A Pipeline for Automatic Lexical Normalization of Swedish Student Writings

Yuhan Liu

Uppsala University
Department of Linguistics and Philology
Master’s Programme in Language Technology
Master’s Thesis in Language Technology
June 2, 2018

Supervisor:
Eva Pettersson
Abstract

In this thesis, we aim to explore the combination of different lexical normalization methods and provide a practical lexical normalization pipeline for Swedish student writings within the framework of SWEGRAM (Näsman et al., 2017).

An important improvement in my implementation is that the pipeline design should consider the unique morphological and phonological characteristics of the Swedish language. This kind of localization makes the system more robust for Swedish at the cost of being less applicable to other languages in similar tasks. The core of the localization lies in a phonetic algorithm we designed specifically for the Swedish language and a compound processing step for Swedish compounding phenomenon.

The proposed pipeline consists of four steps, namely preprocessing, identification of out-of-vocabulary words, generation of normalization candidates and candidate selection. For each step we use different approaches. We perform experiments on the Uppsala Corpus of Student Writings (UCSW) (Megyesi et al., 2016), and evaluate the results in terms of precision, recall and accuracy measures. The techniques applied to the raw data and their impacts on the final result are presented.

In our evaluation, we show that the pipeline can be useful in the lexical normalization task and our phonetic algorithm is proven to be effective for the Swedish language.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>3</td>
</tr>
<tr>
<td>List of Tables</td>
<td>5</td>
</tr>
<tr>
<td>Preface</td>
<td>6</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>7</td>
</tr>
<tr>
<td>1.1 Purpose</td>
<td>7</td>
</tr>
<tr>
<td>1.2 Outline</td>
<td>8</td>
</tr>
<tr>
<td>2 Background</td>
<td>9</td>
</tr>
<tr>
<td>2.1 Normalization of Noisy User-Generated Text</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Normalization of Historical Text</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Lexical Normalization</td>
<td>12</td>
</tr>
<tr>
<td>2.3.1 Levenshtein-based Normalization</td>
<td>12</td>
</tr>
<tr>
<td>2.3.2 Phonetically based Normalization</td>
<td>13</td>
</tr>
<tr>
<td>2.3.3 N-grams models for Normalization</td>
<td>14</td>
</tr>
<tr>
<td>2.3.4 SMT-based Normalization</td>
<td>14</td>
</tr>
<tr>
<td>2.3.5 Deep Learning Method for Normalization</td>
<td>15</td>
</tr>
<tr>
<td>2.3.6 Compound Processing</td>
<td>15</td>
</tr>
<tr>
<td>2.4 SWEGRAM</td>
<td>16</td>
</tr>
<tr>
<td>2.5 Characteristics of Swedish Student Writings</td>
<td>17</td>
</tr>
<tr>
<td>3 Data and Tools</td>
<td>18</td>
</tr>
<tr>
<td>3.1 The Uppsala Corpus of Student Writings</td>
<td>18</td>
</tr>
<tr>
<td>3.2 The SALDO Dictionary</td>
<td>20</td>
</tr>
<tr>
<td>3.3 The SUC Corpus</td>
<td>20</td>
</tr>
<tr>
<td>3.4 TheanoLM</td>
<td>20</td>
</tr>
<tr>
<td>3.5 The Peter Norvig Algorithm</td>
<td>20</td>
</tr>
<tr>
<td>4 System Implementation</td>
<td>22</td>
</tr>
<tr>
<td>4.1 Baseline System</td>
<td>22</td>
</tr>
<tr>
<td>4.1.1 Identification of Out-of-vocabulary Words</td>
<td>23</td>
</tr>
<tr>
<td>4.1.2 Generation of Candidates</td>
<td>23</td>
</tr>
<tr>
<td>4.1.3 Candidate Selection</td>
<td>23</td>
</tr>
<tr>
<td>4.1.4 Training/Evaluation Data Splitting</td>
<td>23</td>
</tr>
<tr>
<td>4.2 Modified Pipeline System</td>
<td>24</td>
</tr>
<tr>
<td>4.2.1 Preprocessing Preparation</td>
<td>24</td>
</tr>
<tr>
<td>4.2.2 OOV Word Identification and Compound Processing</td>
<td>25</td>
</tr>
</tbody>
</table>
4.2.3 Candidate Set Generation .......................... 27  
4.2.4 Candidate Selection ................................ 30  
4.3 Evaluation Method ................................. 31  

5 Experiments and Evaluations ......................... 32  
5.1 Experiments on Different Components of the Pipeline .... 32  
5.2 Experiments on Different Training/Evaluation Data Splitting ........ 33  
5.3 Experiments on Candidate Set Quality ..................... 33  
5.4 Results and Evaluation .............................. 33  
5.4.1 Different Components of the Pipeline ............... 33  
5.4.2 Different Training/Evaluation Data Splitting ......... 36  
5.4.3 Candidate Set Quality .......................... 37  
5.4.4 Candidate Selection Quality ....................... 37  

6 Conclusion ............................................ 39  
6.1 Summary ........................................ 39  
6.2 Future Work .................................... 39  

Bibliography ........................................... 41
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Normalization candidates for the word &quot;bday&quot;</td>
<td>10</td>
</tr>
<tr>
<td>2.2</td>
<td>Levenshtein distance examples</td>
<td>13</td>
</tr>
<tr>
<td>2.3</td>
<td>Overview of Swedish compound rules</td>
<td>16</td>
</tr>
<tr>
<td>3.1</td>
<td>Statistics of the UCSW sub-corpus (sv: native Swedish writings; svas: Swedish as second language writings; k1: high school grade 1; k3: high school grade3)</td>
<td>19</td>
</tr>
<tr>
<td>3.2</td>
<td>Sample format of the UCSW sub-corpus</td>
<td>19</td>
</tr>
<tr>
<td>4.1</td>
<td>Baseline system results</td>
<td>23</td>
</tr>
<tr>
<td>4.2</td>
<td>Phonetic algorithm examples</td>
<td>30</td>
</tr>
<tr>
<td>5.1</td>
<td>Results of different methods combined (BL: baseline; PP: preprocessing; CP: Compounds processing; PA: phonetic algorithm candidates generation; LD: Levenshtein distance candidates generation (default distance =1); FB: frequency based candidate selection; LM: neural language model candidate selection; MP: modified pipeline. These acronym conventions are used in all the tables throughout the whole experiment sessions.)</td>
<td>34</td>
</tr>
<tr>
<td>5.2</td>
<td>Preprocessing examples, errors and correct ones</td>
<td>34</td>
</tr>
<tr>
<td>5.3</td>
<td>OOV words that are correctly identified as compounds</td>
<td>35</td>
</tr>
<tr>
<td>5.4</td>
<td>OOV words that are misidentified as compounds</td>
<td>35</td>
</tr>
<tr>
<td>5.5</td>
<td>Words that are merged into compounds correctly</td>
<td>36</td>
</tr>
<tr>
<td>5.6</td>
<td>Words that are merged into compounds incorrectly</td>
<td>36</td>
</tr>
<tr>
<td>5.7</td>
<td>Results of different training/evaluation sets splitting: BL + PA</td>
<td>37</td>
</tr>
<tr>
<td>5.8</td>
<td>Candidate Set quality</td>
<td>37</td>
</tr>
</tbody>
</table>
Preface

First of all, I would like to express my great gratitude to Dr. Eva Pettersson, as an excellent supervisor and an awesome person, who has helped my thesis with great dedication, warm supports, constructive ideas and inspiration, as well as experimental data and tools. Moreover, I would also like to thank all the teachers of the Department of Linguistic and Philology at Uppsala University, who helped and inspired me during the master’s program.

Special thanks must also go to the Uppsala University IPK scholarship program. With their kind sponsorship, I was able to come and study in Sweden and obtain a master degree at Uppsala University. I wholeheartedly appreciate this great generosity from this distant foreign land. Sverige, tack så jättemycket!

Last but not least, I am grateful to my mother, my family and my beloved friends.

Together we make the world a better place.
1 Introduction

1.1 Purpose

Lexical normalization is the task of transforming non-standard word tokens into their standard forms. It is of great interest not only to the NLP community as a preprocessing step that enables more accurate linguistic analysis, but also to common users, such as second language learners, as a writing aid. Moreover, lexical normalization sheds light on the study of the fundamental bricks of natural languages, namely the vocabulary and their morphological, phonetic and semantical characteristics encoded within.

Despite the large amount of research in spelling normalization, not much of them has targeted at the Swedish language. On that account, this thesis aims to contribute to the Swedish lexical normalization task by exploring commonly used normalization methods as well as adapting them for Swedish with special attention to the unique phonetic characteristics of the language. The whole pipeline consists of preprocessing, identification of out-of-vocabulary words, generation of normalization candidates and candidate selection. A phonetic algorithm for candidate generation is the core component in the pipeline design. The whole process is illustrated in Figure 1.1.

We perform experiments on the Uppsala Corpus of Student Writings (UCSW), which features Swedish school student writings. The performance of this pipeline is compared to the performance of the baseline system adopted in the SWEGRAM project regarding precision, recall and accuracy score (Stynmen et al., 2017).

Figure 1.1: Lexical normalization pipeline overview
The SWEGRAM project (Näsman et al., 2017) aims to build a freely available linguistic tool for automatic analysis of Swedish texts, with a special focus on student writings. The system design, implementation and training/evaluation data are all contributed by Uppsala University. In the SWEGRAM system, spelling normalization is an important component for improving further linguistic analysis. Our thesis is partially motivated by improving the normalization performance to further improve the linguistic analysis in the latter part. With this in mind, our research questions are:

1. How do the common normalization approaches work on Swedish student writings?

2. Could we improve current normalization results by adding phonetic models and compound analysis methods that are specially designed for the Swedish language?

1.2 Outline

In attempting to answer these research questions, we propose a lexical normalization pipeline for Swedish student writings and conduct experiments accordingly. We describe our works in the following chapters.

- In Chapter 2, we review the background of this research and related work in terms of both general text normalization and lexical normalization. We also present the SWEGRAM project and the characteristics of Swedish student writings.

- In Chapter 3, we introduce the data sets and tools used for training and evaluation.

- In Chapter 4, we describe the implementation of the baseline and modified systems with different methods, as well as the evaluation metrics for our experiments.

- In Chapter 5, both the baseline and the modified systems are applied to different training and evaluation sets. We compare and discuss the results.

- In Chapter 6, we summarize the results and discuss future work.
2 Background

Text normalization is the process of transforming non-standard, erroneous or informal text into its standard and correct counterpart. Simply speaking, text normalization is a task of transforming an original non-standard string into its standard target string, and lexical normalization means that we perform the string transformation at the token level. Examples could be found in Example 2 and 4.

From punctuation and spell checking to grammatical correction and stylish modification, text normalization is a broad concept which covers text processing at many different levels and is extensively studied and utilized in both research and commercial applications. Many researches (Larson et al., 2000; Monz and De Rijke, 2001; Pettersson, 2016) have shown that normalization has very positive effects on tasks such as information extraction and data mining and it also helps enhancing the performance of many NLP tools such as part-of-speech tagger and parsers. The main reason is that normalization is able to reduce unknown words for NLP systems and thus decrease the data sparseness.

In this section, we discuss the need for text normalization in different kinds of texts and several common lexical normalization techniques. Additionally, we introduce the SWEGRAM project and the characteristics of Swedish student writings.

2.1 Normalization of Noisy User-Generated Text

As indicated by its name, noisy user-generated text (NUGT) is a text generated by Internet users and contains noise, such as errors and misspellings. Typical NUGT can be found in social media, on-line reviews, crowd-sourced data, web forums, etc. Due to the fast growing of social media and web-services, NGUT normalization has become one of the research focuses in this field for its potential use in data mining.

Clark and Araki (2011) feature the lexical normalization of noisy English social media text. In this work they discuss the "casual" characteristics of social media text and try to deal with the challenges coming along with it. According to the authors, the most distinctive differences between social media text and other user-generated texts include

(1) the creative or individualized use of abbreviations and Internet slangs, e.g. "lol" ("laugh out loud"), "loool" ("laugh out loud" with a stronger emotion).

(2) intentional misspellings, e.g. "playin" ("playing"), "happy bday" ("happy birthday").
Example 2: NUGT normalization examples

Since spell checkers usually use regular lexicons without such abbreviations or Internet slangs and perform normalization based on string similarity, they are unable to handle the cases where the original string has a striking difference with the target string, e.g., the string "bday" needs four more letters to be inserted to become the target string "birthday". Table 2.1 shows the normalization candidates generated by Microsoft Word spell checker 2013 and Apple mobile spell checker within iOS version 11.3.1.

Table 2.1: Normalization candidates for the word "bday"

<table>
<thead>
<tr>
<th>System</th>
<th>Normalization candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>MS word</td>
<td>bay, day, bray, bad, badly</td>
</tr>
<tr>
<td>Apple Mobile</td>
<td>bray, brays</td>
</tr>
</tbody>
</table>

In their experiments, the authors adapt two open source spell checkers to the social media lexical normalization task with the following main approaches. Firstly, they use a preprocessing step to remove alphanumeric character repetitions for more than two characters, e.g., "loool" to "lool" and to deal with special punctuation marks such as # (hashtags) and @ (at). Secondly, they enrich the system lexicon with social media specific vocabulary and build mappings between them and their normalized counterparts from several normalized Twitter corpora. Thirdly, they use a language model to score expressions for determining the final normalization candidate.

For all the out-of-vocabulary (OOV) words that are not in the system vocabulary, the system generates a list of in-vocabulary words and then selects the one most similar to the original word or based on a language model score. Their systems achieve a substantial improvement of performance after the adaptation.

Another important work on this topic by Han and Baldwin (2011) emphasizes the automatic generation of candidate tokens of unknown lexical tokens for English social media and short text messages. Instead of relying only on the string similarities for candidate generation as in Clark and Araki (2011), the authors also take phonetic similarities into consideration when generating candidates.

They are also the first to include “ill-formed words” detection in the normalization process. Ill-formed words are words that are not in the system vocabulary and should be changed by the system. It is a subset of OOV words, excluding the words that are not in the system vocabulary but are valid words and thus should not be changed, e.g. recent new words and name entities. For example, in the sentence "Yuhan Liu studys in Sweden.” the first three words are all OOV words according to traditional lexicon such as Merriam-Webster’s; but only "studies" is ill-formed and "Yuhan Liu" is a name and should not be changed. The identification task of ill-formed words could be seen as a classification problem. The authors train a support-vector-machine on large corpora to do the task based on the context (surrounding tokens) of the input token.
The concept of "ill-formed words" is very helpful to reduce unnecessary changes to the original text. Their work inspired many researchers working on the same topic. In the Workshop on Noisy User-generated Text (Baldwin et al., 2015), researchers investigate the Twitter lexical normalization task with a focus on named entity recognition (NER). Different systems based on neural networks achieve the state-of-the-art performance for lexical normalization and the NER tasks.

Commercial applications of text normalization include different kinds of automatic writing aid softwares, from simple spelling correction, and grammar inspection to complex proofreading, e.g., Microsoft Word and Apple auto spell and grammar checkers for different languages.

2.2 Normalization of Historical Text

Historical documents provide valuable information about different aspects of life and environments of the past. However, historical texts usually to some degree differ from modern texts in vocabulary, spelling and grammar, which makes it difficult for researchers without such knowledge. However, historical text can be seen as a special kind of non-standard text and its modern translation or interpretation can be seen as the target text. In this way, we could use text normalization as a way to study and understand historical text or prepare it for further research.

The challenge for historical text normalization lies mainly in the various characteristics based on different time periods, genres, geography and languages. Based on the study of Pettersson (2016) on historical Swedish, spelling is one of the most important features that tells historical and modern text apart. In this paper, the author illustrates it with a manually normalized Middle English example (3) by historical linguist Markus (1999)

(3) Original: To the moost noble & Worthiest Lordes moost ryghtful & wysest conseille
(4) Normalized: To the most noble and Worthiest Lords most rightful and wisest council

Example 4: Historical text normalization examples

For the original segment in Example 4, the historical spelling for words such as "moost" ("most") and "Lordes" ("Lords") would very likely be OOV words for NLP tools designed for modern English. However, if we normalize these words into their modern spellings, there is a better chance that NLP tools could deal with them. Based on this assumption, Pettersson (2016) designs a historical text normalization pipeline which is applicable to several European languages including, English, Swedish, German, Hungarian and Icelandic, in order to improve the performance of modern taggers and parsers on historical text and to aid further information extraction.

In order to tackle this problem, the author uses several approaches, among which the character-based statistical machine translation (SMT) approach achieves very good results and attracts the most attention. As mentioned before, historical normalization could be seen as a translation task between
historical text and modern text. However, there is a big lack of data for training a word or phrase based SMT system. The author avoids this problem by using a character-based SMT system; The words are dismantled into characters which largely increases the training data size and thus ensures the normalization quality with even a small size of historical data as the training data.

Inspired by Pettersson et al. (2013b) along with other research regarding character-based SMT (Scherrer and Erjavec 2013, Costa-Jussà et al. 2016), Korchagina (2017) applies a character-based method for both an SMT system and a neural machine translation (NMT) system on historical German normalization task and compares their performance given different training data sizes. The experiments show that the SMT system performs better with a smaller training set and the NMT system surpasses the SMT system as the training data size grows.

2.3 Lexical Normalization

Lexical normalization is the focus of this thesis. We refer to lexical normalization as the process of automatically converting non-standard, erroneous lexical token to correct forms. The noisy channel model (Shannon, 1948) has traditionally been the primary approach for text normalization. Suppose that there is an ill-formed token T and our goal is to find its standard form S. From all the possible words in the vocabulary V, we would like to find the word s such that P(s|T) is highest, as shown in Formula 5

\[ s^* = \arg\max_{s \in V} P(s|T) \]  

(5)

We could transform formula 5 with Bayes rule, and thus get formula 6, where P(s) is usually a language model and P(T|s) is an error model (Jurafsky, 2000).

\[ s^* = \arg\max_{s \in V} P(T|s)P(s) \]  

(6)

Mays et al. (1991) extend the noisy channel model by generating a candidate spelling set for every word in a sentence. In this way, we could simplify formula 6 to formula 7 where C stands for candidate set.

\[ s^* = \arg\max_{s \in C} P(T|s)P(s) \]  

(7)

Along with the work of Mays et al. (1991) and many others (Han and Baldwin 2011, Pettersson et al. 2013b, Ahmed 2015), in this work we firstly focus on the generation of candidate set and then the selection of the candidate based on the noisy channel model. Thus we aim to increase both P(T|s), which stands for the candidate generation, and P(s), which stands for the candidate selection, by using as much information as we can gather, e.g. typographical, phonetic and context information.

2.3.1 Levenshtein-based Normalization

The Levenshtein edit distance algorithm and its various versions are among the most widely used typographical methods for lexical normalization. By
comparing two strings in the measure of editing operations (insertion, deletion, substitution) required to transform one string into another, we can determine how alike and different they are to each other (Levenshtein (1966)). In the original Levenshtein edit distance algorithm, all these three operations have a weight of one. The algorithm is summarized in Figure 2.1 by Pettersson (2016).

\[
\begin{align*}
\text{dist}(0, 0) &= 0 \\
\text{dist}(i, 0) &= i \\
\text{dist}(0, j) &= j \\
\text{dist}(i, j) &= \begin{cases} 
\text{dist}(i-1, j) + 1 & \text{deletion} \\
\text{dist}(i, j-1) + 1 & \text{insertion} \\
\text{dist}(i-1, j-1) + 1 & \text{equality} \\
0 & \text{if } i = j \\
1 & \text{otherwise} 
\end{cases} \\
&\quad \text{substitution}
\end{align*}
\]

Figure 2.1: Levenshtein distance algorithm

Some Levenshtein distance examples can be find in Table 2.2.

Table 2.2: Levenshtein distance examples

<table>
<thead>
<tr>
<th>original</th>
<th>normalized</th>
<th>Levenshtein distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>moost</td>
<td>most</td>
<td>1 (1 deletion)</td>
</tr>
<tr>
<td>ryleftful</td>
<td>rightful</td>
<td>1 (1 substitution)</td>
</tr>
<tr>
<td>conseille</td>
<td>council</td>
<td>4 (1 substitution, 3 deletion)</td>
</tr>
</tbody>
</table>

Many important algorithms for finding the potential candidate strings for an unknown string rely on this well-known algorithm and its variations. With the Levenshtein distance algorithm, we could generate candidate strings that are within a certain edit distance limit of an unknown string.

Due to the nature of the algorithm, the candidates that it is able to generate depend on the amount of possible operations we allow. To collect candidate correction candidates, Kernighan et al. (1990) make an assumption that the correct word will differ from the misspelling by a single insertion, deletion, substitution. This limit of one operation is proven to be simple but powerful. Kukich (1992) shows that misspellings are usually within one edit from the intended words. A threshold of one also limits the candidate list to a much smaller amount. In the lexical normalization task by Han and Baldwin (2011), they use different Levenshtein distance threshold to generate candidate sets and show that an increase of edit distance from one to two increases the candidate set size by 10 times. However, these experiments are mostly conducted for English.

2.3.2 Phonetically based Normalization

Besides the apparent spelling pattern presented, the pronunciation behind the spelling has also drawn the attention of many researchers. Kukich (1992)
categorizes spelling errors into three types, namely typographic, cognitive and phonetic. Typographic errors refer to errors caused by the lack of typing or writing accuracy. Cognitive errors happen when the writer has certain spelling knowledge deficiencies. Phonetic errors are a special kind of cognitive errors, for the writer has the knowledge about the pronunciation of the word but use an orthographically wrong spelling. For example, for the OOV word “roppar”, words “hoppar” (“jump”), “droppar” (“drops”), “kroppar” (“bodies”), “doppar” (“dots”), “toppar” (“tops”), “proppar” (“props”), “ropar” (“call out”), “koppar” (“copper”) are all within an edit distance of one. However, if we take the similarity of pronunciation into consideration, we can make an easy guess that “ropar” might be more possible than others, because it has the most similar syllables with the original word.

For many languages, there is a pattern between a word’s pronunciation and its spelling. Phonetic methods originate from the idea of generating possible OOV word candidates with phonetic similarities. Toutanova and Moore (2002) incorporate pronunciation information into a noisy channel model by Brill and Moore (2000) and develop their spelling correction algorithm. The essential part is a letter-to-phone model that converts words to their pronunciation to make a pronunciation dictionary. Boyd (2009) further extends this idea by addressing phonological habits of non-native English speakers.

For the social media and SMS normalization task, Han and Baldwin (2011) incorporate the double metaphone algorithm by Philips (2000) to encode and decode word pronunciations. When an OOV token is encountered, they decode the pronunciation and find similar in-vocabulary (IV) word candidates that share similar phonetic encodings. Some researchers convert text tokens into phonetic or numerical representations and find the corresponding words by looking them up in a phonetic dictionary (Ahmed, 2015; Kobus et al., 2008).

### 2.3.3 N-grams models for Normalization

Levenshtein-based and phonetic-based methods are able to generate candidates of an OOV token at the non-contextual level. In order to find the probable candidate, we could exploit the information surrounding the token as well. N-gram language models play an important role in the process. Given a sentence $S = s_1, s_2, ..., s_k, ..., s_n$, where $s_k$ has alternative spellings $s_{k1}, s_{k2},$ etc., a language model can choose one from among these alternative spellings the spelling that maximizes $P(W)$, by using N-gram models to compute $P(W)$.

Very large contexts are proven to be extremely effective, for example, the 5-grams publicly released by Google Brants and Franz (2006). This database is built upon a 1 trillion words English Web text corpus and it consists of bigrams to 5-grams frequency counts from the corpus.

Many researchers in this field include language model scoring as a step to pick the most probable candidate, e.g. Han and Baldwin (2011); Saloot et al. (2014, 2015); Derczynski et al. (2015).

### 2.3.4 SMT-based Normalization

Statistical machine translation (SMT) based models deal with normalization as a translation task from ill-formed text to standard text. Due to the statistical
nature of SMT systems, such normalization is inherently contextual sensitive. SMT-based normalization is firstly introduced in an SMS text normalization task where text messages (NUGT) as the original language are translated into standard languages as the target language (Aw et al., 2006). It has been heavily investigated by many researchers since (Kaufmann and Kalita, 2010; Saloot et al., 2015).

However, there is usually a lack of large annotated data sets for training an SMT-based system (Han and Baldwin, 2011), more specifically word-based and phrase-based SMT systems. For the character-based SMT-based and NMT-based systems by Pettersson (2016) and Korchagina (2017) in historical text normalization are different for they define the surrounding characters as "context" instead of surrounding words. In this way, their normalization is at token-level instead of contextual-level.

2.3.5 Deep Learning Method for Normalization

Recently, deep learning models, e.g. feed-forward neural networks (FFNN), recurrent neural networks (RNN), long-short term memory model (LSTM), for text normalization have drawn much attention from the NLP community. In a noisy Twitter normalization task, Chrupała (2014) uses an RNN model to generate text embeddings and achieves state-of-the-art results. In the 2015 ACL Workshop on Noisy User-generated Text (WNUT), the LSTM system built by Min and Mott (2015) achieved the best results in a noisy Twitter text normalization task, regardless of exploiting additional resources (Baldwin et al., 2015).

2.3.6 Compound Processing

Compounding is a very common linguistic phenomenon in Germanic languages, such as Swedish, German and Dutch, etc. It describes the process of compound words formation. Compounds, short for compound words, are words formed by joining two or more words together without any word boundaries such as spaces. For instance, the German word "Nacktschnecke (slug)" consists of two word parts "nackt" and "Schnecke", which respectively means "naked" and "snail". The compounds that are flexibly joint together usually post challenges to NLP applications, for they significantly increase the vocabulary size and cause data sparseness problem.

Compound splitting has been studied for many purposes in different languages. In machine translation systems, compound splitting is the task of breaking up compounds in the source language into smaller lexemes in order to build one-to-one correspondences to the words in the target language. For instance, we could split "Nacktschnecke" into "nackt" and "Schnecke" and build one-to-one correspondences from "nackt" to "naked" and "Schnecke" to "snail". The goal is to reduce data sparseness and ultimately improve the overall translation performance.

Koehn and Knight (2003) propose a rule-based recursive method to split German compounds into parts and then compare several empirical methods to pick the correct splittings. Their experiments are evaluated on three metrics: a gold standard between compounds and their splittings, word-based SMT
translation quality, and phrase-based SMT translation quality between German and English. Their best method incorporates a German-English parallel corpus and part-of-speech information achieves an accuracy of 99.1% on the first metric and a big improvement of SMT translation quality measured by BLEU.

Similar work has been done in Swedish by Stymne and Holmqvist (2008) in a phrase-based SMT system between Swedish and English. In their experiments, compounds are split as a preprocessing step for Swedish as the source language and compound merging is performed for Swedish as the target language. The experiments show that their method improves the translation quality for English-Swedish translation, but not quite the other translation direction. However, it reduces untranslated Swedish words in the English translation output by half.

In information retrieval and information extraction tasks, compound splitting is an important method to improve retrieval performance for compounding languages. Monz and De Rijke (2001) investigate compounding as a morphological phenomenon and its influence on retrieval tasks in Dutch and German. Experiments demonstrate that compound splitting has a significant positive impact on the final retrieval performance, 25% of improvements for German and 69% for Dutch. Rosell (2003) uses compound splitting to improve the indexing of clustering of Swedish newspaper articles and gains an improvement of 10%.

Larson et al. (2000) investigate the effects of compounds splitting and word recombination for speech recognition. Their experiments result show that compound splitting does not have a visible influence on the system performance. However, it helps reducing the amount of out-of-vocabulary (OOV) words.

In this lexical normalization task, our goal to recognize and normalize compounds and compound parts with both compound splitting and merging. For an overview of compounding rules you refer to table 2.3 below, summarized from several sources (Holmes and Hinchliffe, 2008; Stymne and Holmqvist, 2008). A more detailed introduction to the compounding rules of the Swedish language can be found in Holmes and Hinchliffe (2008).

<table>
<thead>
<tr>
<th>Type</th>
<th>Suffixes</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concatenation</td>
<td></td>
<td>bilresa (bil + resa) vehicle trip</td>
</tr>
<tr>
<td>Filler letter</td>
<td>-s</td>
<td>parkeringshus (parkering + hus) a multi-storey car park</td>
</tr>
<tr>
<td>Truncation</td>
<td>-e -a</td>
<td>pojknamm (pojke + namn) a boy’s name</td>
</tr>
<tr>
<td>Old case ending</td>
<td>-a/-s -a/-e -a/-u -a/-o -e/-a -e/s</td>
<td>veckoslut (vecka + slut) weekend</td>
</tr>
</tbody>
</table>

2.4 SWEGRAM

SWEGRAM is a freely available on-line tool, contributed by Uppsala University, for linguistic annotation and analysis for Swedish text (Näsman et al., 2017).
The system is able to automatically annotate uploaded text files at different linguistic levels with morphological and syntactic information. Users can gather statistics about the texts regarding the terms, number of words, readability measurements, word statistics, etc.

The SWEGRAM project is an important contribution to the Swedish NLP community, for it greatly enriches humanity and social research.

2.5 Characteristics of Swedish Student Writings

To investigate the characteristics of Swedish student writings, Stymne et al. (2017) have a close look into the UCSW corpus. UCSW consists of 2,500 Swedish students writings from students of different academic years and backgrounds, including native Swedish writers and writers with Swedish as a second language. Their investigation shows that the major error types in Swedish student writings are misspellings, compound splitting and merging, grammatical errors, casing and a mix of two or several of them, e.g. the phrase “jete surt” should be written as a compound “jättesurt” (“very sour”); thus “jete surt” contains a spelling error and a compounding error at the same time. Detailed statistics of the errors is shown in their paper.

Among these error types, misspellings, together with incorrect compounding are the most common error types. Misspellings happen in all languages, while compounding is a unique and important characteristics of the Swedish language, since Swedish is one of the compound rich languages. According to Holmes and Hinchliffe (2008), compounding, along with borrowing, affixation and abbreviation, are the four main ways to make a change to the Swedish vocabulary. In an inspection on a Swedish corpus, Stymne and Holmqvist (2008) show that 37% of all the Swedish words that are longer than 12 characters are compounds. In this way, it could be expected that compound normalization, including splitting and merging would have a positive impact on the overall normalization performance.
3 Data and Tools

To design and test a lexical normalization pipeline for Swedish student writings, we need a Swedish student writings corpus with a manually normalized gold standard. In this thesis, we use the dataset of The Uppsala Corpus of Student Writings (Megyesi et al., 2016). This corpus contains Swedish students writings in different academic years from grade three in primary school to grade twelve in high school, ranging from age nine to age nineteen. Meanwhile, we use the SALDO dictionary, a 1.1-million word lexicon for written Swedish, to determine if a word is present in the Swedish language (Borin et al., 2008). We also use the Stockholm-Umeå Corpus (SUC, Version 2.0) (Gustafson-Capkova and Hartmann, 2006), a balanced collection of representative Swedish texts to train a 5-gram Swedish language model, which supports context-based candidate selection. Additionally, as a crucial tool for language model training and getting the language scores of sentences, TheanoLM Enarvi and Kurimo (2016) is also introduced. Moreover, we modify Peter Norvig’s algorithm to generate Levenshtein distance normalization candidates. These data and tools will be described further below.

3.1 The Uppsala Corpus of Student Writings

The Uppsala Corpus of Student Writings (Megyesi et al., 2016) is a collection of Swedish texts written by students as part of a national test since the year 1996. The texts are written by native Swedish users as well as by users with Swedish as a second language. The whole corpus currently consists of 2,500 digitalized student essays and over 1.5 million tokens and its size is gradually growing with new data. Since this corpus contains private and sensitive personal information, it is strictly limited to research purposes and is not available to the public.

The UCSW corpus has been automatically tokenized, normalized and annotated with lemma, part-of-speech tags and syntactic dependency, etc using state-of-the-art linguistic analyzing tools. The lexical normalization is performed as spelling correction by using an unweighted Levenshtein distance with threshold (Pettersson et al., 2013a). Compounding is handled by using a rule-based system designed by Ohrman (1998).

In our experiments, we use a sub-corpus of 492 essays from the UCSW corpus. These essays contain a gold standard with manual normalization and error annotation. The statistics of the corpus is shown in Table 3.1.

---

Table 3.1: Statistics of the UCSW sub-corpus (sv: native Swedish writings; svas: Swedish as second language writings; k1: high school grade 1; k3: high school grade3)

<table>
<thead>
<tr>
<th>Grade</th>
<th>Essays</th>
<th>Sentences</th>
<th>Tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 sv</td>
<td>86</td>
<td>969</td>
<td>9413</td>
</tr>
<tr>
<td>3 svas</td>
<td>69</td>
<td>969</td>
<td>8984</td>
</tr>
<tr>
<td>6 sv</td>
<td>58</td>
<td>1788</td>
<td>21335</td>
</tr>
<tr>
<td>6 svas</td>
<td>53</td>
<td>1614</td>
<td>19989</td>
</tr>
<tr>
<td>9 sv</td>
<td>79</td>
<td>2818</td>
<td>43322</td>
</tr>
<tr>
<td>9 svas</td>
<td>62</td>
<td>1863</td>
<td>28604</td>
</tr>
<tr>
<td>k1 sv</td>
<td>15</td>
<td>314</td>
<td>5879</td>
</tr>
<tr>
<td>k1 svas</td>
<td>54</td>
<td>1461</td>
<td>26863</td>
</tr>
<tr>
<td>k1 sv</td>
<td>12</td>
<td>464</td>
<td>9999</td>
</tr>
<tr>
<td>k3 svas</td>
<td>4</td>
<td>170</td>
<td>3493</td>
</tr>
</tbody>
</table>

The sub-corpus uses the same CoNLL-U format as the main corpus. The CoNLL-U format is designed for universal dependency projects to clearly represent language data and its linguistic information including tagging and dependency relationships across different language data (Nivre et al., 2016). In the UCSW corpus and the sub-corpus, the CoNLL-U format differs slightly from the original by adding lexical normalization information.

In our sub-corpus, different articles are separated by blank lines and meta data enclosed in "< >". Tokens are shown one per line with indexes indicating its location in the sentence and paragraph. Sentence and paragraph boundaries are marked with blank lines. Besides the sentence and paragraph ID and the token itself, it also shows its manually corrected counterpart, lemma, part-of-speech tags, dependency labels, etc. You can find a concise format sample in the table below.

Table 3.2: Sample format of the UCSW sub-corpus

<table>
<thead>
<tr>
<th>Token id</th>
<th>Index</th>
<th>Token</th>
<th>Manual-correct</th>
<th>Lemma</th>
<th>Pos</th>
<th>Gloss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>1</td>
<td>Ny</td>
<td>Ny</td>
<td>ny</td>
<td>ADJ</td>
<td>new</td>
</tr>
<tr>
<td>1.1</td>
<td>2</td>
<td>teknik</td>
<td>teknik</td>
<td>teknik</td>
<td>NOUN</td>
<td>technique</td>
</tr>
<tr>
<td>1.1</td>
<td>3</td>
<td>och</td>
<td>och</td>
<td>och</td>
<td>CCONJ</td>
<td>and</td>
</tr>
<tr>
<td>1.1</td>
<td>4</td>
<td>nya</td>
<td>nya</td>
<td>ny</td>
<td>ADJ</td>
<td>new</td>
</tr>
<tr>
<td>1.1</td>
<td>5</td>
<td>sätt</td>
<td>sätt</td>
<td>sätt</td>
<td>NOUN</td>
<td>ways</td>
</tr>
<tr>
<td>1.1</td>
<td>6</td>
<td>att</td>
<td>att</td>
<td>att</td>
<td>PART</td>
<td>to</td>
</tr>
<tr>
<td>1.1</td>
<td>7</td>
<td>läsa</td>
<td>läsa</td>
<td>läsa</td>
<td>VERB</td>
<td>read</td>
</tr>
</tbody>
</table>
3.2 The SALDO Dictionary

The Swedish Associative Thesaurus version 2, also known as SALDO, is a large modern Swedish lexical dictionary designed for Swedish NLP research and applications (Borin et al., 2008). It is built up based on the Swedish Associative Thesaurus (SAL) and has been extensively extended with descriptive information about Swedish lexicon.

The SALDO version 2 in our experiments consists of over one million Swedish words in the modern Swedish language. It is free to download and is distributed under a Creative Commons Attribute-Share Alike license.

3.3 The SUC Corpus

The Stockholm-Umeå Corpus (SUC) (Gustafson-Capková and Hartmann, 2006) consists of over one million words of balanced Swedish texts from the 1990’s. The corpus is balanced because it is designed to contain texts from different genres, styles and writing levels. The corpus contains linguistic annotations and structural information.

The scrambled version of SUC corpus is freely available for research. It contains the same information with the original corpus, but the sentence orders are mixed. In our project we use this scrambled version to avoid license problems.

3.4 TheanoLM

TheanoLM is a novel tool for neural network language modeling, implemented using the Python library Theano. It provides the possibility to train different neural network language models, e.g., long short-term memory (LSTM), gated recurrent units (GRU), bidirectional recurrent networks, etc. It can be used for sentence scoring, n-best lists, and text generation and it can be called directly from a Python script.

The tool has been evaluated in difficult tasks such as speech recognition in different languages and obtained over-all good or better results, compared to some other toolkits (Enarvi and Kurimo, 2016). In our experiment, we are going to use it to train an LSTM Swedish language model to aid normalization candidate selection.

3.5 The Peter Norvig Algorithm

The Peter Norvig algorithm is designed by Peter Norvig for the purpose of English spelling correction. This algorithm generates all possible IV correction candidates for OOV words within a certain Levenshtein distance of two and returns the candidate with the highest frequency based on statistics from a big English monolingual corpus. A Python implementation is freely available on-line.

---

In order to adapt Norvig’s algorithm to our Swedish lexical normalization task, several changes need to be made. Firstly, we need to change the model from English to Swedish. This requires the substitution of the English alphabet for the Swedish alphabet when calculating Levenshtein distance and generating terms and a Swedish monolingual corpus to collect statistics instead of English ones. Secondly, we need to change the Levenshtein distance threshold to the one needed in our experiment.
4 System Implementation

The goal of this lexical normalization task is to return the normalization candidate with the highest probability given an OOV word and its surrounding contexts, which is the sentence that the word occurs in. We could represent this process with formula [8] where $t$ is the target candidate, $w$ is the original token and $C_w$ is the context sentence.

$$t^* = \arg\max_t P(t|w, C_w)$$  \hspace{1cm} (8)

In this thesis, we regard lexical normalization as a noisy channel model task as discussed in the related work section. To elaborate the model even more, we could use formula [9] where $t$ is the target candidate, $M_w$ is the morphological information of the original token, $P_w$ is the phonetic information of the original token, and $C_w$ is the context sentence.

$$t^* = \arg\max_t P(t|M_w, P_w, C_w)$$  \hspace{1cm} (9)

In this chapter, we are going to discuss the implementation of the baseline system and the modified pipeline designed for the work.

4.1 Baseline System

In order to keep in line with the SWEGRAM project, in this work we adopt and implement the spelling normalization method as described in Stymne et al. (2017). The whole process can be summarized in Figure 4.1.

![Figure 4.1: Baseline system pipeline overview](image-url)
The baseline system is built upon the Levenshtein distance algorithm and the manual normalization of the student texts as in Pettersson et al. (2013a). The mechanism is described below.

4.1.1 Identification of Out-of-vocabulary Words

Firstly, the system checks the length, content, and casing of each token. Tokens of length one and tokens that contain digits will be left unchanged for normalization. Tokens that are not in the beginning of a sentence but with an initial uppercase letter are also left unchanged. In this way, they avoid changing the proper nouns that may be OOV. All the tokens that do not fit the above-described criteria will be considered for further normalization.

Secondly, the rest of the tokens that are ready for normalization will be compared with two lexical resources to determine whether it is in-vocabulary or out-of-vocabulary. The two lexicons are the training corpus vocabulary and the SALDO lexicon described in the previous section, respectively. The training corpus vocabulary is defined as all the tokens in the manually normalized version of the corpus.

4.1.2 Generation of Candidates

The system collects normalization candidates from several sources. First, if a token is in the training corpus with a normalized spelling which differs from the original one, the normalized spelling is collected as a normalization candidate. Second, if a token is OOV for both lexicon resources, it performs edit distance calculations and collects all the tokens from the two lexicons that are within Levenshtein distance of one with the original OOV token.

4.1.3 Candidate Selection

After generating a set of normalization candidate tokens, the system picks the token with the highest appearing frequency, based on statistics from the SUC corpus. If there are candidate tokens sharing the same frequency, the system randomly picks one of them.

4.1.4 Training/Evaluation Data Splitting

Since the original paper does not state how the authors split the data into 90% training corpus and 10% evaluation corpus, we split the data into 9:1 proportions by randomly picking 90% of the sentences as the training set and the rest as the evaluation set, considering that the normalization in this thesis is designed at sentence level. The result of the baseline system with such data splitting is shown in Table 4.1.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline System</td>
<td>80.4%</td>
<td>37.3%</td>
<td>82.0%</td>
</tr>
</tbody>
</table>
Probably due to the random splitting of the training and evaluation set, our baseline system achieves different precision, recall and accuracy measures from the results in the original paper. Therefore, in our experiments we are going to compare the results with the baseline results of our own implementation.

4.2 Modified Pipeline System

The modified pipeline in our work inherits the basic structure of the baseline system as shown in Figure 4.2. Meanwhile, we apply different approaches to improve the performance of the baseline system. We focus on both generation of probable candidates as well as candidate selection. We test the pipeline with the same UCSW corpus used in the baseline system to see whether it can make improvements in terms of the same precision, recall and accuracy measures. Finally, we integrate all the effective methods together to build our modified system.

4.2.1 Preprocessing Preparation

In the preprocessing step, similar to the baseline system, we check and record several characteristics of tokens: the length, the content and the casing format. A token’s content is identified at the levels of Swedish letters, punctuations, blank space and digits.

Firstly, if the length of a token is one or if it contains any digits, the token is not considered for normalization. This rule is also shared by the baseline system.

Secondly, if the token includes any punctuation except for "," or ";", the token will be split by the punctuation, e.g., ""

Thirdly, if the token includes "," or ";", the token will be marked as special case for normalization.

Lastly, for the rest of the tokens, which only contain letters, hyphens and blank spaces, we check and record their casing format. The casing format is represented with "x", "X" and "o". "x" for a lower-case letter, "X" for uppercase letter and "o" for the other characters, e.g. the casing format of token 24.
"IgelKottsFisken" (hedgehog fish) is represented as "XxxxXxxxxXxxxxx" and "arlanda-gymnasiet" (Arlanda High school) as "xxxxxxxoxxxxxxxxxx".

The above preprocessing steps prepare for the following casing normalization rules.

1. If a word starts with a capital letter and all the other letters are lower-case, do not consider the word for normalization, unless the word is at the beginning of the sentence.

2. If a word consists of no lower-case letters, do not change the word, e.g. "T.V."

3. If a word starts with a capital letter and the other letters are not all lower-case, only keep the first letter as upper-case and change the others to lower-case, e.g. "IgelKottsFisken" (hedgehog fish) will be normalized as "Igelkottfisken".

4. If a word is lower-case and its first-letter-capitalized version is an IV word, use the first-letter-capitalized version as the final normalization candidate, e.g. "jenny", "Jenny".

4.2.2 OOV Word Identification and Compound Processing

The first step for OOV words identification is the same as for the baseline system. We use both the training corpus vocabulary and the SALDO dictionary to determine whether a word is out-of-vocabulary. Moreover, if a word is OOV, we also try to uppercase or lower-case the first letter of it to see if there are matches in the vocabulary, e.g. "jenny", "Jenny".

Additionally, in our modified pipeline we perform a compound processing step to make sure that correct but OOV compound words will not be identified as OOV word and thus remain as they are during the normalization process; meanwhile misspelled OOV compounds can be corrected accordingly.

Inspired by Koehn and Knight (2003), we design an algorithm in an attempt to break up compounds into parts. However, because of the possible misspellings within compounds, our algorithm not only splits a compound into known words as in Koehn and Knight (2013), but also allows unknown OOV words to be split from a word. For example, the OOV compound "jetesurt" (the misspelling of "jättésurt", very sour) will be split into "jete" and "surt", although "jete" is not a IV word. Then the system try to normalize the first part "jete". This algorithm is implemented with an exhaustive recursive search method and we restrict the known words to words of at least length three.

Another big difference is that our objective is not compound splitting, but rather using it as a way to identify valid compounds and normalize any misspelling within the word. Therefore, we do not consider the multiple splitting choices for compound words, e.g. the word "aktionsplan" has the following splitting options:

- No-splitting: aktionsplan
- Use "s" as filler letter: aktion-plan
Split the word into two words: aktions-plan

In short, in our algorithm, if a word can be split into known word parts, it is identified as a compound, and no change will be made to it. If a word has a known word rear, e.g. 'jetesurt', we try to normalized the first part. The detailed implementation of the algorithm is described as follows.

1. If a word can be split into two IV word parts, then we identify it as a possible compound word and keep it as it is during the following normalization.

2. Else if a word can be split into two IV parts with a filler letter "s", then we identify it as a possible compound word, and normalize them to one word concatenated with "s" in between.

3. Else if a word can be split into three IV word parts, then we identify it as a possible compound word and keep it as it is.

4. Else if a word is longer than six letters and can be split into one OOV part and one IV part,
   - if the first part is an OOV word with a truncated IV word, then we identify it as a possible compound;
   - else, the first part of the word is considered for normalization in the next step.

5. Else if a word can be split into one OOV and one IV part with the filler letter "s" or "t", the OOV part of the word is considered for normalization in the next step.

In this way we would like to ensure that we do not change valid OOV compounds. However, as mentioned before, there are also cases where two or more separate parts need to be joined together to form a complete compound. For this problem, we use a simple rule-based method to include some common cases. The prerequisite of this method is a collection of common compounds in Swedish. We use two compound sources. The first one is the compounds collected from the manually corrected training corpus; the second one is the possible compounds we extracted from the SALDO corpus.

The way we extract possible compounds from SALDO is as follows: for each single word token in SALDO dictionary, if the token could be split into two known parts longer than 3 letters with or without filler words, it is stored in a dictionary. This naive way of compound splitting extracts 442,397 possible compounds out of all the SALDO vocabulary.

For the scope of this work, we only deal with possible merging options of the combination of two words.

1. For every sentence we perform a linear search over each token and the token after it and concatenate them together and see if it maps to a complete compound word.
2. For every sentence we perform a linear search over each token and the token after it and combine them together with split letter "s" and see if it maps to a complete compound word.

3. For every sentence we perform a linear search over each token and the token after it and combine them with the first token’s last letter removed and see if it maps to a complete compound word.

4. We try to combine the word "jätte", as well as its misspelling versions in the training corpus, e.g. "jete", "jette" , with the following word and see if it maps to a complete compound word. Because it is a high frequency misspelling in the training corpus.

4.2.3 Candidate Set Generation

In this phase we generate normalization candidates for each OOV word and store them in a candidate set for further selection. The very first token in the candidate set is the token itself. Apart from this, we use three other candidate generation methods.

Memory-based Candidate Generation

This is slightly different from the baseline system. We call it memory-based normalization with reference to the memory-based method introduced by Pettersson [2016], where the author extracts mappings between unnormalized words and their normalized counterparts from the training corpus.

The OOV words remaining at this step will be compared to the the training corpus mappings. If this OOV word has only one corresponding normalization in the training corpus, we adopt its normalization result. If it has more than one normalization candidate, then we store them in a candidate set to wait for further selection.

Levenshtein-Based Candidate Generation

We use the same mechanism as in the baseline system to produce normalization candidates of Levenshtein distance one.

For the purpose of comparison, we also generate candidates of Levenshtein distance of two in some experiments.

Phonetically-Based Candidate Generation

Beside generating candidates with memory-based and Levenshtein-based methods, we also generate normalization candidates based on phonetic similarities between the OOV words and IV words.

As preparation, we convert all the IV words in the SALOD lexicon to their phonetic expressions and make a phonetic dictionary in which the phonetic expression is the key, and the list of words that fit this expression is the value of the key. With such key-value pair, we could use the same phonetic algorithm to get the phonetic expression of an OOV word and look it up in the phonetic dictionary to get IV words of the same phonetic expression.
The phonetic algorithm is initially inspired by the Soundex algorithm [Odell and Russel 1918] for English. This algorithm encodes an alphabetical word into its first letter followed by a sequence of digits assigned according to the following rules Kukich (1992):

1. b, f, p, v -> 1
2. c, g, j, k, q, s, x, z -> 2
3. d, t -> 3
4. l -> 4
5. m, n -> 5
6. r -> 6

As shown above, one of the important characteristics of the Soundex algorithm is that it generally ignores the vowel letters in a string unless they are at special position; in this case, the beginning of the string.

Based on the similarities between the Swedish and the English alphabet, we try to design a phonetic algorithm for Swedish using Soundex as a starting point.

Firstly, we classify Swedish letters into vowel letters and consonant letters. Swedish has 29 letters with different letters representing vowels and consonants than for English, e.g. "y" usually represents a vowel sound in Swedish, e.g. "y" in word "typ (type)", but often a consonant and sometimes a vowel in English, e.g. "y" as a consonant in "yes", "y" as a vowel sound in "type") Holmes and Hinchliffe (2008).

- 9 vowel letters: a e i o u y å ö ä; They represent 21 different vowel sounds.
- 20 consonant letters: b, c, d, f, g, h, j, k, l, m, n, p, q, r, s, t, v, w, x, y, z represent 23 different sounds; consonants b, d, f, g, l, m, n, p, r, s, t, z can be doubled, e.g., "pappa", "glass".

Secondly, we classify the individual consonants in consonant clusters of similar pronunciations into different groups. The groups are extracted and sorted out based on the chapter on pronunciation in the work by Holmes and Hinchliffe (2008).

- s, z, c (when c is before e,i,y), sc : sound like English "s"
- j, gj, dj, hj, lj, g (when g is before e, i, å, ö), -lg, -rg: sound like "y" in word "young" in English
- r: sounds like "r" in Scottish English
- t: sounds like "t" in English
- l: sounds like "l" in English
- x: sounds like "ks" in English
- w, v: sound like "v" in English
• kj-, tj-, ch-, k (when k is before e, i, y, ä, ö): sound like "sh" in English

• sk (when sk is before e, i, y, ä, ö), sj-, skj-, stj-, ch-, sch-: sound like "sh" in English

• when g, k, sk are not before e, i, ä, ö:
  – g: sounds like "g" in English
  – k: sounds like "k" in English
  – sk: sounds like "sk" in English

• ng, gn: "ng" sounds like "singer" in English; "gn" sounds like "ng + n"

• -d, -g, -t, -k, -l are often omitted in the spoken Swedish and pronunciation but not in spelling

• -e in -en after r and l are often omitted in the spoken language

Based on these phonetic rules, we can modify the Soundex algorithm accordingly. Instead of using only digits to represent words, we use a string of both digits and letters to represent them. Before the processing, all letters are lower-cased. The complete rules are shown below with the mappings from the letters or letter clusters to their phonetic representations.

1. if 'g' in the end of the word, drop the 'g'
2. if 'r' in the end of the word, drop the 'r'
3. if 'gn' in the beginning of the word, 'gn':'4n'; else: 'gn': 'n'
4. Vowels are ignored unless it is in the beginning and the end of a word, we use '0' to represent all the vowels
5. when 'g', 'k', 'sk' are not before e, i, y, ä, ö:
   • 'g': '4', 'gg': '4'
   • 'k': '3'
   • 'sk': 's3'
6. 'c':'3' (when c is before a, o, u, å), 'ck': '3'
7. 'b':'b', 'bb':'b', 'p':'p', 'pp':'p'
8. 'd':'d', 'dd':'d'
9. 'f':'2', 'ff':'2'
10. 'v':'v', 'w':'v', 'wh':'v'
11. 'h':'x'
12. 't':'t', 'll':'t'
13. 'm':'m', 'mm':'m'
Some examples of the algorithm are shown in Table 4.2. Here we could see the

<table>
<thead>
<tr>
<th>OOV word</th>
<th>phonetic Expression</th>
<th>IV words examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>jette</td>
<td>jt0</td>
<td>jätte, hjärta, jättar</td>
</tr>
<tr>
<td></td>
<td></td>
<td>very, heart, giants</td>
</tr>
<tr>
<td>petersson</td>
<td>ptsn</td>
<td>Pettersson, Pettersson</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pettersson, Pettersson</td>
</tr>
<tr>
<td>marcus</td>
<td>m93s</td>
<td>Markus, Marx, märkas</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Markus, Marx, branded</td>
</tr>
<tr>
<td>tagana</td>
<td>t4n0</td>
<td>taggarna, tuggorna, tiggarna, etc..</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the tags, the chews, the beggars</td>
</tr>
</tbody>
</table>

advantage of the phonetic algorithm: it can generate multiple candidates with a much longer edit distance, and not as many candidates.

4.2.4 Candidate Selection

Frequency-based Selection

This is the same frequency-based method as the baseline system, where we select the most probable candidate by single word frequency collected from the SUC corpus.

Language Model-based Selection

We use a neural network language model to select the candidate. Due to the computational ability of this project, we train a five gram language model with lstm300 network architecture in TheanoLM, which is proven to be reliable and light-weighted [Enarvi and Kurimo (2016)].

In order to determine the most probable candidate given the context of the OOV word, we use a context of five words including the OOV word itself. The rules are represented as follows,
1. If a sentence is shorter than 5 tokens, use the whole sentence as context.

2. Else if there are four tokens before the current token, use the four former words as context.

3. Else, use as many words that are before this token and take words after this token to make up a context of 5.

We loop over all the candidates and put them in the context and calculate the perplexity scores. Theoretically, a lower perplexity score means a larger test set probability according to the language model (Jurafsky 2000). Thus we pick the candidate with the lowest perplexity score.

### 4.3 Evaluation Method

In this project we adopt the same evaluation method with the SWEGRAM project, which is described in Stymne et al. (2017). Precision and recall are calculated with regard to the identification of misspellings.

The calculation are summarized in formulas 10, 11 and 12 where ”#” means ”the number of”; and the correct and incorrect cases refer to correctly and incorrectly normalized tokens in true positives.

\[
\text{Precision} = \frac{\#\text{true positive}}{\#\text{true positive} + \#\text{false positive}} \tag{10}
\]

\[
\text{Recall} = \frac{\#\text{true positive}}{\#\text{true positive} + \#\text{false negative}} \tag{11}
\]

\[
\text{Accuracy} = \frac{\#\text{correct}}{\#\text{correct} + \#\text{incorrect}} \tag{12}
\]
5 Experiments and Evaluations

In this chapter we describe the experiments we conduct in order to investigate:

1. How does each component of the pipeline affect the final performance?

2. Are there any differences between the systems trained on sv data and svas data, respectively?

3. Does splitting the training and evaluation corpus by academic year affects the result?

4. How does phonetic algorithm affect the candidate generation, regarding the quality and size of candidate set?

5. How does the language model affect the candidate selection, comparing to the frequency based selection?

5.1 Experiments on Different Components of the Pipeline

In order to investigate the effects of each component of the pipeline, we train several systems with all the data available without discrimination regarding Swedish as a native language (SV) or Swedish as a second language (SVAS). The data set is randomly shuffled and split into 90% training set and 10% evaluation set by numbers of sentences.

We train a baseline system and several systems based on the baseline system with combinations of different methods introduced in the previous chapter until we reach the final complete pipeline. The components we are going to explore are as follows.

1. The preprocessing method (PP). We compare the baseline system and the system with preprocessing.

2. The Levenshtein distance method (LD). We compare the baseline system where the Levenshtein distance is one to another baseline system where the Levenshtein distance is two.

3. The compound processing method (CP). We compare the baseline system to the baseline system with the additional compound processing method.

4. The phonetic algorithm method (PA) in the candidate generation step.
5. The language model method (LM) in the candidate selection step. Additionally, we also try some combination of them to gradually progress to the final pipeline.

5.2 Experiments on Different Training/Evaluation Data Splitting

In order to explore the performance of our pipeline on both of the SV and SVAS data set, we train two systems respectively using SV and SVAS language data by shuffling and splitting the two data sets randomly into 90% and 10% by the number of sentences.

Moreover, we also split the whole data set into training and evaluation data based on different academic year. The hypothesis is that student writings from lower academic years might contain more mistakes than those of students from higher grades and thus their writings are better as training data. Considering that student writings of higher grades have longer sentences, instead of splitting the data by sentences, we do the splitting by words. This is to say that the sentences that contain the 90% of all the tokens from the lower grades are used as training data, and the sentences that contain 10% of all the tokens from the higher grades are as evaluation data. This is to avoid undersized training data compared to the baseline system.

5.3 Experiments on Candidate Set Quality

As a novel component of the thesis, the phonetic algorithm plays an important role in the process of candidate generation. Thus we would like to investigate the performance of the phonetic algorithm regarding average candidate set size and candidate set quality. Candidate size is the number of IV candidates generated by the algorithm plus the token itself. Candidate set quality inspects whether the correct normalization result is in the candidate set generated by the algorithm. It is defined as shown in Formula 13, where CQ is candidate set quality and "#" means "the number of".

\[
CQ = \frac{\# \text{candidate sets including the target token}}{\# \text{all candidate sets}}
\]  

(13)

We compare the candidate sets generated by the phonetic algorithm with the candidate sets generated by systems using Levenshtein distance method and we also try the combination of them both.

5.4 Results and Evaluation

5.4.1 Different Components of the Pipeline

The performances of the different components of the pipeline and the whole pipeline are summarized in Table 5.1.
Table 5.1: Results of different methods combined (BL: baseline; PP: preprocessing; CP: Compounds processing; PA: phonetic algorithm candidates generation; LD: Levenshtein distance candidates generation (default distance = 1); FB: frequency based candidate selection; LM: neural language model candidate selection; MP: modified pipeline. These acronym conventions are used in all the tables throughout the whole experiment sessions.)

<table>
<thead>
<tr>
<th>Method</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL</td>
<td>94.4%</td>
<td>35.4%</td>
<td>81.8%</td>
</tr>
<tr>
<td>BL+LD=2</td>
<td>76.4%</td>
<td>42.4%</td>
<td>63.6%</td>
</tr>
<tr>
<td>BL+PP</td>
<td>93.7%</td>
<td>35.1%</td>
<td>82.2%</td>
</tr>
<tr>
<td>BL+CP</td>
<td>95.4%</td>
<td>32.2%</td>
<td>82.1%</td>
</tr>
<tr>
<td>BL+PA</td>
<td>87.2%</td>
<td>42.1%</td>
<td>71.7%</td>
</tr>
<tr>
<td>BL+LM</td>
<td>82.5%</td>
<td>41.7%</td>
<td>62.3%</td>
</tr>
<tr>
<td>BL+PA+LD=2</td>
<td>82%</td>
<td>43.6%</td>
<td>59.2%</td>
</tr>
<tr>
<td>BL+CP+PA</td>
<td>86.9%</td>
<td>36.2%</td>
<td>74.4%</td>
</tr>
<tr>
<td>BL+CP+PA+LM</td>
<td>84.3%</td>
<td>37.0%</td>
<td>66.0%</td>
</tr>
<tr>
<td>MP</td>
<td>83.6%</td>
<td>37.3%</td>
<td>66.5%</td>
</tr>
</tbody>
</table>

Preprocessing

From the table, we see that the preprocessing (PP) step has a small positive influence on accuracy and negative influences on precision and recall. Thus, we argue that this step is not worth taking into our pipeline, because of the noise it introduces. On one hand, the preprocessing step is able to avoid the normalization of those words with casing errors. For instance, the word “johan” is not presented in the vocabulary, but with preprocessing, the system is able to find out that it is an IV word with wrong casing, “Johan”, and thus the normalization could be saved. On the other hand, because of the misspelling involved, it also identifies some OOV words as IV words with wrong casings. More examples of preprocessing in the evaluation corpus could be found in Table 5.2.

Table 5.2: Preprocessing examples, errors and correct ones

<table>
<thead>
<tr>
<th>Word</th>
<th>system output</th>
<th>Gold standard</th>
<th>Gloss</th>
</tr>
</thead>
<tbody>
<tr>
<td>jenny</td>
<td>Jenny</td>
<td>Jenny</td>
<td>Jenny</td>
</tr>
<tr>
<td>tv</td>
<td>TV</td>
<td>TV</td>
<td>TV</td>
</tr>
<tr>
<td>medelhavet</td>
<td>Medelhavet</td>
<td>Medelhavet</td>
<td>Mediterranean</td>
</tr>
<tr>
<td>coca-cola</td>
<td>Coca-Cola</td>
<td>Coca-Cola</td>
<td>Coca-Cola</td>
</tr>
<tr>
<td>tagar</td>
<td>Tagar</td>
<td>taggar</td>
<td>take</td>
</tr>
</tbody>
</table>

Although the first four examples from table 5.2 look promising, we need to realize that these candidates could be generated by both the Levenshtein-based method and the phonetically-based method. Going through the trouble of
putting them into normalization could help us avoid the error of the fifth example. Since the misspelling "tagar" appears several times in our evaluation corpus, this one mistake could do more harm.

Levenshtein-based method

Increasing the Levenshtein distance from one to two increases the recall as expected, because the algorithm is able to generate more candidates. However, the big drop in precision and accuracy indicates that it requires a better candidate selection method to remedy the noise it introduces.

Compound Processing

The compound processing step makes the recall lower. This is possible because the system marks possible compounds as IV and leave them as they are. However, the precision and accuracy stay similar with the baseline system.

In the OOV words detection step, our naive way of compound splitting is able to identify many compounds, it also misidentifies many OOV words as compounds. For these two kinds of examples, see 5.3 and 5.4

<table>
<thead>
<tr>
<th>Compound</th>
<th>Splitting</th>
<th>Gloss</th>
</tr>
</thead>
<tbody>
<tr>
<td>jättelångt</td>
<td>jätte + långt</td>
<td>very long</td>
</tr>
<tr>
<td>hjärtstillestånd</td>
<td>hjärt + stille + stånd</td>
<td>cardiac arrest</td>
</tr>
<tr>
<td>barnmat</td>
<td>barn + mat</td>
<td>child food</td>
</tr>
<tr>
<td>filmvärlden</td>
<td>film + världen</td>
<td>the film world</td>
</tr>
<tr>
<td>fritidsintressen</td>
<td>fritid + s + intressen</td>
<td>spare time interests</td>
</tr>
</tbody>
</table>

Table 5.3: OOV words that are correctly identified as compounds

<table>
<thead>
<tr>
<th>Word</th>
<th>Gold standard</th>
<th>gloss</th>
<th>possible splittings</th>
<th>gloss</th>
</tr>
</thead>
<tbody>
<tr>
<td>varanras</td>
<td>varandras</td>
<td>each others</td>
<td>varan, ras</td>
<td>item, race</td>
</tr>
<tr>
<td>hestar</td>
<td>hästar</td>
<td>horses</td>
<td>hes, tar</td>
<td>hoarse, take</td>
</tr>
<tr>
<td>träfar</td>
<td>träffar</td>
<td>meet</td>
<td>trä, far</td>
<td>tree, father</td>
</tr>
<tr>
<td>ventar</td>
<td>väntar</td>
<td>wait</td>
<td>ven, tar</td>
<td>vein, take</td>
</tr>
<tr>
<td>biljet</td>
<td>biljett</td>
<td>ticket</td>
<td>bil, jet</td>
<td>vehicle, jet</td>
</tr>
</tbody>
</table>

Table 5.4: OOV words that are misidentified as compounds

From 5.3 and 5.4 and our evaluation results, we could see that wrong OOV identification mostly happens to shorter compounds. However, the longer compounds, which can be quite accurately determined by the algorithm, usually stay the same in the normalization process, because it is less likely to find similar candidates using Levenshtein distance or phonetic algorithm. It is the shorter possible compounds that need to be examined.

The compound merging results are in a similar state, it helped solve new problems and as well introduced noises. The correct and some incorrect cases are shown in Table 5.5 and Table 5.6.
Table 5.5: Words that are merged into compounds correctly

<table>
<thead>
<tr>
<th>Compound parts</th>
<th>System output</th>
<th>Gloss</th>
</tr>
</thead>
<tbody>
<tr>
<td>här ifrån</td>
<td>härifrån</td>
<td>from here</td>
</tr>
<tr>
<td>inne hålla</td>
<td>innehålla</td>
<td>content</td>
</tr>
<tr>
<td>ansikts uttryck</td>
<td>ansiktsuttryck</td>
<td>facial expression</td>
</tr>
<tr>
<td>till sammans</td>
<td>tillsammans</td>
<td>together</td>
</tr>
<tr>
<td>små pengar</td>
<td>småpengar</td>
<td>small money</td>
</tr>
</tbody>
</table>

Table 5.6: Words that are merged into compounds incorrectly

<table>
<thead>
<tr>
<th>Incorrectly merged word</th>
<th>Gold Standard</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>storskog</td>
<td>stor skog</td>
<td>jag var i en stor skog</td>
</tr>
<tr>
<td>big forest</td>
<td>big forest</td>
<td>we were in a big forest</td>
</tr>
<tr>
<td>fram till</td>
<td>fram till</td>
<td>jag kom fram till att</td>
</tr>
<tr>
<td>until</td>
<td>front to</td>
<td>I came to the conclusion that</td>
</tr>
<tr>
<td>berättaren</td>
<td>berättar en</td>
<td>tjejer berättar en skakande historia</td>
</tr>
<tr>
<td>narrator</td>
<td>tells a</td>
<td>The girl tells a shaking story</td>
</tr>
<tr>
<td>medmänniskor</td>
<td>med människor with people</td>
<td>Där dödades tusetalls med människor</td>
</tr>
<tr>
<td>fellows</td>
<td></td>
<td>Thousands of people were killed with people</td>
</tr>
<tr>
<td>där borta</td>
<td>där borta</td>
<td>lilla fågeln där borta !</td>
</tr>
<tr>
<td>over there</td>
<td>over there</td>
<td>little bird over there !</td>
</tr>
</tbody>
</table>

However, the rule-based compound merging for word "jätte” are very effective and does not introduce noises.

Phonetically-based method

The phonetic algorithm has quite positive effects on the recall measure, nearly as high as using Levenshtein distance of 2. Meanwhile, the precision does not drop as much. We argue that the phonetic algorithm is able to generate high quality candidates without introducing too much noise. The matter of candidate set quality is further discussed in 5.4.3.

Language Model

The language model does not work well as expected. It has a quite good recall with lower precision and accuracy. The possible causes are further discussed in 5.4.4

5.4.2 Different Training/Evaluation Data Splitting

Table 5.7 shows the performance of a part of our pipeline (BL+PA) given different training and evaluation set splittings.

From the table, it can be seen that the phonetic algorithm works better with SVAS data set regarding precision measure. Since the algorithm is designed based on pronunciation. We could assume that SVAS students make more
Table 5.7: Results of different training/evaluation sets splitting: BL + PA

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Random</td>
<td>93.2%</td>
<td>36.4%</td>
<td>81.9%</td>
</tr>
<tr>
<td>SV only</td>
<td>82.8%</td>
<td>47.5%</td>
<td>75.2%</td>
</tr>
<tr>
<td>SVAS only</td>
<td>88.5%</td>
<td>36.8%</td>
<td>72%</td>
</tr>
<tr>
<td>Academic year</td>
<td>60.3%</td>
<td>23.4%</td>
<td>71.5%</td>
</tr>
</tbody>
</table>

"cognitive errors" as argued by Toutanova and Moore (2002). However, it works better with the SV data set regarding the other two measures. This indicates that SV writings training corpus contain fewer mistakes.

The hypothesis that it might be better to use lower grade writings as training corpus is not correct. We got lower scores on all three measures. This shows that the writings of lower grades students do not share a lot of similarities with the writings from higher grades in terms of vocabulary and misspellings.

5.4.3 Candidate Set Quality

Table 5.8 shows the candidate set quality given different candidate set generation method.

<table>
<thead>
<tr>
<th></th>
<th>Size</th>
<th>CQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD=1</td>
<td>2.2</td>
<td>57.6%</td>
</tr>
<tr>
<td>LD=2</td>
<td>18.2</td>
<td>68%</td>
</tr>
<tr>
<td>PA</td>
<td>10.4</td>
<td>62.1%</td>
</tr>
<tr>
<td>PA+LD=1</td>
<td>11.7</td>
<td>71.3%</td>
</tr>
<tr>
<td>PA+LD=2</td>
<td>30.6</td>
<td>73.5%</td>
</tr>
</tbody>
</table>

From the table, we can see that the highest recall is achieved by the combination of PA + LD=2 and the smallest average candidate size is achieved by LD=1.

However, it is clearly seen that the phonetic algorithm has done well. It has a fairly low average candidate size and a fairly high candidate set quality. The PA + LD=2 combination cannot rival in terms of average candidate size and the LD=1 method cannot compete with regard to candidate set quality.

5.4.4 Candidate Selection Quality

In our pipeline the language model is outperformed by the frequency-based regarding precision and accuracy. This is counter-intuitive at first sight. We purpose several reasons for this.

1. Firstly, the errors in the surrounding context may introduce noises, which are unknown tokens to the language model. Thus the system is not able to perform well without a good smoothing strategy.
2. Secondly, the language model is not well-trained with only 10 iterations on the neural network. The hyper parameters of the language model are not well tuned for this task.

3. Last but not least, the one-million-token training data set is too small and thus causes data sparseness problem.
6 Conclusion

6.1 Summary

Lexical normalization is an effective approach to complement the overall performance of general NLP systems, such as part-of-speech taggers and parsers. Under the framework of SWEGRAM, we try to improve the performance of its lexical normalization process with different methods based on the special characteristics of Swedish language and evaluate them. The most fundamental theoretical base of the whole work is the noisy channel model.

In the thesis, focusing on the lexical normalization task for Swedish students writings, we build a normalization pipeline with freely removable components upon the baseline system in SWEGRAM project. Our Levenshtein-based baseline achieve a precision of 94.4%, recall 35.4% and accuracy 81.8% according to our evaluation metrics. Additionally, we use four approaches to modify the baseline system: preprocessing, compound processing, phonetically-based candidate generation and language model-based candidate selection. The baseline + phonetic algorithm achieves the best precision of 95.4%. The best recall is achieved in baseline + phonetic algorithm + Levenshtein distance of 2. The best accuracy is achieved by the system of baseline + compound processing.

The experiments are performed on a manually corrected sub-corpus of the UCSW corpus. We also make the splitting of SV and SVAS writings. The results show that the phonetic algorithm is more effective on SVAS data.

The core and most original component is the phonetic algorithm designed in this pipeline. With the hope of filling the blank of phonetic algorithms for Swedish, we present our design and evaluate it within the lexical normalization task. The phonetic algorithm is proven to be quite effective in the pipeline.

The methods presented in this thesis, although are not all positively effective, have shown their pro and cons.

Overall, we have successfully built the lexical normalization system and made it applicable to Swedish language with satisfying performance of the phonetic algorithm. The experiment results did not reached our initial expectation. More ideas about possible improvements are discussed in the next section.

6.2 Future Work

Even though our normalization pipeline does not outperform the baseline in all the evaluation measures, there is potential to improve its performance.

Firstly, for the OOV words detection step, we could make use of the automatic part-of-speech tags of the words or simply a dictionary with part-of-
speech. Such information is able to help identify compounds with more accuracy. For the Swedish language, the parts that can form a compound are combined with certain rules. According these rules, words like "hes" (hoarse) and "tar" (take) should not be combined into a compound, because an adjective and a verb are not common combinations.

Additionally, we could also add Swedish affixation rules in the OOV words detection step. For instance, in our test set, the valid but OOV word "oelegant" (not elegant) is normalized to IV word "elegant" (elegant), which is of Levenshtein distance of one with the original word. This completely altered the meaning of the word and would cost trouble. However, with affixation rules, we could identify it as a valid word.

Secondly, in the candidate generation step, the phonetic algorithm can be improved with more vowel information. Pettersson et al. (2013a) uses a weighted Levenshtein approach to find vowels with similar pronunciation. According to this work, the Swedish letter "å" is more likely to be misspelled as "o" than being misspelled as "ä". This can potentially improve the candidate set quality by reducing its size.

Last but not least, in the candidate selection step, we could collect bigger corpus and train different language models. However, this can be quite time consuming.
Bibliography


Lars Borin, Markus Forsberg, and Lennart Lönngren. Saldo 1.0 (svenskt associationslexikon version 2). Språkbanken, University of Gothenburg, 2008.


Thorsten Brants and Alex Franz. Web 1t 5-gram version 1. 2006.


Marta R Costa-Jussà, Cristina España-Bonet, Pranava Madhyastha, Carlos Escalona, and José AR Fonollosa. The talp–upc spanish–english wmt biomedical


