

Context-Aware Predictive Text Entry for Swedish using Semantics and Syntax

Sebastian Ganslandt & Jakob Jörwall

Examensarbete för 30 hp
Institutionen för datavetenskap, Naturvetenskapliga fakulteten, Lunds universitet

Thesis for a diploma in Computer Science, 30 ECTS credits
Department of Computer Science, Faculty of Science, Lund University

Context-Aware Predictive Text for Swedish using Semantics and Syntax

Abstract

For mobile phones the most common input method of today, single-tap, uses frequency statistics and a dictionary to disambiguate key sequences. This report investigates the combined use of language modeling, semantic knowledge and grammars for context aware ordering of the predicted candidates. The results are compared to pure frequency-based word list ordering (baseline), both with and without utilization of completions.

The reduction of extra key presses, for example used when skipping down to a word in the prediction list, reached 13.63%, compared to the baseline, when using models based on frequency, semantics, and grammar. When the models are used together with completions, the keystroke savings rate increased with 12.65% compared to the baseline.

Comparisons have also been made with bigrams, showing that bigrams are superior to all of these models combined but can be improved further with the addition of the grammatical and semantic models.

The contribution of the syntactic, dependency grammar based, model is by the KSPC metric comparable to the contribution of the semantic model. The syntactic model can however, in contrast to the semantic- and bigram model, be greatly optimized in terms of memory footprint and might therefor be preferred in situations where the resources are limited.

Kontextkänslig prediktiv textinmatning för svenska med hjälp av semantik och syntax

Sammanfattning

Den vanligaste inmatningsmetoden som används idag utnyttjar ordfrekvenser och en ordlista för att klargöra flertydiga knappsekvenser. Den här rapporten försöker utröna huruvida kombinerade lingvistiska modeller kan användas för att ordna om ord i prediktionslistan med hänseende till kontexten. Resultaten jämfördes med en baslinje baserad på ordfrekvenser, både med och utan användning av ordkompletteringar.

Minskningen av extra knapptryckningar, som används t.ex. när ett ord i prediktionslistan ska väljas, var 13,63%, jämfört med baslinjen, när modeller baserade på frekvens, semantik och grammatik utnyttjades. När modellerna användes tillsammans med ordkompletteringar minskade användandet av knapptryckningar med 12,65%, jämfört med baslinjen.

Jämförelser har också gjorts med bigram, och visar att bigram är överlägsen de andra modellerna men kan förbättras med hjälp av modeller baserade på grammatik och semantik.

Bidraget från den syntaktiska modellen, baserad på dependensgrammatik, är efter KSPC-måttet jämförbar med bidraget från den semantiska modellen. Den syntaktiska modellen kan dock, i motsats till bigram- och semantik-modellen, optimeras med avseende på minnesåtgång och borde därför föredras i situationer då resurserna är begränsade.

Contents

1	Introduction	1
1.1	Text input on small devices	1
1.2	Problem	2
1.3	Outline	4
2	Language processing in text entry	5
2.1	Corpus	5
2.2	Dictionaries and frequency based prediction	6
2.3	Language model	7
2.4	Semantic affinity	9
2.5	Part of speech	10
2.5.1	Model	10
2.5.2	Sparse data	10
2.5.3	Granska	11
2.6	Syntax	11
2.6.1	Syntax in predictive text entry	12
3	Dependency grammars	13
3.1	Introduction	13
3.2	About parsing	14
3.2.1	The algorithm	14
3.2.2	Classifiers	15
3.2.3	Features	17
3.2.4	N-Best search	18
3.3	Experimental setup and results	18
3.4	Discussion and future work	19
3.5	Dependency grammars in predictive text	21
4	Experimental setup and results	23
4.1	System overview	23
4.2	The models	24
4.3	Introducing completions and next word predictions	25
4.4	Metrics	26
4.5	Simulations	26
4.6	Results	27
4.6.1	Without completions	27
4.6.2	With completions	29

5	Discussion	31
5.1	About the results	31
5.2	Complications	32
5.3	Future work	32
5.4	Conclusion	33
A	Dependency parsing	37
A.1	Feature model syntax	37
A.2	Feature models	39
B	Word list	41

List of Figures

1.1	Standard 12-button keypad layout	1
2.1	Example of an ABCTrie	6
3.1	Dependency structure example	14
3.2	Linear classification	17
3.3	Classes that are not linearly separable	17
4.1	System overview	24
A.1	The feature model grammar	37

List of Tables

2.1	Examples of bigrams for the word <i>där</i>	8
2.2	Examples of trigrams for the word <i>där</i>	8
3.1	Parser transitions	16
3.2	A simple feature model	17
3.3	A simple feature model adjusted for a linear classifier	18
3.4	Parse results	20
3.5	Parse times	20
3.6	Parse results, Talbanken Malt	20
3.7	Parse results, Talbanken small	20
4.1	Configurations with word frequency	28
4.2	Results with word frequency	28
4.3	Configurations with bigrams	28
4.4	Results with bigrams	28
4.5	Results with completions and word frequencies	29
4.6	Results with completions and bigrams	29
A.1	FM2, our feature model for predicting parser actions	39
A.2	FM3, our feature model for predicting labels	40

Acknowledgements

Our greatest acknowledgements will go to Doc Pierre Nugues for supporting us during this thesis work. He has been inspiring and encouraging in our work despite the fact that we had little knowledge in his field before we started working. We would also like to thank Sony Ericsson and our supervisor Daniel Moberg for giving us the opportunity to perform the study at their software development site in Lund. Other people that deserve mentioning are Richard Johansson and Pär Spjuth. Richard for help on classifiers, and Pär for valuable conversations. Without them the work would be a lot more tiresome. Further acknowledgements go to Joakim Jörwall for proof-reading.

Chapter 1

Introduction

When the market for mobile phones was fairly new, most people used their phones only for making calls. This changed when short messaging service (SMS) was introduced and people could send messages instead of calling one another. For a relatively long time, text input was slow and required much time and effort from the user making it tedious to write long messages. When more and more people began writing SMS, the companies developing phones understood that they needed to extend and improve their input methods, thus more advanced input methods were introduced. Nowadays people input text on their phones for many different reasons; writing mails, browsing the Internet, making notes, etc.

Other methods such as handwriting, QWERTY-keyboard, shapewriting, speech recognition, etc. have been introduced recently but still, the most commonly used input method was introduced almost 10 years ago. In this report, further developments of this method will be explored.

1.1 Text input on small devices

When one inputs text on a mobile phone, several techniques are available. 12-key input is the most common technique used. The alphabet is divided into eight sets as seen in Figure 1.1, and each key corresponds to a set of characters. Since several characters are assigned to each key, one single press on a key is ambiguous. The software in the phone does not know the intention of the user and therefore a method of disambiguation is needed.

Multi-tap is a simple input method for a 12 button keypad layout. Each key acts as a loop, cycling through each letter on the key for every press on the same key. This makes the input unambiguous, because a key stroke sequence corresponds to only one sequence of characters. When the user wants to type a letter, he or she presses the corresponding key until the loop reaches the intended letter. After this, he or

1 .!?	2 ABC	3 DEF
4 GHI	5 JKL	6 MNO
7 PQRS	8 TUV	9 WXYZ
*	0 _	#

Figure 1.1: Standard 12-button keypad layout.

she presses another key or waits a predefined time to verify that the correct letter is selected. For example, the key sequence 8 – 4 – 4 – 3 – 3 leads to the word *the*.

Multi-tap is easy to implement and no dictionary is needed. At the same time, it is slow and tedious for the user, since each character corresponds to one or more key presses. Another problem with multi-tap arises when the user wants to enter two consecutive characters that are placed on the same key, for example an *f* followed by an *e*. In this case, the user has to, depending on the choice of solution, wait a predefined time or press a *next-key* to continue the input. This also increases the time it takes to input text using this method.

Single-tap is another input method for a 12 button keypad layout. With this method, each key is pressed only once for every character. This makes the input ambiguous since two different words may have the same key stroke sequence. When the user presses a combination of buttons, the phone retrieves the possible words (prediction candidates) from a dictionary. For example, the key sequence 8 – 4 – 3 is ambiguous and leads to a number of alternatives including the word *the*. The list of predictions is then sorted according to some sense of relevance, for example how frequent the words are in the language. If the word does not exist in the dictionary, the user has to multi-tap the word. Recent implementations of single-tap make use of completions. These try to increase the typing speed by giving the user words that complete a keystroke sequence.

1.2 Problem

In a sentence, some words may not fit the current context and should therefore not be at the top of the list. When one talks about context it could mean different things, such as the weather, the country one is in at the moment, the music one is listening to, etc., but here, by context we mean the current sentence the user is typing. Most current implementations do not use any context at all.

The traditional ambiguous text input setup is that a sequence of keys, ks_i , are pressed to enter a desired word, w_i . Since the key sequence might be ambiguous, the words matching the key sequence, $match(ks_i) = \{w_0, \dots, w_n\}$ is presented in an alternative, list where it takes k extra key presses to reach candidate w_k . Each $w \in match(ks_i)$ will be assigned a score

$$Score(w|Context) = \sum_{s \in S} \lambda \cdot s(w|Context),$$

measuring its validity in the current context, and the prediction list is then sorted according to the word scores. s is a scoring function from a set of scoring functions S , λ the weight of the corresponding scoring function, and $S(w|Context)$ is the total score or rate of w in the current context.

Our work is connected to the ambiguous key sequences. More clever scoring functions will result in a more clever sorting of the prediction list, and the user intended word will be presented as the first candidate more often. The questions we are trying to answer are

- *By exploiting the combined information from various linguistic models, how can we more accurately disambiguate the key sequence?*

- *What information should these models be based on?*
- *How can the syntax of a sentence be used to disambiguate the key sequence?*

Our hypothesis is that context sensitive sorting of the words in the alternative list would result in less key strokes by the user (compared to current implementations), and this should in return make text input a more pleasant experience.

The set of functions that evaluates and assign scores to the words consists of:

- Language model
- Semantic affinity
- Part-of-speech validity
- Dependency validity

Language models are based on sequences of words and are the most commonly used in text input today. Semantic affinity is based on the likelihood of two words appearing together in a sentence. Part of speech (POS) is a way of dividing words into sets where each set represents common grammatical properties, for example *verb*, *adjective* and *noun*. Just as some sequences of words are more common than others, some sequences of POS tags are more common than others. Gathering statistics on these sequences creates a sort of implicit grammar for the language. Dependency grammars are more linguistically rooted theories about the inherent structure of a sentence and the dependency validity is our attempt to exploit this structure, when finding the probability of a given prediction candidate being the user intended word. The functions will be more thoroughly examined in the following chapters.

An evaluation method for the system is needed. This is based on simulations of a user performing input and measuring the keystrokes needed. Since our work is about minimizing the number of keystrokes by the user, a metric for evaluation is key strokes per character (KSPC) (MacKenzie, 2002)

$$KSPC = \frac{\sum_{w \in L} K_w \cdot Occ_w}{\sum_{w \in L} C_w \cdot Occ_w}, \quad (1.1)$$

where C_w is the number of characters in w , K_w the number of keystrokes to enter w and Occ_w the number of occurrences of w in the language L .

The improvements in disambiguation is subtle in that the differences between the baseline and the improved version might be hard to spot in a usability test, where the user only types in a few sentences. For this reason, we did not conduct any usability tests. Instead simulations were performed to establish approximations of KSPC for different sets of scoring functions to find out which contributed the most, and if they contributed with disjointed information.

There are a number of other metrics as well, such as words per minute (WPM) and an minimum string distance (MSD) based error metric (Gong and Tarasewich, 2006). WPM is either measured in usability tests on users with different levels of expertise or by keystroke level modeling (Card et al., 1980).

In the latter, case a theoretical maximum performance of error free input from an uninterrupted user, based on finger movement speed and the placement of the physical keys, is measured. Since we have not made any changes to the physical device or conducted any usability studies, these metrics have not been applied.

1.3 Outline

In this thesis work, we have performed a reimplementaion and evaluation for Swedish, of the grammatical and semantic models described in Gong et al. (2008). We have also implemented a dependency parser based on Nivre's algorithm (Nivre, 2003) together with a maximum entropy classifier (Fan et al., 2008) and evaluated impact of a more sophisticated grammatical analysis of the context, not solely based on sequences of words and POS.

A brief outline of the intent and work done has been presented in this chapter. Chapter 2 will cover previous research and theory connected to language modelling, POS validity (POSV) and semantic affinity (SemA). In Chapter 3, theory on dependency grammars and parsing, together with our scoring function based on this theory, will be investigated. In Chapter 4, the simulations will be described together with the results, and in the final Chapter 5, the results will be discussed together with ideas on future work in this area.

Chapter 2

Language processing in text entry

Linguistics is the study of language structures. Computational linguistics is a research area which combines knowledge from linguistics and computer science. By creating statistical models of languages one can make language processing automatic by a computer. Training data is used to train these models so that conclusions about previously unseen data can be drawn. In this chapter, theory and previous research in the field that is applicable to text entry will be handled.

2.1 Corpus

A corpus (plural corpora) is a large collection of natural language text. It can be collected from different sources such as novels, newspapers, discussion forums, etc. Some corpora are only collected from one source while others come from multiple sources.

Different corpora apply to different needs, e.g. an application used in the financial industry would not benefit from a corpus based on sports articles since the different lingo match poorly. A suitable choice of corpus is therefore important and has an impact on the application.

Some corpora hold meta-information about the words in them, so called annotated corpora. Here each word is associated with a set of tags that describes, for example, the grammatical function of the word. Usage of the meta information could for example involve training of models.

There are a number of different corpora to choose from, more for English than the smaller languages of the world, like Swedish. One corpus widely used is the British National Corpus (BNC). BNC consists of over 100 million tagged words collected from various sources¹. Other corpora in English are the Oxford English Corpus² and the Corpus of Contemporary American English³. Quite recently Google⁴ announced that they would make their n -gram statistics public. They are based on their corpus consisting of over one trillion words of running text,

¹<http://www.natcorp.ox.ac.uk>

²<http://www.askoxford.com/oec>

³<http://www.americancorpus.org>

⁴<http://www.google.com>

collected from websites and not annotated. Although it is very large compared to the other mentioned above, the smaller annotated corpora will still be needed.

For Swedish, a number of corpora exist. The Stockholm Umeå Corpus (SUC) consists of one million words POS annotated. Talbanken05 is a modernized version of a corpus from 1976 containing 300,000 words. This is the largest dependency structure annotated Swedish corpus.

For statistical models used in mobile phones, a corpus based on SMS messages would be beneficial. The language in SMS messages differs a lot from ordinary language. Therefore a number of projects are involved in the task of creating large SMS corpora. A corpus in French⁵ contains 75,000 SMS messages, and is also annotated with information about the author’s gender, age, occupation, etc. A Swedish SMS corpus does not yet exist.

2.2 Dictionaries and frequency based prediction

Dictionary-based text entry goes back to the beginning of text messaging systems. In the late 1990s, most mobile phones incorporated a system for sending SMS. The input methods were first based on multi-tap but later single-tap was introduced to create more efficient input. Single-tap with a model based on word frequencies and dictionaries is still the standard of today.

When the user presses a sequence of keys the sequence is transformed into a set of combinations of letters, i.e. a set of possible words. This transformation is done by using a tree data structure also known as a *Trie*. The data structure is used for storing large lexicons and to do fast searches in these. The idea of the Trie is to store the words as trees of characters and to share branches as far as the letters of two words are identical (Fredkin, 1960). A minor modification of the Trie is the ABCTrie⁶ (Hasselgren et al., 2003) where on each level in the tree every node corresponds to a button on the phone keypad. Each node has links to a set of children, representing further key presses, and/or a set of leaves, representing words that match the key given key sequence. Figure 2.1 is an example of an ABCTrie where the key sequence 4 – 6 – 2 could mean either *eld* or *eke*. This sequence could also be followed by a 1, 4 or 5 to get longer words, but not, for example by a 2. The words found in the Trie are presented to the user in a list.

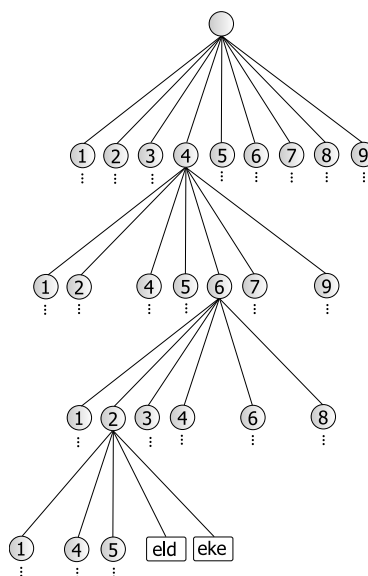


Figure 2.1: Example of an ABCTrie.

To determine how to sort the list of alternatives one needs to determine which word is more relevant. For the most commonly used single-tap input

⁵<http://www.smspouurlscience.be>

⁶The name comes from that the classic Trie is used in the context of an ABC keypad layout, where the first key represents A, B and C and so on.

method, T9⁷, the words are sorted by frequency. That is, two words with the same key sequence are compared as to how common their appearance is in the language. This is probably done in a similar way for other methods but since we have no insight into the implementations no conclusions can be drawn. These systems are also currently changing and newcomers such as ExB⁸, are trying to gain their share of the market by utilising more intelligent methods. The frequencies are in general found by corpus statistics. The words' frequencies determine their probability by

$$P(w) = \frac{C(w)}{\sum_{w_i \in L} C(w_i)} \quad (2.1)$$

where $C(w)$ is the frequency of word w , L is the set of distinct words in the language, and $P(w)$ is the probability of word w based on word frequency. The alternatives are sorted according to their probabilities. This method results in a KSPC just above 1, depending on the evaluation method (MacKenzie, 2002; Mittal and Sengupta, 2007; Hasselgren et al., 2003). Other input methods using dictionary and word frequencies are *iTap*⁹, and *eZiText*¹⁰.

New releases of the input method described above make use of enhanced functionality in the form of word completion. The input method finds possible words that completes the current key sequence. This means that the whole subtree of the current key sequence is given as an alternative list. When the whole subtree is used this results in a longer list of word alternatives. A good disambiguation method will therefore be needed. Since the user does not have to write the whole word, KSPC can drop below 1.

2.3 Language model

Input methods based on word frequencies are not context sensitive. This means that previous words in a sentence are not considered when sorting the word alternatives. A better way would be to increase the context and thereby achieving better disambiguation. From statistical theory comes the notion of the maximum likelihood estimate (MLE). This can be used to make estimations of how probable a sequence of words is. The estimation is based on how frequent the sequence is in a corpus. For a sequence of n words the MLE is

$$P_{MLE}(S) = \frac{C(w_1, \dots, w_n)}{N}$$

where N is the number of sequences of length n in the corpus. For the estimate one needs a corpus which contains all possible sequences to produce the probability. This is practically impossible and no corpus big enough exists.

One may decompose $P(S)$ by

$$P(S) = P(w_1)P(w_2|w_1)\dots P(w_n|w_1, \dots, w_{n-1}) = \prod_{i=1}^n P(w_i|w_1, \dots, w_{i-1})$$

⁷<http://www.nuance.com>

⁸<http://www.exb.de>

⁹<http://www.motorola.com>

¹⁰<http://www.zicorp.com>

where S is a sentence, or sequence of words, $P(S)$ the probability of sentence S to appear in the corpus, and $P(w_n|w_1, \dots, w_{n-1})$ the probability of the word w_n to appear after the words w_1, \dots, w_{n-1} .

This will be an impossible model to implement since there does not exist a corpus big enough for n-grams when n is large. The model can simply not say whether a combination of for example 10 words are probable when the combination does not exist in the corpus. For large n this results in sparse matrices, i.e. a large part of the elements will be zero. An approximation of large n-grams to smaller ones is needed by

w_{i-1}	w_i	#
,	där	387
den	där	104
och	där	83
det	där	77
de	där	24
så	där	21

$$P(w_i|w_1, w_2, \dots, w_{i-1}) \approx P(w_i|w_{i-1})$$

Table 2.1: Examples of bigrams for the word *där*.

where the n-grams are approximated by bigrams. Bigrams are sequences of two words, and trigrams sequences of three words. This will reduce memory usage but still capture some of the word sequences in a sentence. This leads to the MLE models

$$P_{MLE}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i)}{\sum_{w \in L} C(w_{i-1}, w)} = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})} \quad (2.2)$$

$$P_{MLE}(w_i|w_{i-1}w_{i-2}) = \frac{C(w_{i-2}, w_{i-1}, w_i)}{\sum_{w \in L} C(w_{i-2}, w_{i-1}, w)} = \frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})} \quad (2.3)$$

The statistics are collected from a corpus. Examples of n -grams are shown in Tables 2.1 and 2.2.

N-grams are exponential in memory usage. Consider a corpus containing 10,000 distinct words. To save all possible combinations and their occurrences we need a matrix of size $10,000^2$ for bigrams, $10,000^3$ for trigrams and so forth up to $10,000^i$ for n-grams of size i . This means that a model based on n-grams larger than 3 is practically impossible to keep in memory during runtime.

w_{i-2}	w_{i-1}	w_i	#
allt	det	där	10
,	och	där	10
på	den	där	8
det	var	där	7
,	den	där	7
här	och	där	7

A linear interpolation of the different n-grams would be

Table 2.2: Examples of trigrams for the word *där*.

$$P(w_i|w_{i-n+1} \dots w_{i-1}) = \lambda_1 P(w_i) + \dots + \lambda_n P(w_i|w_{i-n+1} \dots w_{i-1})$$

where λ_i is a weight for a specific n-gram and

$$0 \leq \lambda_i \leq 1$$

and

$$\sum \lambda_i = 1$$

so that the score of the language model function may be compared to other scoring functions. One finds the weights by iterating a subset of possible combinations of weights. This is done in small incremental steps finding weights that

result in lowest KSPC for a development set. The set of weights resulting in lowest KSPC is used further on when the language model is used in combination with other scoring functions.

The bigram and trigram data will be sparse matrices since the corpus does not contain all possible combinations. Often a cutoff function removes n-grams with frequency lower than a threshold. This leads to matrices with a large number of elements equal to zero. Therefore *Laplace's rule* is applied which increments all elements in the matrices with *one* (Laplace, 1812)

$$P_{Laplace}(event) = \frac{Occ(event) + 1}{\sum_{e \in Events} Occ(e) + |Events|} \quad (2.4)$$

or in this special case

$$P_{Laplace}(w_i|w_{i-1}) = \frac{Occ(w_{i-1}, w_i) + 1}{\sum_{w \in L} Occ(w_{i-1}, w) + |L|} \quad (2.5)$$

This results in a model which interprets word combinations not seen in the corpus as very unlikely, instead of being completely unlikely. Unfortunately this shifts the probabilities so that low-frequent n-grams become almost as probable as n-grams never seen before.

2.4 Semantic affinity

By using n-grams of no greater length than *three*, the model only takes into account words in groups of three in the corpus. This makes it impossible to find relations between words that are semantically connected but in the sentence far away from each other. It is likely that a sentence contains words that are linked semantically. For example in the sentence

The mouse, who was afraid of the big cat, ate cheese.

the word *mouse* and *cheese* are semantically linked but far from each other. Since the word n-gram model cannot tell us anything about words far from each other, it will not help us in determining whether or not *cheese* is a relevant alternative. Therefore we need a model which verifies the semantic relatedness between two words. One such model was introduced by Li and Hirst (2005), and which is defined as

$$Relatedness(w_i, w_j) = \frac{C(w_i, w_j)}{C(w_i)C(w_j)} \quad (2.6)$$

where $C(w_i, w_j)$ is the number of times the words w_i and w_j exist together in a sentence in the corpus, and $C(w_i)$ is the number of times the word w_i exists in the corpus. The relations are symmetrical, i.e.

$$C(w_i, w_j) = C(w_j, w_i).$$

The estimated semantic affinity of a word in a context is

$$SemA(w|H) = \sum_{w_j \in H} Relatedness(w, w_j) \quad (2.7)$$

where H is the context of the word w .

Li and Hirst (2005) used a context of changing size and found a keystroke saving rate (KSR) of between 64.86% and 65.80% depending on context size, compared to a rate of 59% for just using word n -grams. KSR is a measure of saying how many keystrokes the user saves with a certain input method compared to a baseline method. Gong et al. (2008) used a similar model in a predictive text application with a slight modification to the *Relatedness* function.

$$\text{Relatedness}(w_i, w_j) = \frac{C(\text{stem}(w_i), \text{stem}(w_j))}{C(\text{stem}(w_j))} \quad (2.8)$$

The $\text{stem}(w)$ function removes suffixes from words, a crude form of lemmatization well suited to get approximations of basic word forms in English.

2.5 Part of speech

Grammar classifies words into different categories, or parts of speech (POS), according to how the words are used in a sentence. In this section, we will describe a software library with functions for finding lemmas and tagging words with POS. The tagged words will be used in a model that evaluates the relevance of different words in a context according to their POS.

2.5.1 Model

In Chapter 2, the language model exploited word sequences of length two and three. To take advantage of semantically linked words, we extended this using the semantic affinity model. To further exploit sequences of words, we will add a POS validity model. This model will, like the word n -gram model, use sequences for disambiguation but instead of word sequences the POS validity model will use POS-tag sequences. This leads to a similar maximum likelihood estimate

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})},$$

where t_i is a POS tag and $P(t_i|t_{i-1})$ is the probability of the POS tag sequence $t_i t_{i-1}$. The corresponding linearly combined POS n -gram model would be

$$P(t_i|t_{i-n+1} \dots t_{i-1}) = \lambda_1 P(t_i) + \dots + \lambda_n P(t_i|t_{i-n+1} t_{i-1}),$$

but since the model will be based on n -grams of longer length than for word n -grams, we need a better way of handling sparseness than simply using weights.

2.5.2 Sparse data

We cannot be certain that the elements that are zero in the sparse data actually mean a very unlikely sequence or is just due to properties of the corpus. Katz (1987) found that another way to handle sparseness is to use a backoff-model

$$P_{\text{backoff}}(w_i|w_{i-1}) = \begin{cases} P(w_i|w_{i-1}), & \text{if } C(w_{i-1}, w_i) \neq 0 \\ \alpha P(w_i), & \text{otherwise} \end{cases}$$

where α is a variable used to ensure that the probabilities sum up to 1. When the n-gram does not exist the model “backs” off to the next $(n - 1)$ -gram. We use a slightly modified version of (Katz, 1987)

$$P_{\text{modifiedbackoff}}(w_i|w_{i-1}) = \begin{cases} P(w_i|w_{i-1}), & \text{if } C(w_{i-1}, w_i) \neq 0, \exists w_i \in M \\ P(w_i), & \text{otherwise} \end{cases} \quad (2.9)$$

where M is the set of words that match the current keystroke sequence. When no bigram exists for the current word set, the model “backs off” to unigrams. Here the modified backoff is, just like before, shown for bigrams but the principle can be used recursively for any n-gram model.

For part of speech the n-gram models will not be as sparse as the word n-gram model. The backoff model should therefore work better than *Laplace’s Rule*, under the assumption that the n-gram model is accurate when present.

2.5.3 Granska

Granska is a system essentially made for grammar checking in Swedish. The system consists of a tokenizer that break up the sentence in words and punctuation marks; a POS tagger; a rule matcher which assigns rules to find grammatical errors in Swedish. The POS tagger is based on a hidden Markov model (Carlberger et al., 2002). The most probable POS tagging of a sequence of words w_1, \dots, w_n is determined by

$$\text{Tagwords}(w_1, \dots, w_n) = \arg \max_{t_1, \dots, t_n} \prod_{i=1}^n P(t_i|t_{i-2}, t_{i-1})P(w_i|t_i)$$

where t_1, \dots, t_n is the sequence of POS tags giving the maximum product probability. The Granska tagger has a precision of 97% when the word is known to the tagger, and 93% when the word is unknown (Carlberger et al., 2002). As seen in the sentences,

- *Får jag ett glas öl?*
- *Du är ett får.*

the word *får* play different roles in speech; *verb* in the first sentence but *noun* in the second. The tagging of a word is not as straightforward as one may think since the POS is to some extent dependent on the context.

2.6 Syntax

The models presented above are all helpful when processing natural language but they all use knowledge of sequences and semantic affinity, and do not look at the relationships between the words. Syntax is about these relationships, or simply put *who does what to whom*. By looking at the relationships we should be able to discard words that do not fit the dependency structure. Evaluation of different syntactic structures seems like a natural next step in our work.

In the following chapter, we will look more deeply into syntax. Specifically we will look into dependency grammar which is a way to represent syntax.

In this section, previous research of syntax connected to predictive input will be discussed. This should not be viewed as complete, but instead as a small selection.

2.6.1 Syntax in predictive text entry

Studies of syntax has been a major part of linguistic research during the 20th century by for example Chomsky (1957) and Tesnière (1959). The goal has been and still is to find and describe the inherent structure of the sentence. In computational linguistics the aim is also to develop algorithms that automatically can extract these structures to use them in language processing applications.

At the same time, two different schools were developed; the American school based on Chomsky's constituency grammar, and the European school based on dependency grammar. The constituency grammar divides words in a sentence into different sets, with each set representing a phrase. The dependency grammar sees a sentence as connections between words, where each connection describes a dependency between two words.

The knowledge of syntax has later on been used in different areas of NLP and has been used in systems since the early 1990s (Tyvand and Demasco, 1993; Arnott et al., 1993). One example is grammar correction when writing text on a computer. Text prediction especially targeted at people with motor disabilities has been another aim of the studies of syntax. This is interesting since word prediction is helpful for all text input systems and therefore interesting to our problem.

A good example is FASTY, an EU-funded project which aims to help disabled people with a predictive text input environment. At first, FASTY only relied on language models (Matiasek et al., 2002) but later a syntax part based on grammar rules was evaluated (Gustavii and Pettersson, 2003). The grammar rules were based on common grammatical errors and were used to rerank the next word to be predicted. The results were not as good as expected and therefore the work of Gustavii and Pettersson (2003) was not added to the system because of the large overhead introduced (Matiasek, 2006).

Another example is (Sundarkantham and Shalinie, 2007) which implemented a system where the grammar rules were inferred from a corpus. These rules were then used to discard infeasible grammatical constructions. The authors evaluated the system by playing Shannon's Game (Shannon, 1951) and achieved better results than previously seen. Shannon's Game is a way to evaluate prediction systems by giving the system a partial sentence and seeing how often the system correctly guesses the next word.

Chapter 3

Dependency grammars

The sentence is an organized whole, the constituent elements of which are words. Every word that belongs to a sentence ceases by itself to be isolated as in the dictionary. Between the word and its neighbours, the mind perceives connections, the totality of which forms the structure of the sentence. - Tesnière (1959)

The first section of this chapter will be a gentle introduction to dependency grammars and automatic parsing, extraction of such structures. This will be followed by our work on using Nivre's algorithm enhanced with an N-Best search together with the *liblinear* (Fan et al., 2008) classifier. The chapter will be rounded off with ideas on how this can be put into use in the area of predictive text.

3.1 Introduction

Dependency grammars is a family of theories which are all gathered around the assumption that the syntactic structure of a sentence consists of lexical nodes (such as words) that are linked together by binary relationships called dependencies (Nivre, 2005). The two constituents of a dependency relationship are called, among other things, head and dependent and the following list contains examples of rules, or criteria that make up a dependency grammar, the relations between the head (H) and dependent (D) in a construct (C) (Hudson, 1990).

1. H determines the syntactic category of C and can often replace C .
2. H determines the semantic category of C ; D gives semantic specification.
3. H is obligatory; D may be optional.
4. H selects D and determines whether D is obligatory or optional.
5. The form of D depends on H (agreement or government).
6. The linear position of D is specified with reference to H .

As can be seen, these rules are partly semantic and partly syntactic in nature. An example of a dependency structure annotated sentence is presented below,

where the arrows point from dependent to head. The arrow that points up means that this is the root, governor of whole sentence.

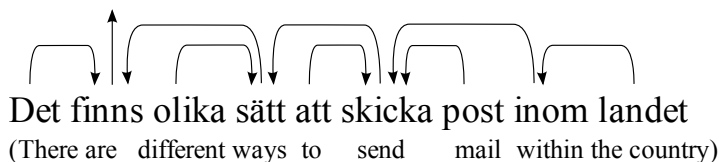


Figure 3.1: The dependency structure of the Swedish sentence “Det finns olika sätt att skicka post inom landet”.

The dependency structure annotation of this sentence shows for example that *are* is the main verb, what is being primarily communicated, and that *different* is the modifier of *ways*, i.e. it is not the *mail* or *the country*, but the *ways* that are different. It also tells us that *send* is the head of *mail* and if we would try to suggest *röst* (voice) or *port* (gate) instead of *post* (which in Swedish both match the same key sequence) as possible predictions, this would not make a very probable parse tree.

In the complete theory, the arcs are also labeled with grammatical functions, describing not only that the words *are* syntactically related but also *how* they are related. Since these functions will not be exploited in our analysis they will not be described further although our efforts in finding them will be evaluated in the section about parsing.

3.2 About parsing

3.2.1 The algorithm

Dependency grammars make up the rules for how to interpret a sentence in terms of a syntax, its inherent structure. Unfortunately, when parsing natural language text there is not a set of deterministic rules that can be applied from left to right (or any other order for that matter) to extract this structure unambiguously. In case of graph based parsing, a search through the space of all possible parse trees must be conducted. Searching in this space is very hard and effective algorithms use for example dynamic programming (Hays, 1964) and constraint programming (Foth et al., 2004) to solve the problem.

Another approach is the transition-based parser that instead of searching for a more probable, complete parse tree, splits the task down to finding one transition at the time, where a sequence of such transitions creates the complete parse tree. Nivre (2003) presented such an algorithm that deterministically parses a sentence not only in linear time but also achieves competitive results. It is based on the Shift-Reduce parsing algorithm and extended by two graph building functions LeftArc and RightArc. Table 3.1 and Algorithm 1 describe how the parse tree A , the set of arcs between pairs of words, is created from an input sentence W , the linear representation of a natural text sentence.

Given an oracle that can correctly predict the next parser transition, we would always end up with a correct parse tree. In the parser’s first appearance,

this oracle came in the shape of a hand crafted grammar consisting of 90 left-headed rules and 36 right-headed rules. Together with these rules it used a fixed priority where LeftArc was always preferred over RightArc, RightArc always preferred over Reduce and Reduce always preferred over Shift. Two additional rules to resolve situations where both Shift and Reduce could be performed and when both Shift and RightArc could be performed, were also added. This parser without and with the two addons achieved an unlabelled attachment score of 80%, 87.3%, and 89% respectively.

In a more recent attempt to estimate this oracle, support vector machines (SVM) have been used (Nivre et al., 2006) and achieved top scores in the CoNLL-X shared task¹. CoNLL is an annual conference on computational natural language learning and the tenth edition was dedicated to dependency parsing. A parser implementing Nivre’s algorithm (Nivre et al., 2006) reached 1st and 2nd place for LAS² and USA³ respectively. The remaining portion of this section will be dedicated to the general idea behind classifiers and how they are utilized in dependency parsing.

3.2.2 Classifiers

As mentioned earlier, another way of estimating the oracle is with the help of machine learning techniques. The basic idea is to supply a classifier with a set of feature vectors and desired outcomes $\langle x_i, y_i \rangle$ where

$$x_i = \langle f_1, \dots, f_n \rangle$$

is a tuple of n features and

$$y_i \in \{y_0, \dots, y_k\}$$

is the desired outcome given x_i . The set

$$S = \{\langle x_0, y_0 \rangle, \dots, \langle x_l, y_l \rangle\}$$

will be referred to as the training set of l instances. The classifier then tries to generalize this information in a training phase to be able to make statements about previously unobserved examples. Examples of classifiers are induced decision trees (Quinlan, 1986), perceptrons (Rosenblatt, 1958; Collins, 2002), and SVMs (Vapnik et al., 1992). SVMs have terrific classification performance but are very slow both when it comes to training and the actual classification phase. We have used a logistic regression model implemented in *liblinear* and compared this with the result of *libsvm* using a polynomial kernel.

¹<http://nextens.uvt.nl/~conll>

²Labelled Attachment Score, percentage of correct head- and correct label-pairs predicted.

³Unlabelled Attachment Score, percentage of correct heads predicted

Name	Action	Condition
Initialization	$\langle nil, W, \emptyset \rangle$	
Termination	$\langle S, nil, A \rangle$	
Left-Arc	$\langle n S, n' Q, A \rangle \rightarrow \langle S, n' Q, A \cup \{\langle n', n \rangle\} \rangle$	$\neg \exists n'', \langle n, n'' \rangle \in A$
Right-Arc	$\langle n S, n' Q, A \rangle \rightarrow \langle n' n S, Q, A \cup \langle n, n' \rangle \rangle$	$\neg \exists n'', \langle n', n'' \rangle \in A$
Reduce	$\langle n S, Q, A \rangle \rightarrow \langle S, Q, A \rangle$	$\exists n', \langle n, n' \rangle \in A$
Shift	$\langle S, n Q, A \rangle \rightarrow \langle n S, Q, A \rangle$	

Table 3.1: Parser transitions. W is the original input sentence, n , n' and n'' are lexical tokens, A is the dependency graph, $\langle n, n' \rangle$ is an arc from the dependent n to the head n' , S is the stack and Q is the queue.

Algorithm 1 Nivre’s algorithm.

```

1:  $Queue \leftarrow W$ 
2:  $Stack \leftarrow nil$ 
3: while  $\neg Queue.isEmpty()$  do
4:    $features \leftarrow ExtractFeatures()$ 
5:    $action \leftarrow oracle.Predict(features)$ 
6:   if  $action = RightArc \wedge canRightArc()$  then
7:      $RightArc()$ 
8:   else if  $action = LeftArc \wedge canLeftArc()$  then
9:      $LeftArc$ 
10:  else if  $action = Reduce \wedge canReduce()$  then
11:     $Reduce()$ 
12:  else
13:     $Shift()$ 
14:  end if
15: end while
16:  $return(A)$ 

```

Algorithm 2 Reference parsing.

```

1:  $Queue \leftarrow W$ 
2:  $Stack \leftarrow nil$ 
3: while  $\neg Queue.isEmpty()$  do
4:    $x \leftarrow ExtractFeatures()$ 
5:   if  $\langle Stack.peek(), Queue.get(0) \rangle \in A \wedge canRightArc()$  then
6:      $t \leftarrow RightArc$ 
7:   else if  $\langle Queue.get(0), Stack.peek() \rangle \in A \wedge canLeftArc()$  then
8:      $t \leftarrow LeftArc$ 
9:   else if  $\exists w \in Stack : \langle w, Queue.get(0) \rangle \in A \vee \langle Queue.get(0), w \rangle \in A \wedge$   

 $canReduce()$  then
10:     $t \leftarrow Reduce$ 
11:   else
12:     $t \leftarrow Shift$ 
13:   end if
14:   store training example  $\langle x, t \rangle$ 
15: end while

```

If $y \in \{-1, +1\}$, we can visualize the classifier by laying out all the feature vectors in the n dimensional space and trying to find the hyperplane of dimension $n - 1$, that separates the positive from the negative examples. Figure 3.2, where the *os* is our -1 s and *x*s are our $+1$ s, illustrates this in the case where $n = 2$. This figure also illustrates the common phenomena that the examples in S are not entirely separable and we will have to find the line that *best* generalizes the idea of what is an *x* and what is an *o*, according to some criteria of what is a good generalization.

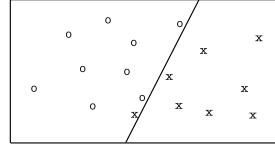


Figure 3.2: Example of a linear classifier.

It might also be the case that the classes are not at all linearly separable, like the example in Figure 3.3. SVM classifiers solve this problem by applying a nonlinear feature function $\phi(x)$ (kernel) that maps the features to a space where they actually are linearly separable. You could also solve this by explicitly combining features that are to be considered together, thereby mapping the features to a higher dimension and resolving the problem.

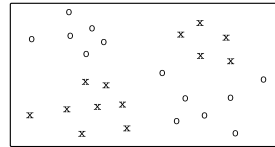


Figure 3.3: Classes that are not linearly separable.

The idea of good generalization is in *liblinears* logistic regression model defined in terms of

$$\min_w \frac{1}{2} w^T w + C \sum_{i=1}^l \log(1 + e^{-y_i(w^T x_i)})$$

where w is the weight of the different features. The classification now becomes finding the most probable class where

$$P(y = \pm 1 | x, w) = \frac{1}{1 + e^{-y(w^T x)}}.$$

In the case of $k > 2$, a one-vs-the-rest method described in Keerthi et al. (2008) is used, a method when deployed in computational linguistics is referred to the maximum entropy model (ME).

3.2.3 Features

The training set is extracted during the phase of reference parsing, where the correct sequence of parsing actions is extracted according to Algorithm 2, given a gold standard parsed sentence.

POS INPUT 0
POS INPUT 1
POS STACK 0
POS STACK 1

In the case of dependency parsing, and Nivre’s algorithm specifically, the classes that the classifier has to choose from are the different transitions encoded as numbers. The features that the decisions are based on are pieces of information from the input sentence and the partially created dependency graph, typically described in a feature set such as the one in Table 3.2. This table states that we should consider the POS of the first and second token in the input queue and the POS

Table 3.2: FM1, a simple feature model.

of the top and the second top element on the stack. For the complete feature set grammar and its description, see section A.1 in the appendix. Table 3.3 shows how to manually combine features to compensate for the weakness of a linear classifier (this is actually equivalent to using an SVM with a quadratic kernel). To create good examples for the classifier, the features need to be binarized, i.e. encoded as vectors of 1s and 0s.

3.2.4 N-Best search

One of the problems with deterministic, transition based parsing is that of error propagation. A possible remedy is to produce complete parse trees, and instead of considering the probability of one parser transition at the time, maximizing the probability of the whole parse tree. Algorithm 3 outlines the Nivre algorithm together with an N-Best search (also known as beam search).

This approach has been tried together with an online learning algorithm in Johansson and Nugues (2007); Zhang and Clark (2008), with excellent results. It has not been evaluated for Swedish and ME classifiers but a similar setup for Japanese has been tried in Sekine et al. (2000), actually with a small decrease in the token level results but with the correct parse tree among the 20 best in 99% of the cases.

POS INPUT 0
POS INPUT 1
POS STACK 0
POS STACK 1
POS INPUT 0; POS INPUT 1
POS STACK 0; POS STACK 1
POS STACK 0; POS INPUT 0
POS STACK 0; POS INPUT 1
POS STACK 1; POS INPUT 0
POS STACK 1; POS INPUT 1

Table 3.3: FM1, adjusted to fit the weakness of a linear classifier.

Algorithm 3 Beam parse.

```

1: Agenda.add(InitialParserState)
2: while  $\neg done$  do
3:   for parserState  $\in$  Agenda do
4:     Output.add(parserState.doLeftArc())
5:     Output.add(parserState.doRightArc())
6:     Output.add(parserState.doReduce())
7:     Output.add(parserState.doShift())
8:   end for
9:   Sort(Output)
10:  Clear(Agenda)
11:  Take  $N$  best parse trees from Output and put in Agenda.
12: end while
13: Return best item in Agenda.

```

3.3 Experimental setup and results

The corpus for this task has been Talbanken05 (Nivre et al., 2005), the largest of the two Swedish dependency structure annotated corpora consisting of 11431 sentences (about 300,000 lexical tokens). This was split into a training set

and development set of about 96% and 4% of the sentences respectively. The Nivre parsing algorithm was extended with a beam search strategy outlined above, where we measured the validity of a parse tree as the product of the probabilities of the parser transitions.

Feature set engineering and finding the optimal width of the beam was done on the development set. Traditional binarization techniques was used. Zhang and Clark (2008) provided us with good starting sets of features and composites for linear classifiers even if we could not fit all the lexical features, because of the memory consumption during both the training and the classification phase. The rest of the features in model FM2 (see Table A.1 on page 39) was found during experimentation, adding what intuitively (for a nonlinguist) seemed like reasonable pieces of information. To find the labels on the arcs, we trained another classifier that takes similar features and, instead of returning 1 of 4 possible actions, returns 1 of the 64 possible labels. Since the memory consumption in the *liblinear* implementation is $O(|Classes| * |Features|)$, we had to remove some of the entries in the label feature set, resulting in Table A.2. This feature set performed almost as well as the top scoring ones in the CoNLL-X shared task (on the development set), but generalized quite badly when it came to testing.

In the evaluation phase, we conducted random subsampling on the previous training set where 96% became the new training set and 4% became the test set. This was repeated 10 times and the averages are presented with LAS and UAS. For reference, the best results from the CoNLL-X shared task, parsing of Swedish, is also presented.

As mentioned, one of the greater advantages with the ME classifier is its efficiency. To highlight this, Table 3.5 shows the amount of time taken to parse each sentence, using both the *liblinear* and *libsvm* classifier.

Since our goal is to use this parser in our predictive text application, we have to make it understand the POS tags that Granska uses. This was done by re-training the parser on Talbanken Malt, a subset (about 50%) of Talbanken that is retagged using an HMM tagger compliant with the Granska tag set. The tags were stripped of features and only contain the 27 base POS groups. Talbanken Malt also uses a smaller label set than the original Talbanken consisting of 17 named functions, compared to Talbanken's 64 functions. Table 3.6 shows the results from a 10-fold random subsampling on this data. To investigate whether the decreased UAS was caused by the smaller training set or the more crude tag set, we performed the same random subsampling process on Talbanken with the original tags and achieved the results in Table 3.7.

3.4 Discussion and future work

We have shown that by manually combining features that are to be considered composite features by the classifier, you can do without the more elaborate SVM kernel based classifiers and instead use much more primitive but efficient, ME classifiers. In our case *liblinear* actually outperformed *libsvm* in parsing accuracy. This opens up for the advent of transition based, linear time, dependency parsers in applications where time and processing power is a factor, as in the case of predictive text (in particular on embedded devices), where we do not have a lot of time to wait for the possible predictions to be sorted.

The extension of the deterministic parser to consider the best sequences of

	liblinear		libsvm		CONLL-X	
Beam width	LAS	UAS	LAS	UAS	LAS	UAS
1	79.37%	87.67%	78.643%	87.328%	84.58%	89.54%
2	79.63%	87.97%				
4	79.69%	88.05%				
8	79.69%	88.05%				
16	79.70%	88.07%				
32	79.70%	88.08%				
64	79.70%	88.07%				

Table 3.4: Parse results using the *liblinear* and *libsvm* classifier presented in LAS and UAS for different widths of the beam with *liblinear*.

Beam width	liblinear	libsvm
1	1.0	255.5
2	1.2	
4	2.4	
8	5.0	
16	8.4	
32	17.8	
64	34.6	

Table 3.5: Parse times per sentence for *liblinear* and *libsvm*, together with different widths of the beam for *liblinear*.

Beam width	LAS	UAS
1	82.04%	86.41%
2	82.33%	86.69%
4	82.43%	86.81%
8	82.46%	86.83%
16	82.44%	86.80%
32	82.47%	86.82%
64	82.48%	86.82%

Table 3.6: Parse results using Talbanken Malt, with Granska tags without their morphological and lexical features.

Beam width	LAS	UAS
1	79.45%	88.05%
2	79.76%	88.41%
4	79.75%	88.40%
8	79.77%	88.41%
16	79.78%	88.42%
32	79.77%	88.41%
64	79.79%	88.44%

Table 3.7: Parse results using Talbanken with equally much smaller training set.

actions instead of only one action at the time, gave small but not inconsiderable improvements. We have found no evaluations of max entropy classifiers together with beam search for parsing of Swedish, but a similar experiment for Japanese (Sekine et al., 2000) actually achieved worse results when beam search was deployed. It seems that online learning algorithms such as those described by Johansson and Nugues (2007) and Zhang and Clark (2008), are able to benefit much more from the beam search method. A combination of the ME model in *liblinears* efficient implementation, which has just as good classification performance as its SVM cousins, and an online learning method might be something to look into.

The base tags without features from the Granska tag set have proven to be a bit too crude, degrading the UAS by about 1.5%. It is likely that this could be compensated for by carefully choosing relevant features in addition to the base tags.

Further on, investigating which probabilities to consider when measuring the validity of a parse tree should be done. By involving the probabilities of the label assignments as well as the individual parser transition, we should be able to find increases in the LAS as well. Feature set engineering is also something that should be given greater care. In FM2, 98% of the weights are actually 0 and could therefore be discarded and replaced with features that might improve the scores.

Last but not least, we have the area of applications. With linear time parsing and efficient classification, dependency grammars can be employed in applications such as automatic natural text translations, predictive text systems, text categorization, etc.

3.5 Dependency grammars in predictive text

As stated in Section 2.6.1 on Page 12, different variations of grammar have been used in predictive text entry, with varying results. Dependency grammars have also been used in the area of speech recognition and proved to be effective, according to Wang and Harper (2003).

One of the main reasons for using the ME classifier *liblinear* was that it also provided us with probabilities of its suggested transitions. The idea was to use the probability of the parse tree as a measure of how relevant a predicted word was in the current context. Since this model proved somewhat weak, we extended it with an amplification of the probability of the link from the predicted word to its suggested head. The statistics for this dependent/head relationship could then be retrained on SUC, which should provide us with a better model since SUC is a much larger corpus than Talbanken. The final model is

$$\begin{aligned}
 DepV(w) = & \lambda_1 * P_{Parse}(LeftContext, w) + \\
 & + \lambda_2 * P_{Link}(w, Head(w)) + \\
 & + \lambda_3 * P_{Link}(POS(w), POS(Head(w)))
 \end{aligned}
 \tag{3.1}$$

where the last two terms describes $P_{Link}(w, Head(w))$ based on the two features LEX and POS of w and $Head(w)$.

Chapter 4

Experimental setup and results

In this chapter, we will go through our experimental setup and results from the simulations. The simulation system is described in the first section. Further on, models used in the system and how to collect statistics for these are described. At the end of the chapter, results from the experiments are shown.

4.1 System overview

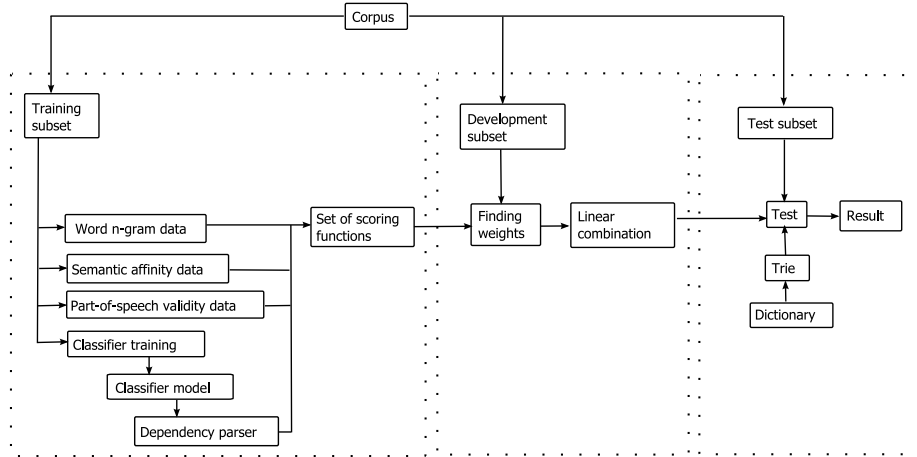
An overview of the system can be seen in Figure 4.1. As depicted by the dotted lines, the system can be divided into three sets representing different stages of the system. In the first stage, the system takes a corpus and uses a subset of this for training.

As noted earlier, it is important to find a corpus that matches the application. However, since there exists no large enough SMS corpus in Swedish we have chosen to use SUC (Ejerhed and Källgren, 1997) because it is the largest available POS tagged Swedish corpus and it is collected from a diverse collection of sources. We believe that if our model performs well on this very general, balanced corpus, it will also perform well on specific areas of the Swedish language, given that a large enough training corpus exists for this area.

SUC, which contains 1,032,494 hand annotated lexical tokens, has been split into a training set (80%), development set (10%), and a test set (10%). The training set was used to gather statistics on word n -grams (for $n \in \{1, 2, 3\}$), POS n -gram (for $n \in \{1, 2, 3, 4, 5\}$), collocations, lemma frequencies, dependent/head relationships. Relationships and sequences only occurring once are not counted. This was done mainly to keep the sizes of the models realistic since keeping the unique occurrence most often led to a better score.

In the second and the final stage a development and test subset of the corpus is used. This is where the simulations are performed. Sentence by sentence is taken from the corpus subsets. For each word, the corresponding keystroke sequence is given to the Trie and a list is returned with words matching the sequence. The set of scoring functions then orders the list so that, ideally, alternatives matching the context come higher up in the list.

The second stage involves using the development set to find suitable weights for the scoring functions. This is done by an exhaustive search through all possible linear combinations in steps of 0.1. The linear combination that results in lowest KPSC will be used in the final stage. Here the test subset of the corpus is used to find KPSC values that are valid and not biased by the development or training subsets.



$$\text{Set of scoring functions } S = \{score_{LM}, score_{SemA}, score_{POSV}, score_{DepV}\}$$

$$\text{Linear combination} = \sum_{s \in S} \lambda_s \cdot s(w)$$

Figure 4.1: Overview of the system.

4.2 The models

The models are the major part of the scoring functions.

Some shorthand;

- The left context of a word is assumed to be present at all times, i.e. $Score(w)$ means the same as $Score(w|LeftContext(w, S))$
- All the individual scores from the different models have been properly normalized, and from here on $Score(w)$ actually refers to

$$\frac{Score(w)}{\sum_{pc \in L} Score(pc)}$$

for all the scoring functions.

- L in the above equation, and everywhere else here onward, refers to the list of possible predictions, and not the entire language.

The frequency model

$$Score_{Freq}(w) = P_{Freq}(w)$$

has been used as in its definition in 2.1 on page 7 with the addition of smoothing with *Laplace's Rule* according to 2.4.

For the LM, we have used linear interpolation of frequencies and bigrams as in

$$Score_{LM}(w) = \alpha * P_{Freq} + \beta * P_{bigram} \quad (4.1)$$

together with *Laplace's Rule*.

Some trials with trigrams were also conducted but with no considerable performance differences and a great overhead in both memory usage and time taken to find the respective weights.

POSV was used together with statistics of POS-5-grams and the same backoff model as described in 2.9.

$$P_{POS}(w|Context) = \begin{cases} P(w_i|w_{i-4}, \dots, w_{i-1}) & \text{if } \exists w_i \in L, C(w_{i-4}, \dots, w_i) \neq 0 \\ P(w_i|w_{i-3}, \dots, w_{i-1}) & \text{if } \exists w_i \in L, C(w_{i-3}, \dots, w_i) \neq 0 \\ \vdots & \\ P(w_i), & \text{otherwise} \end{cases} \quad (4.2)$$

Also the semantic affinity model was used as presented in 2.7 with the modification that lemmas are used instead of stems. This was done partly because using Granska for POS tagging provided us with free lemmatization and partly because it is harder to create a good stemmer for Swedish than for English.

4.3 Introducing completions and next word predictions

Presenting possible word completions, where the system populates the prediction list with words which might not *match* but at least begins with the same key sequence the user has currently pressed, has been investigated numerous times, for example by Hasselgren et al. (2003). By doing this you could actually get $KSPC < 1$. Next word prediction is just a special case of word completion where the completion is presented before the user has pressed any keys at all.

As noted in Section 2.2, one possible way of facilitating word completion is to return the whole subtree under the current node in the ABCTrie. Such an implementation is impractical because when, for example, starting a word with the key 7, representing *p,q,r,s*, about 1/4 of the complete dictionary will be returned. This amount of words is neither sortable nor apprehensible by the user. Another way would be to only present completions that actually match the context we are in, where "matching the context" would be defined in terms of the scoring functions already presented.

When using frequency based disambiguation we use frequency based completions, filling up the prediction list with the most common words that start with the given prefix. This is only done when the user has pressed three keys or more. When using bigram based disambiguation, this is also the information

that we use when looking up completions, first populating the prediction list with words that have been seen following the previous word before, then filling it up with the most frequent unigrams. Again the unigrams are not used until the user has pressed three keys of the word.

When completions are used, the simulated user will choose completions in an optimal way. This means that choosing a completion is done at the point where the sum of the currently entered number of key presses plus the position in the alternatives list is minimized.

4.4 Metrics

The KSPC metric defined in 1.1 on page 3 has to be redefined since the number of characters needed to input a word is now dependent on the word’s left context of the sentence. Let $S = (w_0, \dots, w_n) \in L$ be a sentence in the language. The KSPC for the language then becomes

$$KSPC = \frac{\sum_{S \in L} \sum_{w \in S} KS(w|LeftContext(w, S))}{\sum_{S \in L} \sum_{w \in S} Chars(w)}$$

where $KS(w|LeftContext)$ is the number of key strokes needed to enter a word in a given context, $LeftContext(w, S)$ is the left context of w in S and $Chars(w)$ is the number of characters in w .

Another performance measure is the disambiguation accuracy (DA), which is the percentage of words that are correctly disambiguated after all the keys have been pressed

$$DA = \frac{\sum_{S \in L} \sum_{w \in S} PredictionHit(w|LeftContext(w, S))}{|L|}$$

where

$$PredictionHit(w|Context) = \begin{cases} 1 & \text{if } w \text{ is the top prediction} \\ 0 & \text{otherwise} \end{cases}$$

A good DA means that you can more often simply accept the default proposed word instead of having to scan the prediction list for your desired word.

4.5 Simulations

It has now become infeasible to calculate the KSPC in any other way than actually running a simulation where we input sentence after sentence, word after word, which is exactly what we have done. The results are presented in the following sections, divided into results where we enforce pressing 1 key per character, to see how close we will get to the 1, and another, where we utilise completions to see how well our scoring functions perform when they have many more candidates to choose from.

The simulations can be divided into two sets, one with completions and one without. We compare the results from the baseline, frequency based disambiguation, with different combinations of the scoring functions. We have also

considered improvements over the bigrams model separately. The configurations are listed and labeled in Tables 4.1 and 4.3.

As scoring tokens we have chosen to keep the ones that actually have the ability to differentiate the models, i.e. we have not counted the KSPC and DA for words that are not present in our dictionary and we have not counted white spaces and punctuation marks which will always be presented in the same position in the prediction list.

In the case where completions were used, words that have the same length as the entered key sequence are to promoted by adding a constant to their score. This actually degraded the total KSPC performance, but we found it unlikely that any user would accept that the 3-character, intended word would be found in the 10th position of the prediction list just because longer words seemed more relevant in the context.

4.6 Results

Comparing results between simulations not using and using completions should be done to see the advantage of completion usage, but also demonstrate whether the scoring functions perform better or worse in the specific setting.

4.6.1 Without completions

The KSPC decrease is quite small but since we are so close to 1 from the beginning, we must not be blinded by this. Instead we choose to study the KSPC Error Reduction Rate (ERR), i.e. the number of extra key presses needed, beyond the 1 that is of course needed when we don't utilize completions. All optimizations has been done considering KSPC but we can also observe that the KSPC ERR is closely related to the DA ERR.

Table 4.1 presents the different combinations of scoring functions that have been used, together with their respective optimal weights. The DepV weights are in order, the weight for P_{Parse} , the weight for the LEX feature of P_{Link} and the weight of the POS feature of P_{Link} .

Table 4.2 shows KSPC and DA together with KSPC decrease, KSPC ERR and DA ERR compared with the baseline.

Table 4.3 shows the different combinations of scoring functions that have been used together with the bigram language model and their respective optimal weights. In addition to the DepV weights this table also shows the Bigram weights, which are the α and β from 4.1.

Table 4.4 shows the respective values when using a bigram based disambiguation instead of just frequency. The ERR is still compared to the baseline but attention should also be put on the relative increases, how much the new models can improve bigram based disambiguation.

As can be seen, we have not stuck to the 0.1 steps when searching for weights. This was because it turned out to be too coarse grained to find a set of weights that actually improved the scores. For example removing 0.1 from *Bigrams* and adding 0.1 of *SemA* degraded the score more than it helped.

Configuration	Score model	DepV weights
F1	1*Freq	-
F2	0.9*Freq + 0.1*POSV	-
F3	0.7*Freq + 0.3*SemA	-
F4	0.6*Freq + 0.4*DepV	(0.3, 0.7, 0.0)
F5	0.6*Freq + 0.1*POSV + 0.3*DepV	(0.0 1.0 0.0)
F6	0.5*Freq + 0.2*SemA + 0.3*DepV	(0.2 0.7 0.1)
F7	0.4*Freq + 0*1*POSV + 0.3*DepV + 0.2*SemA	(0.2, 0.8, 0.0)

Table 4.1: The different combinations of score models using frequency based disambiguation in bottom.

Configuration	KSPC	DA	KSPC decrease	KSPC ERR	DA ERR
F1	1.015559	94.15%	0.00%	0.00%	0.00%
F2	1.014829	94.31%	0.07%	4.69%	2.72%
F3	1.014902	94.36%	0.06%	4.22%	3.62%
F4	1.014462	94.56%	0.11%	7.05%	7.04%
F5	1.013625	94.75%	0.19%	12.43%	10.28%
F6	1.014159	94.62%	0.14%	9.00%	8.10%
F7	1.013438	94.86%	0.21%	13.63%	12.16%

Table 4.2: Results for the disambiguation based on word frequency plus the semantic and syntactic models.

Configuration	Score model	Bigram weights	DepV weights
B1	1*Bigram	(0.1, 0.9)	-
B2	0.9*Bigram + 0.1*POSV	(0.2, 0.8)	-
B3	0.95*Bigram + 0.05*SemA	(0.2, 0.8)	-
B4	0.9*Bigram + 0.1*DepV	(0.2, 0.8)	(0.2, 0.8, 0.0)
B5	0.8*Bigram + 0.1*POSV + 0.1*SemA	(0.2, 0.8)	-
B6	0.81*Bigram + 0.08*POSV + 0.11*DepV	(0.2, 0.8)	(0.2, 0.8, 0.0)

Table 4.3: The different combinations of score models using bigram based disambiguation in bottom.

Label	KSPC	DA	KSPC decrease	KSPC ERR	DA ERR
B1	1.012159254	95.48%	0.33%	21.85%	22.81%
B2	1.011434213	95.75%	0.41%	26.51%	27.41%
B3	1.011860573	95.50%	0.36%	23.77%	23.20%
B4	1.011698693	95.62%	0.38%	24.81%	25.19%
B5	1.011146932	95.80%	0.43%	28.36%	28.23%
B6	1.010980592	95.91%	0.45%	29.43%	30.09%

Table 4.4: Results for the disambiguation based on bigrams plus the semantic and syntactic models.

4.6.2 With completions

When testing our input method with completions, the same configurations as in the previous section are used, i.e. the weights of the different models are not optimized for completions. Configuration F1C refers to configuration F1 in Table 4.1 but with a completion setup outlined in Section 2.2. The measures that are presented are all based on KSPC and the ones except for KSPC are only presented to help the interpretation. KSR is the key stroke savings rate compared to the baseline, i.e. how many key presses the user do not use thanks to the completions. KSPC decrease is compared to the baseline with completions to highlight the fact that the models are more helpful when they have additional candidates to choose from.

DA1 is the average amount of key presses needed to get the word into the first position of the prediction list and DA5 is the corresponding amount needed to get the word into top 5 of the prediction list, given that the words ever reach the first position or top 5 respectively. Both of these measures are compared to the baseline, configuration F1. The amount of words that ever reach first position is the same as DA in the previous experiment. The amount of words that ever reach top 5 is > 99.8% for all combinations of models.

Configuration	KSPC	KSR	KSPC decrease	DA1	DA5
F1C	0.872714	14.07%	0.0000%	89.10%	73.56%
F2C	0.864478	14.88%	0.9436%	88.47%	72.61%
F3C	0.868345	14.50%	0.5005%	88.92%	73.19%
F4C	0.865657	14.76%	0.8086%	88.67%	73.57%
F5C	0.858134	15.50%	1.6706%	87.95%	72.20%
F6C	0.8633	14.99%	1.0787%	88.50%	72.77%
F7C	0.854684	15.84%	2.0659%	87.68%	72.01%

Table 4.5: Results with completions and word frequencies.

Configuration	KSPC	KSR	KSPC decrease	DA1	DA5
B1C	0.792678	21.95%	9.17%	80.10%	56.57%
B2C	0.789895	22.22%	9.49%	78.85%	55.63%
B3C	0.791415	22.07%	9.32%	80.36%	56.48%
B4C	0.789848	22.23%	9.50%	80.17%	56.34%
B5C	0.787413	22.47%	9.77%	79.01%	55.53%
B6C	0.778567	23.34%	10.79%	79.03%	55.54%

Table 4.6: Results with completions and bigrams.

Chapter 5

Discussion

This chapter will discuss the results presented in the previous chapter. We will also discuss complications during the work and how our work can be improved further.

5.1 About the results

We have shown that a simple model based on dependency grammars can improve the prediction considerably. The DepV model is actually the most effective one when applied together with the frequency counts. Furthermore, the improvements from the POSV, SemA, and DepV model are almost disjunct, in that the sum of their respective improvements is achieved when combining them. However, we have not been able to identify any specific features that the different models contribute.

The 4.2% ERR observed when adding the SemA model is consistent with the result from Gong et al. (2008), where a 4.6% ERR was found. The POSV model on the other hand, only contributed with a 4.7% ERR in our case, whereas Gong et al. (2008) observed an 12.6% ERR. One possible explanation for this discrepancy is that they grouped closely related POS tags into 19 groups. By performing this grouping, you can effectively ignore morphological and lexical features that have no relevance, when deciding which word should come next. Other possible explanations include that our backoff model is not well suited for this problem or that the POS sequences are not an applicable model for Swedish.

Quite extraordinary is that the language model, bigrams, has such a large impact on the performance. The ERR for bigrams alone is higher than for all the other, in some sense more sophisticated, models combined. This will probably not hold for all languages and for the ones which have a much freer word order than Swedish, the DepV and SemA models might be preferable.

Even though bigrams are superior to the other models, the other models still have the ability to contribute on top of the bigrams model. For example POSV increases the ERR about 5% both when using bigram- and frequency based disambiguation, suggesting that this information is not at all captured in the bigrams model. On the other hand, DepV only increases the ERR by about 3% when using bigrams instead of the 7% noted before. This is likely due to

that about 50% of the dependency links only stretch to the next preceding or succeeding word (in Talbanken).

The most effective combination of models that we have found, is bigrams together with the implicit, POS sequence, grammar and the explicit, dependency structure, grammar. With this combination, we are able to reduce the number of erroneous disambiguations by almost one third and together with this, reduce the number of extra key presses needed to enter a word by the same amount.

It should be noted that although we have not conducted any significance tests, the figures from the development set and the test set are coherent.

5.2 Complications

Even if SUC is a very diverse corpus reflecting many areas of Swedish text, it exclusively represents written texts such as news paper articles, scientific articles, memoirs and prose. Since we know that written and spoken language differs a lot, it remains an open question if good performance on these types of texts would also imply good prediction results on the more verbally based SMS texting language.

The dependency parser is trained for parsing sentences that are complete, which is not the case in our application. A parser trained for sentences that are incomplete might perform better, or has the ability to decide whether the suggested word actually has a head in the current context. This also gives rise to the idea to dynamically adjust the weight depending on how far into the sentence we are.

5.3 Future work

As already mentioned the results obtained do not only apply to text input on small devices, but to all areas where the redundancy of natural language can be utilized. Examples are, speech recognition, communication accessories for disabled people, etc. That is, there are a lot of areas where the use of these combined models should be investigated. Still, there is a lot to do in improving the models also.

Everywhere where smoothing with *Laplace's Rule* is used, some more clever method like Good-Turing estimation should be used and would probably improve the results.

The DepV model is rather primitive and better use can probably be made of both the dependency arcs and their labels.

A similar grouping of related POS tags with features, as mentioned above, should definitely be looked into. Finding optimal such groupings would require a big effort and another approach would be to consider all the features and POS tags by them self. For example looking at sequences the *definite* or *indefinite* feature. Yet another approach would be to could a decision tree classifier, that based on the POS and all the features considered separately, decides which POS with which features is more likely to come next.

Further on, studies on how the scoring functions could be applied to the text input method in a mobile phone, have to be done in a give and take manner, using approximations of the models, so that memory and CPU consumption is

kept at a reasonable level. Because it has not been the main objective in this study to keep the models sizes down we can not give any accurate figures. The POSV model is estimated to a size of 700KB according to Gong et al. (2008). The DepV can be made as small as the feature model, and the accuracy of FM1 presented in Table 3.3 might actually be enough for this task. This model is also about 700KB big and computational power required by *liblinear* is certainly something that might be possible to run on a modern hand held device.

The semantic affinity, how well a predicted word fits into the current context according to semantic relatedness with the context, might be improved by looking at different, possibly larger contexts. A natural approach would be to consider the SMS the user is currently writing. Two problems with this is that it will require greater computational power, since its running time is linear relative to the number of words considered. Another and greater problem is that we require a whole new type of training data consisting of small contexts such as few-sentence paragraphs, or preferably SMSs.

As we mentioned in the introductory section, the actual perceived difference for a user might be subtle but however subtle, it is still there. Reports about a reluctance to use the single-tap input methods because of their “unpredictable” behaviour has been made (Gutowitz, 2003) and introducing context aware disambiguation certainly will enhance this perception of “unpredictability”. To get a feel of how well this feature will be received on the market, quantitative usability studies must be carried out.

The same author also concluded that 36% of the interviewees did not use dictionary based input methods because they presented wrong, unwanted, strange or stupid words. Another 13% stated that they did not use this methods because the dictionary was too small. These are two problems that both might be remedied with our system.

5.4 Conclusion

We have implemented a set of models that describe features of the Swedish language. The models are used to disambiguate key stroke sequences. The results of this work is twofold. We have confirmed that bigrams actually are outstanding and superior to all the other models we have evaluated. Despite this fact, the semantic and grammatical models can still be used to improve the performance of text prediction even further.

We have also shown that we can reach close to state-of-the-art results in dependency parsing with an efficient maximum entropy classifier and manually combined features. This parser has then been used to achieve disambiguation results comparable to the semantic model of Gong et al. (2008). In contrast to the semantic model, the syntactic model can be greatly optimized in terms of memory footprint and should therefor be preferred in situations where the resources are limited.

Bibliography

- J. Arnott, J.M. Hannan, and R.J. Woodburn. Linguistic prediction for disabled users of computer-mediated communication. In *Proceedings of the ECART2 Conference*, 1993.
- Stuart K. Card, Thomas P. Moran, and Allen Newell. The keystroke-level model for user performance time with interactive systems. *Communications of the ACM*, 23(7):396–410, 1980. ISSN 0001-0782.
- Johan Carlberger, Rickard Domeij, Viggo Kann, and Ola Knutsson. A Swedish grammar checker. 2002.
- Noam Chomsky. Syntactic structures. *Linguistic Society of America*, 1957.
- Micheal Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, pages 1–8, Philadelphia, USA, 2002.
- Eva Ejerhed and Gunnel Källgren. Stockholm umeå corpus version 1.0, suc 1.0, 1997.
- Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Wang, Xiang-Rui and Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.
- Kilian A. Foth, Michael Daum, and Wolfgang Menzel. A broad-coverage parser for German based on defeasible constraints. In *In KONVENS 2004, Beiträge zur 7. Konferenz zur Verarbeitung natürlicher Sprache*, pages 45–52, 2004.
- Edward Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, 1960. ISSN 0001-0782.
- Jun Gong and Peter Tarasewich. A new error metric for text entry method evaluation. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 471–474, New York, NY, USA, 2006. ACM. ISBN 1-59593-372-7.
- Jun Gong, Peter Tarasewich, and I. Scott MacKenzie. Improved word list ordering for text entry on ambiguous keypads. In *NordiCHI '08: Proceedings of the 5th Nordic conference on Human-computer interaction*, pages 152–161, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-704-9. doi: <http://doi.acm.org/10.1145/1463160.1463177>.

- Ebba Gustavii and Eva Pettersson. A Swedish grammar for word prediction. Technical report, Department of Linguistics, Uppsala University, 2003.
- Howard Gutowitz. Barriers to adoption of dictionary-based text-entry methods; a field study. In *EACL 2003 Workshop on Language Modeling for Text Entry Systems*, pages 33–41, 2003.
- Jon Hasselgren, Erik Montnemery, Pierre Nugues, and Markus Svensson. HMS: A predictive text entry method using bigrams. In *Association for Computational Linguistics*, pages 43–49, 2003.
- David Hays. Dependency theory: A formalism and some observations. *Language*, 40:511–525, 1964.
- Richard A. Hudson. *English Word Grammar*. Blackwell, 1990.
- Richard Johansson and Pierre Nugues. Incremental dependency parsing using online learning. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, Prague, Czech Republic, June 28-30 2007.
- Slava M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 35(3):400–401, Mar 1987. ISSN 0096-3518.
- S. Sathiya Keerthi, S. Sundararajan, Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. A sequential dual method for large scale multi-class linear SVMs. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 408–416, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-193-4.
- Pierre Simon de Laplace. *Théorie analytique des probabilités*. 1812.
- Jianhua Li and Graeme Hirst. Semantic knowledge in word completion. In *Assets '05: Proceedings of the 7th international ACM SIGACCESS conference on Computers and accessibility*, pages 121–128, New York, NY, USA, 2005. ACM. ISBN 1-59593-159-7.
- I. Scott MacKenzie. KSPC (Key Strokes Per Character) as a characteristic of text entry techniques. pages 195–210. Springer-Verlag, 2002.
- Johannes Matiasek. The language component of the FASTY predictive typing system. In Karin Harbusch, Kari-Jouko Raiha, and Kumiko Tanaka-Ishii, editors, *Efficient Text Entry*, number 05382 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.
- Johannes Matiasek, Marco Baroni, and Harald Trost. FASTY - A multi-lingual approach to text prediction. In *ICCHP '02: Proceedings of the 8th International Conference on Computers Helping People with Special Needs*, pages 243–250, London, UK, 2002. Springer-Verlag. ISBN 3-540-43904-8.
- Arpit Mittal and Arijit Sengupta. Optimized layout for keypad entry system. In Hamid R. Arabnia and Laurence Tianruo Yang, editors, *ESA*, pages 127–133. CSREA Press, 2007. ISBN 1-60132-052-3.

- Joakim Nivre. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160, 2003.
- Joakim Nivre. Dependency grammar and dependency parsing. Technical report, Växjö University, 2005.
- Joakim Nivre, Jens Nilsson, and Johan Hall. Talbanken05, 2005. URL <http://www.msi.vxu.se/users/nivre/research/Talbanken05.html>.
- Joakim Nivre, Johan Hall, Jens Nilsson, Gülsen Eryigit, and Svetoslav Marinov. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 221–225, June 2006. URL <http://acl.ldc.upenn.edu/W/W06/W06-2933.pdf>.
- John Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. ISSN 0885-6125.
- Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 6(65):386–408, 1958.
- Satoshi Sekine, Kiyotaka Uchimoto, and Hitoshi Isahara. Backward beam search algorithm for dependency analysis of Japanese. In *Proceedings of the 18th conference on Computational linguistics*, pages 754–760, Morristown, NJ, USA, 2000. Association for Computational Linguistics.
- Claude Elwood Shannon. Prediction and entropy of printed English. *The Bell System Technical Journal*, pages 50–64, January 1951.
- K. Sundarakantham and S. Mercy Shalinie. Word predictor using natural language grammar induction technique. *Journal of Theoretical and Applied Information Technology*, 3:1–8, 2007. ISSN 19928645.
- L. Tesnière. *Éléments de syntaxe structurale*. Klincksieck, Paris, 1959.
- S. Tyvand and P. Demasco. Syntax statistics in word prediction. In *Proceedings of the ECART2 Conference*, 1993.
- Vladimir N. Vapnik, Bernhard E. Boser, and Isabelle M. Guyon. A training algorithm for optimal margin classifiers. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, New York, NY, USA, 1992. ACM. ISBN 0-89791-497-X. doi: <http://doi.acm.org/10.1145/130385.130401>.
- Wen Wang and Mary P. Harper. Language modeling using a statistical dependency grammar parser. *Automatic Speech Recognition and Understanding, 2003. ASRU '03. 2003 IEEE Workshop on*, pages 519–524, Nov.-3 Dec. 2003.
- Yue Zhang and Stephen Clark. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of EMNLP*, Hawaii, USA, October 2008.

Appendix A

Dependency parsing

A.1 Feature model syntax

The feature model syntax used is a slight modification of the one used for MaltParser¹. We have removed some functionality that we did not actually use and added support for composition of several features into one, to make up for the “linear-classifier-weakness.” Its grammar and explanations of its tokens are described below.

```
<fspec> ::= <feat>+

<feature> ::= <simplefeat> | <simplefeat> ";" " <feature>

<simplefeat> ::= <lfeat> | <nfeat>

<lfeat> ::= (LEX) \t <dstruc> \t <offs>

<nfeat> ::= (POS|DEP) \t <dstruc> \t <offs>

<dstruc> ::= (STACK|INPUT)

<offs> ::= <nnint> \t <int> \t <nnint> \t <int> \t <int>

<int> ::= (...|-2|-1|0|1|2|...)

<nnint> ::= (0|1|2|...)
```

Figure A.1: The feature model grammar.

The features are defined in terms of a data type, source location and offset. The data type can be either a part of speech, lexical value (being the word itself) or a dependency type. The source of the feature can be either the input queue or the stack. The offset is defined in terms of 5 integers a, b, c, d, e which in order represent

¹<http://w3.msi.vxu.se/~nivre/research/MaltParser.html>

- a steps to the left in the input queue or a steps down in the stack
- b steps to the right in the original sentence if $b > 0$ and $|b|$ steps to the left if $b < 0$
- c applications of the HEAD function
- d applications of the RC function if $d > 0$ and $|d|$ applications of the LC function if $d < 0$
- e application of the RS function if $e > 0$ and $|e|$ applications of the LS function if $e < 0$

The offsets are applied in the order they are presented above, thus
 POS INPUT 0 0 1 1 1 represents the part of speech of the rightmost sibling of
 the rightmost child of the head of the first token in the queue.

A.2 Feature models

LEX INPUT
LEX INPUT 1
LEX INPUT; POS INPUT 1; POS INPUT 2
LEX STACK
LEX STACK; LEX INPUT
POS INPUT
POS INPUT 1
POS INPUT 1; LEX INPUT 1
POS INPUT 1; POS INPUT
POS INPUT; LEX INPUT
POS INPUT; POS INPUT 1; POS INPUT 2
POS STACK
POS STACK 0 0 1; POS STACK; POS INPUT
POS STACK 1; POS INPUT
POS STACK 1; POS INPUT 1
POS STACK 1; POS STACK
POS STACK; LEX INPUT; POS INPUT
POS STACK; LEX INPUT; POS INPUT 0 0 0 -1
POS STACK; LEX STACK
POS STACK; POS INPUT
POS STACK; POS INPUT 1
POS STACK; POS INPUT 1; POS INPUT
POS STACK; POS INPUT; POS INPUT 0 0 0 -1
POS STACK; POS STACK 0 0 0 1; LEX INPUT
POS STACK; POS STACK 0 0 0 1; POS INPUT

Table A.1: FM2, our feature model for predicting parser actions, with explicitly combined features to fit the weakness of a linear classifier.

POS INPUT
 POS INPUT; LEX INPUT
 POS INPUT; POS INPUT 1; POS STACK
 POS INPUT 0 0 0 -1
 POS INPUT 0 0 0 -1
 POS INPUT 1
 POS INPUT 1; LEX INPUT 1
 POS INPUT 1; POS INPUT
 POS INPUT 1; POS STACK
 POS INPUT 1; POS STACK 1
 POS STACK
 POS STACK; POS INPUT
 POS STACK 0 0 0 1
 POS STACK 1; POS INPUT
 POS STACK 1; POS INPUT 1; DEP INPUT 0 0 0 1
 LEX INPUT
 LEX INPUT 1
 LEX STACK
 LEX STACK; DEP INPUT 0 0 0 1
 DEP INPUT 0 0 0 1; POS STACK
 DEP STACK 0 0 0 1; LEX STACK
 DEP STACK 0 0 0 1; POS INPUT

Table A.2: FM3, our feature model for predicting labels

Appendix B

Word list

ABCTrie A modified *Trie* where each node represent a button on the keypad instead of a character in the word.

Annotated corpus Corpus where each word is associated with meta information.

Completion Words that complete a sequence of characters.

Context The set of facts or circumstances that surround a situation or event. In this paper the context is the current sentence.

Corpus Large collection of natural language text.

Disambiguation accuracy Metric saying how often the method of disambiguation places the intended word in the top of the prediction list.

Dependency validity (DepV) Scoring function saying how probable a word is in a context, according to a model based on syntax.

Grammar Studies of the rules that make up the structure of natural language text.

Granska System that, among other things, annotates a sentence with POS tags.

Key Strokes Per Character (KSPC) Metric saying how efficient an input method is.

Language model Model based on word N -grams.

Lemma Base form of a word.

liblinear Library for efficient large scale linear classification

libsvm Library for support vector machine classification.

Multi-tap Unambiguous input method for 12-button keypad. Every press on a key cycles through the corresponding letters on that key.

N -best search Graph search that in every step saves the N best partial solutions and discards the rest.

***N*-gram** Sequence of N something, usually words.

Part of speech (POS) Classification of words into categories according to how the words are used in a sentence.

Part of speech validity (POSV) Scoring function saying how probable a word is in a context, according to a model based on POS n-grams.

Semantic affinity (SemA) Scoring function saying how probable a word is in a context, according to a model based on semantics.

Semantics The perceived meaning of a lexical unit such as a word or sentence.

Single-tap Ambiguous input method for 12-button keypad. Input is done by pressing the key that corresponds to the intended character.

Stem The part of a word that is common to all its inflected variants.

Syntax Studies about the structure of natural language text.

Trie Tree structure for storing words of a dictionary by letting each node represent a character in a word.