

Preprocessing on bilingual data for Statistical Machine Translation

Table of contents

1	INTRODUCTION	3
1.1	MACHINE TRANSLATION	3
1.2	SMT, ALIGNMENT AND PREPROCESSING.....	5
1.3	OVERVIEW OF FURTHER CHAPTERS.....	8
2	STATISTICAL MACHINE TRANSLATION	9
2.1	BASIC THEORY	9
2.2	LANGUAGE MODELING	10
2.3	TRANSLATION MODELING.....	12
2.4	ALIGNMENT	14
2.5	THE EXPECTATION MAXIMIZATION ALGORITHM	15
2.6	GIZA++	18
2.7	PARAMETER ESTIMATION.....	20
3	PREPROCESSING	24
3.1	TOKENIZATION AND SENTENCE ALIGNMENT	24
3.2	STEMMING	24
3.3	NAMED ENTITY RECOGNITION USING CONDITIONAL RANDOM FIELDS.....	25
3.4	NER FOR SMT	28
3.5	NER ALGORITHMS FOR BILINGUAL DATA	29
3.6	UPPERCASE AND LOWERCASE	31
4	EXPERIMENTS AND EVALUATION	32
4.1	THE EUROPARL PARALLEL CORPUS	32
4.2	ALIGNMENT ERROR RATE	32
4.3	PRECISION, RECALL AND F1-SCORE	34
4.4	BASIC EXPERIMENTS	35
4.5	EXPERIMENTS ON BILINGUAL DATA	36
5	EXPERIMENTAL RESULTS	37
5.1	TOKENIZATION.....	37
5.2	NAMED ENTITY RECOGNITION	37
5.3	STEMMING	39
6	REVIEW AND CONCLUSIONS	40
6.1	PREPROCESSING EFFECTIVENESS	40
6.2	USING BILINGUAL DATA FOR PREPROCESSING	41
6.3	FUTURE RESEARCH.....	41
7	REFERENCES.....	43
	APPENDIX A: DUTCH LIST OF NON-BREAKING PREFIXES	45

1 Introduction

1.1 Machine translation

Machine Translation (MT) is the translation of text from one human language to another by a computer. Computers, like all machines, are excellent at taking over repetitive and mundane tasks from humans. As translating long texts from one language to another qualifies as such a task, Machine Translation is a potentially very economic way of translation. Unfortunately natural languages are not very suitable for processing by a machine. They are ambiguous, illogical and constantly evolving, qualities that are difficult to handle with a machine. This makes the problem of Natural Language Processing, and by extension MT, a difficult one to solve.

A theoretical method that can analyze a text in a natural language and decipher its semantic content can store this semantic content in a language-independent representation. From this representation, another text with the same semantic content can be generated in any language for which exists a generation mechanism. Such an MT architecture would provide high quality translations, and be modular; a new language could be added to the pool of inter-translatable languages simply by developing an analysis and generation method for that language.

Unfortunately this method does not exist. Some existing MT attempts to approach it to a degree, but as long as semantic analysis remains an unsolved problem in the field of Natural Language Processing there can be no true language independent representation. Figure 1 shows the Machine Translation Pyramid, which is a schematic representation of the degree of analysis performed on the input text. The MT method described in these paragraphs is at the top of the pyramid.

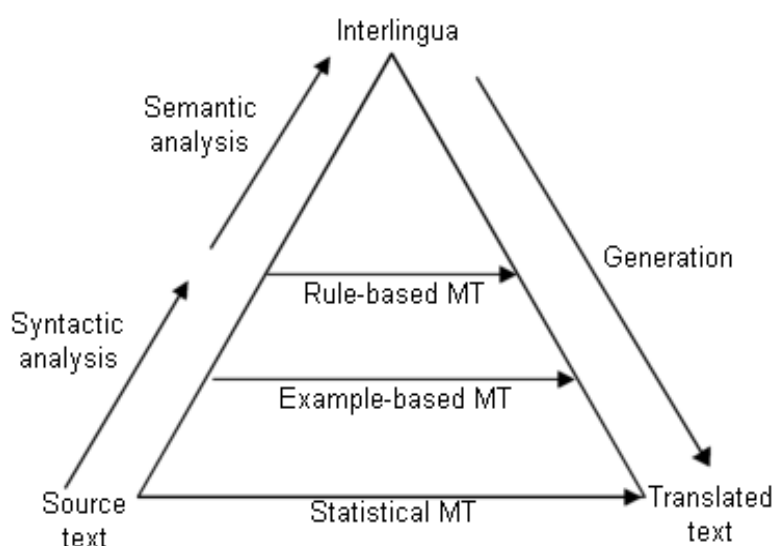


Figure 1. *The Machine Translation Pyramid, which is an indication of the level of syntactic and semantic analysis performed by various MT methods.*

Existing MT can be categorized into three fields: Rule-based Machine Translation, Example-based Machine Translation and Statistical Machine Translation.

Rule-based Machine Translation

Rule-based MT is a method that focuses on analyzing the source language by syntactic rules. It typically creates an intermediary, symbolic representation from that analysis that represents the content of the text, and then builds a translation from the intermediary representation. In the Machine Translation Pyramid (figure 1), this method is highest up of all the existing MT varieties. It can do syntactic analysis, but since semantic analysis is still an unsolved problem, there is still a need for language-specific translation steps.

Rule-based MT is one of the most popular methods for practical use. Well known rule-based systems include Systran and METEO [1]. Systran was fairly successful, being utilized for a time by both the United States Air Force and the European Union Commission. The system was never abandoned, as today it is used in Altavista's Babelfish and Google's language tools. METEO is a system developed for the purpose of translating weather forecasts from French to English, and was used by Environment Canada. It continued to serve its purpose until 2001.

The main advantage to this method of translation is that it works fast. A translation can be produced within seconds, which makes it an attractive method for the casual user. However, the translations produced by Rule-based MT tend to be of poor quality, as Rule-based MT deals poorly with ambiguity.

Example-based Machine Translation

Example-based Machine Translation operates on the philosophy that translation can be done by analogy. An Example-based Machine Translation system breaks down the source text into phrases, and translates these phrases analogous to the example translations it was trained with. New sentences are created by substituting parts of a learned sentence with parts from other learned sentences. This basic principle is explained in a paper by Nagao [2]. In the Machine Translation Pyramid (figure 1), this method is lower than Rule-based MT, because there is little in the way of analysis of the text.

There have been few commercially used example-based translation systems, but the techniques involved are still being researched. A recent proposal for an Example-based MT system was submitted by Sasaki and Murata [3].

The advantage of Example-based MT is that it can produce very high-quality translations, as long as it is applied to very domain-specific texts such as product manuals. However, once the texts become more diverse, the translation quality drops quickly.

Statistical Machine Translation

Statistical Machine Translation (SMT) is a type of MT similar to Example-based MT in the sense that it translates an input text according to what it has learned from training data. Unlike Example-based MT, SMT aims to be able to translate phrases it has not specifically seen before.

With advances in statistical modeling the translation quality of SMT systems has risen above that of alternative methods. A paper by Alshawi and Douglas shows this difference in performance [4]. However, the drawback of this method of MT is that it requires massive amounts of processing time and training material to produce a translation. This makes it unsuitable for time-critical applications. Furthermore, SMT is not very effective for language pairs that have little training data available.

In the Machine Translation Pyramid (figure 1), SMT is all the way at the bottom. This reflects the fact that SMT does not syntactically or semantically analyze the input text at all. It simply uses statistics obtained during model training to find a sequence of words it deems the best translation.

Two well-known SMT systems include Moses and Pharaoh. In addition there exists a variety of other systems that focus on components of an SMT system, such as language model trainers and decoders.

1.2 SMT, Alignment and preprocessing

This thesis will investigate preprocessing methods for SMT, in an attempt to find ways to increase the performance of this method of MT. Before we can go into the details of preprocessing we must introduce the workings of SMT.

SMT translates based on information it has trained from example translation data. This example translation data takes the form of a parallel corpus. Such a corpus consists of two texts, each of which is the translation of the other. In this work, this corpus is the Europarl corpus [5], which is freely available for the purposes of SMT research.

By statistically analyzing such parallel corpora, one can estimate the parameters for whatever statistical models one chooses to employ (see Brown et al [6] and Och and Ney [7]).

The trained statistical models are then used by the system to calculate the sentence that has the highest probability of being the translation of the input sentence. The part of the system that does this is called the decoder. Decoding is a difficult problem that is the subject of much research, but it falls outside the scope of this thesis. It is important to note, however, that the performance of the decoder is a function of the quality of the statistical models. Preprocessing on the training data can help improve model quality and by extension the performance of an SMT system.

To understand how preprocessing has an effect on the performance of an SMT system, one must understand the concept of alignment. In the remainder of this text there will be mention of two types of alignment: the sentence-level alignment and the word-level alignment.

The sentence-level alignment refers to the way the sentences in the corpus are sequenced. If a given sentence in one half of the parallel corpus is in the same sequential position as a sentence in the other half of the corpus, then those sentence are said to be a sentence pair. If the sentences in a sentence pair are translations of each other, those sentences are said to be aligned. In order to train an SMT system, one requires a parallel corpus the sentences of which are properly aligned. In the remainder of this thesis, we will assume the training data has a correct sentence-level alignment.

Following Och and Ney [8], the word-level alignment is defined as a subset of the Cartesian product of the word positions. This can be visualized by printing a sentence pair and drawing lines between the words. Such a visualization is given in figures 2 and 3. Both visualizations will be used in the remainder of this thesis.

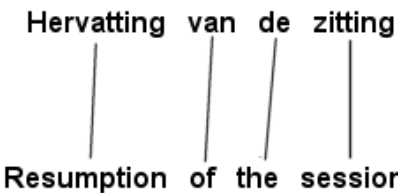


Figure 2 An example of a graphical representation of an alignment on a sentence pair. Words connected by a line are considered to be translations of each other.

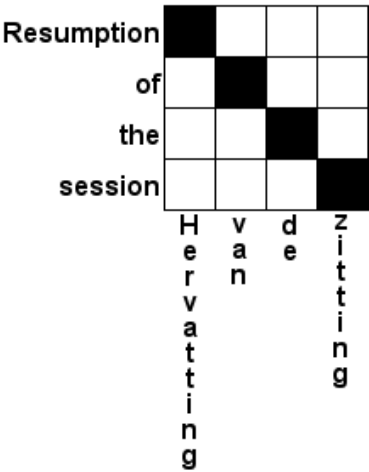


Figure 3 The same alignment as shown in figure 2, represented as a subset of the Cartesian product of the word positions.

As is explained in more detail in chapter 2, a word-level alignment is essential for training the statistical models. As training data does not contain a word-level alignment from the get-go, such an alignment must be created by the system.

Note that there is not necessarily one specific “good” alignment for any given sentence pair. When asked to create an alignment, human aligners may well come up with different alignments, and it may well be that one alignment is as good as another, depending on one’s point of view. A human will create an alignment based on meaning, while a machine cannot do this. Clearly, this means that it’s not easy to define what a “good” alignment is for a machine. In practice, the algorithm simply tries to come up

with an alignment that has a low perplexity. Perplexity is a measure of how complex an alignment is. To give a simple example where we only consider word mapping, alignments in which words are mapped to multiple other words will have a higher perplexity than alignments that have few word mappings. The example in figures 2 and 3 has exactly one word associated with each word in the source language, and therefore has a low perplexity.

Perplexity will be formally introduced in chapter 2.

When creating a word-level alignment, a lot depends on the quality of the corpus itself. Things such as spelling errors, missing words or sentences, garbage and mistranslations may negatively influence the accuracy of the alignment, which in turn may negatively affect the translation models that are trained from the alignment. To minimize these negative influences, the corpus can be adjusted prior to the alignment step. Spelling errors can be corrected or simply tokenized away. Incomplete sentence pairs can be removed, as can garbage such as punctuation or formatting codes. It is tasks such as these that are performed by preprocessing.

In addition to eliminating elements that reduce the corpus' quality, preprocessing can analyze the corpus, often on a semantic level, thereby stimulating a certain tendency in the creation of the word-level alignment. For example, number tagging can ensure that numbers will be aligned to other numbers. For detailed descriptions of preprocessing steps that are relevant in this thesis, refer to chapter 3.

Previous research into preprocessing steps includes work by Habash and Sadat [9], who investigated the effect of preprocessing steps on SMT performance for the Arabic language. Research into preprocessing for automatic evaluation of MT has also been done by Leusch et al [10].

The goals of this work are twofold.

Firstly, it investigates the impact of preprocessing on the performance of SMT training, if the preprocessing is applied to both halves of a parallel corpus. The objective is to judge whether such preprocessing steps are a useful addition to a typical training process. The preprocessing methods examined in this thesis are Stemming, Tokenization and Named Entity Recognition.

Secondly, it investigates whether the efficiency of preprocessing in the context of a bilingual corpus can be improved by making use of the bilingual corpus and the assumption that the two halves are accurate translations of each other. This research goal will focus on the Named Entity Recognition preprocessing method.

1.3 Overview of further chapters

Chapter 2 will give an outline of the SMT theory underlying the experiments. It describes the basic SMT theory, which is later used to explain how preprocessing steps can influence the performance of an SMT system.

Chapter 3 describes the theoretical foundations of the experiments performed. It introduces the techniques involved in the preprocessing steps and explains their working.

Chapter 4 is a description of the experimental setup. It lists the experiments that were performed as well as a predicted result, along with a motivation.

Chapter 5 will show the results of the experiments and provide an explanation as to what they mean and how they reflect the impact of the preprocessing steps.

Finally, Chapter 6 will conclude this thesis, review the results and give recommendations on future research.

2 Statistical machine translation

To understand why changing certain properties of a corpus can have an influence on the accuracy of an SMT system, it is important to understand how an SMT system works. An SMT system can roughly be considered as a training process and a decoding process. Because preprocessing has its effect during the training process, this chapter will focus on that and forego a detailed explanation of the decoding process.

2.1 Basic theory

MT is about finding a sentence e that is the translation of a given sentence f . The identifiers f and e originally stood for French and English because those were the languages used in various articles written on the subject (Brown et al [6], Knight [11]). This thesis deals with Dutch and English, but will adhere to the convention.

SMT considers every sentence e to be a potential translation of sentence f . Consider that translating a sentence from one language to another is not deterministic. While a typical sentence can usually only be interpreted in only one way when it comes to its meaning, the translation may be phrased in many different ways. In other words, a sentence can have multiple translations. For this reason, SMT does not, in principle, outright discard any sentence in the foreign language. Any sentence is a candidate. The trick is to determine which candidate has the highest probability of being a good translation.

For every pair of sentences (e, f) we define a probability $P(e | f)$ that e is the translation of f . We choose the sentence that is the most probable translation of f by taking the sentence e for which $P(e | f)$ is greatest. This is written as:

$$\underset{e}{\text{Argmax}} P(e | f) \quad (1)$$

SMT is essentially an implementation of the noisy channel model, in which the target language sentence is distorted by the channel into the source language sentence. The target language sentence is “recovered” by reasoning about how it came to be by the distortion of the source language sentence. As the first step, we apply Bayes’ Theorem to the formula given above. Because $P(f)$ does not influence the argmax calculation, it can be disregarded. The formula then becomes:

$$\underset{e}{\text{Argmax}} P(e | f) = \underset{e}{\text{Argmax}} P(e)P(f | e) \quad (2)$$

The highest probability that e is the translation of f has been expressed in terms of the probability of e a priori and the probability of f given e . At first glance this does not appear to be beneficial. However, the introduction of factor $P(e)$ lets us find translations that are well-formed. To understand this, remember that $P(e | f)$ is never zero – after

all, every e is a potential translation of f . It isn't zero even if e is complete gibberish. In effect, this means that some of the probability mass is given to translations that are ill-formed sentences – a sizable portion of the probability mass, in fact. The probability $P(e)$ compensates for this. It is called the *language model probability*. The language model probability can be thought of as the probability that e would occur. As gibberish is less likely to occur than coherent, well-formed sentences, $P(e)$ is higher for the latter than for the former. The probability $P(f | e)$ is called the *translation model probability*. The translation model probability is the probability that the sentence e has f as its translation. Evidently the product $P(e)P(f | e)$ will be greatest if both $P(e)$ and $P(f | e)$ are high – in other words, if f is a translation of e and if e is a good sentence.

It is especially $P(f | e)$ that is of interest for this thesis. As stated in the introduction, preprocessing affects the word-level alignment that the system creates on a corpus, and the word-level alignment is used to estimate models that are used to calculate $P(f | e)$. $P(e)$ gives statistical information about a single language, and as such is not determined from the alignment. Often, language models are trained separately from translation models, on different data.

Figure 4 is a graphical representation of the above. This figure shows a basic SMT system, including preprocessing for the translation model training.

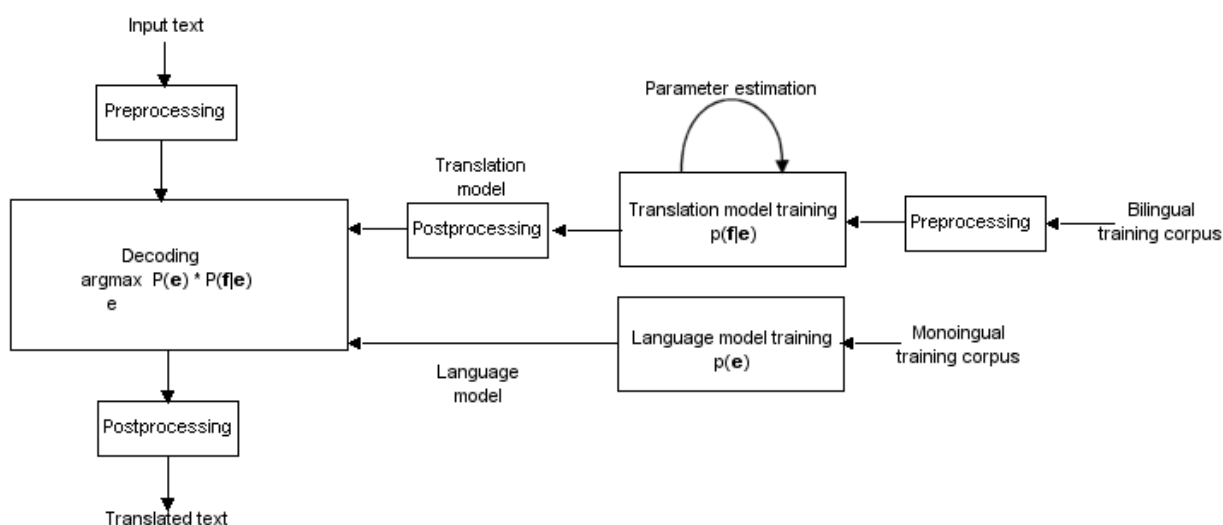


Figure 4 A more detailed schematic of an SMT system's architecture. Note that the language model is trained from separate (monolingual) training data.

As will be clear, a good SMT system requires a good language model as well as a good translation model. The remainder of this chapter describes how one may obtain such models.

2.2 Language modeling

The language model $P(e)$ is largely determined by the training corpus that was used to train the language model. The more similar the sentence is to the training data, the higher its $P(e)$ score will be.

An important part of this probability is the “well-formedness” of the sentence. A sentence that is grammatically correct is a well-formed sentence, whereas a sentence that is a mere collection of words that bear no relation to each other is ill-formed.

While well-formedness is important, it is not all that matters for the language model probability. The words used in the sentence also have their impact. If a sentence uses many uncommon words, it may be given a lower $P(e)$ score than a sentence that only uses more common words. For example, a sentence that uses the words “mausoleum”, “nanotechnology” and “comatose” together may be given a lower $P(e)$ score than a sentence that uses the words “cooking”, “house” and “evening” together. Of course, if the training corpus was largely on the subject of people being held comatose in a mausoleum by means of nanotechnology, the opposite might be true, as the former sentence would be using common words given that training corpus.

The language model can be trained by simple counting. Training requires a training corpus, preferably as large a corpus as possible, which contains sentences in the language for which the language model is being trained. From this corpus a collection of n-grams is constructed. An n-gram is a fragment of a sentence that consists of n consecutive words. For example, the sentence “Resumption of the session” contains five 2-grams: “<s> Resumption”, “Resumption of”, “of the”, “the session” and “session <s>”, where <s> and </s> indicate the absence of a word at the start and the end of the sentence, respectively.

The n-grams can then be assigned probabilities as follows:

$$P(X_n | X_0 \dots X_{n-1}) = \frac{\#(X_0 \dots X_n)}{\#(X_0 \dots X_{n-1})} \quad (3)$$

Where X is a word in the sentence, $X_0 \dots X_n$ is an n-gram and $\#$ is the number of occurrences of in the corpus. Any sentence that can be constructed from the n-grams that the system has learned can be assigned a $P(e)$ that is a function of the probabilities of its component n-grams.

However, this isn't sufficient. A sentence that cannot be built out of learned n-grams will be given a probability of zero, which means the system will not be able to generate those sentences. As training data is finite and the number of possible sentences is not, it will always be possible to construct a sentence that has one or more n-grams that do not occur in the training data, no matter how large the training corpus is. Therefore, we employ a technique called smoothing, which assigns a nonzero probability to every possible n-gram given the words in the training corpus, even those that don't actually occur. This allows the system to generate sentences that contain n-grams that weren't in the training corpus, as long as those n-grams contain known words. There are many possible approaches to smoothing, the most simple being the addition of a very small value to every n-gram that did not appear in the corpus.

There are other methods of building a language model, though the smoothed n-gram method is prevalent. These methods are outside the scope of this thesis. However, it is worth pointing out that Eck et al [12] investigated a method that expands on the n-gram method by adapting the language model to be more domain specific, thereby achieving better results in that domain.

2.3 Translation modeling

The purpose of the translation model is to indicate for a given sentence pair (e, f) the probability that f is the translation of e . It assigns a probability to each potential translation of the input sentence, and if the model is any good, better translations will have higher probabilities. Training models that will yield good probabilities is not easy, and in fact a great deal of the research done in the field of SMT is related to translation modeling.

The approach used by the SMT translation models is called string rewriting. It is described in detail by Brown et al [6]. String rewriting essentially replaces the words in a sentence with their translations, then reorders them. While string rewriting cannot explicitly map syntactic relationships between words from the source sentence to the target sentence, it is possible to approximate such a mapping statistically. The upside to this method is that it's very easy in principle, and it can be learned from available data. This means that as long as appropriate training data is available this method applies to any language pair.

In string rewriting, there are four parameters that are calculated by the translation model.

The first parameter is the amount of translated words that are associated with every source word. This is called the *fertility* of that word. For example, a word with a fertility of 3 will have 3 words associated with it as a translation of that word. The fertility for a word is not directly dependent on the other words in the sentence or their fertilities, but as the sum of all fertilities must be equal to the amount of words in the target sentence, fertilities indirectly influence each other by competing for words when estimating the fertility probabilities during model training.

The part of the translation model that decides on the fertility is called the fertility model. This model assigns to each word e_i a fertility ϕ_i with probability

$$n(\phi_i | e_i) \quad (4)$$

Secondly, the translation model decides which translation words are generated for each word in the source sentence. This is called the *translation* probability, not to be confused with the translation model probability. More formally, for each word e_i the generation model chooses k foreign words τ_{ik} with probability

$$t(\tau_{ik} | e_i) \quad (5)$$

With $1 \leq k \leq \phi_i$

Thirdly, the translation model decides the order in which these translated words are to be placed. This part of the translation model is called the *distortion* model. The distortion model chooses for each generated word τ_{ik} a position π_{ik} with probability

$$d(\pi_{ik} | i, l, m) \quad (6)$$

Where l is the amount of words in the source sentence and m is the sum of all fertilities.

Finally, the translation model causes words to be inserted spuriously. To understand this, consider that sometimes, words that appear in a translation may not be directly generated from a word in the original sentence. For example, a grammatical helper word that exists in one language may have no equivalent in the other language, and will therefore not be generated by any of the words in that language. For this reason, all sentences are assumed to have a NULL word at the start of the sentence. This NULL word can have translations like any other word, which allows words without a counterpart in the original sentence to be generated. This is called *spurious insertion*. Every time a word is generated normally in the target sentence, there is a probability that a word is generated spuriously. This probability is denoted with

$$p_1 \quad (7)$$

The probability p_0 is the probability that spurious generation does not occur, given by

$$p_0 = 1 - p_1 \quad (8)$$

In the following section we will see how these parameters can be turned into the translation model probability $P(f | e)$ that we're looking for, by means of a word-level alignment.

2.4 Alignment

An actual translation model is trained from a word-level alignment. The parameters described in the previous section can be estimated from a word-level alignment.

$n(\phi | e)$ for a certain e and ϕ can be obtained simply by checking the word-level alignment on the entire corpus, counting all the occurrences that e is aligned to exactly ϕ words in the foreign language, then dividing this count by the amount of probabilities n in the translation model.

$$n(\phi | e) = \frac{\#(\phi | e)}{\#n} \quad (9)$$

$t(\tau | e)$ for a certain τ and e can be obtained by counting how many words are generated by all occurrences of e in the alignment and then dividing the amount of τ by the total count.

$$t(\tau | e) = \frac{\#(\tau | e)}{\#(x | e)} \quad (10)$$

Where x means “any word”.

$d(\pi | i, l, m)$ for a certain π, i, l and m can be obtained by counting the occurrences of $(\pi | i, l, m)$ and dividing it by the count of all occurrences of $(j | i, l, m)$, with $j = 1 \dots m$.

$$d(\pi | i, l, m) = \frac{\#(\pi | i, l, m)}{\#(j | i, l, m)} \quad (11)$$

p_1 can be obtained by looking at the foreign corpus. This corpus consists of N words. We reason that M of these N words were generated spuriously, and that the other $N - M$ words were generated from English words. M can be obtained from the word-level alignment by counting the occurrences of translation parameter (x | "NULL"). This leads to the value for p_1 :

$$p_1 = \frac{M}{N - M} \quad (12)$$

The above shows that it is vitally important for the word-level alignment to be as accurate as possible. The ideal scenario is that every word in a sentence is aligned to a word that is a translation of that word, or if there is no translation of that word available in the translation sentence, that it not be aligned to another word at all.

In practice, prefabricated word-level alignments do not exist. It falls to the SMT system training process to estimate one from the sentence-aligned corpus. Because the trainer has no knowledge of the languages involved at all, it must determine which alignment is the best one based on patterns that exist in the corpus. For this purpose we employ the Expectation Maximization (EM) algorithm (Al-Onaizan et al [13]).

2.5 The Expectation Maximization algorithm

EM is an iterative process that attempts to find the most probable word-level alignment on all sentence pairs in the parallel corpus. It attempts to find patterns in the corpus by statistically analyzing the component sentence pairs, and considers alignments that conform to these patterns to be better than alignments that don't. This is why preprocessing on the corpus has an effect on the overall translation model quality. By modifying the corpus we modify certain patterns, with the intent to direct the EM algorithm produce a better alignment.

In creating a word-level alignment on a sentence-aligned corpus, we consider that each sentence pair has a number of alignments, not just a single one. Some of these alignments we may consider "better" than others. To reflect this, we introduce alignment weights. An alignment with a higher weight is considered better than an alignment with a lower weight. The sum of the alignment weights for all alignments on a sentence pair is equal to 1. These weights will help us estimate the translation model parameters by collecting *fractional counts* over all alignments. The basic method is the same as described at the beginning of this section, but we do it for all alignments. Furthermore, we multiply the counts by the weight of the alignment that we count the parameter from, and then add the fractional counts for a parameter together to get the final count. In this manner, we can estimate parameters even if we have more than a single alignment on a sentence pair.

The question arises where these alignment weights come from. Let us express these weights in terms of *alignment probabilities*. The probability of an alignment on a sentence pair (e, f) is the probability that the alignment would occur given that sentence pair. We write this probability as

$$P(a | e, f) \quad (13)$$

Where a is the alignment. We can use the definition of conditional probability to rewrite this probability as

$$P(a | e, f) = \frac{P(a, e, f)}{P(e, f)} \quad (14)$$

Because e is statistically independent of both f and a , we can write

$$P(a | e, f) = \frac{P(a, f | e)P(e)}{P(f | e)P(e)} \quad (15)$$

After dividing out $P(e)$ we end up with

$$P(a | e, f) = \frac{P(a, f | e)}{P(f | e)} \quad (16)$$

It is easy to see that taking the sum over a of all probabilities $P(a, f | e)$ is the same as $P(f | e)$:

$$P(f | e) = \sum_a P(a, f | e) \quad (17)$$

In other words:

$$P(a | e, f) = \frac{P(a, f | e)}{\sum_a P(a, f | e)} \quad (18)$$

Finally, $P(a, f | e)$ is calculated as follows:

$$P(a, f | e) = \binom{m - \phi_0}{\phi_0} \cdot p_0^{m-2\phi_0} \cdot p_1^{\phi_0} \cdot \prod_{i=1}^l n(\phi_i | e_i) \cdot \prod_{j=1}^m t(f_j | e_{a_j}) \cdot \prod_{i=1}^m d(j | a_j, l, m) \cdot \prod_{i=0}^l \phi_i \cdot \frac{1}{\phi_0!} \quad (19)$$

Where

- e is the source sentence
- f is the foreign sentence
- a is the alignment
- e_i is the source word in position i
- f_j is the foreign word in position j
- l is the number of words in the source sentence
- m is the number of words in the foreign sentence
- a_j is the position in the source language that connects to position j in the foreign language in alignment a
- e_{a_j} is the word in the source sentence in position a_j
- ϕ_i is the fertility for the source word in position i given alignment a
- p_1 is the probability that spurious insertion occurs
- p_0 is the probability that spurious insertion does not occur

Note that, in deducing the formula for $P(a|e, f)$, we introduced a formula for calculating $P(e|f)$ (formula 17). Remember that this is the translation model probability that we ultimately aim to establish by training the translation models.

In summary, the alignment probability $P(a|e, f)$ can be expressed in all the translation model parameters. As we already asserted, these translation model parameters can be calculated given the alignment probability. If we have one, we can compute the other. Needless to say we start out with neither, which presents a problem. This is often referred to the chicken-and-egg problem. We will need a method for bootstrapping the training, and Expectation Maximization is exactly that.

EM begins with a set of uniform parameters. Every word in the corpus will be given the same fertility, the same translation probabilities and the same distortion probabilities. With this set of parameters, alignment probabilities can be computed for every sentence pair in the corpus, as described above. From these alignments we can collect fractional counts, and with the fractional counts we can compute a new set of parameter estimates. This new set of parameters is going to be better than the one we started with, because the process takes into account the correlation data in the parallel corpus. For example, if a certain word always shows up with a certain other word in the other language, the translation parameter for those two words will get a higher count. As a result, the EM process will give a higher probability to alignments that connect those words with each other.

EM searches for an optimization of numerical data. As EM iterates it will produce alignments it considers “better”. In this context, “better” means a lower perplexity. In the introduction, perplexity was described as a measure of complexity. With the theory described in this section, we are ready for a more formal definition:

$$2^{-\log \frac{P(f|e)}{N}} \quad (20)$$

Remember that $P(f | e)$ can be expressed in terms of $P(a, f | e)$ (equation 14). N is the amount of words in the corpus. The higher $P(f | e)$ is, the lower the perplexity will be. $P(f | e)$ is higher if the parameters that make up $P(a, f | e)$ have higher values. Finally, the parameters will have higher values when their fractional counts – over the entire corpus – are high. Because simple relationships between words will show up more often than complex ones, alignments with such simple relationships will yield higher parameter values.

What EM does is find the lowest perplexity it can. Each iteration lowers perplexity. However, because perplexity is a measure over a product of the parameters, the EM algorithm is only guaranteed to find a local optimum, rather than the global optimum. The optimum it finds is partly a function of where it starts searching, or to put it in other words, what parameter values it starts with.

There are several EM algorithms imaginable. For example, there could be an EM algorithm that simplifies the translation model by ignoring fertility probabilities, probabilities for spurious insertion and distortion probabilities. This EM algorithm will only optimize perplexity in terms of the translation probability parameter. As there is only one factor to optimize, this EM algorithm will be guaranteed to find the global optimum for its perplexity. This EM algorithm exists, and it is the EM algorithm used in IBM Model 1.

2.6 GIZA++

In practice the alignment is generated by a program called GIZA++. GIZA++ is an extension of GIZA, which is an implementation of several IBM translation models. In addition to the IBM models, GIZA++ also implements Hidden Markov Models (HMMs). By request this thesis acknowledges Franz Josef Och and Hermann Ney for GIZA++. The theory of their implementation is described in [8].

GIZA++ produces a word-level alignment on a sentence aligned parallel corpus. GIZA++ will produce a one-to-many alignment, in which words in the “target” sentence may only be aligned to a single word in the “source” sentence. This is illustrated in figure 5.



Figure 5. Two one-to-many alignments, one for English-Dutch and one for Dutch-English. Note that these alignments are not optimal. Some errors exist, such as the alignment of "naar" to "be".

To achieve a many-to-many alignment from GIZA++ it is necessary to produce two one-to-many alignments, one for each translation direction, and combine them into a single many-to-many alignment. This process is referred to as symmetrization. There are two methods of symmetrization used in these experiments: Union and Intersection symmetrization.

Union symmetrization assumes that any alignment the two one-to-many alignments do not agree on should be included in the many-to-many alignment. Formally:

$$A_{MTM} = A_{OTM1} \cup A_{OTM2} \quad (21)$$

Intersection symmetrization assumes that any alignment the two one-to-many alignments do not agree on should be discarded. Words that no longer have any alignment after symmetrization are aligned to NULL. Formally:

$$A_{MTM} = A_{OTM1} \cap A_{OTM2} \quad (22)$$

The Union and Intersection many-to-many alignments for the two one-to-many alignments given in figure 5 are shown in figure 6.



Figure 6. Two many-to-many alignments created from the one-to-many alignments in figure 5. The upper figure is the Union alignment and the lower figure is the Intersection alignment.

2.7 Parameter estimation

When optimum perplexity has been achieved, the alignment with the highest probability is called the Viterbi alignment. The Viterbi alignment found by Model 1 when starting off with uniform parameter values may be a very bad alignment. For example, all words could be connected to the same translation word. Model 1 has no way of knowing that this is not a probable alignment, because it ignores all the parameters that show this improbability, such as the fertility parameter. However, the Model 1 Viterbi alignment can be used as the starting position for more complex EM algorithms. More complex algorithms have more parameters that weigh into the perplexity, and they are not guaranteed to find a global optimum. From the most probable alignment given a local optimum we can get a new set of parameters. This new set of parameters can then be fed back to Model 1, which may find a new Viterbi alignment as a result of its new starting parameters.

This last part is an important aspect of translation model training. By using the parameters from a training iteration of one model, we can start a new training iteration, with that same model or with a different one, which will hopefully yield improved parameter values. This process is called *parameter estimation*. A simple, schematic representation of this process is given in figure 7.

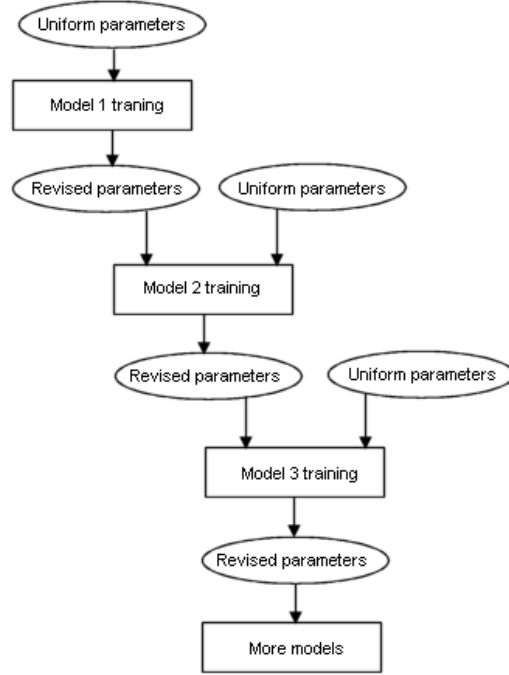


Figure 7. A schematic representation of the parameter estimation process. The models can each be trained a number of times, taking the results from the previous iteration as the starting point for the new iteration. When a model estimates parameters that were not estimated by a previous model, it starts the first training iteration with uniform values for those parameters.

There is one practical problem with starting the parameter estimation process. Recall that $P(a | e, f)$ can be expressed in terms of $P(a, f | e)$ (see formula 18). In the Model 1 EM algorithm, the denominator of that formula, $\sum_a P(a, f | e)$, can be written as

$$\sum_a \prod_{j=1}^m t(f_j | e_{aj}) \quad (23)$$

As is implied by this formula, the EM algorithm needs to enumerate over every alignment in the corpus. In a corpus of N words in language e and M words in language f , the amount of alignments is equal to

$$(N + 1)^M \quad (24)$$

To illustrate, in a single sentence pair with 20 words in each sentence, the amount of alignments is $2.78 \cdot 10^{26}$. For a corpus with 120,000 sentence pairs the amount of alignments is astronomical, and enumerating all of them is impractical. Fortunately, we can optimize the enumeration process.

As formula 21 sums over a product that contains independent elements, we can factor out the elements independent to the product and take the product over the independent element of the sum of the factored expression:

$$\prod_{j=1}^m \sum_{i=0}^l t(f_j | e_i) \quad (25)$$

With this formula, the amount of alignments to be enumerated to get the fractional counts for all alignments is equal to

$$N + 1 \bullet M \quad (26)$$

This formula has a quadratic order of magnitude, whereas formula 22 has an exponential order. To illustrate, consider again the single sentence pair with 20 words in each sentence. The amount of alignments to enumerate is now only 420.

The above means that we can enumerate all the alignments for IBM Model 1 within reasonable time, and therefore find the Viterbi alignment. The same is true for IBM Model 2, which is like Model 1, but handles distortion probabilities as well as translation probabilities. Unfortunately, this manner of simplification cannot be performed for complex models like IBM Model 3, and so we cannot find their Viterbi alignment in reasonable time. However, there is a technique called *hill climbing* that can be used to find the (local) optimum for such models. Hill climbing takes for every sentence pair a single alignment to start with. A good place to start would be the Model 2 Viterbi alignment. The model then makes a small change to the alignment, for example by moving a connection from one position to another position close by. Then the model computes the perplexity for the new alignment. This can be done fairly quickly with formula 19. If the new alignment is worse, it is discarded. If it is better, it replaces the old alignment. The model repeats this process until no better alignment can be found by making a small change. The alignment the model ends up with is considered the Viterbi alignment for this model, even though there is no guarantee that the alignment is, in fact the best alignment given the current parameter values. From this “Viterbi” alignment and a small set of alignments that are “close” to it we can collect fractional counts and estimate a new set of parameters. This new set of parameters can be used as the starting point for a new training iteration or, if no more training is deemed necessary, to calculate the final $P(f | e)$.

There are more models than IBM Models 1, 2, and 3, such as the models presented by Och and Ney [7]. However, for the purpose of preprocessing it is not necessary to enumerate and explain each of these models.

In summary, this chapter shows that Statistical Machine Translation is about calculating the highest probability that a given sentence is the translation of another sentence. This is done by analyzing monolingual data to obtain language models and bilingual data to obtain translation models. Translation model training is by far the most complex task, because it requires a word-level alignment, which has to be estimated from a sentence-level alignment by analyzing patterns. The EM algorithm is used for this analysis. By modifying the patterns we influence the EM algorithm, and by extension the word-level alignment and the translation model.

3 Preprocessing

Preprocessing is literally to process something before it is processed by something else. In computer science a preprocessor is a program that processes its input data to produce output that is used as input to another program. In the specific context of these experiments the output data of the preprocessors serves as the input data for GIZA++. This chapter gives an overview of the preprocessing techniques used in the experiments.

3.1 Tokenization and sentence alignment

Following Webster and Kit [14], tokenization is defined as a type of preprocessing that decomposes parts of a given text into more basic units. An example of tokenization on English is decomposing the contraction “it’s” into “it is”.

The tokenization employed in these experiments is limited to the removal of punctuation and words that do not bear any semantic significance, such as corpus markup.

Tokenization on punctuation is a trivial task in itself. However the scripts that are included with the Europarl corpus take a slightly more involved approach, making use of a list on non-breaking prefixes. These non-breaking prefixes indicate words that do not mark the end of a sentence when encountered with a period. The list is used not only in tokenization but also for sentence alignment, and later it will be used during Named Entity Recognition as well. Europarl includes a list of non-breaking prefixes for English, but not for Dutch.

Although a Dutch list is not required to remove punctuation on the corpus, the list can also be used during the sentence-alignment of the text. If no suitable list is found for a language, the sentence alignment script falls back to English. This can result in a bad sentence alignment, which is effectively useless as a base to train statistical translation models.

A list of Dutch non-breaking prefixes is shown in Appendix A.

3.2 Stemming

Stemming is the process of reducing a word to its stem. In linguistics a stem is the part of a word that is common to all its inflected variants. This is also called the morphological root. The morphological root is the primary lexical unit of a word, which carries the most significant aspects of semantic content and cannot be reduced into smaller constituents. Any word in natural languages such as Dutch and English can be considered to be composed of a stem, optionally inflected by an affix (be it a prefix, a suffix or a circumfix).

In the context of Natural Language Processing, a word's stem is taken to be the first part of that word, stripping off any trailing letters that might constitute inflections, conjugations or other modifiers to the word. It is important to note that the stem thus generated is not necessarily the same as the morphological root of that word. In stemming for NLP, it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root.

Example 8 illustrates the result of stemming on a small selection of words.

consist consisted consistency consistent consistently consisting consists	consist consist consist consist consist consist consist consist
knock knocked	knock knock
knocker knockers	knocker knocker

Example 8. *Examples of stemming applied to a few English words and their variations.*

The stemming algorithm used in most NLP related tasks is the Porter stemmer, or a stemmer derived from the Porter stemmer. The stemmer used in this experiment is no exception; the stemmer used is the Porter2 stemmer, which is a slightly improved version of the Porter stemmer. The Porter stemming algorithm is described by Porter et al [15].

Because stemming directly changes the corpus, it requires a postprocessing step to restore the words in the corpus to their original forms once the alignment has been computed, as estimating the translation model parameters from a stemmed corpus would not result in a very good SMT system.

3.3 Named Entity Recognition using Conditional Random Fields

Named Entity Recognition (NER) is a subtask of information extraction that seeks to locate and classify Named Entities (NEs), which are expressions that refer to the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.

NER has been a topic of research for years on conferences and in workshops, most notably the Message Understanding Conference (MUC), the Conference on Natural Language Learning (CoNLL) and the Multilingual Entity Tasks (MET). One of the methods evaluated on these conferences and workshops is NER by means of Conditional Random Fields (CRFs) (Lafferty et al, [16]).

CRFs are conditional probabilistic models not unlike HMMs for labeling or segmenting sequential data, such as a plaintext corpus. One of the problems with labeling sequential data is that the data often cannot be interpreted as independent units. For example, an English sentence is bound by grammatical rules that impose long-range relationships between the words in the sentence. This makes enumerating all observation sequences intractable, which in turn means that a joint probability distribution over the observation and label sequences, as would be the case in an HMM, cannot be calculated in reasonable time. On the other hand, making unwarranted independence assumptions about the observation sequences is not desirable either. A solution to this problem is to define a conditional probability given a particular observation sequence rather than a joint probability distribution over two random variables.

Let X be a random variable that ranges over observable sequences of words, and let Y be a random variable that ranges over the corresponding sequences of labels, in this case named entity tags. A CRF defines a conditional probability $P(Y|x)$ given a particular observed sequence of words x . The model attempts to find the maximal probability $P(y|x)$ for a particular label sequence y . Let $G = (V, E)$ be an undirected graph such that there is a node $v \in V$ corresponding to each of the random variables representing an element Y_v of Y . If each random variable Y_v obeys the Markov property with respect to G , then (Y, X) is a CRF. The Markov property entails that the model is memoryless, meaning each next state in the model depends solely on its previous state.

While in theory G may have any structure, in practice it always takes the form of a simple first-order chain. Figure 9 illustrates this.

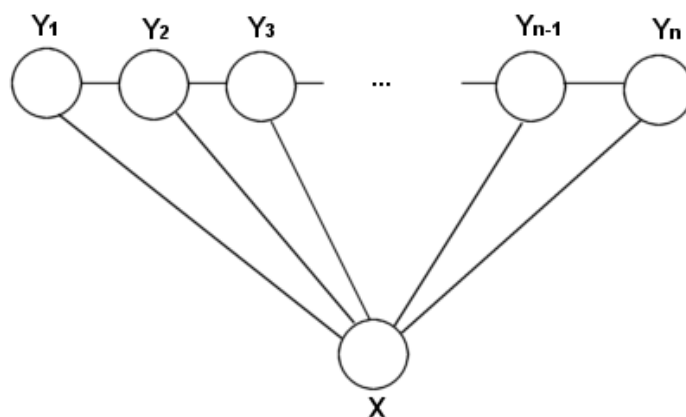


Figure 9. A simple first-order chain CRF architecture. The states at the top are the label sequences generated by the model, while the state X at the bottom represents the observed data sequences. The states Y_n are only dependent on their neighboring states, thereby satisfying the Markov property.

Lafferty et al. [16] define the probability of a particular label sequence y given observation sequence x to be a normalized product of potential functions, each of the form

$$\exp(\sum_j \lambda_j t_j(y_{i-1}, y_i, x, i) + \sum_k \mu_k s_k(y_i, x, i)) \quad (27)$$

Where $t_j(y_{i-1}, y_i, x, i)$ is a transition feature function of the entire observation sequence and the labels at positions i and $i-1$ in the label sequence, and $s_k(y_i, x, i)$ is a state feature function of the label at position i and the observation sequence. λ_j and μ_k are parameters that are obtained from the training data. These feature functions take on the values of one of a number of real-valued features. Such features are conditions on the observation sequence that may or may not be satisfied. For example, a feature $b(x, i)$ could be

$$b(x, i) = \begin{cases} 1, & \text{if the word at position } i \text{ starts with a capital letter} \\ 0, & \text{otherwise} \end{cases} \quad (28)$$

A state feature function for this feature could be

$$s_k(y_i, x, i) = \begin{cases} b(x, i), & \text{if } y_i = \text{IN} \\ 0, & \text{otherwise} \end{cases} \quad (29)$$

Similarly, a transition feature function $t_j(y_{i-1}, y_i, x, i)$ is defined on two states and a feature.

With such features and feature functions, we can write the probability of a label sequence y given an observed sequence x as

$$P(y | x, \lambda) = \frac{1}{Z(x)} \exp(\sum_j \lambda_j F_j(y, x)) \quad (30)$$

Where $\frac{1}{Z(x)}$ is a normalization factor and $F_j(y, x)$ is the sum over all feature functions, both state and transition.

Given this probability we can calculate the maximal probability by maximizing the logarithm of the likelihood, given by:

$$\Lambda(\lambda) = \sum_k \left[\log \frac{1}{Z(x^{(k)})} + \sum_j \lambda_j F_f(y^{(k)}, x^{(k)}) \right] \quad (31)$$

Because this function is defined on the probabilities of local label sequences, we can ensure we get the general label sequence with the highest probability by maximizing this function. It is concave, which ensures convergence to a global maximum.

An NER system that implements CRFs is the Stanford Named Entity Recognizer (Rose Finkel et al [17]), which scored 86.86 F-score on the CoNLL2002 shared task and 92.29 on the CMU Seminar Announcement dataset (these are the highest overall scores only). This system will be used in the CRF NER experiments described in this thesis.

3.4 NER for SMT

An NE is special from the perspective of SMT in the sense that each NE can only have exactly one translation, no more and no less. Though some entities may have more than one name, they are typically indicated with only one of the possible names throughout the use of the language. For example, In English the Belgian town of Bruges is always referred to by that name, though in Dutch the town is invariably referred to by its Dutch name, Brugge. We can say that in this specific example, Bruges in English is the correct translation of Brugge in Dutch, and that any other translation is wrong.

Linguistically speaking NER is aimed towards finding and classifying exactly those words that are indeed NEs, and no others. In tasks such as information retrieval it is important to achieve a high precision and recall under those constraints, because of the semantic significance of the words. In the context of training an SMT system, however, there is a different consideration that comes into play.

NER is not a means for information extraction but for exerting influence over the EM algorithm. An NER algorithm that scores very badly, but can be shown to improve the quality of the word-level alignment is still an effective algorithm, even if it is uninteresting for the NER task itself. In an ideal world a word tagging algorithm would be able to recognize which words are translations of each other in all cases. If such an algorithm existed there would effectively no longer be a need for the EM algorithm. Clearly such an algorithm is not realistic, but by applying NER an attempt is made to perform a small portion of this task.

3.5 NER algorithms for bilingual data

Sentence pair based lexical similarity

This algorithm tags words as NEs by checking for a lexically similar word in the translation sentence. For each word it calculates the maximum lexical similarity score for that word given all the words in the translation. Lexical similarity is a measure of how closely two words resemble each other. The words “Parlement” and “Parliament”, for example, are lexically quite similar because they have many letters in common. In the experiments, lexical similarity is defined as

$$\text{similarity} = 1 - \frac{D}{\max(N_1, N_2)} \quad (32)$$

Where D is the Levenshtein distance (Navarro, [18]) between the two words and N_1 and N_2 are the lengths of the words.

The Levenshtein distance is a measure about how different the words are in terms of how many elementary operations to one word need to be preformed to obtain the other word. An elementary operation is either the substitution of any letter with a different letter, the addition of a letter at any position in the word or the removal of any letter in the word.

In equation 30, the Levenshtein distance is normalized by the length of the longer of the two words. The reason for this is that the Levenshtein distance only indicates the absolute difference between two sequences. For example, the words “European” and “Europa” have a Levenshtein distance of 2, because it takes 2 elementary operations to obtain one from the other. However, the same can be said for the words “of” and “in”, which are clearly not very lexically similar. By normalizing for word length we obtain a score that gives an indication of how many letters the two words have in common relative to how many are different. An example of this algorithm is shown in example 10.

English: One of the people assassinated very recently in Sri Lanka was Mr Kumar Ponnambalam, who had visited the European Parliament just a few months ago.	
Dutch: Een van de mensen die zeer recent in Sri Lanka is vermoord, is de heer Kumar Ponnambalam, die een paar maanden geleden nog een bezoek bracht aan het Europees Parlement.	
recently	recent
Sri	Sri
Lanka	Lanka
Kumar	Kumar
Ponnambalam	Ponnambalam
European	Europees
Parliament	Parlement

Example 10. A sentence pair and its NEs, recognized by lexical similarity, given a similarity threshold of 0.4.

While all NEs in this example have been successfully recognized, note how “recently” and “recent” were also recognized as NEs because they are sufficiently similar. This shows that judging NEs by lexical similarity is prone to false positives.

Corpus based lexical similarity

Corpus based lexical similarity is like sentence based lexical similarity, but it attempts to avoid some of the problems inherent to the sentence based approach.

To combat false positives, the algorithm imposes two conditions on the word pairs. Firstly, a matching word pair must occur at least a certain amount of times in the corpus. Singletons are statistically insignificant, and are even likely to decrease the alignment accuracy. Secondly, the algorithm calculates an occurrence score for each matching word pair. Even if a matching word pair is found, this pair cannot be considered a NE if the component words appear unpaired too often in the text. The NER algorithm will not consider a matching word pair to be a NE pair if its occurrence score is too low. Formally, for every unique word w and for all sentence pairs (i, j) in the corpus the occurrence score for that word is calculated as:

$$score(w_i) = \frac{2 \cdot \sum_{(i,j)} \min(|w_i|, |w_j|)}{\sum_i |w_i| + \sum_j |w_j|} \quad (33)$$

Example 11 shows the recognized NEs in the sentence pair from example 10.

Sri	Sri
Lanka	Lanka
Kumar	Kumar
Ponnambalam	Ponnambalam
European	Europees

Example 11. *Lexical similarity NER using minimum occurrence and minimum length constraints.*

Note that “Parlement” and “Parliament” are now not recognized as NEs. This can be remedied by reducing the similarity threshold.

Re-classifying unclassified tags

Most of the tagging algorithms described here do not classify the NEs, as the classification is not immediately relevant to the word level alignment. However, classification can help establish a correct alignment.

The re-classification algorithm compares two unclassified tagged corpora to their respective untagged versions and classifies the tags according to the lexical similarity score of the tagged words only. A tagged word that has a lexically identical counterpart in the translation sentence will be marked as <NE_VERBATIM>, a tagged word that is not lexically identical yet lexically similar above a given threshold will be marked as <NE_SIMILAR> and all tagged words that have no lexically similar counterparts will remain unclassified. The expectation is that such a re-classification will lead to a better Alignment Error Rate while not reducing the NER performance.

3.6 Uppercase and lowercase

The sentences in the Europarl corpus are punctuated and capitalized. While punctuation is only markup that benefits a human reader, capitalization can be used to analyze the text from a MT point of view. For example, capitalization might be used to identify named entities, or to facilitate part-of-speech tagging. However, because capitals are also used at the beginning of sentences and sentences can begin with practically any word, a capitalized corpus is inherently more complex than a corpus with the capitals removed.

Capitalization typically is highly language specific. For one thing, it is limited to languages that use the alphabet. And even amongst alphabet-based languages there are often differences. In English it is customary to capitalize the names of days and months, as well as the first person pronoun “I”, while in Dutch this is not the case. In German every noun is capitalized.

Because capitals carry information that can be useful for Named Entity Recognition, the NER experiments were performed on a corpus with the capitals intact. However, because capitals carry no information in the context of stemming – in fact they counter the intended effect of stemming – the stemming experiments were performed on a lowercase corpus.

4 Experiments and Evaluation

Chapters 2 and 3 provided the theoretical background of an SMT system and (some types of) preprocessing. This chapter describes the data used for the experiments, how the various preprocessing techniques are applied to this data, what the expected effect is on the alignment and how the performance is evaluated.

4.1 The Europarl parallel corpus

The experiments in this thesis were performed on a parallel corpus called Europarl (Koehn, [19]). The corpus is extracted from the proceedings of the European Parliament from April 1996 to December 2001. Most of the sentences are the sentences spoken by the politicians in the Parliament, translated if the speaker was speaking in a language other than the language used in the corpus. The translation was created manually by teams of freelance translators employed by the European Parliament. The sentences cover political debates and procedural conversation between the politicians and the president of the Parliament. In addition, the corpus contains descriptive sentences that indicate non-verbal action taken by the Parliament at large.

The corpus contains about 20 million words in 743,880 sentences per language. The corpus is available in a sentence aligned format.

4.2 Alignment Error Rate

The effectiveness of a preprocessing step can be expressed as the improvement in the alignment that it causes.

To determine how “good” any generated alignment actually is, a point of reference is needed. For this purpose, a reference alignment must be created. This reference alignment represents the “correct” alignment. The alignment created by the system should ideally be identical to this reference alignment. Because the reference alignment must be made manually, it is relatively costly to produce. For that reason, only a small portion of the corpus is used for the reference alignment.

The accuracy of the generated alignments is the degree in which they differ from the reference alignment. This is called the Alignment Error Rate. The Alignment Error Rate is the most important evaluation measure used in this thesis.

210 sentences of the first part of the Europarl corpus have been aligned manually in many-to-many format. This is the reference that will be used to calculate the Alignment Error Rate.

The Alignment Error Rate is a measure of how similar two alignments are to each other. The definition of the AER is given by Och and Ney [8]:

$$AER(S, P; A) = 1 - \frac{|A \cap S| + |A \cap P|}{|A| + |S|} \quad (34)$$

Where S is the Sure alignment, P is the Possible alignment and A is the generated alignment for which the AER is being calculated. A Sure alignment is an alignment that is always considered to be correct. A Possible alignment is an alignment that may be correct. Possible alignments can be used to resolve differences in opinion between multiple human aligners. An example of an alignment that combines Sure and Possible alignments is shown in figure 12.

That												
is												
why												
my												
Group												
moves												
that												
this												
item												
be												
taken												
off												
the												
agenda												
	D	v	m	f	d	p	v	d	a	a	t	v
	a	r	i	r	i	u	a	e	g	f	e	o
	r	a	j	a	t	n	n	a	e	n	r	e
	o	a	n	c	t	t	a	a	a	a	e	n
	m	g	e	t	e	e	n	d	a	n	n	n

Figure 12. A sentence pair that was aligned using Sure and Possible alignments. The black fields are the Sure alignment and the grey fields are the Possible alignment. While the Possible alignment may seem to be largely in error, it could have been established like this because some human aligners chose to align certain sequences to other sequences, where other human aligners did not.

The manual alignment used as the reference for the experiments described here only includes Sure alignments, no Possible ones, due limited time and resources. This means that for the purpose of calculating the Alignment Error Rate in this instance, formula 25 should be rewritten as

$$\text{AER}(S; A) = 1 - \frac{2 \cdot |A \cap S|}{|A| + |S|} \quad (35)$$

Some sentence pairs provide a poor match on the semantic level (and as a result have few words that can be aligned). These sentences contain many words that are aligned to NULL. The AER score for a standard training on these sentences is likely to be low, but the score could increase as a result of preprocessing, because words are properly aligned to NULL.

4.3 Precision, recall and F₁-score

Precision and recall are standard evaluation measures used for many NLP related tasks. In the context of NER, precision is the portion of the words recognized that are in fact NEs. Recall is the portion of NEs in the text that was successfully recognized. The F_1 -score is a measure that combines precision and recall into a single scalar score.

Precision is defined as

$$\text{precision} = \frac{\#(M \wedge A)}{\#A} \quad (36)$$

Where M denotes an NE that was manually tagged and A denotes an NE that was automatically tagged. Classification, if present, is ignored.

Recall is defined as

$$\text{recall} = \frac{\#(M \wedge A)}{\#M} \quad (37)$$

The F_1 -score is defined as

$$F_1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad (38)$$

Precision and recall are a good measure to gain insight in the performance of the NER algorithms themselves. However, it is important to note that in the context of these experiments neither precision nor recall are the most important evaluation measure. The AER is the evaluation measure that matters most.

4.4 Basic Experiments

The following is a list of the preprocessing experiments and the expected effect. The experiments in this section are strictly limited to preprocessing steps that apply to monolingual data.

Tokenization

This is a GIZA++ training on the corpus after tokenization for punctuation. No other tokenization was applied to the corpus. The AER score for this training is considered the baseline. If an AER score is said to improve or worsen, it does so relative to the AER for this training.

Stemming

In this experiment the porter2 stemmer described in section 3.2 was applied to both halves of the parallel corpus. The stemmed corpus was then used to train an alignment, following which a postprocessing step was performed to restore the corpus to its original state.

As stemming reduces the amount of unique words in a corpus, it reduces the complexity of the alignment training task, in the sense that the EM algorithm no longer has to distinguish between morphological variants of words. This will result in a stronger bias towards alignments that align words that were previously morphological variants, which in turn reduces the perplexity of the alignment.

Simple NER

The simple algorithm considers any word that starts with a capital letter an NE, as long as it isn't at the start of a sentence. It judges the latter condition by checking if the word is the first in the input, and subsequently by checking for a period at the end of the preceding word, excepting those words listed in the list of non-breaking prefixes.

As the name suggests this algorithm is unsophisticated and will likely yield a low performance in terms of precision and recall as well as AER improvement.

Conditional Random Field NER

This is an experiment with a corpus annotated by the CRF NER algorithm. The corpus was annotated using the standard classifier for the Stanford CRF NER implementation, which uses three tag types: <PERSON>, <ORGANIZATION> and <LOCATION>.

Because the algorithm tags the NEs with three different kinds of tags and does so independently on both halves of the parallel corpus, it is very likely that a pair of NEs that are meant to be aligned to each other get different classifications. In these cases the NEs will not be aligned to each other because the EM algorithm will not favor alignments that match up different tags. Therefore it is likely that this kind of NER will not yield a large improvement in AER, and may even cause the AER score to worsen.

4.5 Experiments on bilingual data

These experiments attempt to improve on the performance of the experiments described in section 4.4 by using bilingual information.

NER with sentence based lexical similarity

This experiment uses the lexical similarity algorithm described in section 3.5. Each word in a given sentence pair is examined for lexical similarity with all words in the other sentence. If a word pair is found which is lexically similar to a sufficient degree, specified by a similarity threshold, that pair is considered an NE pair and will be tagged. In addition a minimum word length may be specified. Words that are smaller than the minimum specified word length will never be considered as part of an NE pair. The minimum word length is a measure to avoid false positives among very short words. Precision may be increased by increasing this parameter, but recall is likely to suffer as a result.

However even by doing so, it is expected that too many non-NE words will be falsely tagged. The AER may not improve much, or even worsen as a result.

NER with corpus based lexical similarity

This is an experiment using the lexical similarity algorithm described in section 3.5, which takes into account occurrence information of word pairs in the corpus. The algorithm tags words based on a lexical similarity threshold as well as an occurrence threshold. The occurrence threshold is a measure to improve precision by disqualifying suspected false positives.

This algorithm is expected to perform substantially better than the sentence based approach in terms of precision. Recall, however, is expected to be roughly the same because the lexical similarity algorithm that judges NEs is identical.

Reclassification of unclassified NER

By reclassifying unclassified NE tags, an attempt is made to ensure that NEs that look the same will always be aligned to one another, rather than unrelated NEs that may be present in the sentence pair.

Because this algorithm does not tag untagged words, the impact on precision and recall is nil. By the same token the alignment generated on the untagged words is unlikely to be affected. The only gain to be had is in the alignment among the tagged words. Therefore, it is expected that AER will improve, albeit to a modest degree.

5 Experimental Results

5.1 Tokenization

Table 13 shows the result of the tokenization step, in terms of the amount of unique words encountered in the corpus and the average amount of unique words on each line.

	Amount of unique words in the entire corpus	Average amount of unique words per sentence
Normal English corpus	109,491	21.33
Tokenized English corpus	55,397	21.18
Normal Dutch corpus	168,007	21.46
Tokenized Dutch corpus	100,713	21.39

Table 13. *Unique word count across the corpus and on average per sentence before and after tokenization for punctuation.*

This table shows a dramatic decrease in the amount of unique words in the corpus. The effect is that the complexity of the alignment task is also reduced, showing a 50% reduction in the amount of unique Dutch words and a 40% reduction of unique English words.

Table 14 shows the effect the tokenization step had on alignment accuracy.

	Intersection	Union
Normal corpus	38.80	39.74
Tokenized corpus	30.64	33.29

Table 14. *AER scores in % for a non-tokenized and a tokenized corpus.*

Clearly, the reduction of the corpus' complexity has significantly improved the alignment accuracy.

5.2 Named Entity Recognition

Table 15 shows precision, recall and F_1 -scores for the NER tagging experiments. The sentence based lexical similarity was judged at 70% similarity or more with a minimum word length of 2. The corpus based lexical similarity experiment used the same similarity threshold and also enforced an occurrence threshold of 75%.

	Precision	Recall	F_1 -score
CRF English	0.77	0.74	0.76
CRF Dutch	0.73	0.53	0.61
Simple English	0.72	1.00	0.84
Simple Dutch	0.67	0.84	0.75
Sentence based LexSim English	0.32	0.56	0.41
Sentence based LexSim Dutch	0.34	0.66	0.45
Corpus based LexSim English	0.66	0.42	0.52
Corpus based LexSim Dutch	0.73	0.48	0.58

Table 15. Precision, recall and F_1 -score for the various NER algorithms.

Precision is low in general. In the case of CRF, this is mostly because the algorithm tags sequences of words, and considers the entire sequence part of the same NE. In these sequences there are often non-NE words which are considered false positives during evaluation. An example of this is “the European Parliament”, which is tagged as an organization as a whole, including the article “the”. The lexical similarity algorithm suffers badly from the similarity between languages. Frequently words that look the same but have nothing to do with each other will be tagged as an NE pair, as is the case with “near” and “naar”. The corpus-based algorithm eliminates a lot of the false positives, but not enough to rival the performance of the CRF algorithm. In addition recall considerable suffers compared to the sentence-based approach.

Table 16 shows the AER scores for the different NER algorithms.

	Intersection	Union
Baseline	30.64	33.29
Simple	42.42	41.37
CRF	34.24	35.95
Uniform CRF	35.35	36.19
Sentence based LexSim	33.12	35.33
Corpus based LexSim	31.18	33.70
Reclassified Simple	42.23	41.18
Reclassified CRF	34.57	36.00

Table 16. AER scores for the NER experiments.

The best alignment was in fact the baseline alignment that was obtained without the use of NER. Only corpus based lexical similarity achieves an AER score that approaches the baseline score. Simple AER scores worse than even the non-tokenized corpus.

5.3 Stemming

The stemming preprocessing step reduced the amount of unique words in the corpus. Table 13 shows the total amount of unique words in the English and Dutch corpora, as well as the average amount of unique words per sentence.

	Amount of unique words in the entire corpus	Average amount of unique words per sentence
Normal English corpus	55,397	21.18
Stemmed English corpus	38,920	21.08
Normal Dutch corpus	100,713	21.39
Stemmed Dutch corpus	93,219	21.36

Table 17. Unique word count across the corpus and on average per sentence, before and after stemming.

As is immediately apparent, the amount of unique words in the entire corpus has been reduced significantly while the average amount of unique words in the individual sentences has not. Because the perplexity of the individual sentence has been mostly preserved, the alignment accuracy of the stemmed corpus should not decrease.

This alignment was made after stemming preprocessing was applied as described in section 4.3. Table 14 shows the AER scores for the corpus prior to and after stemming.

	Intersection	Union
AER on normal corpus	30.64	33.29
AER on preprocessed corpus	29.49	31.90

Table 18. AER scores for stemming preprocessing

6 Review and Conclusions

6.1 Preprocessing effectiveness

One of the goals of this work was to examine the general effectiveness of different kinds of preprocessing on the training process of an SMT system.

Tokenization and stemming are two forms of preprocessing that have considerable effect. The reason for this can be found in tables 13 and 17. Because the amount of unique words is drastically reduced the EM algorithm has an easier task. This is especially true for words that have punctuation affixed, as those words will be considered different from words that are not punctuated, even if those words are identical otherwise.

With stemming this effect is enhanced because inflected word variants are eliminated. Because stemming only removes word variants to a common stem the complexity of the corpus decreases. Because the amount of unique words per sentence remains almost the same we can conclude that the EM algorithm has no reason to confuse words, while at the same time having a better base to judge the alignment probabilities.

Named Entity Recognition did not fare well across the board. It is immediately apparent from table 16 that sentence based lexical similarity performs dramatically poorly, with both precision and recall being disappointingly low. Corpus based lexical similarity performs better, but still well below par. Simple NER tagging, however, outperforms even Conditional Random Fields in terms of the F_1 -score.

The most noticeable aspect of the results for NER is that none of the NER algorithms decreased the AER score, which means every algorithm introduced more errors than it solved. This effect can be attributed to a precision problem. As shown in table 15, the precision scores for all algorithms are below 80%. The false positives significantly distort the EM algorithm's process, resulting in a worse alignment.

The simple algorithm scores very poorly in terms of AER because words in English and in Dutch aren't capitalized in the same way. In English, many significant words such as names are capitalized, while in Dutch they often are not. Therefore, the simple algorithm will tag a significant number of words in the English corpus while not tagging their corresponding translations in the Dutch corpus. The precision and recall scores for the simple algorithm support this observation. This tagging behavior causes the EM algorithm to prefer alignments that do NOT align the words in question to their translations, which hurts the AER score. This experiment clearly shows that the simple algorithm is extremely sensitive to the language on which it is performed. While in Dutch the algorithm has low recall and low precision, it is likely to yield a high recall by a very low precision on a language like German, which capitalizes all nouns.

In the case of the lexical similarity algorithm the result is highly dependent on the language pair on which it is performed. While it is theoretically sound, it is prone to incorrectly tagging word pairs that just happen to be lexically similar while not being translations of each other. This effect is likely to be strong between language pairs that bear similarity to each other, as is the case with Germanic and Romantic languages. Arguably it will be less pronounced between lexically different languages, such as Germanic languages versus East-European languages. However, because lexical similarity is based on the alphabet, it will be useless on languages that do not make use of the alphabet, such as Arabic or Asian languages.

6.2 Using bilingual data for preprocessing

The second goal of this work was to research if the use of bilingual data could improve on preprocessing that normally is applied to monolingual data. Given the results, the conclusion must be that this approach does not look promising. On its own, a bilingual NER tagging algorithm such as lexical similarity is not a substitute for monolingual tagging such as CRF.

The use of bilingual data in the context of tokenization and stemming is not helpful. Both forms of preprocessing apply to aspects of a language that bear no relationship to the same aspects in the other language. In both cases we test for certain syntactic properties of the text, which differ between language pairs almost by definition.

The results do show that reclassification by lexical similarity on the uniform CRF tagging performs comparable to the original CRF classification. This indicates that reclassification by lexical similarity can constitute a gain in AER score relative to an unclassified tagging. The AER scores for simple and reclassified simple confirm this. This means that given an NER algorithm that focuses on obtaining a high precision without regard to classification, the use of bilingual data can improve on such an algorithm.

6.3 Future research

This thesis examined a select number of preprocessing steps, those steps being basic stemming, basic tokenization and NER. Further research into the effect of preprocessing on SMT performance could expand the borders into other techniques offered by the field of NLP. These techniques include:

- **Chunking**

By breaking sentence pairs into multiple smaller sentence pairs, each of which is a contained translation, the complexity of the alignment task can be significantly reduced. Roughly 70,000 out of the 120,000 sentences in the English corpus are longer than 50 words, and about 1,400 sentences are over 100 words, which is the maximum amount of words GIZA++ will accept before truncating the sentence. A good chunking algorithm is likely to produce considerable gain in training times, and possibly alignment quality.

- **Part-of-speech Tagging**

Part-of-speech tagging in itself is unlikely to contribute to SMT training directly, as translations typically don't use the same amount of noun phrases, verbs or other types of words, nor do the patterns in which such word classes occur generally match. However, Part-of-speech Tagging might be helpful as part of another type of preprocessing, such as Named Entity Recognition.

- **Decompounding**

Some languages such as Dutch and Finnish feature compounded words where other languages tend to use the compound words separated by spaces. By analyzing the compound words it may be possible to either decompounds the words into their components, or alternatively arrange for the word-level alignment to take compounded words into account. Alignment quality could increase as a result, and decompounding a compounded word is likely to reduce the complexity of the alignment task as well.

- **Number recognition**

Numbers can be written in a variety of ways, including written-out words, numeric characters and Roman numerals. By standardizing the way numbers are represented in the corpus it may be possible to improve alignment accuracy.

- **Named Entity Recognition**

Though the experiments in this thesis failed to attain an improvement in alignment quality by means of NER, there is merit in further research into this type of preprocessing. As the decline in AER scores can be largely attributed to low precision, an attempt could be made at a system that attains 100% precision at any nonzero recall.

- **Using bilingual data for preprocessing**

The use of bilingual data has not shown promising results in the experiments described in this thesis. However, there are many other ways to use bilingual data, and future research may produce an approach that produces better results. For example one could use cross-reference information between gazetteers in NER. In addition it may be worthwhile to research an adaptation of monolingual algorithms such as CRF that makes use of bilingual data directly.

7 References

- [1] John Chandioux, *METEO, an operational system for the translation of public weather forecasts*, In: FBIS Seminar on Machine Translation, 8-9 March 1976, Rosslyn, Virginia. American Journal of Computational Linguistics, microfiche 46; pp.27-36
- [2] (1984) Makoto Nagao, *A framework of a mechanical translation between Japanese and English by analogy principle*, Artificial and human intelligence: edited review papers presented at the international NATO Symposium, October 1981, Lyons, France; ed. A. Elithorn and R. Banerji. Amsterdam: North Holland, 1984; pp. 173-180
- [3] Miki Sasaki & Toshiki Murata, *A pattern-based machine translation system - Yakushite Net MT engine*, International Workshop on Spoken Language Translation: Evaluation Campaign on Spoken Language Translation [IWSLT 2005], 24-25 October, 2005, Pittsburgh, PA, USA
- [4] Hiyan Alshawi & Shona Douglas, *Speech translation performance of statistical dependency transduction and semantic similarity transduction*, ACL-2002 workshop "Speech-to-speech translation", 11 July 2002, Philadelphia, USA; pp. 31-38
- [5] *European Parliament - Plenary debates: Note to the reader*,
http://www.europarl.europa.eu/cre/info_en.htm
- [6] P. Brown et al, 1993, *The Mathematics of Statistical Machine Translation: Parameter Estimation*. Computational Linguistics Volume 19 , Issue 2 (June 1993)
- [7] F.J. Och and H. Ney 2000, *Improved Statistical Alignment Models*. Proceedings of the 38th Annual Meeting on Association for Computational Linguistics, pages 440-447
- [8] F.J. Och and H. Ney 2003, *A Systematic Comparison of Various Statistical Alignment Models*. Association for Computational Linguistics, 2003
- [9] Habash, N and Sadat, F, *Arabic preprocessing schemes for Statistical Machine Translation*, Proceedings of Human Language Technology Conference/North American Chapter of the Association for Computational Linguistics (HLT/NAACL) 2006. New York City, New York, USA.
- [10] G. Leusch, N. Ueffing, D. Vilar, & H. Ney: *Preprocessing and normalization for automatic evaluation of machine translation*. ACL-2005: Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization, University of Michigan, Ann Arbor, 29 June 2005; pp. 17-24.
- [11] Kevin Knight, *A Statistical MT Tutorial Workbook*, JHU summer workshop, April 30, 1999

- [12] Matthias Eck, Stephan Vogel, & Alex Waibel, *Language model adaptation for statistical machine translation based on information retrieval*, LREC-2004: Fourth International Conference on Language Resources and Evaluation, Proceedings, Lisbon, Portugal, 26-28 May 2004; pp.327-330
- [13] Y. Al-Onaizan and J. Curin and M. Jahr and K. Knight and J. Lafferty and I. D. Melamed and F. J. Och and D. Purdy and N. A. Smith and D. Yarowsky, *Statistical Machine Translation, Final Report*, {JHU} Workshop, 1999
- [14] Jonathan J. Webster & Chunyu Kit. *Tokenization as the initial phase in NLP* Proceedings of the 14th conference on Computational linguistics - Volume 4 pp 1106 – 1110, 1992
- [15] C.J. van Rijsbergen, S.E. Robertson and M.F. Porter, 1980. *New models in probabilistic information retrieval*. London: British Library. (British Library Research and Development Report, no. 5587).
- [16] J. Lafferty, A. McCallum, and F. Pereira. *Conditional random fields: probabilistic models for segmenting and labeling sequence data*. In International Conference on Machine Learning, 2001.
- [17] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. *Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling*. Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL 2005), pp. 363-370.
- [18] Gonzalo Navarro. *A guided tour to approximate string matching*. ACM Computing Surveys, 33(1):31–88, 2001.
- [19] P. Koehn, *A Parallel Corpus for Statistical Machine Translation*, MT Summit 2005

Appendix A: Dutch list of non-breaking prefixes

#Anything in this file, followed by a period (and an upper-case word), does NOT indicate an end-of-sentence marker.

#Special cases are included for prefixes that ONLY appear before 0-9 numbers.

#any single upper case letter followed by a period is not a sentence ender. U can be an exception, but we leave it in.

#usually upper case letters are initials in a name.

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z

#List of titles. These are often followed by upper-case names, but do not indicate sentence breaks

#Source: <http://taaladvies.net/taal/advies/tekst/21/>

LADM

gen

VADM
lgen
SBN
genm
bgen
KTZ
kol
KLTZ
lkol
LTZ1
maj
LTZ2OC
kap
ritm
LTZ2
elnt
LTZ3
tlnt
DR
Dr
dr
Drs
drs
Prof
prof
Prof.dr
prof.dr
Dhr
dhr
Mevr
mevr
kand
bacc
ir
mr
dr
dr.h.c
ds
ing
bc
BA
BSc

MPhil
LLB
MA
MSc
PhD

#misc – odd period-ending items that NEVER indicate breaks (p.m. does NOT fall into this category – it sometimes ends a sentence)

vs
b.v
bv
n.b
dwz
mbt
c.q

#Numbers only. These should only induce breaks when followed by a numeric sequence

add NUMERIC_ONLY after the word for this function

#This case is mostly for the english "No." which can either be a sentence of its own, or

#if followed by a number, a non-breaking prefix

Art #NUMERIC_ONLY#

Nr

pp #NUMERIC_ONLY#