

Undergraduate Thesis

Improved Dehyphenation of Line Breaks for PDF Text Extraction

Mari Sverresdatter Hernæs

Examiner: Prof. Dr. Hannah Bast

Adviser: Claudius Korzen

University of Freiburg

Faculty of Engineering

Department of Computer Science

Chair of Algorithms and Data Structures

December 2nd, 2019

Examiner

Prof. Dr. Hannah Bast

Adviser

Claudius Korzen

Declaration

I hereby declare, that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.

I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date

Signature

Abstract

Words in layout-based documents can contain hyphens that divide the word into two parts across two lines. PDF documents only store information about individual characters, which makes it difficult to extract text correctly. Words which are hyphenated at the end of a line are especially problematic because in English there are some words where the hyphen should be kept, while others need to be merged. For example, “high-quality” can be a compound word. If this word splits on the hyphen across two lines, then the hyphen should be retained in the extracted text. The correct dehyphenation of line breaks requires the recognition of either words or sequences of characters. In this thesis, vocabulary-based baseline algorithms, logistic regression on the word level and a bi-LSTM Language Model on the character level are used to solve this problem. On the ClueWeb12 Extract data set, the vocabulary-based algorithm achieved a balanced accuracy (bACC) of 66.87%, the logistic regression 92.38% and the bi-LSTM Language Model 90.14%. Our investigations showed a trade-off effect between recognising words which naturally contain hyphens and predicting words without hyphens correctly.

Zusammenfassung

Wörter in layoutbasierten Dokumenten können Bindestriche enthalten, so dass sie in zwei Teilen über zwei Zeilen erscheinen. Diese Arbeit beschreibt verschiedene Methoden, um diese Wörter korrekt zusammenzuführen. PDF-Dokumente enthalten nur Informationen über einzelne Zeichen, was die korrekte Extrahierung von Texten erschwert. Besonders problematisch sind Wörter, die am Ende einer Zeile mit einem Bindestrich versehen sind. Im Englischen gibt es Wörter, in denen der Bindestrich beibehalten werden soll, andere, die zusammengeführt werden müssen. Wenn sich beispielsweise “high-quality” auf den Bindestrich über zwei Zeilen verteilt, sollte der Bindestrich im extrahierten Text erhalten bleiben. Um Zeilenumbrüche korrekt zusammenzuführen, müssen entweder Wörter oder Zeichenfolgen erkannt werden. In dieser Arbeit werden wörterbuchbasierte Algorithmen, logistische Regression auf Wortebene und ein bi-LSTM-Sprachmodell auf Zeichenebene zur Lösung dieses Problems verwendet. Auf dem ClueWeb12 Extract Datensatz erzielte der wörterbuchbasierte Algorithmus eine ausgeglichene Genauigkeit (bACC) von 66,87%, die logistische Regression 92,38%, und das bi-LSTM Sprachmodell 90,14%. Unsere Untersuchungen zeigten einen Trade-off-Effekt zwischen dem Erkennen von Wörtern, die allgemein mit Bindestrich geschrieben werden und der korrekten Vorhersage von Wörtern ohne Bindestrich.

Contents

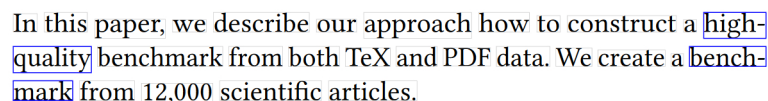
1	Introduction	1
1.1	Overview of the thesis	4
2	Related Work	6
2.1	Noisy text correction	6
2.2	Conversion tools	7
2.3	Hyphenation tools	8
2.4	Spelling correction tools	8
3	Background	9
3.1	Typography	9
3.2	Electronic documents	10
3.2.1	PDF	11
3.2.2	EPUB	11
3.2.3	The ideal electronic document	11
3.3	Liang’s hyphenation algorithm	12
3.4	Hy-phens	13
3.5	Natural Language Processing	15
3.5.1	Motivation	16
3.5.2	Hidden Markov Model	16
3.5.3	Deducing POS-tags with the Viterbi algorithm	18
3.5.4	Maximum Entropy Markov Model	19
3.5.5	Bidirectionality	19
3.6	Conditional Random Field	21
3.7	Bidirectional LSTM	22
4	Approach	25
4.1	Data sets	25
4.1.1	ClueWeb12	25
4.1.2	Ontonotes	26

4.2	Data preprocessing	27
4.2.1	Filtering	28
4.2.2	Cleaning	28
4.3	Adding hyphens to sentences	29
4.3.1	The final hyphenated data set	31
4.3.2	Training and test data	31
5	Implementation	33
5.1	Vocabulary-based baseline algorithms	33
5.1.1	Creating a vocabulary	33
5.1.2	Baseline algorithm	35
5.1.3	Supplemented baseline algorithm	36
5.2	Logistic regression	37
5.3	Using the bi-LSTM	38
6	Evaluation	41
6.1	Metrics	41
6.2	Results	43
6.3	Analysis of types of errors	44
7	Future Work	47
8	Conclusion	49
9	Acknowledgments	50
	Bibliography	51

1 Introduction

The topic of this thesis is inspired by the need for even more precise text extraction as a part of Bast and Korzen’s *Icecite* system [1]. Icecite is a research management system that precisely extracts metadata, references, annotations and full texts from the PDF files of scientific research papers.

The system has a fast search-as-you-type functionality which is driven by CompleteSearch [2]. To be able to search in the documents with high accuracy, however, it becomes necessary to extract the text with even more precision. Here emerges the topic of this thesis. The subproblem emphasized in this thesis is the correct extraction of words with hyphens at the end of a line. Consider Figure 1.1, which shows a typical PDF hyphenation.



In this paper, we describe our approach how to construct a high-quality benchmark from both TeX and PDF data. We create a benchmark from 12,000 scientific articles.

Figure 1.1: **An example of PDF hyphenation.** This example shows text with two end-of-the-line hyphenations: *high- quality* and *bench- mark*. The challenge is to extract these correctly as *high-quality and benchmark*.

A PDF document is like printed pages on a computer screen. The file does not include the text as it is but rather information on how to draw the page to obtain the desired layout. To be specific, the PDF provides information about the single letters with their position and font (if it is not scanned). Icecite already composes words out of these letters, in the correct reading order. However, words split between two lines are still two separated word parts. The subproblem in this thesis is to find out how to assemble these two-word parts, for which there are four different approaches:

1. Do not merge the parts. For example: extract *bench- mark* as the two words *bench-* and *mark*
2. Always merge the parts without a hyphen. For example: extract *bench- mark* as *benchmark* and *high- quality* as *highquality*
3. Always merge the parts with a hyphen. For example: extract *bench- mark* as *bench-mark* and *high- quality* as *high-quality*
4. Merge the parts with or without hyphen depending on whether the actual word has a hyphen or not. For example: extract *bench- mark* as *benchmark* and *high- quality* as *high-quality*

We implement the fourth approach because we aim to merge the parts accurately. Our main point of interest was to see if a machine learning model could solve the problem. We experimented with a Conditional Random Field, a Language Model, and simple linear regression. We will describe these in chapter 3. But first, we define the problem formally, and we take a closer look at some of the challenges.

We represent a word as a sequence of characters, ordered by left-to-right reading direction. Thus, the formal task definition of this thesis is defined as follows:

Given a sequence of characters with a line-break hyphen on position i

$$S = [c_1, c_2, \dots, c_{i-1}, -, c_{i+1}, \dots, c_n]$$

the task is to decide if this hyphen should be deleted or kept (1.1)

$$\hat{S} = [c_1, c_2, \dots, c_{i-1}, c_{i+1}, \dots, c_n] \vee \hat{S} = [c_1, c_2, \dots, c_{i-1}, -, c_{i+1}, \dots, c_n]$$

so that \hat{S} is identical to the expected output.

An obvious solution to the problem is a vocabulary-based algorithm, which stores words along with frequencies to indicate how common the word is. The algorithm looks up the parts merged with and without a hyphen, respectively, and chooses the most common word. For example, for the word parts *ele* and *phant*, that is, for the sequence $S = [e, l, e, -, p, h, a, n, t]$, the algorithm looks up *ele-phant* and *elephant*, and chooses the most common spelling. Since *ele-phant* is not a word, the choice is simple: delete the hyphen. For most cases, this method works well. As long as the original word is well known, and there is no other alternative spelling, it should be in the dictionary with the highest frequency.

Unfortunately, this approach will not always work. One obvious problem is that the algorithm only knows a limited amount of words, and it would not recognise misspellings, such as *elhephant*. Furthermore, it usually does not know plural or conjugated words, unless these are explicitly defined. There is a risk that the algorithm will not recognise *elephants* even though it knows the word stem *elephant*. For this reason, it is crucial to obtain a good dictionary so that the baseline approach can work well. In the rest of this thesis, we will refer to the dictionaries as vocabularies. We define a vocabulary to be a specific collection of words. No dictionary describes all words perfectly with frequencies: language is unique, and some people use some spelling styles more than others. Hence, we tested different vocabularies with different frequency scores.

The idea of frequency scores leads us to the next problem, illustrated in figure 1.2. It is a screenshot from a robotics research paper which analyses the stability of walking robots based on their leg-end movements. For the prefix *leg-* and the suffix *end*, the algorithm described above would obviously choose to merge the two parts because *legend* is a more common word than *leg-end*. The result is an odd sentence, not about the forces of the leg-ends, but about “the legend forces” which “can be sensed by the force sensors mounted on leg-ends”.

leg-end sensors, which is presented in the case of leg-end motion planning and control.

Several simulation cases are set up to evaluate the effectiveness of the LSM method. In Section 2, the concept of LSM is introduced and the procedure to obtain the LSM_m is given. Based on a mechanical model shown in Section 3 as well as the gait and computed leg-end forces described in Section 4, simulation results are obtained in Section 5. Four conditions are considered: walking on flat terrain without or with external forces; walking on slope with or without external forces. Conclusions are given in Section 6.

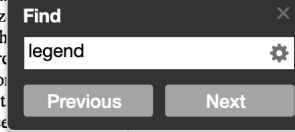
2 Leg-end Supporting Moment (LSM)

2.1 The Concept of LSM

As shown in Fig. 1, during the walk of a n -legged robot, when legs i and u , v are the supporting legs, the moment generated by the leg-ends around the line uv can be expressed as:

$$M_{uv} = \sum_{i \in \{u, v\}} \left\{ \left(\frac{|r_{iu}| |r_{iv}| \sin \alpha}{|r_{iu} - r_{iv}|} \hat{r}_{i,uv} \right) \times F_i \right\} \cdot \hat{r}_{uv}. \quad (3)$$

case (4, 7). The general the sum of the generaliz acceleration forces of th in Eqs. (3-4) are the force Furthermore, the relation the LSM method is t external forces and acc by the forces acting on leg-ends (F_i, F_u, F_v). As mentioned in reference [6], (M, F) can be obtained by using the leg-end forces, which can be sensed by the force sensors mounted on leg-ends. But in the SLM method, the leg-end forces are directly used to compute the stability margin. It can be then concluded that the LSM method is more direct and practical than the TSM method.



2.2 Computation of LSM

Four coordinate frames are defined in Fig. 1 for modeling purposes: the *Terrain Fixed Frame* (TFF) (O_t, X_t, Y_t, Z_t), the *Body-Terrain Frame* (BTF) ($O_{bt}, X_{bt}, Y_{bt}, Z_{bt}$), the *Body Fixed Frame* (BFF) (O_b, X_b, Y_b, Z_b) and the *Hip-Body frame* (HBF) ($O_{ib}, X_{ib}, Y_{ib}, Z_{ib}$). The TFF is fixed on the ground. The orientation of the BTF, which is a point-attached frame, is always the same as the TFF while the motion of the BTF is always translation. The origin of the

Figure 1.2: **An example of challenging hyphenation.** This screenshot from the research paper “A stability analysis of walking robots based on leg-end supporting moments” by Zhou et al. demonstrates challenging hyphenation. The search result marked in blue shows that the word *leg-end* is incorrectly recognised as *legend*.

Important to note about this example is that both words are grammatically correct,

but with different meanings. To know whether to choose leg-ends or legends, one would have to analyse the context. Another possibility is to use the document itself as a dictionary, hoping that the hyphenated word occurs somewhere else. For the leg-end example, this would be a possible solution, since the word *legend* does not appear anywhere else in the running text of the paper. The word *leg-end*, though, appears in the same sentence.

Another difficulty is words which have two correct spellings and the same meaning. For example, *long-standing* or *longstanding*, *north-east* or *northeast* and *e-mail* or *email* can be used interchangeably. Again, one could use the document itself as a dictionary, hoping that it would give clues about the desired spelling. Other than that, it would be an educated guess. To always choose the most frequent spelling would be one strategy. Another strategy is to use Approach 2 or 3 for these words; either always keep or always discard the hyphen. Tendentially, newly made-up words are spelled with a hyphen, such as *e-mail* in the 1970s. As the word becomes widely used, the hyphen disappears: *email*. As Donald Knuth argues, “it’s high time for everybody to stop using the archaic spelling ‘e-mail’”. Think of how many keystrokes you will save in your lifetime if you stop now!” [3]. This suggests removing the hyphens in the words where both spellings are correct would be the better option. However, this is not always the case. For instance, if this quote from Donald Knuth were to be split precisely at the hyphen, the hyphen should remain. If not, the result would have been “the archaic spelling email”, which is not what he wanted to say.

1.1 Overview of the thesis

The goal of this thesis is to use Approach 4 and merge the two-part words correctly. As demonstrated in the previous paragraphs, this is a challenging task. Without a ground truth, that is to say, the original text, it is not always clear, even for humans, if the author meant to put a hyphen in a specific word or not. We explore the problem with a vocabulary-based baseline algorithm and machine learning approaches and do a thorough evaluation of the results.

Chapter 2 discusses different fields of research which are related to the work in this thesis.

Chapter 3 outlines the history of typography and different digital document formats. It describes a computer-based hyphenation routine in detail. Furthermore, it considers the grammar and correct use of hyphens. Last but not least, it outlines the theoretical background of Natural Language Processing, Conditional Random

Fields and LSTM networks.

Chapter 4 presents the data sets used in this thesis, as well as the filtering and cleaning preprocessing steps we did. Furthermore, it describes the modifying steps we did to get a data set suitable for our purposes.

Chapter 5 gives a thorough explanation of our implementations of the vocabulary-based baseline algorithms and of the machine learning model.

Chapter 6 evaluates the results thoroughly by comparing the baseline algorithms, a machine learning approach and a language model.

Chapter 7 presents ideas for future research. It discusses the possibilities of our algorithms and it considers the application of them on other languages.

Chapter 8 concludes the thesis.

2 Related Work

As far as we know, there are not many published methods that attempt to merge two-part words with or without hyphens correctly. In this chapter, we describe two papers that have also used approach 4, however with different methods than ours. Furthermore, we describe what other PDF extraction tools do. Most of the tools do not stress the problem at all, using either approach 1, 2 or 3. However, there are some conversion tools which do address the problem. In addition, there are several works about the reversed problem of this thesis, namely inserting hyphens correctly into words. Though created for other purposes, the field of spelling correction tools is also relevant for our work

2.1 Noisy text correction

Clark [4] focused on the problem of preprocessing noisy text. He proposed to solve the hyphenation problem in a unified tool, based on a noisy channel model, which at the same time corrects other issues in general noisy text. However, the publication does not contain a proper evaluation.

In “A Method for Correcting Broken Hyphenations in Noisy English Text” Micher [5] developed a rule-based algorithm to solve the hyphenation problem. Similar to our baseline approach, Micher’s algorithm uses a long list of valid words to indicate whether a spelling is correct or not. The algorithm takes one of three different actions: if the merged word is known, it merges. If not, it tries to look up parts of the word. If this succeeds, the hyphen stays. If not, the algorithm does nothing.

The algorithm solves one problem which we do not address in this thesis: hanging hyphens. For example: “first- and second-order planning”. Specifically, if the second word is “and”, “or” or a comma, the algorithm does nothing.

Micher tested the algorithm on an English military training text, and achieved 98.19% accuracy for the words with non-expected hyphens and 93.18% accuracy for words with expected hyphens. However, there were only 43 words with expected hyphens in the ground truth. We assume that the high recall value for these occurs

because there were few, or none, words with ambiguous spellings and no words with creative language in the data set.

2.2 Conversion tools

Tiedemann [6] developed the open-source tool pdf2xml ¹, which aims to convert PDF text to a linguistically useful XML format. This tool uses three different PDF rendering software; *Apache Tika*; pdfXtk ², which is built upon pdfbox ³; and tools from *Poppler Developers*, which includes pdftotext ⁴. Through a post-processing procedure, Tiedemann’s tool aims to fix various imperfections and to extract a cleaner text. It creates a vocabulary based on the output from the three conversion tools. Concerning the hyphenated words, the tool merges the two parts if the resulting word is a part of its vocabulary. This tactic is similar to Approach 4 using a dictionary-based algorithm, albeit without a frequency score.

Likewise, the tools which aim to convert PDF files to the ebook file format EPUB faces the same difficulties as in this thesis. The EPUB format is designed to be paper-format independent. It supports custom layouts that are influenced by page orientation and the size and resolution of the mobile device. Specifically, the lines can be longer or shorter, leading to a different hyphenation than in the original PDF document. Therefore, the tools aim to extract the text as cleanly as possible. One example is the open-source converting tool Calibre ⁵. It enables heuristic processing that removes unnecessary hyphens. It uses the document itself as a dictionary and removes the hyphen if the merged version is present somewhere else. If the word occurs only once within the document, nothing is changed, and the hyphen stays. Therefore, this tool would succeed with the leg-end example, demonstrated in Figure 1.2. But in our experiments, we found that if a text was randomly hyphenated, there was a less than 3% chance that the inserted hyphen was on the same position as an original one. Hence, we consider Calibre’s choice of keeping the hyphens in the cases where the word is unknown, as disadvantageous.

¹<https://bitbucket.org/tiedemann/pdf2xml/src>

²<https://github.com/tamirhassan/pdfxktk>

³<https://pdfbox.apache.org/>

⁴<https://www.xpdfreader.com/pdftotext-man.html>

⁵<https://calibre-ebook.com/>

2.3 Hyphenation tools

There are several papers about the reversed problem in this thesis, that is, to find correct hyphenation breakpoints in any word. We were interested to see if these methods could be used to find out if a hyphen was suitable in a word.

Specifically, one idea was to see if we could look at the word at a character sequence level to determine whether a hyphenation was valid or not. For example, a hyphen likely follows prefixes like *high*, *pre*, *post*, and similar. These prefixes have specific properties, such as combinations of syllables. And such properties of character sequences were examined in these works.

Trogkanis and Elkan [7] used Conditional Random Fields (CRF) to decide possible hyphenation points in a word, outperforming the current state-of-the-art systems. They report a sixfold improvement compared to the systems currently in use. Németh et al. [8] used different deep learning-based methods to hyphenate Hungarian words and compares three neural network models: Feedforward Neural Network (FFNN), Convolutional Neural Network (CNN) and Long Short-Term Memory Network (LSTM). All models achieve more than 95% accuracy, with CNN being the best model.

2.4 Spelling correction tools

Finally, the topic of this thesis can also be considered a spelling correction problem. Assuming that we always keep the hyphen, the problem is to remove misplaced hyphens in a sentence. That is to say, fixing a possibly misspelt sentence.

Taghva and Stofsky [9] proposed a spelling correction system specifically made for correcting optical character recognition (OCR) mistakes. The system is interactive. This means it has, for example, a second mode for texts with many hyphenations. The evaluation of the tool is on a word- and character basis, but not specifically on hyphens.

Furthermore, many grammar check tools which are on the market today would recognise words with invalid hyphenations. But they would not always differentiate between common and rare spellings. Nonetheless, when we tested the Grammarly [10] spell checker manually, the results were surprisingly good. That is, the tool could correct most false hyphenation. However, it is not open-source, and therefore not applicable to our work. The system uses machine learning techniques, but it is not specified how.

3 Background

This chapter outlines the history of typography and the emergence of different electronic document formats. We show the challenge of implementing typographical routines with computers. Specifically, we explain how Frank Liang’s computer-based algorithm is used to hyphenate words in electronic documents. Moreover, we describe the grammar and correct use of hyphens in the English language. Last but not least, we describe the tasks of Natural Language Processing, and how Conditional Random Fields and LSTM networks can be used to solve these tasks.

3.1 Typography

The main goal of typography is to make written language readable and visually appealing. Although the topic has been relevant ever since the invention of the written language, it emerged as a craft with Gutenberg’s printing works in the middle of the 15th century. Gutenberg’s printing press made it possible to mass-produce manuscripts. The art of setting the text in written material became a new craft: typography.

During the mid-1980s, computers replaced the traditional typography craft. Desktop publishing, the creation of documents using a personal computer, increased dramatically. The typography skills were no longer needed in the print shops but in offices everywhere. At the same time, the designers started to typeset documents by themselves, using different software applications. However, they did not have the typography skills to set the texts properly. Therefore, this led to a phase of insensitivity in respect of character construction and spacing [11]. On the other hand, the digital shift made it drastically cheaper and faster to make new fonts and layouts. It allowed the typographers to create more experimental designs.

Typography has three fundamental aspects. Legibility points out how easy it is to distinguish two characters. The characters should be similar enough to look uniform but at the same time different enough to be easily separated. Aesthetics refers to the general layout of the page, aiming to make it look visually appealing. Readability

considers how easy it is to read the text as a whole. The latter aspect is the most relevant for this thesis. Readability is affected by the use of margins, line length, line spacing and the design of the right-hand edge.

There are different typographical techniques to make the right-hand edge look beautiful. A beautiful and readable edge is not too ragged, that is to say, it is even. For a left-aligned text, hyphenation can help to achieve this. Justified text has lines with spacing between words and characters so that both the left and the right edge are perfectly even. Here, hyphenation is quite necessary. If hyphenation is not used, the spaces can be extremely large.

Figure 3.1 shows the difference between hyphenated and unhyphenated justified text. There is no doubt that hyphenation improves readability. If hyphenation is disabled, there are fewer points where you can break the sentence. This makes inconvenient line breaks more likely.

HYPHENATED

Section 513 extends the time
in which to run away if the ap-
plicant was outside Califor-
nia when the kitten appeared
or leaves the state after it ap-
peared. It reads: “If, when the
cute kitten appears beside a
person, he is out of the State,
he may run away at the earliest

UNHYPHENATED

Section 513 extends the time
in which to run away if the
applicant was outside
California when the kitten
appeared or leaves the state
after it appeared. It reads:
“If, when the cute kitten
appears beside a person, he is
out of the State, he may run

Figure 3.1: **An example of justified paragraphs.** This example shows how hyphenation affects the look of justified paragraphs, as demonstrated in M. Butterick’s online book “Practical Typography”. [12]

3.2 Electronic documents

The craft of typography shifted to digital formats in the space of about ten years [11]. In this time, there were different approaches on how to make layout-based documents look typographically satisfactory. But none of the standards succeeded in being compatible with all kinds of systems. The PDF format focused on keeping a fixed layout as it is, disregarding the need for extracting the raw text. E-book file-formats

preserves the raw text as it is, but does not support fixed-layout typography. In this section, we describe the two file formats and the ideal one.

3.2.1 PDF

Portable Document Format (PDF) is an open-source file format created by Adobe in 1993, based on the PostScript language. Each PDF file describes a fixed-layout document, including text, fonts, vector graphics, raster images, and other information that is required to visualise it. Since 2008 it is an ISO 32000 standard [13].

The newest version of PDF supports metadata, annotations and even three-dimensional content. However, text extraction remains a challenge. As explained in the introduction, the text is only available in letters. Usually, it is therefore not clear which letters belong to which word, and there are no spaces. To assemble the words you have to, for example, analyse the distances between the letters.

3.2.2 EPUB

The electronic publication format EPUB is an open file format standardized by the International Digital Publishing Forum (IDPF) in 2007 [14]. EPUB is an archive file format. It contains XML- and XHTML-based files which specifies the content, image files (if any) and files which specify metadata. The final document does not have a fixed layout. It adjusts according to the (usually small) screen and the resolution of the e-book reading device. This feature, called reflowable text, is necessary for a satisfying reading experience. With a PDF document, you would have to zoom and scroll all the time.

3.2.3 The ideal electronic document

Today, the ideal digital document somehow preserves the original text as it is. Unfortunately, PDF documents were designed to be printed out in their physical form. But nowadays, people mostly read research papers on a computer. The times have changed, and the way we read texts has changed. The best method to ensure that people can use search- and copy-functions with the computer, as usual, is to adapt the file format.

The newest PDF versions currently do support extensive metadata notes, including page content annotations. At least with Adobe's Acrobat Pro DC ¹. It is indeed possible to tag headers, figures, paragraphs and references as what they are. It is

¹<https://acrobat.adobe.com/us/en/acrobat/acrobat-pro.html>

also possible to denote the reading order. In this way, the document is accessible for people with blindness, low vision or other disabilities. Personal computers make it possible to read digital texts differently.

Ideally, the electronic document would have annotations in such a way that it is digitally accessible and useful for the way we consume digital text today. However, these annotations are usually not used. There are several reasons for that. It means additional work for the creator of the document to create these annotations. Of course it is possible to automate this, but then the problem is how to calculate these annotations reliably. Even if the PDF was created from TeX, it is not so easy.

3.3 Liang’s hyphenation algorithm

Figure 3.1 shows the importance of hyphenation in justified paragraphs. However, for a computer, it is not clear how a word should be hyphenated. It does not understand grammar or phonetics. Take the word *misleading* as an example. Without routines, a computer could hyphenate the word at any place: *mi-sleading*, *misl-eading*, *misle-ading*. It would ignore whether the hyphenation is readable or not. Therefore, it was necessary to implement hyphenation routines.

Frank Liang [15] designed the hyphenation routine which is used in TeX today. The computer-based algorithm uses phonetic patterns to suggest where a word can be broken in two whilst still being easily readable. To be precise, the patterns are a set of short sequences of letters and digits. For example, `mis5l`. Liang created this set during the work on his thesis “Word Hy-phen-a-tion by Com-put-er”. He distinguished between two types of phonetic patterns: inhibition- and hyphenation. The inhibition patterns are sequences of letters where hyphenation is inappropriate. Furthermore, he gave the patterns five levels of increasing strength. Hence, they have digits in the range one to five. The odd numbers indicate allowable hyphen points, the even the opposite.

For example, the pattern `mis5l` suggests that a word which has the letter sequence *misl* should be hyphenated between the *s* and the *l*. For the word *misleading*, this would force the hyphenation *mis-leading*, which is better than the examples given above.

We demonstrate Liang’s algorithm in action on the word *hyphenate* in Figure 3.2. First, the algorithm pads the word with two punctuation marks, symbolising the beginning and the end of the word. Then, it finds all patterns from Liang’s set, which match parts of the word. In this example, there are seven matching patterns.

We have listed them below the word in arbitrary order. Notice that some patterns overlap, for example, **4te.** and **n2at.** Also, notice that **4te.** includes a punctuation mark. This pattern only applies to the very end of a word. The algorithm chooses the highest number at each intercharacter position. Since hyphens are only allowed at odd-valued positions, the resulting hyphenation is *hy-phen-ate*.

.	h	y	p	h	e	n	a	t	e	.
				h	e	2	n			
				h	e	n	a	4		
				h	e	n	5	a	t	
	h	y	3	p	h					
						1	n	a		
						n	2	a	t	
									4	t e .
.	h	y	3	p	h	e	2	n	5	a 4 t e .
	h	y	-	p	h	e	n	-	a	t e

Figure 3.2: **A demonstration of Liang’s hyphenation algorithm** This illustration, inspired by Németh[16], shows Liang’s algorithm on the word *hyphenate*.

Today, the quality of the phonetic patterns has improved. There are more patterns and exceptions, some of them having level six. Besides, there are custom patterns for different languages. Nevertheless, the basic algorithm remains the same.

3.4 Hy-phens

The word “hyphen” origins from the Greek “huphen”, which means “together”. A hyphen is a punctuation mark which is used to point out that two words have a combined meaning. As seen in the previous paragraphs, a hyphen can be used to indicate a word division between two lines. It can also be used to indicate that a word has repetitive elements which belong together, for example *short- and long-term*. This section describes the use of hyphens in compound words and the difference between American English and British English concerning word division.

But first, we will describe what is not a hyphen: a dash. A dash looks like a hyphen, but it is wider. The en-dash (–) is about the length of an upper-case N; the em-dash (—) is roughly as long as an M. The en-dash is often mistaken to be a hyphen. It mostly occurs in formal publications, where it serves as an even stronger connection between compound words. The em-dash has an entirely different function. It indicates a longer pause in a sentence. This pause is stronger than a comma but

weaker than a period or semicolon [17]. The em-dash is not interchangeable with a hyphen. We will come back to this when we describe the data preprocessing steps in section 4.2.2.

Going back to the hyphens: perhaps their most important function is to form compound words, such as *high-quality* or *brother-in-law*. When combining a noun or an adjective with a present participle (a word ending in *-ing*), the hyphen makes it clear that the two words are one unit of meaning. For example, in the sentence “we were told a story about a man eating shark”, it is not clear whether a man is eating a shark or if a shark is eating a man. An unambiguous version could be “we were told a story about a man-eating shark”.

Another function of the hyphen is to split a word between two lines. It is not clear how to do this correctly. There are even different guidelines for British English and American English. The prior tends to focus on the origin of a word, the latter on the sounds [18]. But, even when emphasising the sounds, British and American pronunciations can be different. Take the word *vitamin* as an example. In British English, the first syllable *vit* rhymes with *fit*, and the hyphenation is *vit-amin*. In American English, on the other hand, the first syllable sounds more like *wide*, and the hyphenation is *vi-ta-min*.

Most importantly, the hyphenation should not be misleading. For example *readjust*, which means to adjust something again, should be hyphenated as *re-adjust*, not as *read-just*. To understand this, take a look at figure 3.3 and try to read the words out loud.

Most importantly, the hyphenation should not be misleading. For example *read-just*, which means to adjust something again, should be hyphenated as *re-adjust*, not

Figure 3.3: **An example of a hyphenation flaw** This example shows our bad luck when setting the preceding paragraph with the L^AT_EXhyphenation routine

It is indeed simpler to read *read* as the first syllable. But it is misleading because the correct pronunciation of *readjust* has two syllables in the beginning: *re* and *ad*, with a small break in between. The reader is likely to miss this break because *read* is a well-known word by itself. Section 3.3 demonstrated a computer-based hyphenation algorithm which focused on the sounds of a word. This method works well for most words, but not for words like *readjust*, because the algorithm would not understand that the syllable pattern, which is harder to pronounce, is indeed the correct one. The algorithmic approach does not supersede common sense and a

human understanding of the language.

Nevertheless, there is a dream of “understanding” human language using a computer. The next sections will describe various computer-based approaches which seek to accomplish this.

3.5 Natural Language Processing

Statistical Natural Language Processing (NLP) is concerned with the automatic processing of the natural languages of humans. It applies probabilistic methods, information theory, and linear algebra. The history of NLP is exciting, but out of the scope of this thesis. We will limit ourselves to two concepts: Part-of-Speech Tagging and Named Entity Recognition. We will use these to explain how a Hidden Markov Model, a Maximum Entropy Markov Model, and a Conditional Random Field, works. But firstly, we outline some of the challenges in this field of science.

Automatic processing of natural languages is considered to be a difficult problem. The reason for this is that the human language has idioms, abstract, sarcastic or ambiguous sentences. It is difficult to describe this to a computer. Consider the phrase *out of sight, out of mind*. It does not mean *invisible, insane*. For a human, this is obvious, but for a computer, it remains a challenge to understand such concepts. “For this reason, much research in NLP has focused on intermediate tasks that make sense of some of the structure inherent in language without requiring complete understanding” (Manning and Schütze, p. 341) [19]. One of these intermediate tasks is part-of-speech tagging: to assign grammatical labels to words in a sentence. That is, to decide if a word is a verb, a noun, a determiner, or something else. An example of a POS-tagged version of the sentence “I can understand this” is:

I/PRP can/MD understand/VB this/DT ./PUNCT.

Each word has a grammatical tag. The first word, *I*, is tagged as PRP (personal pronoun). *Can* is tagged as MD (a modal verb), and *understand* is a regular verb VB. *This* is a determiner DT. Last but not least, there is a PUNCT (punctuation). The tags can be more or less specific. For example, one can distinguish between NN (a noun) and NNP (a proper noun). The goal of Named Entity Recognition (NER) is to identify proper nouns in a sentence. That is to say, to find out if a proper noun is a real-world object, for example, an organization, a person, a place, or something else.

The challenge of POS-tagging is that a word might have a different tag in a different context. Consider the classic example “time flies like an arrow”. You can

interpret *time* and *flies* as a noun and a verb, suggesting that time passes very quickly. However, *flies* is also a noun in plural, and *like* is a verb. A computer could just as well assume that the sentence is about a unique type of flies, time flies, which happen to like arrows.

3.5.1 Motivation

At this point, please notice the relation between POS-tagging and the hyphenation challenge proposed in this thesis. Imagine a sentence with one hyphenated word. We wanted to find out if the grammatical structure of the sentence could indicate whether or not this hyphen is correct. For example, consider the word *high-quality*, which is written with a hyphen because it is a compound word. Since we know that compound words usually come before a noun in a sentence, we wanted to see if this could help us solve the problem. In other words: if a word is hyphenated, and it is most likely a compound word, then the hyphen should probably stay.

We can already reveal that we changed the approach throughout the thesis, not looking at sentences any more, but at single words or sequences of character. However, we will now describe how a sequential model works at the word level. To some extent, we will focus on the words to explain why we changed our approach. But also, we keep it on this level because it makes it less abstract, and more intuitive to understand how the model works.

So the question is, how do you compute the most likely sequence of POS-Tags? A solution which solves the problem with high accuracy is to use a Hidden Markov Model and the Viterbi Algorithm.

3.5.2 Hidden Markov Model

A Hidden Markov Model (HMM) is a sequence model. In other words, it is a classifier which assigns a label or class to each unit in a sequence. For example, it can assign a POS-tag to each word in a sentence. It is a generative approach; it learns a probability distribution over the data and uses this to deduce a label or a class. In this case, the model first estimates the probability of a word given a POS-tag. Later, the model uses these probabilities to predict a POS-tag when given a word.

The basis of an HMM is a Markov chain. For a set of states, which can be words, labels, or anything else, a Markov chain computes the probability of a specific sequence of states. It makes one strong assumption, called the Markov property: it assumes that the current state only depends on the previous one. Or, if you see it

the other way, it assumes that the future can be predicted from the current state only, ignoring the history before.

Formally, the Markov assumption is defined as follows: consider a sequence of state variables $q_1, q_2 \dots q_i$. The probability of state q_i only depends on state q_{i-1} .

$$P(q_i | q_1 \dots q_{i-1}) = P(q_i | q_{i-1}) \quad (3.1)$$

A Markov chain is made up of the following components:

$Q = q_1, q_2 \dots q_n$	a set of n states
$A = a_{11} a_{12} \dots a_{n1} \dots a_{nn}$	a transition probability matrix A , each a_{ij} representing the probability of moving from state i to state j . Each row sum to one: $\sum_{j=1}^n a_{ij} = 1 \ \forall i$
$\pi = \pi_1, \pi_2 \dots \pi_n$	an initial probability distribution over the states. π_i is the probability that the Markov chain will start in state i . Some j may have $\pi_j = 0$, meaning that they cannot be initial states. The initial probabilities sum up to one: $\sum_{i=1}^n \pi_i = 1$

as defined by Jurafsky and Martin[20].

The Markov chain computes the probability of an observable sequence. In many cases, however, the events in which we are interested are hidden: we can not observe them directly. For example, we can not usually see the POS tags in a sentence, only the words. An HMM allows us to consider both observable and hidden events. It consists of the Markov chain components and the following extensions:

Q, A and π	the Markov chain components
$O = o_1 o_2 \dots o_t$	a sequence of t observations , each one drawn from a vocabulary $V = v_1, v_2 \dots v_v$
$B = b_i(o_t)$	a sequence of observation likelihoods , expressing the probability of an observation o_t being generated from a state q_i

Two assumptions simplify the model. The Markov property 3.1, and also the assumption that the probability of an output observation only depends on the state which produced this observation.

$$p(o_i|q_1...q_i...q_t, o_1...o_i...o_t) = P(o_i|q_i) \quad (3.2)$$

as defined by Jurafsky and Martin[20].

We will not explain in detail how to learn the probability distributions over the data. However, we will outline the procedure of deducing POS-tags in a sentence, given the set of probability distributions.

3.5.3 Deducing POS-tags with the Viterbi algorithm

Assume that we have a trained HMM and an observed sentence. To compute the most likely sequence of POS-tags is to find the series of tags maximizing the probability of seeing the sentence. More formally, given a sequence of observations, the task is to find the most likely order of hidden states.

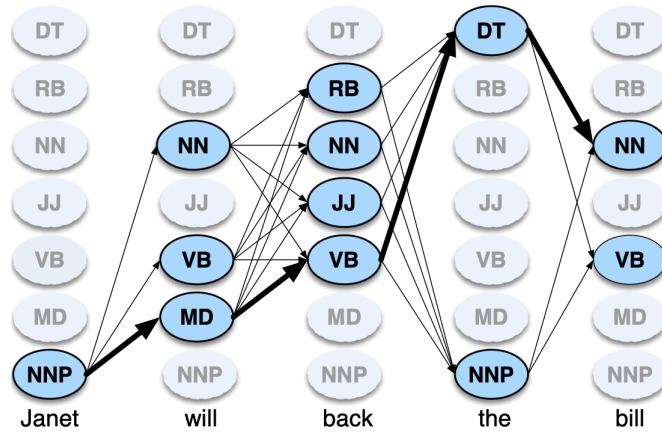


Figure 3.4: **A sketch of how the Viterbi algorithm works** This sketch, made of Jurafsky and Martin[20], shows the procedure of the Viterbi algorithm on the sentence “Janet will back the bill”. The edges are thicker for the most likely sequence of hidden states.

A dynamic programming algorithm which solves this is the Viterbi algorithm. It is similar to the dynamic minimum edit distance algorithm. Consider figure 3.4. Each cell represents the probability of a specific hidden state after seeing the first t observations and passing through the most likely sequence q_1, \dots, q_{t-1} . The algorithm fills out the cells recursively. For each step, that is, for each word, it only considers the most likely path which leads to a particular state.

Notice that some cells are grey because the observation likelihood is zero. For

example “Janet” is always a proper noun, never a verb or an adjective. From this, it is easy to observe a problem of HMMs: unknown words. Imagine if the word “Janet” never occurred during the training. All the cells for this word would be grey, and the probability of the whole sequence would be zero. That is to say, all of the observation probabilities for this word in the model would be zero. Without a proper probability distribution, the generative model does not know which label to predict. This can be solved using smoothing or with other extensions to the model. Or, by using another approach, such as a maximum entropy Markov model or a conditional random field.

3.5.4 Maximum Entropy Markov Model

The idea behind the maximum entropy Markov model (MEMM) is to add additional features to each word, that is, to combine logistic regression and HMMs. As we know, HMM use a generative approach. However, logistic regression is a discriminative approach: it learns a decision boundary between classes with which it can later make predictions.

Notice the contrast between the two strategies: an HMM learns a probability distribution over words and labels. Logistic regression learns how to estimate a tag directly, depending on the properties of the word itself.

However, one can classify the words consecutively, using a feature which tells the label for the previous word. A MEMM applies logistic regression in this way. This model can deal with unknown words because it incorporates many different features; the word shape, if it contains a specific prefix, digits, hyphens, and more.

3.5.5 Bidirectionality

One problem with HMMs and MEMMs is that they only run left-to-right. It can be useful to know what will happen in the future, that is to say, to use the information about the next tag. One solution is to run the MEMM twice: once left-to-right, and once right-to-left, and then choose the prediction with the highest scoring. This idea is taken further with the bi-LSTM model, which we will outline in section 3.7.

Furthermore, MEMMs have a so-called label bias problem [21]. It is a type of unwanted explaining-away effect. As opposed to the HMMs, the MEMMs cannot downgrade a sequence of hidden states based on something that will happen in the future, for example, based on a word in the very end of a sentence. Following the Markov property, the classifier merely decides one class for each time step (or one tag for each word). It does not distinguish between different probability scores but

makes a discriminative decision. In other words, one sequence is explained away because the path of influences is severed [22].

The problem is that the probabilities are locally normalized, and therefore, the MEMM prefers states with fewer transitions. Conditional random fields solve the label bias problem by globally normalizing the model instead.

To sum up, we compare the definitions of the HMM and the MEMM, before we will explain how the conditional random fields work. Let \mathbf{w} be the sequence of words and \mathbf{t} the sequence of tags. The goal is to compute the best tag sequence $\hat{\mathbf{t}}$ that maximises $P(\mathbf{t}|\mathbf{w})$. An HMM uses Bayes' rule and $P(\mathbf{w}|\mathbf{t})$, the likelihood of a word given a tag [20]:

$$\begin{aligned}
\hat{\mathbf{t}} &= \arg \max_{\mathbf{t}} P(\mathbf{t}|\mathbf{w}) \\
&= \arg \max_{\mathbf{t}} P(\mathbf{w}|\mathbf{t})P(\mathbf{t}) \\
&= \arg \max_{\mathbf{t}} \prod_i P(word_i|tag_i) \prod_i P(tag_i|tag_{i-1}) \\
&= \arg \max_{\mathbf{t}} \prod_i P(word_i|tag_i)(tag_i|tag_{i-1})
\end{aligned} \tag{3.3}$$

A MEMM computes the posterior $P(\mathbf{t}|\mathbf{w})$ directly. It considers a set of features within l neighbouring words, and it takes the previous k tags into account:

$$\begin{aligned}
\hat{\mathbf{t}} &= \arg \max_{\mathbf{t}} P(\mathbf{t}|\mathbf{w}) \\
&= \arg \max_{\mathbf{t}} \prod_i P(tag_i|t_{i-k}^{i-1}, w_{i-l}^{i+l}) \\
&= \arg \max_{\mathbf{t}} \prod_i \frac{\exp(\sum_j \theta_j \cdot f_j(t_i, t_{i-k}^{i-1}, w_{i-l}^{i+l}))}{\sum_{t' \in \text{tagset}} \exp(\sum_j \theta_j \cdot f_j(t', t_{i-k}^{i-1}, w_{i-l}^{i+l}))} \\
&= \arg \max_{\mathbf{t}} \prod_i \frac{1}{Z_i(t_{i-1}, w_{i-l}^{i+l})} \cdot \exp(\sum_j \theta_j \cdot f_j(t_i, t_{i-k}^{i-1}, w_{i-l}^{i+l}))
\end{aligned} \tag{3.4}$$

Notice that $Z_i(t_{i-1}, w_{i-l}^{i+l})$ is the normalisation factor which forces the probabilities of a particular preceding tag to sum to one.

By having a theoretical overview of the HMMs and MEMMs, we are now ready to discuss the conditional random fields.

3.6 Conditional Random Field

A conditional random field is an undirected graphical model. It was introduced by Lafferty et al. [21] as an improvement to the MEMM model. If the graph is a single line, it is called a linear-chain CRF. Instead of computing one tag for each time step, that is, for each word, the model computes log-linear functions over a clique, a set of relevant features [20]. These features might include information about words in future steps. Consider figure 3.5, which compares the two sequence models we have discussed until now with the linear-chain CRF.

The figure shows how the most probable tag sequence for the sentence “I like dogs” is generated. The HMM computes the likelihood of each word conditioned on the hidden states, whereas the MEMM computes the most likely subsequent tag, given a label and a word. The CRF computes the most likely sequence of tags using log-linear functions.

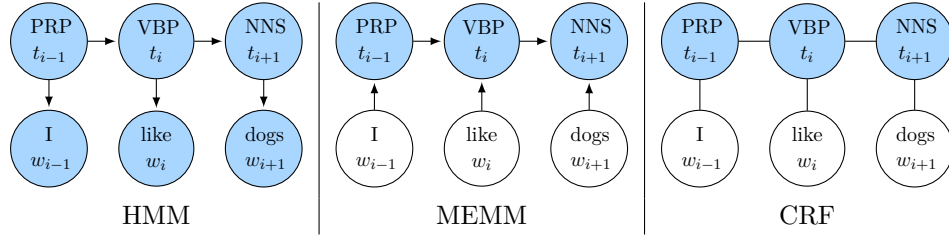


Figure 3.5: **A graphical representation of the sequence models** This figure shows the difference between HMM, MEMM and linear-chain CRF. The first two models are Bayesian networks, with arrows indicating conditional dependencies; the third is an undirected graphical model. A filled circle indicates a variable generated by the model.

Since CRFs are globally normalised, in contrast to the locally normalised MEMMs, they avoid the label bias problem. To understand the reason why, one can consider the CRF as a probabilistic automaton with unnormalised transition probabilities [23]. Each transition can magnify or reduce the total probability mass of a sequence. Due to the global normalisation factor, the weights of the final sequences are still a valid probability distribution.

We use the notation of Hanna Wallach [24] to describe the CRFs formally. The probability of a label sequence \mathbf{t} and a sequence of observations \mathbf{w} can be written as

$$p(\mathbf{t}|\mathbf{w}\lambda) = \frac{1}{Z(\mathbf{w})} \exp\left(\sum_j \theta_j F_j(\mathbf{t}, \mathbf{w})\right) \quad (3.5)$$

where $Z(\mathbf{x})$ is a normalisation factor. F is a global feature vector which maps the entire sequence:

$$F_j(\mathbf{t}, \mathbf{w}) = \sum_{i=1}^n f_j(t_{i-1}, t_i, \mathbf{w}, i) \quad (3.6)$$

Each function $f_j(t_{i-1}, t_i, \mathbf{w}, i)$ is a set of real-valued features $b(\mathbf{w}, i)$ which expresses characteristics of the data. For example:

$$b(\mathbf{w}, i) = \begin{cases} 1, & \text{if the word in position } i \text{ is uppercase} \\ 0, & \text{otherwise} \end{cases} \quad (3.7)$$

There is no limit of feature functions. Therefore, the predictions can be based on properties anywhere in the sentence, which is why, for example, the Stanford Named Entity Recogniser by Finkel et Al. [25] is CRF-based.

Just like the MEMM, the CRF is a combination of logistic regression and a sequential model. In other words, if the CRF is a chain with only one label, it is merely logistic regression. When we tested the linear-chain CRF, we quickly found out that the properties of a single word were sufficient enough to predict a label with high accuracy. To be precise: we ran the model on one-word sequences, which is plain logistic regression. Since this gave surprisingly good results (discussed in chapter 6), we did not use the model further but changed our strategy.

We therefore adapted our approach, looking at sequences of characters instead. We tested a character-based bidirectional LSTM language model to see if this approach would give similar results.

3.7 Bidirectional LSTM

We are grateful that we could use Matthias Hertel’s bidirectional LSTM [26] to explore the possibilities of character-based neural language models for the problem stated in this thesis.

Before we explain how his model works, we give a general outline of bidirectional LSTMs (bi-LSTMs). To be consistent with the examples in the preceding sections, we will still consider sequences of words. To be clear, this is different from Hertel’s model, which is on the character level. We use the word example because it is less abstract and more intuitive. Consider Figure 3.6. On the left side, there is a regular unidirectional Long Short-Term-Memory (LSTM) model. The new elements here are the grey boxes: the LSTM memory cells. These cells are designed to store information;

that is, it can keep its hidden state, and it can learn long-term dependencies. To learn long-term dependencies is the same as learning how to connect information, although the gap between the information which is needed, and the information which is relevant to predict it, is large.

To use Christopher Olah’s [27] example, consider the problem of predicting the last word in the sentence “I grew up in Norway... I speak fluent *Norwegian*”. The last part suggests that the last word should be the name of a language. But to know which one, you need to remember the information in the beginning.

A unidirectional LSTM network has one layer of cells. A bidirectional LSTM network has two layers, one forward, and one backward layer. Notice the dependencies (arrows) between these layers in Figure 3.7. For each element of the (finite) input sequence, the bidirectional model can access more information. That is, it can access information about the future as well as the past. Interestingly, this is comparable to the advantage CRFs have over MEMMs [28].

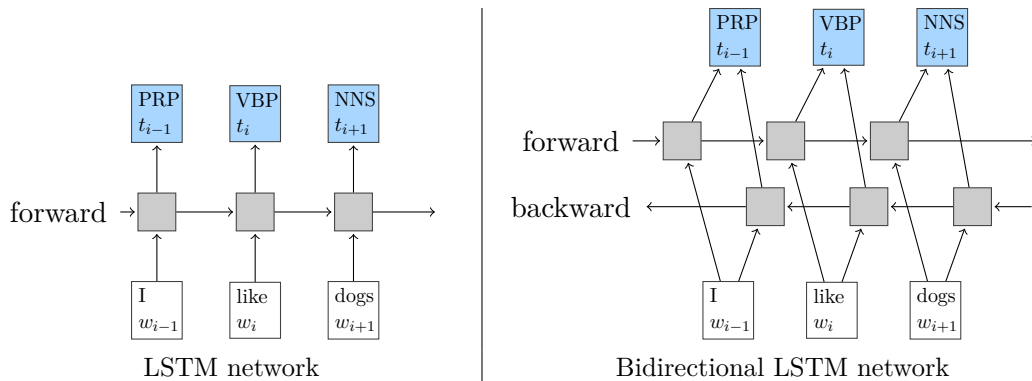


Figure 3.6: **LSTM Networks.** This example, inspired by Huang et Al. [28], shows a unidirectional and a bidirectional LSTM. Each grey box represents a LSTM memory cell. These cells can make use of long-range dependencies in the data.

Hertel used (among other models) a bidirectional LSTM to perform tokenization repair, a special case of spelling correction. For each character at each position in a sequence, the model predicts the likelihood of all other characters. In this thesis, we are solving a similar task (see section 1.1). For a sequence of characters, we want to determine if a hyphen at position i is correct or wrong. That is, we want to find out how likely the character is.

Hertel’s bidirectional LSTM works on a character level. More specifically, it considers a set of the hundred most common English letters, as well as three special characters: one for the beginning of a sentence, one for the end, and one for the

non-set characters. The training, validation and test data were from the English part of the Europarl German-English Parallel Corpus and an English Wikipedia dump. The prior was 287MB in raw text size, the latter 12.7GB.

The model learns a probability distribution over all characters in all sequences (in the training data). The output layer is sigmoidal so that the model also can predict that “no character is likely”. There are 103 output neurons, one for each element in the character set. In chapter 5, we will describe how we use the likelihoods of these 103 characters to solve our task. The results are evaluated in chapter 6.

4 Approach

We approached the problem by designing a data set of sentences with hyphenated words. We used Frank Liang’s algorithm, described in Section 3.3, to obtain a realistic hyphenation. In this way, we could flexibly hyphenate several words in the same sentence. And, most importantly, we had the original text available, which served as ground truth for the evaluation. This chapter describes ClueWeb12 and Ontonotes, the two data sets of sentences used in this work, and the necessary preprocessing steps. Finally, it describes in detail how we created and structured the final data set.

4.1 Data sets

We used two data sets which consists of sentences in natural language: ClueWeb12 and Ontonotes 5.0.

4.1.1 ClueWeb12

The ClueWeb12 dataset consists of 733,019,372 English web pages. It was created by researchers at Carnegie Mellon University, through a web crawl done between February 10, 2012 and May 10, 2012.

We used a smaller version of the data set, which was already parsed by our group. This version contained Named Entity Recognition (NER) tags (see section 3.5), intended for other purposes. We removed these tags in the cleaning step of the data preprocessing. However, there were some sentences with broken NER tags, lacking the closing brackets. We removed these sentences from the dataset.

This resulted in a dataset of 389,718,096 sentences with a total of 10,794,281,624 words before preprocessing. The dataset consists of all sorts of sentences which have been posted online. Therefore, the dataset includes misspellings and improper formatting. And although the web pages which were crawled are English, the sentences contain quotes, names and translations in other languages. Nevertheless, this is a huge and resourceful database of natural language. Figure 4.1 shows a small sample from this version of ClueWeb12 before we started working with it:

I 'm travelling to [m.0345h|Germany] for vacation and would like to know if there 's any way that I can take the river instead of the road or train from [m.02z0j|Frankfurt] to Freiburg .

Afterwards the [m.09zyn5|Norwegian] Prime Minister , Jens Stoltenburg , made a public announcement that the baby would be called [m.095p0v|Sverre Magnus] , pronounced ' Svair-ah Mahg-noos ' .

Another way to say HAPPY [m.018y5m|VALENTINE 'S DAY] in [m.06bnz|Russian] is ПОЗДРАВЛЯЮ С ДНЁМ СВЯТОГО ВАЛЕНТИНА pronounced as PAH-ZDRAH-VLYA-YOU S DNYOM SVYA-TON-VAH VAH-LYEN-TEE-NAH .

Figure 4.1: **Sample of ClueWeb12 sentences** This sample shows three sentences from our version of ClueWeb12, before the data preprocessing steps.

There were certainly several preprocessing steps needed before we could use the dataset for our purposes. We will describe these in Section 4.2.

4.1.2 Ontonotes

Ontonotes Release 5.0 consists of sentences in English, Chinese and Arabic, manually annotated by a collaborative team from BBN Technologies, Brandeis University, the University of Colorado, the University of Pennsylvania and the University of Southern California's Information Sciences. The corpus consists of text from different genres: news, broadcasts, talk shows, conversational telephone speech, among others. For each sentence and each source, there is an extensive amount of grammatical and structural information, as well as other annotation layers. The English corpus consists of 1,745 texts.

Unfortunately, the data set has a format which does not fit our purposes. Therefore, we did several steps to reformat it. In the following paragraph, we will describe these steps.

The data set includes five annotation layers for each text. Specifically, these layers are distinct files in a database. They contain different types of semantic annotation: named entities, coreference, word sense, treebank- and proposition bank annotations. We will not explain what the different layers are (see [29]). The point is that this was not useful for us because we only needed sentences. Although it indeed exists a summary file of each text, these are very inconvenient to parse. Sameer Pradhan et Al. [30], a part of the team which produced Ontonotes, proposed a script for joining the different annotation layers into a single CoNLL 2012-formatted file. Although this format was more what we needed, it was not ideal. Therefore, we used a second

script ¹ to convert the CoNLL 2012-data into a BIO tagging scheme. These BIO formatted files were the ones we used in the further preprocessing.

We extracted the raw sentences, ignoring most of the annotation layers, but keeping the Part-Of-Speech Tags (see section 3.5). This resulted in a data set of 130,540 sentences with a total of 2,262,674 words before preprocessing. We separated the strings and the corresponding tags by a tab character. Figure 4.2 shows a sample of four sentences from this version of Ontonotes before we cleaned the sentences, and before we replaced the bracket tags with “(” or “)”.

```
oh thank you . UH VBP PRP .
Agree ahead of time to NOT smash the cake into each other 's faces .
VB RB IN NN TO RB VB DT NN IN DT JJ POS NNS .
It 's a post-Pop world , `` ism '' - free and with no end in sight .
PRP VBZ DT JJ NN , `` NN '' HYPH JJ CC IN DT NN IN NN .
-LRB- anyway . -RRB- -LRB- UH . -RRB-
```

Figure 4.2: **Sample of Ontonotes 5.0 sentences with corresponding POS-tags** This sample shows four sentences from our version of Ontonotes, before the data preprocessing steps. The special POS-tags -LRB- and -RRB- symbolise left- and right round brackets.

This corpus does not have the same amount of misspellings and foreign characters as in ClueWeb12, because the sentences were annotated and transcribed by linguists. Still, further preprocessing was needed, mainly due to the extensive annotations of brackets and hyphens, which were undesired for this work. Notably, the HYPH tag sometimes symbolises a hyphen, but not always. The third sentence in Figure 4.2 shows a difference between the tags for hyphenated words. *post-Pop* is tagged as JJ (adjective), and “*ism*”-*free* as NN HYPH JJ (noun, hyphen and adjective).

4.2 Data preprocessing

We filtered the data sets in two steps, removing specific sentences and characters. Then, we cleaned the remaining sentences for unusual words and symbols which does not represent natural language.

¹<https://github.com/yuchenlin/OntoNotes-5.0-NER-BIO>

4.2.1 Filtering

Although the datasets are English, they had foreign words and characters. Consider figure 4.1. The last two sentences include both non-words, namely descriptions of pronunciation, and, for the last one, even Russian characters. Another mixture of languages occurs when poems, names of persons and places, and similar, occur in their original language. We focused our work and algorithms on the English language only. Therefore, we attempted to remove as many foreign-character sentences as possible.

- We filtered out all sentences with characters in the Unicode sections U+0370 (Greek and Coptic) until U+1FFF (Greek Extended); which equals most of the non-Latin languages. Note that non-English languages with Latin alphabet were not filtered out. These languages are more difficult to filter because the character sets are the same. Therefore, the second sentence in figure 4.1 was not filtered out even though it contained the non-words *Svair-ah Mahg-noos*.
- We also removed all soft hyphens, meaning all hidden hyphenation marks.

4.2.2 Cleaning

In the cleaning step, we reformatted the parts of the sentences which were inconsistent, or different than natural language. The intention was to avoid hyphenating these words in step 4.3.

- We replaced all links and emails with the character x. More specifically, we replaced all words which had ., @ or : in the middle. After all, it is highly uncommon to break a link or an e-mail in two.
- Most of the punctuation marks had space before and after the symbol. An exception was the slashes. We inserted spaces in between the slashes directly between words.
- For our version of ClueWeb12, we removed the named entity tags, seen in figure 4.1. Furthermore, we corrected the characters which should have been em-dashes (see section 3.4). Specifically, there were some hyphens between two whitespace characters, indicating a longer pause in the sentence. We replaced these hyphens with em-dashes.

- From our parsed version of Ontonotes, which included sentences with corresponding POS-tags, we used the tags to clean the sentences, converting it to one sentence per line. Consider figure 4.2. We replaced the left- and right bracket tags with the proper symbols. To be consistent with ClueWeb12, we removed the spaces around the hyphens which had a HYPH tag.

In this way, the two data sets had similar formatting, one sentence per line, without grammar- or entity tags and links and emails.

4.3 Adding hyphens to sentences

The last preprocessing step was to hyphenate the sentences from ClueWeb12 and Ontonotes. We had three main intentions: to insert the hyphens realistically, with a special symbol, in a random manner. This section describes how we met these three aims.

A realistic hyphenation is similar to what occurs in real documents. We used Frank Liang’s hyphenation algorithm, described in section 3.3, to achieve this authenticity. More specifically, we used Ned Batchelder’s implementation ² of this algorithm, which includes hyphenation patterns up to level six and can, therefore, handle more exceptions. In this way, we ensured an authentic hyphenation process.

We used a special symbol, the Greek Ano Telia, to denote the hyphens. It is comparable to an interpunct character or a centred dot, but the Unicode code point is in the Greek range. Remember from section 4.2.1, that we filtered out all sentences with Greek letters. Therefore, we were sure that the Greek Ano Telia did not occur anywhere else in the data set.

Algorithm 1 shows, in a simplified way, how we randomised the hyphenation procedure. The basic principle is as follows: For each sentence, we started at a random point in the first half of it. We ran the hyphenation algorithm on the word at that point. If it was not possible to hyphenate that specific word, we tried the next word, until a word could be hyphenated. After each succeeding hyphenation, we moved not only to the next word but to the word in a predefined distance.

²<https://nedbatchelder.com/code/modules/hyphenate.html>

Algorithm 1 Hyphenate sentence

Input : A sentence, as string**Output :** The same sentence, with hyphenshyphen_symbol \leftarrow .hyphen_interval \leftarrow 13**begin**

```
  tokens  $\leftarrow$  sentence.split()
  n  $\leftarrow$  len(tokens)
  /* add random slack to the hyphen interval */
  hyphen_interval += random(-3, 2)
  /* start at a random point in the first sentence half */
  pointer  $\leftarrow$  random(0, n // 2)
  while pointer  $\leftarrow$  n do
    try to hyphenate tokens[pointer]
    if hyphenation not possible then
      pointer += 1
      continue while loop
    else if several possible hyphenations then
      choose one hyphenation at random
    end
    /* insert hyphen in between two characters, or on top of an
       existing hyphen */
    insert hyphenation with hyphen_symbol in tokens[pointer].
    pointer += hyphen_interval
  end
  return joined tokens
```

end

In this way, we did not hyphenate every single word, but roughly one or two words per sentence, depending on the length of the input sentence. The final output was a collection of hyphenated sentences and the corresponding original sentences, structured as <hyphenated sentence>TAB<original sentence>. As an example, consider the sentence “Let’s have a time-out to re-evaluate!”, which could have the following output:

Let’s have a time-out to re-evaluate! Let’s have a time-out to re-evaluate!

This example has exactly one new hyphen, namely in *time-out*, on top of the existing hyphen. It simulates a word break on the end of a line. Imagine it as follows:

Let’s have a time·
out to re-evaluate!

where the centred dot represents a hyphen at the end of the line. Notice that *re-evaluate* is kept as it is. In the evaluation, we only consider words with centred-dot hyphens.

4.3.1 The final hyphenated data set

These procedures resulted in two hyphenated data sets. For ClueWeb12, we kept 370,958,448 sentences with 10,182,157,839 words. We inserted 543,538,945 new hyphens. For Ontonotes, we kept 117,450 sentences with 2,234,528 words and inserted 126,028 new hyphens. Because ClueWeb12 is huge, we also extracted roughly five hundred thousand sentences evenly distributed over the data set, to evaluate our algorithms and language models. The data sets are compared in table 4.1.

Hyphenated data set	ClueWeb12 total	ClueWeb12 extract	Ontonotes Release 5.0	Wikipedia extract
size	104GB	152MB	64MB	63MB
sentences	2 · 370,958,448	2 · 529,933	2 · 117,450	–
words	2 · 10,182,157,839	2 · 14,553,968	2 · 2,234,528	2 · 5,294,052
new hyphens	543,538,945	776,700	126,028	300,009

Table 4.1: **A comparison of the hyphenated data sets** This table compares the hyphenated data sets. Remember that each sentence is written twice: once in the hyphenated, and once in the original form. Therefore, the number of sentences in total is two times the number of individual sentences, and the size of the data sets are accordingly larger.

We also gained access to the dataset on which Hertel’s language model was trained. For testing, we used the validation section of this dataset. As before, we used our algorithm to add new hyphens. We will refer to this hyphenated dataset as “Wikipedia extract”. The data set is slightly different because each line consists of a whole paragraph, instead of only a sentence. There were 99,289 paragraphs. In the table, the cell for these sentences is left intentionally blank.

4.3.2 Training and test data

From the hyphenated data set, we also prepared test- and training data for machine learning approaches. To make the training faster and easier, we reformatted the

data in a tab-separated format: <label>TAB<prefix>TAB<suffix>. The prefix was the part of the hyphenated word before the centred dot, while the suffix was the part after. The label was either 1 (indicating a correct hyphen) or 0 (indicating a misplaced hyphen).

There was one entry per line. Of course, this resulted in a data set which had as many lines as hyphenated words. Therefore, for the ClueWeb12 extract, the corresponding machine learning data set had 776,700 lines.

We did not explicitly split the data in training and test parts because we had data from several different sources. That is, we could train on Ontonotes and test on ClueWeb12, or vice versa. We never used the same data for training and testing; that is, we did not do overfitting.

However, we made other versions of the training data with more 1-labels. We did this by including every regular hyphenated word as well. That is, we included words hyphenated with “.” and also the words with “-”. In this way, our machine learning models had more examples of words with correct hyphens. We will note the type of training- and test data explicitly in the evaluation in chapter 6.

5 Implementation

This chapter describes our two different baseline algorithms in detail. Furthermore, we explain how we implemented CRF-based logistic regression. Finally, we describe how the bi-LSTM output probabilities were used to determine whether to delete or retain a hyphen.

5.1 Vocabulary-based baseline algorithms

We implemented two baseline algorithms, based on the same simple idea: to look up the hyphenated word in an English vocabulary. If the hyphenation is known, and quite common, it should probably stay. If not, then the word without the hyphen should be familiar. This chapter describes how we created a vocabulary, a collection of words, specially designed for this purpose. Furthermore, it describes in detail how we implemented the baseline algorithms.

5.1.1 Creating a vocabulary

A vocabulary suitable for our purposes is a collection of most written words. A simple word list, for example, the 171,476 words in the Oxford Dictionary, is not enough, because most words come in different variants. Conjugated verbs, plural nouns, and adjectives with comparative and superlative forms; we needed a vocabulary covering all these forms.

Therefore, we created a vocabulary from scratch, based on all words in ClueWeb12. We registered all unique words, and how often each word occurred. Please note that we only added words which exclusively consisted of letters and hyphens. Words like *ClueWeb12* and *Zipf's* were therefore **not** registered. In this way, we created an extensive list of common words with scores, indicating how frequent a specific word (spelling) is. True to Zipf's law, the word frequencies were inversely proportional to the rank in the list. When sorting the vocabulary after the frequencies in descending order, the first word occurred about twice as often as the second one, which occurred

roughly three times as often as the third one, and so on. Therefore, we assumed that the vocabulary was indeed a good representation of the English language.

As the last step, we discarded all words with the lowest frequency scores. This threshold was set manually, in a trial-and-error style, aiming to remove the worst spelling mistakes and nonce words. Since we were not particularly restrictive, this resulted in a ClueWeb12-based vocabulary of 4,218,204 hyphenated and non-hyphenated words. This number is much higher than the number of English words. A lot of artistic and unusual compound words (such as *hype-the-lemon-matrix*) were included, as well as spelling mistakes and abbreviations. But since the vocabulary included frequencies, this was not directly a problem. We will discuss the bias effects at the end of this section.

Adding words with numbers

Usually, vocabularies do not include words with numbers. But, to handle common word constructions like *17-years-old* or *19th-century*, it was necessary to register these as well. We created a second vocabulary, aiming to include words like these. One obstacle was to get reasonable frequency scores for these words. There are 89 different versions of *xx-years-old*, and even more versions of *xxx-billion-dollar*. We wanted the commonness of the construction *years-old* itself, not only the unrelated frequencies of *17-years-old*, *23-years-old*, and so on. Therefore, we replaced all digits with the Greek character chi. Again, due to the filtering step in section 4.2.1, we were sure this character did not occur anywhere else in the data set. For example, *χχ-years-old* was registered, with a reasonable frequency score. We used a threshold again to remove the worst spelling mistakes and nonce words. This ClueWeb12-based vocabulary had a total of 4,604,374 words with and without hyphens, including representations of words with digits.

Testing bias

There was an obvious risk of a vocabulary bias. We were going to evaluate the baseline algorithm on a ClueWeb12 dataset, with a vocabulary made from the very same sentences. Hence, it was needed to find an external collection of words with which we could test the bias.

We used Maas et al.’s Large Movie Review Dataset [31], a collection of 50,000 highly polar movie reviews. The data set includes a vocabulary of 89,527 words, sorted after frequency in descending order. We used the inverse rank number to

indicate the commonness. Therefore, *the* had a score of 89527, *and* a score of 89526, *a* a score of 89525, and so on.

The IMDB-based vocabulary did not include numbers, and it was much smaller than the ClueWeb12-based one. However, it did include expressive words like *fancy-dancy* and *thousand-cuts-a-minute*, found in the negative reviews. These words are not present in our ClueWeb12 data set. Therefore, we concluded that although ClueWeb12 is web page-based, it did not include sentences from these movie reviews. And we had a collection of words with which we could test the algorithm with less bias.

5.1.2 Baseline algorithm

The simplest variant of our baseline algorithm merely looks the words up in a vocabulary. The vocabulary can come from different sources. Until now, we have described the IMDB-based and the ClueWeb12-based ones. In the evaluation in chapter 6, we will use other versions as well.

In the following paragraphs, when we refer to “the vocabulary”, we always mean “the vocabulary with digits”. Think about it: for the basic algorithm, which only looks up words as they are, it does not make a difference to use the extended vocabulary. As mentioned before, we were sure that χ never occurred in our data set. Unless the algorithm has methods to convert words with numbers into χ , it will never find these words in the vocabulary.

Algorithm 2 describes the baseline algorithm in a simplified way. It returns the word without a hyphen if only this version is in the vocabulary. Likewise, if just the hyphenated word is there, this is returned. If both variants are there, then the algorithm returns the word with the highest frequency. Otherwise, it returns the merged word. The latter is a choice based on the nature of our data. In more than 97% of the cases, the correct answer is to remove the hyphen.

Algorithm 2 Baseline algorithm

Input : A word hyphenated with \cdot

Output : The corresponding word either merged or regularly hyphenated

$\text{vocab} \leftarrow$ load vocabulary from file

begin

$\text{merged_word} \leftarrow$ remove \cdot in the (lowercased) word

$\text{hyphenated_word} \leftarrow$ replace \cdot with $-$ in the (lowercased) word

if merged_word *in* vocab **then**

if hyphenated_word *in* vocab **then**

 /* return the variant with highest frequency score */

$\text{freq_merged} \leftarrow$ frequency of merged_word in vocab

$\text{freq_hyphenated} \leftarrow$ frequency of hyphenated_word in vocab

if $\text{freq_merged} > \text{freq_hyphenated}$ **then return** merged_word **else return**
 hyphenated_word ;

else

return merged_word

end

else if hyphenated_word *in* vocab **then**

return hyphenated_word

else

 /* if neither is found, return the merged variant */

return merged_word

end

end

5.1.3 Supplemented baseline algorithm

The second version of the baseline algorithm has two more features. It can recognize words with numbers, and it can look up only a part of a word if the word has several hyphens.

First, it replaces all numbers in input words with χ , formatting them the same way as in the vocabulary. Then, it proceeds like algorithm 2. However, if neither the hyphenated nor the merged variant is known, but the word has several hyphens, then the algorithm looks up a part of it. For example, for *thou-sand-cuts-a-minute* it looks up *thou-sand* on its own. The ClueWeb12 vocabulary does not include the whole word. However, it knows *thousand*, and would, therefore, remove the hyphen.

Running time

Assuming that the baseline algorithm itself is only loaded once, that is, the vocabulary is the same, the running time is linear in the size of the input data. To read the vocabulary into an internal dictionary takes time $O(m)$ for m words. A lookup in a dictionary has a complexity of $O(1)$, in the average case. Therefore, for n hyphenated words in the input and m words in the vocabulary, the running time of the baseline algorithm is $O(n + m)$.

5.2 Logistic regression

Although we planned to implement a CRF model, we stopped when we noticed how well the simple logistic regression worked (see section 3.6). Therefore, the features which we describe in this section are functions from a CRF framework.

We prepared the features for a sequential model, but we only used two labels (0 and 1), and we only trained on sequences of length one. Therefore, the results which we present in Chapter 6 is comparable with a simple logistic regression (see section 3.6). However, it is easy to extend the code to a sequential model.

In the following, we will describe the features we used. We used the prefix and the suffix (see section 4.3.2) as one input, and predicted one label based on these two. We distinguished between the prefix and suffix features. For the suffix, we used the +1bigram, +2bigram and +3bigram, if possible. For the prefix, we used the same bigrams but in a backward direction. We used an “s” to denote a suffix feature and a “p” to denote a prefix feature. As an example, consider the hyphenation *19th-cen-tury*. The prefix is *19th-cen* and the suffix is *ture*. The suffix bigram features are: *s:+1bigram:tu*, *s:+2bigram:ur*, and *s:+3bigram:ry*. Furthermore, we registered if the prefix and suffix were uppercase, if (the whole part) was a digit or if they included another hyphen. As an example, consider *19th-cen-ture* again. The prefix features were, in addition to the bigram features, *p:uppercase:False*, *p:isdigit:False*, *p:lower:19th* and *p:hashyphen:True*. The last feature, the only one computed from both word parts, was the “word shape”. We replaced all characters except hyphens with x. In this example, it would be *word-shape:xxxx-xxx-xxxx*. These features proved to be sufficient to decide whether a hyphen fitted in a sequence or not.

We used `sklearn-crfsuite`¹, which is a scikit-learn-similar wrapper of `python-crfsuite`² which in turn is a python binding to CRFsuite, “A fast implementation of CRFs” by

¹<https://github.com/TeamHG-Memex/sklearn-crfsuite>

²<https://github.com/scrappinghub/python-crfsuite>

Naoaki Okazaki [32].

We tested two training algorithms. The default L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno, proposed by Nocedal [33]) algorithm with L2 regularization, and OWL-QN (Orthant-Wise Limited-memory Quasi-Newton) with Elastic Net (a combination of L1 and L2 regularization proposed by Zou et Al. [34]). We used the test data sets described in section 4.3.2. Hence, we already had a set of extracted, hyphenated words, and it was not necessary to parse sentences to test the algorithm.

5.3 Using the bi-LSTM

As explained in section 3.7, Hertel’s bi-LSTM has 103 output neurons, one for each element of the character set (the hundred most common English characters, and three special characters). Algorithm 3 shows the procedure we used. We based the hyphen-decision not only on the hyphen but also on the likelihood of other characters in the same word.

Algorithm 3 Usage of the Language Model

Input : A sentence, with word(s) hyphenated with ·

Output : The word(s) either merged or regularly hyphenated

language_model \leftarrow load language model

target_word_indexes \leftarrow empty list

result \leftarrow empty list

begin

foreach *word in sentence* **do**

 /* mark the words which have · hyphens */

foreach *character in word* **do**

if *character is ·* **then** append word-index to target_word_indexes;

end

end

 /* make two new sentences, with and without hyphens for · */

 v1_sentence \leftarrow replace all · with – in sentence

 v2_sentence \leftarrow remove all · in sentence

 /* the language model predicts each character in the sentence */

 v1_all_probs \leftarrow predict character probabilities of v1_sentence with language_model

 v2_all_probs \leftarrow predict character probabilities of v2_sentence with language_model

 /* consider all the target words, which used to have a · hyphen */

foreach *word-index in target_word_indexes* **do**

 v1_word \leftarrow word in v1_sentence on word-index

 v1_word_sum = 0

 /* sum up probabilities of characters in v1_word.. */

foreach *character in v1_word* **do**

 i \leftarrow character index in v1_all_probs

 v1_word_sum += v1_all_probs [i][character]

end

 v2_word \leftarrow word in v2_sentence on word-index

 v2_word_sum = 0

 /* ..and repeat the process for v2_word */

foreach *character in v2_word* **do**

 i \leftarrow character index in v2_all_probs

 v2_word_sum += v2_all_probs [i][character]

end

 /* choose the variant with the highest probability sum */

if v1_word_sum > v2_word_sum **then**

 append v1_word to result

else

 append v2_word to result

end

39

end

end

Our procedure was as follows: for each sentence, we marked the words which had centred-dot hyphens. Then, we made two new versions of the sentence. The first version had regular hyphens in those places where it had been a centred dot. The second version had no such symbols. In other words, the first sentence had many hyphens, and the second version had only natural hyphens.

We used the language model to predict all character probabilities of each of the sentences. Then, we looked up the words which used to have centred-dot hyphens in both of the new sentence versions. For each of these words, we summed up the probabilities of each character. As an example, consider the hyphenation “high-quali·ty”. In the first version of the sentence, the word was: “high-quality”. We added the probabilities of an ‘h’ on this position in the sentence, of an ‘i’, of ‘g’, and so on.

Finally, we chose the word from the first or the second version of the sentence with the highest probability sum. We did this for all the hyphenated words. That is, for each word, we made separate decisions if the hyphenated or the non-hyphenated version was the most likely.

6 Evaluation

This chapter analyses the results of our different methods. First, we describe the metrics we used. Then, we discuss the results and analyse the error types.

6.1 Metrics

We use the metrics: “Total accuracy”, “Accuracy Expected Hyphen”, “Accuracy Expected Non-Hyphen” and “Balanced Accuracy”. Our data sets are very unbalanced. Specifically, the algorithm reaches an overall accuracy of over 97% when it always merges the words. But this value does not show how well our algorithms recognise words that have hyphens that should remain. Some well-known alternatives are “Recall”, “Precision” and “F1-Score”. But these scores will not catch the fact that we are interested in an algorithm which predicts both positive and negative classifications well. To understand this, consider the confusion matrix in figure 6.1.

		expected 0	expected 1	
predicted 0		50.000 (TN)	100 (FN)	
		1000 (FP)	500 (TP)	Precision 33.3%
predicted 1				
				Recall 83.3%
				F1-Score 47.2%

Figure 6.1: **An example of inappropriate metrics** This confusion matrix demonstrates a situation where the recall, precision and F1-metrics does not capture what we want. The green cells are correct predictions, the red cells are mistakes. Precision (P) is defined as $TP/(TP + FP)$ and recall (R) is $TP/(TP + FN)$. The F1 score is the harmonic mean of the latter values, $2 \cdot P \cdot R/(P + R)$

The 0-label denotes a non-hyphen, and the 1-label denotes a hyphen. Notice that the second matrix row is sufficient to calculate the precision. And the second column is enough to calculate the recall.

The problem with these metrics is that they ignore a particular type of word, namely the words correctly classified as 0, shown in the True Negative (TN) class. In our case, this group is very, very much larger than the other ones. If ignored, the metric will not describe how well our algorithm works overall. Because if our algorithm recognises all the words with a correct hyphen, but it neglects words with wrong hyphens, then it is overall worse. Consider figure 6.1 again. The value of TN is neither used to calculate the precision, recall, nor the F1 score. Therefore, these metrics do not reflect the fact that 50,500 choices were correct and that there were only 1,100 errors. But this should be taken into account.

Therefore, we chose to use accuracy as our metric, as well as accuracy for “expected non-hyphen” (specificity), accuracy for “expected hyphen” (recall), and balanced accuracy (bACC). In that way, we could break down the results in a transparent manner. Figure 6.2 illustrates, again with a confusion matrix, what these metrics are. Notice that the first matrix column is enough to calculate the specificity, analogue to the recall and precision calculations demonstrated in figure 6.1.

	expected 0	expected 1	
predicted 0	50.000 (TN)	100 (FN)	Accuracy 97.8%
predicted 1	1000 (FP)	500 (TP)	
	Specificity 98.0%	Recall 83.3%	bACC 90.7%

Figure 6.2: **An example of appropriate metrics** This confusion matrix demonstrates better metrics than in figure 6.1. Specificity (accuracy for expected 0) is defined as $TN/(TN + FP)$; recall (accuracy for expected 1) = $TP/(TP + FN)$; accuracy is $(TN + TP)/(TN + FP + FN + TP)$; balanced accuracy (bACC) is defined as $(specificity + recall)/2$.

It is the same situation as before, where the algorithm did a pretty good job. The

difference is, that the accuracy of “expected non-hyphen” (also called the specificity) is taken into account. The accuracy of “expected hyphen”, which is the same as the recall, is calculated just as before. The balanced accuracy bACC, that is, the average value of specificity and recall, shows that the algorithm made many right choices. The improvement is that the true negatives have affected the score.

6.2 Results

In the following sections, we will present the most remarkable of the results. We tested the models with different types of training data (or different vocabularies), on different data sets. First, we present the results with distinct training and test data. Then, we analyse the types of errors.

Consider table 6.1, which shows some of the results from the evaluations on the ClueWeb12 Extract data set (see table 4.1 for details about the data). The baseline algorithms have the lowest results; however, IMDB and Ontonotes are also the smallest vocabularies. That is, the baseline is never better than its vocabulary. The bi-LSTM gets the highest total accuracy, but the recall is slightly lower than the CRF models.

Model	Version	Accuracy	Specificity	Recall	bACC
Baseline1	VOC-I	98.76%	99.91%	31.67%	65.79%
Baseline2	VOC-O	98.79%	99.91%	33.83%	66.87%
bi-LSTM	–	99.25%	99.56%	80.71%	90.14%
CRF OWL-QN	O+W	98.75%	98.98%	85.78%	92.38%
CRF OWL-QN	OT+W	98.86%	99.11%	84.27%	91.69%
CRF OWL-QN	O+WT	99.14%	99.55%	75.38%	87.47%
CRF L-BFGS	OT+W	98.78%	99.07%	81.45%	90.26%

Table 6.1: **Evaluation results on ClueWeb12 Extract.** The results on the hyphenated data set ClueWeb12. It consisted of 776,700 hyphenated words from which 13,112 were expected hyphens. Abbreviations: VOC-I=vocabulary IMDB; VOC-O=vocabulary Ontonotes. Training data O+W, OT+W, O+WT and OT+W are different combinations of words from Ontonotes and Wikipedia. Specifications: O+W: 11.64% expected hyphen; OT+W: 9.53% expected hyphen; O+WT: 4.42% expected hyphen.

There are results from three CRF models with the same learning algorithm, however, trained on data sets with different portions of expected hyphens. The data

sets are sorted after the portion of expected hyphens, in descending order. They demonstrate the trade-off effect between total accuracy and recall. That is, when the model learns more words with natural hyphens, it recognises such words easier. Therefore, the CRF OWL-QN model trained on O+W (Ontonotes and Wikipedia with 11.64% expected hyphen) has a recall of 85.78% and a bACC of 92.38%. These are the best results for these categories. However, this means that it does make the right decision for as many words without hyphens. Therefore, total accuracy is smaller. On the other side, CRF OWL-QN trained on O+WT (Ontonotes and Wikipedia with 4.42% expected hyphen) recognises fewer words with hyphens. Still, it has a better total accuracy score.

Table 6.2 shows the evaluation results for the Wikipedia extract. The baseline algorithm achieves better results with a more extensive vocabulary. It gets the best bACC score as well as the best accuracy score. The CRF OWL-QN performs worse than on the ClueWeb12 data set, but the training data was only Ontonotes-based, and accordingly smaller. The bi-LSTM gets bACC and total accuracy comparable to the ones from the baseline.

Model	Version	Accuracy	Specificity	Recall	bACC
Baseline2	VOC-C	99.51%	99.95%	76.82%	88.39%
bi-LSTM	–	99.17%	99.67%	73.86%	86.77%
CRF OWL-QN	OEA	98.91%	99.60%	63.78%	81.69%

Table 6.2: **Evaluation results on Wikipedia Extract** The results on the hyphenated data set Wikipedia Extract. It consisted of 299,919 hyphenated words from which 5,734 were expected hyphen. Abbreviations: Baseline2+Voc-C: Second baseline algorithm with ClueWeb12 vocabulary; bi-LSTM: algorithm described in section 5.3 (Note: it was only validated on 272,225 hyphenated words, 27,694 less than total. Further, the model was validated on Wikipedia Extract, which makes overfitting feasible); CRF OWL-QN+OEA: The CRF classifier. trained on all words from Ontonotes.

6.3 Analysis of types of errors

We had the impression that many words were impossible to decide due to the nature of the data set. As explained before, words can have different spellings, for example.

To investigate this effect, we examined our models and algorithms differently. Specifically, we tested the machine learning models on the same data we had trained

them on, and the rule-based algorithm with a vocabulary derived from the same dataset. Theoretically, this should force accuracies close to 100%. Table 6.3 shows the results.

Model/ version	Dataset/ Size	Accuracy	Specificity	Recall	bACC
CRF OWL-QN W	Wikipedia Extract 299,919	99,52%	99,60%	95,01%	97,31%
Baseline2 VOC-C	ClueWeb Extract 776,700	99,67%	99,95%	83,40%	91,68%
Baseline2 VOC-C	ClueWeb Total 543,537,395	99,66%	99,94%	83,55%	91,74%

Table 6.3: **Evaluation on training data** This table demonstrates the relationship between training data and results. The models are tested on the same data set on which they are trained. Baseline 2 is tested on Clueweb with VOC-C (the Clueweb vocabulary), and the CRF OWL-QN model is trained on Wikipedia Extract.

We see that the CRF model, tested and trained on Wikipedia, has 95% recall. This is in line with the theory that the accuracy values should be close to 100%. With the baseline, we tested the same effect with a larger data set. Note that the vocabulary was derived from the total ClueWeb12 dataset, not from the ClueWeb extract. The frequency scores of each word were therefore also based on the total dataset, not the extract. However, the recall is only 83.40%. We could expect a higher number because all the words are in the vocabulary. This suggests that although the algorithm recognizes the words, it is not always certain which spelling variant is the correct one, and this has a big impact on the final numbers.

To confirm this theory, we also tested the Baseline algorithm on the total ClueWeb dataset (see section 4.3). Overall, this resulted in a slightly higher recall value and a slightly higher bACC. But the trend is the same: there are many cases where the algorithm, even though it knows the word with a hyphen, selects a different variant because it is more common. To make it clear, consider the word *email*. It can be written with a hyphen, or without. If the word is written without a hyphen in 70% of the cases, and the model or algorithm learns this, then the 30% of the choices will necessarily be considered “wrong”.

Another thing that affects the datasets is which words occur frequently and which words are rare. For example, *email* or *e-mail* occurs very often in our data sets.

Hence, a 30% error for this word has a very large total impact. However, in the tables presented in this chapter, we chose to include all unique words. Because this represents reality in the best possible way: if a particular word occurs very often, our algorithm must be able to put that word together in a sensible way.

Finally, it is often difficult to know whether the name of a person or a company is with or without a hyphen. The same applies to abbreviations and similar. For example, ADAC does not have a hyphen, in contrast to APD-C.

All in all, we can see that our methods recognise many words with hyphens. This is an improvement compared to the current tools, which do not handle the problem at all. Today's tools do not have a bad overall accuracy value, but, the accuracy of the "expected hyphen" is not satisfactory. The overall accuracy of our methods is better. This can be read based on recall and bACC values. It is uncertain whether it is possible to improve our accuracy scores much at all. Because of the nature of the problem, there are many situations and many words where one cannot say with certainty which decision was the right one.

7 Future Work

This section describes ideas for future work, and it addresses other problems which we discovered during this project. Furthermore, we discuss the extension of the methods to non-English languages.

Hanging hyphens

Due to the limited time and resources within the framework of a Bachelor thesis, the problem of “hanging hyphens” was not further explored. For example, words like “full- and part-time”. Our algorithms looked only at a sequence of characters, ignoring the white space after the line break hyphen. However, we assume that there is only a limited amount of words which follows after such hanging hyphens. Our methods should be effortlessly adaptable to this problem as well.

Conditional Random Field on a sentence

We only used the CRF for logistic regression. It would be interesting to see if the sequentiality of the model can improve performance. Thus, examining whether using a longer sequence as input, either at the word level or the character level, can lead to better results. The good results of the bi-LSTM model indicate this. In this thesis, we have thoroughly examined different sequential models, and we have described the possibilities and dilemmas of these models concerning our problem. The methods we propose are easy to apply to sequences, with most of the work consisting of creating reasonable datasets.

Other classification methods

The performance of the CRF on a single-word level indicates that regular classification methods could be an adequate solution. We explored SVM, linear regression, Naive Bayes, Random Forest, and other algorithms, but only superficially; on a small test set, and without satisfactory results. However, it would be interesting to see how the

results improved if we used features similar to the ones which are common in CRF modelling.

Extensions for non-English languages

The work in this thesis only relates to the English language. We describe techniques which applies to English hyphenation rules. However, as discussed previously, hyphenation rules vary within languages. In English, there are minor differences between American and British spelling. But no matter how you hyphenate a word, the characters remain the same.

This does not apply to all languages. In Norwegian, for example, there are words which are spelt differently when they are hyphenated. Namely compound words which have three similar consonants in a row. These words are, for the sake of convenience, only written with two consonants. This is confirmed by the Norwegian Language Council [35]. For example, the word *trafikkork*, meaning traffic jam, is a compound word of *trafikk* (traffic) and *kork* (cork). Though if the word is hyphenated, it is spelled *trafikk-kork*, adding one more *k* before the hyphen. The same applies to words like *nattog* (night train), *bussjåfør* (bus driver) and *gresstrå* (blade of grass), where one more *t* or *s* is added, respectively. Our baseline algorithm merges the hyphenated words and looks them up in a dictionary. Merging the special hyphenated word *gress-strå* to *gressstrå* (with three consecutive identical consonants) would not give any results, although the spelling without a hyphen is by far the most common one.

We assume there are other peculiarities in other non-English languages. To adapt to this, it is usually necessary to have native speakers.

8 Conclusion

Extracting text from PDF files is a surprisingly difficult problem. We have investigated the possibilities of correcting words with hyphens. In most cases, it is quite obvious which word was the right one. But not always.

The simple algorithm that only looked up the words in a dictionary worked very well, as long as the vocabulary was large enough. Machine learning methods were more flexible. They could recognise words that were not spelt quite as one would think.

It is important to have procedures to remove hyphens, to ensure that people can search through PDF files, and to make the text available for various computer systems. Just as Liang created methods in 1983 to put hyphens into words, it is high time to have methods to remove them.

In this work, we have come up with suggestions on how to solve this small but very significant problem. Our methods work better than those used today. But there will always be a challenge to figure out which word someone was going to write, in retrospect, with very high precision.

9 Acknowledgments

First of all, I would like to say my sincere thank you to Claudius Korzen for his creative input, assistance and our almost daily conversations.

I would also like to thank Prof. Dr. Hannah Bast for enabling me to work on this project and for providing me with a great environment where I could work. My special thanks to everyone at the Chair of Algorithms and Data Structures who welcomed me warmly.

Thank you to Mimi Ford for proofreading.

Special mentions go to Joseline Veit, who has been by my side throughout the studies.

Finally, I would like to thank my friends and family who always support me and encourage me.

Bibliography

- [1] H. Bast and C. Korzen, “The icecite research paper management system,” in *International Conference on Web Information Systems Engineering*, pp. 396–409, Springer, 2013.
- [2] H. Bast and I. Weber, “The completesearch engine: Interactive, efficient, and towards ir & db integration,” in *Third Biennial Conference on Innovative Data Systems*, pp. 88–95, 2007.
- [3] D. Knuth, “Email (let’s drop the hyphen).” <https://www-cs-faculty.stanford.edu/~knuth/email.html>. (accessed: 3d November 2019).
- [4] A. Clark, “Pre-processing very noisy text,” in *Proc. of Workshop on Shallow Processing of Large Corpora*, pp. 12–22, 2003.
- [5] J. C. Micher, “A method for correcting broken hyphenations in noisy english text,” tech. rep., U.S. Army Research Laboratory, April 2012.
- [6] J. Tiedemann, “Improved text extraction from pdf documents for large-scale natural language processing,” in *International Conference on Intelligent Text Processing and Computational Linguistics*, pp. 102–112, Springer, 2014.
- [7] N. Trogkanis and C. Elkan, “Conditional random fields for word hyphenation,” in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pp. 366–374, Association for Computational Linguistics, 2010.
- [8] G. Németh and J. Ács, “Hyphenation using deep neural networks,” October 2017.
- [9] K. Taghva and E. Stofsky, “Ocrspell: an interactive spelling correction system for ocr errors in text,” *International Journal on Document Analysis and Recognition*, vol. 3, no. 3, pp. 125–137, 2001.

- [10] Grammarly, “How we use ai to enhance your writing.” <https://www.grammarly.com/blog/how-grammarly-uses-ai/>, August 2018. (accessed: 29th November 2019).
- [11] K. Clair and C. Busic-Snyder, *A typographic workbook: A primer to history, techniques, and artistry*. John Wiley & Sons, 2012.
- [12] M. Butterick, *Butterick’s Practical Typography*. 2nd ed., 2010. (accessed: 5th November 2019).
- [13] “Pdf 32000-1:2008.” http://wwwimages.adobe.com/www.adobe.com/content/dam/acom/en/devnet/pdf/pdfs/PDF32000_2008.pdf. (accessed: 7th November 2019).
- [14] “Epub 3 overview.” <https://w3c.github.io/publ-epub-revision/epub32/spec/epub-overview.html>. (accessed: 7th November 2019).
- [15] F. M. Liang, “Word hy-phen-a-tion by com-put-er,” tech. rep., Calif. Univ. Stanford. Comput. Sci. Dept., 1983.
- [16] L. Németh, “Automatic non-standard hyphenation in openoffice. org,” *TUGboat*, vol. 27, no. 1, pp. 32–37, 2006.
- [17] Grammarly, “What’s the difference between dashes and hyphens?.” <https://www.grammarly.com/blog/hyphens-and-dashes/>. (accessed: 26th November 2019).
- [18] M. Keary, “On hyphenation - anarchy of pedantry,” *PC Update*, December 1991.
- [19] C. D. Manning, C. D. Manning, and H. Schütze, *Foundations of statistical natural language processing*. MIT press, 1999.
- [20] D. Jurafsky and J. H. Martin, “Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition,” 2008.
- [21] J. Lafferty, A. McCallum, and F. C. Pereira, “Conditional random fields: Probabilistic models for segmenting and labeling sequence data,” 2001.
- [22] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer, “Feature-rich part-of-speech tagging with a cyclic dependency network,” in *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pp. 173–180, Association for computational Linguistics, 2003.

- [23] H. Wallach, *Efficient training of conditional random fields*. PhD thesis, Master’s thesis, University of Edinburgh, 2002.
- [24] H. M. Wallach, “Conditional random fields: An introduction,” *Technical Reports (CIS)*, p. 22, 2004.
- [25] J. R. Finkel, T. Grenager, and C. Manning, “Incorporating non-local information into information extraction systems by gibbs sampling,” in *Proceedings of the 43rd annual meeting on association for computational linguistics*, pp. 363–370, Association for Computational Linguistics, 2005.
- [26] M. Hertel, “Tokenization repair using character-based neural language models.” <http://ad-blog.informatik.uni-freiburg.de/post/tokenization-repair-using-character-based-neural-language-models/>, February 2019. (accessed: 29th November 2019).
- [27] C. Olah, “Understanding lstm networks.” <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>, 2015. (accessed: 29th November 2019).
- [28] Z. Huang, W. Xu, and K. Yu, “Bidirectional lstm-crf models for sequence tagging,” *arXiv preprint arXiv:1508.01991*, 2015.
- [29] R. Weischedel, S. Pradhan, L. Ramshaw, J. Kaufman, M. Franchini, M. El-Bachouti, N. Xue, M. Palmer, J. D. Hwang, C. Bonial, *et al.*, “Ontonotes release 5.0,” 2012.
- [30] S. Pradhan, A. Moschitti, N. Xue, H. T. Ng, A. Björkelund, O. Uryupina, Y. Zhang, and Z. Zhong, “Towards robust linguistic analysis using ontonotes,” in *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pp. 143–152, 2013.
- [31] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, “Learning word vectors for sentiment analysis,” in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, (Portland, Oregon, USA), pp. 142–150, Association for Computational Linguistics, June 2011.
- [32] N. Okazaki, “Crfsuite: a fast implementation of conditional random fields (crfs),” 2007.

- [33] J. Nocedal, “Updating quasi-newton matrices with limited storage,” *Mathematics of computation*, vol. 35, no. 151, pp. 773–782, 1980.
- [34] H. Zou and T. Hastie, “Regularization and variable selection via the elastic net,” *Journal of the royal statistical society: series B (statistical methodology)*, vol. 67, no. 2, pp. 301–320, 2005.
- [35] T. L. C. of Norway, “Bokmålsordboka (monolingual dictionary for norwegian bokmål).” https://ordbok.uib.no/info/om_ordbokene_en.html. (accessed: 28th October 2019).