 characters, Ocropus 52 characters, and Tesseract 36 characters.

An interesting observation is that the post-processed Ocropus data seems to contain many long words compared to Abbyy and Tesseract. A closer look at the post-processed data shows that the method tends to find more words in the lexicon when processed with the OCR output from Tesseract and consequently reduces the length of the spitted sting, for example: “måttebehöfwafålångeatleta” has been replaced with [“behöfwa”, “långpanna”], “fördennagängcn” has been replaced with [“förena”, “gången”]. While in the post-processed Ocropus dataset we find many examples like: “Fruentimretföraktademmittlilla” that has been replaced with [“Fruentimretföraktademmitt”, “lilla”]. A possible improvement of the split method is the perform additional splits on processed the strings.

A closer look at the successfully generated strings shows that the method successfully recognizes shorter strings that are up to 4 characters. From the statistical overview of the results, we note that four character long strings constitute half of the strings in each dataset (OCR output). When we compare the correctly post-processed strings that are up to 4 characters long, we learn that the method successfully recognizes and corrects them regardless of the output from the OCR system, some examples are given in Table 5.8.

The content of each output varies significantly, when we look at the statistics over how many strings are included in our database, i.e., they are correct words, we learn that the percentage of words that are found in the database is highest for Tesseract. This also explains why the evaluation performance only shows an improvement for

OCR-word	Corrected-word	Ocropus	Tesseract	Abbyy
cn	en	58	2	26
pd	på	34	1	38
rn	en	11	1	2
mcd	med	29	5	0
ndr	när	16	2	8
ych	och	17	4	0
Hwad	Hwar	30	29	0
fttt	fått	0	0	29

Table 5.8: Examples of correctly recognized words for each engine.

Tesseract. A more objective evaluation would be to evaluate the performance on only strings that are up to 4 characters long.

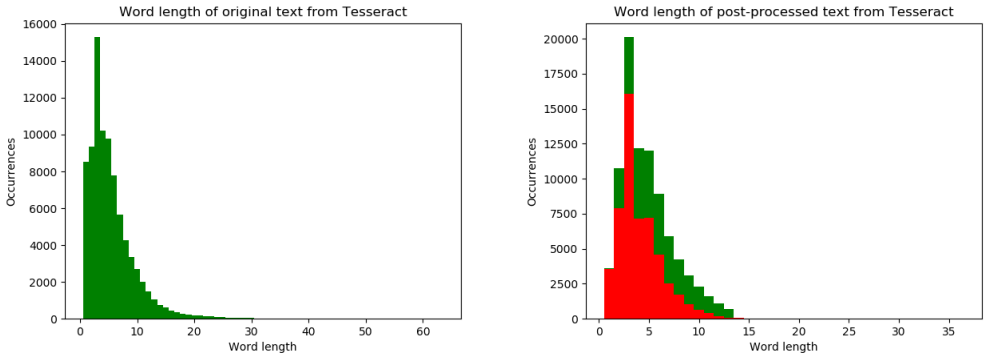


Figure 5.9: Histogram of the word length distribution of texts from Tesseract before and after the post-processing. The red part of the right figure indicates how many of the words are represented in our word database.

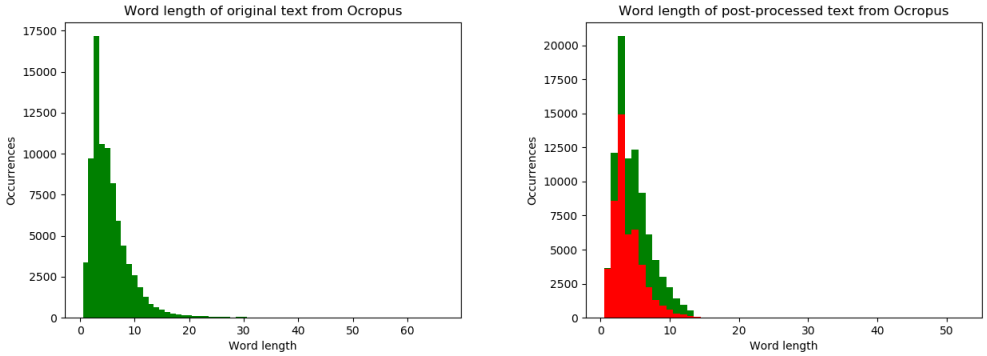


Figure 5.10: Histogram of the word length distribution of texts from Ocropus before and after the post-processing. The red part of the right figure indicates how many of the words are represented our word database.

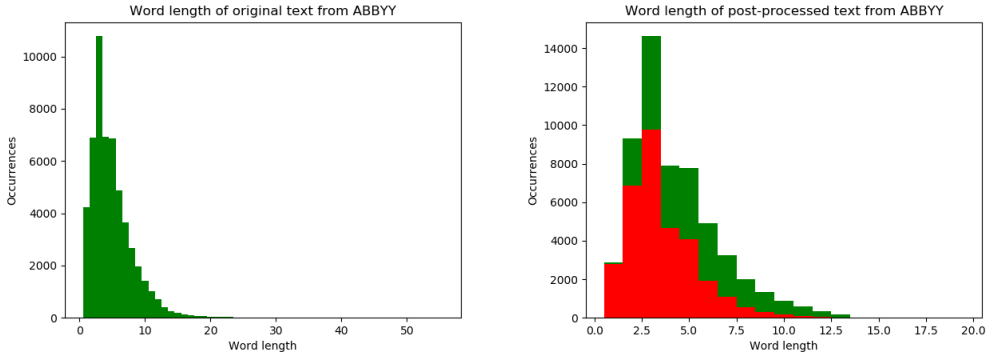


Figure 5.11: Histogram of the word length distribution of texts from ABBYY before and after the post-processing. The red part of the right figure indicates how many of the words are represented in our word database.

6
Discussion

This chapter analyses and discusses the results and method of this project. The discussion follows the same structure as the Chapter 5.

6.1 Error detection

The first part of the post-processing system is the error detection stage. This stage labels all words in the input text as either valid or erroneous. The error detection system is evaluated in Section 5.2, both the results and method are discussed in this section.

6.1.1 Word features

In Section 5.1.1 the word features for this project were evaluated and the finalized performance of the word features was presented in Figure 5.8. Overall, we observe that all features perform very well considering that a value of 1 is the highest possible. The two features that perform the worst are the *#Vowel* and *#Uppercase*. Below each feature is discussed and analyzed whenever it performed well or not.

The *#Alphanumeric* feature returns the number of characters in a given word that are not normal characters or numbers. The motivation for this feature is that no valid word contains any non-alphanumeric characters and that OCR-tools sometimes misrecognize normal characters with special characters that have similar shape, e.g., "till" gets misrecognized with "t;ll". The *#Alphanumeric* feature yields great results which is presented in Figure 5.8.

The *#Vowel* feature is the worst performing metrics. The motivation for the use of the *#Vowel* feature is that no Swedish word contains zero vowels. So the absence of vowels would indicate that the word is erroneous. The poor performance of the *#Vowel* could most likely be attributed to that most erroneous words also contain vowels. The chance that an erroneous word contains a vowel is even higher since the OCR-tool tends to merge adjacent words and thus creating very long words. Other contributing factors to the poor performance are that words that only contain non-alfa character get filtered by the *non-alfa*-filter. Since these words are often erroneous and contain no vowels this would increase the performance of the feature. This can be observed when comparing the result with *non-alfa*-filter in Figure 5.4

and without in Figure 5.2.

The other word feature that shows poor performance is the *#Uppercase* feature, however the performance is much greater than the *#Vowel* feature. The motivation for this feature is that a word with multiple capital letters is probably erroneous, e.g. "JagSka" should be "Jag Ska". The reasons for the poorer performance is probably that most words only have lower case letters and some valid words are written in all uppercase letters. This causes the feature to not give a clear indication of the validity of the word.

One of the most high performing features is the *Word Frequency* feature. The high performance could be contributed to the database of words and frequencies that are generated from texts from the same language and time period. The performance would probably drop if the database was not representative of the targeted text. With a database generated from modern text some historical spelling variants of words would not be present and valid words would get a low score in this feature. The *Swedishness* feature relies on a database with character tri-grams and their frequencies. This database is also generated from the same texts as the word frequency database and impact the *Swedishness* feature in a similar way.

The *Word length* feature returns one if the word is over 13 characters and the motivation is that very few valid words are that long. This is further strengthened by the fact that OCR-tools tend to sometime merge adjacent and thus creating long words.

6.1.1.1 Tri-gram database size

The size of the character tri-gram database was evaluated and the result is presented in Figure 5.6. The performance of the feature is unsteady and low for small sizes but stabilizes after 10000 words. The performance starts to decline slightly after 30000 words but the optimal size is in the range of 12000 to 22000. The unstable performance when the size is low is expected since the data is not yet representative of the reality. However, as the size of the database increases the data gives a better picture of which tri-grams are common and not.

6.1.1.2 Word frequencies database size

The optimal size for the word frequency database when considering the performance of the *Word frequency* feature is evaluated in Section 5.1.1.3. As is displayed in Figure 5.7, the performance is more or less the same after the size reaches 100 words. Since the most common words make up a relatively large portion of the total text, the database does not need to be very large in order to produce great results. As long as it is large enough to contain the most common words like "och" (and) and "jag" (I) it is sufficient to produce a difference in the average between the error and valid class.

6.1.1.3 Training data filtering

Two filters that are used to filtrate the training data were evaluated in Section 5.1.1.1. These filters are called *last char* filter and *non-alfa* filter. This section analyzes and discusses the filter's impact on the performance of the SVM.

The *last char* filter removes the last character of the word if it is a punctuation symbol, e.g. ´?', ´,' ´!'. The motivation for this filter is that our method is designed to process real words. This filter causes valid words at the end of a sentence like "how?" to be changed to "how" and thus easier for the system to correctly identify it as a valid word. However it removes some information from the words that can be crucial, consider the erroneous word generated by am OCR-tool "to." which should actually be "too". By removing the last character of the erroneous word it produces a word that is represented in the database and is valid in other contexts. There will thus be two versions of "to" in the training data, one labeled as valid and one as erroneous. This may cause the word features to perform worse. The analysis of the impact of the *last char* filters on the word features is done by comparing the result, without the filter in Figure 5.2 and with the filter in Figure 5.3. The impact of the filter is that *#Alfanumeric* feature is improved and is probably a consequence of that different instances of the valid word gets unified, i.e. "example?" and "example," gets both represented as "example".

The *non-alfa* filter removes words that only contain symbols and numbers. The desired functionality for this filter is to remove some strings that are labeled valid but not actual words like individual symbols or numbers, e.g. "?" or "1893". Just like the *last char* filter this is to limit the valid words in the training data to actual words and thus, improving the performance of the word features. The analysis of the impact of the *non-alfa* filter word features is done by comparing the results, without the filter in Figure 5.2 and with the filter in Figure 5.4. The significant differences in the performance of the word features are the *#Numbers*. This is probably a consequence of that all years and dates from the valid words are removed. This causes very few valid words to contain any numbers at all, however the erroneous words still contain numbers since characters sometimes get recognized as numbers. This highlights a difference in the occurrences of numbers in erroneous and valid words, in valid words it is often an entire string, e.g. "1855" whilst in erroneous word the numbers are intertwined with other characters, e.g. "4fter".

6.1.1.4 SVM

The error detection is based on a binary classification conducted by an SVM that classifies each word as valid or erroneous. The performance of the SVM is evaluated using *precision, recall* and *F1-score* metrics in Section 5.2. Compared to previous work by Khirbat [26] the result is much higher and has better values in all three metrics. However, there is still room for improvements and a better error detection stage would greatly impact the overall performance of the system. Improvements of the SVM could be achieved by extending the feature vectors with more and better word features. This will provide the SVM with more information that could

help it to find a solution to the binary classification problem with higher accuracy. Word features that could potentially extend the feature vectors are discussed in Section 6.7.1 as future work.

6.2 Error correction

The second part of the system is the error correction stage which tries to find correct replacement for the erroneous words detected by the previous step. The error detection stage was evaluated in Section 5.2.

6.2.1 Input filtering

The impact of filtering the input before it enters the system is evaluated in Section 5.2.1. The filters are identical to those that filter the training data and are called *last-char* filter and *non-alpha* filter. Similar to the training data filtering four different configuration were tested: no filter, only *last-char* filter, only *non-alpha* filter and both filters. Both of the filters impact the output of the system positively and the optimal configuration is to use both filters. Below the impact of the filters is discussed.

The OCR-tool occasionally misrecognize images, decorative patterns or ink bleed-through as text and generates many short words. This causes the error correction algorithm to try correcting these words which may introduce more character errors if the replacing word is longer than the erroneous. However, a large part or all of these misrecognized words are non-alpha characters which make the *non-alpha* filter useful. In Table 5.2.1 the benefit of the *non-alpha* filter could be observed in the improvement of the word based result from no filter to only *non-alpha* filter.

A possible negative consequence of the filters is that they remove essential information from the input. Consider the word erroneous word “mode!” which should be corrected to “model”, the *last-char* filter would remove the last character and the thus generating the word “mode”. This word is represented in the database and if the error correction has a minimum edit distance of zero it will not be corrected further and thus creating an error.

6.2.2 Correction algorithm

Two error correction algorithms were evaluated in Section 5.2.2. The first algorithm is called *no-split* and simply tries to find a replacement for the erroneous word. The other algorithm is called *split* and splits the erroneous word one time in every possible way and tries to correct the splits as well. The *split* algorithm can separate words that have been wrongfully merged by the OCR-tool.

The apparent advantage of the *split* algorithm over the *no-split* version is the ability to split up wrongfully merged words. However, since there exist some short and very common words in the Swedish language this split might become disadvantageous.

Consider the erroneous word “jagi” which should be corrected to “jag”, the split algorithm would split the erroneous word into “jag” and “i” since they both are common Swedish words. However, the no-split algorithm is not able to split the erroneous word and would therefore correct “jagi” into “jag” which would be correct in this case.

6.2.3 Word database size

Evaluation of different word database sizes is conducted in Section 5.2.3 and the optimal size of 10000 words was chosen. Analysis of the values in Table 5.6 shows that the performance increases marginally if at all when the database size grows. This points to that errors caused by the error correction system is most likely not due to a lack of potential candidate words. The errors caused by the error correction stage is probably due to the way candidates are selected or ranked.

6.3 Overall performance

The overall performance of the system is summarized in Table 5.7. The system shows an improvement of seven of the eighteen possible metrics. Most of the improvement were made on the data from Tesseract which points to that Tesseract produces errors that our system manages to correct. The result shows that the method has the ability to improve the quality of historical Swedish but also that there exists room for improvement. As stated throughout the discussion of this project, many of the subsystems offer room for improvements. And since each subsystem directly influences the next subsystem the overall performance of the system is therefore affected by each subsystem.

6.4 Bias for any OCR-tool

Part of the project aims is to develop a method that is not biased towards any single OCR-tool. To accomplish this each tool is considered as a “black box” and the post-processing is developed as a completely separate entity. As stated in Section 5.3 our system seems to perform best on short words up to 4 characters. That would explain the poorer performance on the Ocropus output since it contains many long words. The conclusion is that our OCR-tool is not biased towards any OCR-tool but excels at correcting a specific type of errors.

6.5 Suitability of a machine learning approach

Part of the project aims is to determine if a machine learning approach is suitable for solving the problem of post-processing historical Swedish texts. An important requirement for machine learning is access to an extensive dataset. The availability of large dataset depends greatly on the target language and time period. New data can be constructed given there exist more literature from the desired time period,

however manual transcription is required which makes it a slow and expensive process. The chosen machine learning techniques for this project is an SVM which requires less training data than other approaches. So for some machine learning approaches the restricted training data available might be a challenge to overcome.

During this project machine learning was used to classify each word as either valid or erroneous as the error detection stage. This process produces satisfactory results but shows room for improvement as well. A machine learning approach is deemed to be suitable for post-processing of historical Swedish texts given that enough annotated data is produced.

6.6 Limitations caused by the targeted text

This project suffers from several limitations which originate from the target texts. Since the targeted text is Swedish text from the 19th century it poses some difficulties described in Section 2.1. These difficulties reflect the performance of the OCR-tools which in some cases is very poor and even to the point that most of the text is erroneous. Understandably, this makes the post-processing harder since much of the information is lost due to the OCR-tool.

The targeted text is also not annotated with extra information about the class of each word (*noun, verb, adjective, etc.*). If such information was available it could have been used to improve the performance of the system further. Considering these limitations it is unreasonable to expect that the method proposed here would find and correct all the OCR errors.

6.7 Future work

This Section discusses future work to potentially improve the method and result.

6.7.1 Word features

There are many types of word features that could be used to indicate the validity of a word. During this project only two types of features were considered, word-based and statically based. However, there exist context-based features as well that consider the text surrounding the word. Example of this is presented by Mei et al. [29] which utilize a sliding window to construct a word based n-gram context for each word. The frequency of the context for that word is then extracted from a context frequency database. A word is more likely to be correct if it occurs with that context in other places. These types of context-based features open up another dimension for the error detection since it can detect real-word errors. The addition of context-based word features could improve the performance of the error detection method that is explored in this project.

Another possible improvement is to change the *word length* feature to simply return the word length instead of one if the value is over 13 characters. This would possibly give a more nuanced feature that could provide more information to the SVM.

6.7.2 Error detection

This project method for detecting error is to use an SVM to conduct a binary classification of the words in order to predict their validity. But there exist other machine learning methods for binary classification. Popular python library Keras [20] offers useful tools that allow users to construct neural networks that solve a wide range of problems including binary classification problems.
The error correction stage of this project uses an algorithm that is capable of splitting the word one time. An improvement to this algorithm would be to increase the number of possible splits. By considering all possible splits of a word it can potentially correct more errors caused by wrongfully merged words. However, the computational time would increase since the number of splits that needs to be evaluated would increase drastically.

Another improvement that would potentially increase the performance of the error correction algorithm is to change the way the algorithm finds suitable candidates. The current way is to select candidates that are closest by Levenshtein edit distance to the given erroneous word. By extending the information that is used to find candidates the accuracy would increase. A potential extension is to use character bi-gram edit distance which is yet another version of the normal Levenshtein edit distance but for two characters instead of one. By considering bi-gram edit distance consecutive correct characters would be taken into account as well. More extensions could include an improved Levenshtein edit distance which gives lower edit distance for characters that have visual similarities and often get misrecognized, e.g. “i” and “l”. Higher edit distance would be given for characters that have few visual similarities and do not get frequently misrecognized, e.g. “W” and “H”.

The currently implemented error correction algorithm cannot remove words. This is a desired feature when trying to correct historical texts since the OCR-tool miss recognizes images, decorative patterns or ink bleed-through as text. This causes many erroneous words that should not be corrected but rather removed. As mentioned the input filters previously, partially solve this problem but a more effective solution is needed.

7
Conclusion

This project aimed to develop a method for OCR post-processing of historical Swedish texts using machine learning techniques. The two main problems of OCR post-processing is to detect and correct errors caused by the OCR-tool. Our method for error detection is to utilize a Support Vector Machine (SVM) that classifies each word to either valid or erroneous. This method originates from previous work by Khirbat [26] but is extended with more extensive training data. The training data is a set of feature vectors that contains several word features. Word features are computed for each word, all together they indicate the validity of the given word. The generation of training data is a fully automatic process that utilizes a modified version of Språkbankens evaluation script to extract erroneous words. The chosen approach for error correction is an adaptation of the frequency based correction algorithm proposed by Kettunen et al. [25]. This approach searches for potential candidates that could replace the erroneous word. The selection of candidates is based on the closest word in Levenshtein distance (LD). If multiple candidates have the same LD the one with the highest frequency gets selected. The database of potential candidates is extracted from 19th-century Swedish literature that was supplied and proofread by the Project Runeberg [7]. The targeted texts are Swedish texts from the 19th century and called *Swänska Argus* and *Grepect* and they amount to a total of 400 pages. Ground truth for each page was available which is crucial for evaluation.

The system is constructed in a modular fashion and each part of the system was evaluated, the results of each part are presented. The final evaluation of the post-processing method was done by using three different OCR evaluation systems. To get a reliable evaluation results we considered multiple OCR evaluation tools since they tend to produce different values [25]. The finalized result is presented in Table 5.7, showing the method was able to improve 7 of 18 possible values. Moreover, the method seems to perform best on short words with a length of up to 4 characters as presented in the Section 5.3.

Another part of the aim is to determine whether a machine learning approach is suitable for post-processing of historical Swedish texts. This is discussed in Section 6.5. The conclusion is yes, but if larger access of training material was available, the machine learning approach, like the one proposed here, would have probably been more successful.

The last part of the project aims is not to be biased towards any OCR-tool. The

method for achieving this was to consider all the OCR-tools as black boxes and not consider any of the inner workings. Since the method excels at solving certain types of errors the performance differs depending on the OCR-tool since each OCR-tool is prone to make certain types of errors.

Bibliography

[1] Abbyy finereader. <http://finereader.abbyy.com/>. Accessed: 2019-04-01.

[2] Grepect. <http://www.grepect.de/>. Accessed: 2019-04-26.

[3] IMPACT project. <http://www.impact-project.eu/about-the-project/concept/>. Accessed: 2019-04-22.

[4] Nationalencyklopedin, Then swänska Argus. <https://www.ne.se/uppslagsverk/encyklopedi/l%C3%A5ng/then-sw%C3%A4nska-argus/>. Accessed: 2019-04-26.

[5] Ocropus. https://github.com/tmbdev/ocropy. Accessed: 2018-12-05.

[6] PrimA text evaluation. <https://www.primaresearch.org/tools/PerformanceEvaluation>. Accessed: 2019-03-25.

[7] Project runeberg. <http://runeberg.org/>. Accessed: 2019-04-26.

[8] Språkbanken evaluation script. <http://demo.spraakdata.gu.se/svedd/ocrproject/script/align.py>. Accessed: 2018-12-14.

[9] Språkbankens image database. <http://demo.spraakdata.gu.se/gerlof/ocrprojektet/complete/>. Accessed: 2019-04-26.

[10] Tesseract. https://github.com/tesseract-ocr/. Accessed: 2018-12-05.

[11] Trove database. <https://trove.nla.gov.au//>. Accessed: 2019-04-22.

[12] Andrew D. Bagdanov, Stephen V. Rice, and Thomas A. Nartker. The OCR frontiers toolkit version 1.0.

[13] Hildelies Balk. Responding to the challenges in mass digitisation of historical printed text. In *The IMPACT workshop at the EVA MINERVA Conf*, 2008.

[14] Lars Borin, Gerlof Bouma, and Dana Dannélls. A free cloud service for OCR/en fri molntjänst för OCR project report. Technical report, Institutionen för svenska språket, Göteborgs universitet, 2016.

[15] Lars Borin and Markus Forsberg. A diachronic computational lexical resource for 800 years of Swedish. In *Language technology for cultural heritage*, pages 41–61. Springer:Berlin, 2011.

[16] Derek Bradley and Gerhard Roth. Adaptive thresholding using the integral image. *Journal of graphics tools*, 12(2):13–21, 2007.

[17] Thomas M. Breuel. The OCRopus open source OCR system. In *DRR*, 2008.

[18] Jean-Christophe Burie, Joseph Chazalon, Mickaël Coustaty, Sébastien Eskenazi, Muhammad Muzzamil Luqman, Maroua Mehri, Nibal Nayef, Jean-Marc Ogier, Sophea Prum, and Marçal Rusiñol. Icdar2015 competition on smartphone document capture and OCR (smartdoc). In *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, pages 1161–1165. IEEE, 2015.

[19] Cyril Goutte and Eric Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In *European Conference on Information Retrieval*, pages 345–359. Springer, 2005.

[20] Antonio Gulli and Sujit Pal. *Deep Learning with Keras*. Packt Publishing Ltd, 2017.

[21] Rose Holley. How good can it get? Analysing and improving OCR accuracy in large scale historic newspaper digitisation programs. *D-Lib Magazine*, 15(3/4), 2009.

[22] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification. 2003.

[23] Ines Jerele, Tomaž Erjavec, Daša Pokorn, and Alenka Kavčič-Čolić. Optical character recognition of historical texts: end-user focused research for Slovenian books and newspapers from the 18th and 19th century. In *6. SEEDI Conference: Proceedings (16–20 May 2011, Zagreb, Croatia)*, page 11, 2011.

[24] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998.

[25] Kimmo Kettunen, Timo Honkela, Krister Lindén, Pekka Kauppinen, Tuula Pääkkönen, Jukka Kervinen, et al. Analyzing and improving the quality of a historical news collection using language technology and statistical machine learning methods. In *IFLA World Library and Information Congress Proceedings 80th IFLA General Conference and Assembly*. IFLA, 2014.

[26] Gitansh Khirbat. OCR post-processing text correction using simulated annealing (opteca). In *Proceedings of the Australasian Language Technology Association Workshop 2017*, pages 119–123, 2017.

[27] Gurpreet S Lehal and Chandan Singh. A post-processor for Gurmukhi OCR. *Sadhana*, 27(1):99–111, 2002.

[28] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, pages 707–710, 1966.

[29] Jie Mei, Aminul Islam, Yajing Wu, Abidalrahman Moh'd, and Evangelos E Milios. Statistical learning for OCR text correction. *arXiv preprint arXiv:1611.06950*, 2016.

[30] Frederic P Miller, Agnes F Vandome, and John McBrewster. Levenshtein distance: Information theory, computer science, string (computer science), string metric, damerau? levenshtein distance, spell checker, hamming distance. 2009.

[31] Diego Mollá-Aliod and Steve Cassidy. Overview of the 2017 ALTA shared task: Correcting OCR errors. In *Proceedings of the Australasian Language Technology Association Workshop 2017*, pages 115–118, 2017.

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[33] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[34] Ashis Pradhan. Support vector machine-a survey. *International Journal of Emerging Technology and Advanced Engineering*, 2(8):82–85, 2012.

[35] Sukhpreet Singh. Optical character recognition techniques: a survey. *Journal of emerging Trends in Computing and information Sciences*, 4(6):545–550, 2013.

[36] Ray Smith. An overview of the tesseract OCR engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 629–633. IEEE, 2007.

[37] Oeivind Due Trier and Torfinn Taxt. Evaluation of binarization methods for document images. *IEEE transactions on pattern analysis and machine intelligence*, 17(3):312–315, 1995.

[38] Tommi Vatanen, Jaakko J Väyrynen, and Sami Virpioja. Language identification of short text segments with n-gram models. In *LREC*, 2010.

A

Appendix 1

Tesseract (Word length, Occurrences): (1, 8533), (2, 9332), (3, 15288), (4, 10206), (5, 9789), (6, 7767), (7, 5652), (8, 4274), (9, 3358), (10, 2714), (11, 2019), (12, 1472), (13, 1046), (14, 751), (15, 630), (16, 449), (17, 375), (18, 282), (19, 239), (20, 202), (21, 171), (22, 136), (23, 122), (24, 105), (25, 75), (26, 68), (27, 60), (28, 58), (29, 55), (30, 44), (31, 30), (32, 24), (33, 19), (34, 14), (35, 10), (36, 13), (37, 14), (38, 16), (39, 9), (40, 14), (41, 10), (42, 6), (43, 2), (44, 6), (45, 3), (46, 4), (47, 8), (48, 5), (49, 6), (50, 2), (51, 2), (52, 3), (53, 2), (55, 2), (56, 1), (57, 1), (63, 1)

Ocropus (Word length, Occurrences): (1, 3388), (2, 9737), (3, 17171), (4, 10585), (5, 10357), (6, 8206), (7, 5888), (8, 4391), (9, 3282), (10, 2610), (11, 1845), (12, 1277), (13, 852), (14, 633), (15, 492), (16, 355), (17, 272), (18, 226), (19, 171), (20, 152), (21, 108), (22, 108), (23, 85), (24, 72), (25, 50), (26, 58), (27, 42), (28, 30), (29, 36), (30, 37), (31, 25), (32, 25), (33, 23), (34, 25), (35, 18), (36, 24), (37, 12), (38, 11), (39, 13), (40, 13), (41, 8), (42, 7), (43, 8), (44, 5), (45, 1), (46, 7), (47, 3), (48, 8), (49, 5), (50, 3), (51, 7), (52, 1), (53, 3), (54, 1), (55, 2), (56, 1), (57, 1), (58, 1), (60, 3), (61, 2), (62, 1), (64, 2), (66, 1)

ABBYY (Word length, Occurrences): (1, 4238), (2, 6896), (3, 10778), (4, 6941), (5, 6852), (6, 4865), (7, 3665), (8, 2667), (9, 1953), (10, 1408), (11, 1021), (12, 717), (13, 390), (14, 263), (15, 181), (16, 143), (17, 96), (18, 73), (19, 66), (20, 50), (21, 33), (22, 24), (23, 23), (24, 15), (25, 13), (26, 20), (27, 10), (28, 15), (29, 8), (30, 4), (31, 5), (32, 5), (33, 9), (34, 1), (35, 4), (36, 5), (37, 3), (39, 3), (40, 1), (41, 1), (42, 3), (45, 1), (47, 1), (48, 5), (49, 1), (55, 1)

Ground truth (Word length, Occurrences): (1, 2564), (2, 11661), (3, 21986), (4, 12708), (5, 12824), (6, 9523), (7, 6579), (8, 4512), (9, 3269), (10, 2419), (11, 1527), (12, 937), (13, 508), (14, 238), (15, 150), (16, 67), (17, 35), (18, 13), (19, 13), (20, 12), (21, 2), (22, 2), (24, 2)

OutputABBYY (Word length, Occurrences, #Represented in lexicon): (1, 2872, 2812), (2, 9334, 6855), (3, 14623, 9789), (4, 7904, 4677), (5, 7778, 4065), (6, 4929, 1903), (7, 3230, 1076), (8, 1982, 536), (9, 1342, 311), (10, 872, 183), (11, 593, 101), (12, 320, 42), (13, 160, 20), (14, 18, 18), (15, 2, 2), (16, 3, 3), (17, 2, 2), (18, 2, 2), (19, 1, 1)

OutputOcropus (Word length, Occurrences, #Represented in lexicon): (1, 3644, 3583), (2, 12121, 8565), (3, 20671, 14899), (4, 11717, 6115), (5, 12349, 6453), (6,

9195, 3875), (7, 6131, 2220), (8, 4268, 1314), (9, 3011, 884), (10, 2259, 587), (11, 1412, 326), (12, 946, 240), (13, 561, 104), (14, 63, 61), (15, 32, 29), (16, 21, 18), (17, 16, 11), (18, 10, 10), (19, 5, 2), (20, 2, 0), (21, 4, 3), (22, 1, 0), (23, 5, 0), (24, 3, 0), (27, 1, 0), (29, 1, 0), (30, 1, 0), (31, 1, 0), (32, 2, 0), (34, 1, 0), (36, 1, 0), (37, 2, 0), (41, 1, 0), (48, 1, 0), (50, 1, 0), (52, 1, 0)

OutputTesseract (Word length, Occurrences, #Represented in lexicon): (1, 3622, 3525), (2, 10764, 7889), (3, 20104, 16082), (4, 12169, 7153), (5, 12024, 7209), (6, 8894, 4602), (7, 5916, 2519), (8, 4266, 1706), (9, 3090, 1011), (10, 2269, 617), (11, 1610, 428), (12, 1075, 202), (13, 710, 96), (14, 80, 79), (15, 32, 26), (16, 34, 29), (17, 14, 10), (18, 10, 7), (19, 6, 3), (20, 1, 0), (21, 4, 3), (23, 3, 0), (28, 2, 0), (29, 2, 0), (30, 2, 0), (34, 1, 0), (36, 1, 0)