Predictive Text Entry using Syntax and Semantics

Sebastian Ganslandt **Jakob Jörwall** **Pierre Nugues**
Department of Computer Science
Lund University
S-221 00 Lund, Sweden
sebastian@ganslandt.nu pierre.nugues@cs.lth.se
d02jjr@student.lth.se

Abstract

Most cellular telephones use numeric keypads, where texting is supported by dictionaries and frequency models. Given a key sequence, the entry system recognizes the matching words and proposes a rank-ordered list of candidates. The ranking quality is instrumental to an effective entry.

This paper describes a new method to enhance entry that combines syntax and language models. We first investigate components to improve the ranking step: language models and semantic relatedness. We then introduce a novel syntactic model to capture the word context, optimize ranking, and then reduce the number of keystrokes per character (KSPC) needed to write a text. We finally combine this model with the other components and we discuss the results.

We show that our syntax-based model reaches an error reduction in KSPC of 12.4% on a Swedish corpus over a baseline using word frequencies. We also show that bigrams are superior to all the other models. However, bigrams have a memory footprint that is unfit for most devices. Nonetheless, bigrams can be further improved by the addition of syntactic models with an error reduction that reaches 29.4%.

1 Introduction

The 12-key input is the most common keypad layout on cellular telephones. It divides the alphabet into eight lists of characters and each list is mapped onto one key as shown in Figure 1. Since three or four characters are assigned to a key, a single key press is ambiguous.

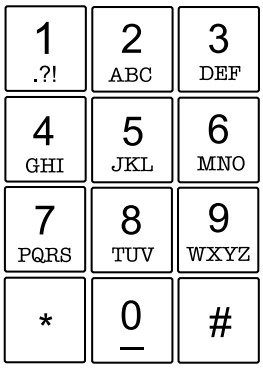

Figure 1: Standard 12-button keypad layout (ISO 9995-8).

1.1 Multi-tap

Multi-tap is an elementary method to disambiguate input for a 12-button keypad. Each character on a key is assigned an index that corresponds to its visual position, e.g. 'A', 1, 'B', 2, and 'C', 3 and each consecutive stroke – tap – on the same key increments the index. When the user wants to type a letter, s/he presses the corresponding key until the desired index is reached. The user then presses another key or waits a predefined time to verify that the correct letter is selected. The key sequence 8-4-4-3-3, for example, 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, notably when two consecutive characters are placed on the same key.

1.2 Single Tap with Predictive Text

Single tap with predictive text requires only one key press to enter a character. Given a keystroke sequence, the system proposes words using a dictionary or language modeling techniques.

Dictionary-based techniques search the words matching the key sequence in a list that is stored by the system (Haestrup, 2001). While some

Proceedings of the 11th International Conference on Parsing Technologies (IWPT), pages 37–48,
Paris, October 2009.

keystroke sequences produce a unique word, others are ambiguous and the system returns a list with all the candidates. The key sequence 8-4-3, for example, corresponds to at least three possible words: *the*, *tie*, and *vie*. The list of candidates is then sorted according to certain criteria, such as the word or character frequencies. If the word does not exist in the dictionary, the user has to fall back to multi-tap to enter it. The T9[1] commercial product is an example of a dictionary-based system (Grover et al., 1998).

LetterWise (MacKenzie et al., 2001) is a technique that uses letter trigrams and their frequencies to predict the next character. For example, pressing the key 3 after the letter bigram 'th' will select 'e', because the trigram 'the' is far more frequent than 'thd' or 'thf' in English. When the system proposes a wrong letter, the user can access the next most likely one by pressing a *next-key*. LetterWise does not need a dictionary and has a $KSPC$ of 1.1500 (MacKenzie, 2002).

1.3 Modeling the Context

Language modeling can extend the context from letter sequences to word n-grams. In this case, the system is not restricted to the disambiguation or the prediction of the typed characters. It can complete words and even predict phrases. HMS (Hasselgren et al., 2003) is an example of this that uses word bigrams in Swedish. It reports a $KSPC$ ranging from 0.8807 to 1.0108, depending on the type of text. eZiText[2] is a commercial example of a word and phrase completion system. However, having a large lexicon of bigrams still exceeds the memory capacity of many mobile devices.

Some systems use a combination of syntactic and semantic information to model the context. Gong et al. (2008) is a recent example that uses word frequencies, a part-of-speech language model, and a semantic relatedness metric. The part-of-speech language model acts as a lexical n-gram language model, but occupies much less memory since the vocabulary is restricted to the part-of-speech tagset. The semantic relatedness, modified from Li and Hirst (2005), is defined as the conditional probability of two stems appearing in the same context (the same sentence):

$$SemR(w_1|w_2) = \frac{C(stem(w_1), stem(w_2))}{C(w_2)}.$$

The three components are combined linearly and their coefficients are adjusted using a development set. Setting 1 as the limit of the $KSPC$ figure, Gong et al. (2008) reported an error reduction over the word frequency baseline of 4.6% for the semantic model, 12.6% for the part-of-speech language model, and 15.8% for the combination of both.

1.4 Syntax in Predictive Text

Beyond part-of-speech language modeling, there are few examples of systems using syntax in predictive text entry. Matiasek et al. (2002) describes a predictive text environment aimed at disabled persons, which originally relied on language models. Gustavii and Pettersson (2003) added a syntactic component to it based on grammar rules. The rules corresponded to common grammatical errors and were used to rerank the list of candidate words. The evaluation results were disappointing and the syntactic component was not added because of the large overhead it introduced (Matiasek, 2006).

In the same vein, Sundarkantham and Shalinie (2007) used grammar rules to discard infeasible grammatical constructions. The authors evaluated their system by giving it an incomplete sentence and seeing how often the system correctly guessed the next word (Shannon, 1951). They achieved better results than previously reported, although their system has not been used in the context of predictive text entry for mobile devices.

2 Predictive Text Entry Using Syntax

We propose a new technique that makes use of a syntactic component to model the word context and improve the $KSPC$ figure. It builds on Gong et al. (2008)'s system and combines a dependency grammar model with word frequencies, a part-of-speech language model, and the semantic relatedness defined in Sect. 1.3. As far as we are aware, no predictive text entry system has yet used a data-driven syntactic model of the context.

We used Swedish as our target language all over our experiments, but the results we obtained should be replicable in any other language.

[1] www.t9.com

[2] www.zicorp.com/ezitext.htm

2.1 Reranking Candidate Words

The system consists of two components. The first one disambiguates the typed characters using a dictionary and produces a list of candidate words. The second component reranks the candidate list. Although the techniques we describe could be applied to word completion, we set aside this aspect in this paper.

More formally, we frame text input as a sequence of keystrokes, $\mathbf{ks}^i = ks_1^i \dots ks_n^i$, to enter a desired word, w_i. The words matching the key sequence in the system dictionary form an ordered set of alternatives, $match(\mathbf{ks}^i) = \{cw_0, \dots, cw_m\}$, where it takes k extra keystrokes to reach candidate cw_k. Using our example in Sect. 1.2, a lexical ordering would yield $match(8-4-3) = \{the, tie, vie\}$, where two extra keystrokes are needed to reach *vie*.

We assign each candidate word w member of $match(\mathbf{ks}^i)$ a score

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

to rerank (sort) the prediction list, where s is a scoring function from a set S, λ_s, the weight of s, and $Score(w|Context)$, the total score of w in the current context.

In this framework, optimizing predictive text entry is the task of finding the scoring functions, s, and the weights, λ_s, so that they minimize k on average.

As scoring functions, we considered lexical language models in the form of unigrams and bigrams, s_{LM1} and s_{LM2}, a part-of-speech model using sequences of part-of-speech tags of a length of up to five tags, s_{POS}, and a semantic affinity, s_{SemA}, derived from the semantic relatedness. In addition, we introduce a syntactic component in the form of a data-driven dependency syntax, s_{DepSyn} so that the complete scoring set consists of

$$S = \{s_{LM1}, s_{LM2}, s_{SemA}, s_{POS}, s_{DepSyn}\}.$$

2.2 Language and Part-of-Speech Models

The language model score is the probability of a candidate word w, knowing the sequence entered so far, $w_1, \dots, w_i$:

$$P(w|w_1, w_2, \dots, w_i).$$

We approximate it using unigrams, $s_{LM1}(w) = P(w)$, or bigrams, $s_{LM2}(w) = P(w|w_i)$ that we derive from a corpus using the maximum likelihood estimate. To cope with sparse data, we used a deleted interpolation so that $s_{LM2}(w) = \beta_1 P(w|w_i) + \beta_2 P(w)$, where we adjusted the values of β_1 and β_2 on a development corpus.

In practice, it is impossible to maintain a large list of bigrams on cellular telephones as it would exceed the available memory of most devices. In our experiments, the s_{LM2} score serves as an indicator of an upper-limit performance, while s_{LM1} serves as a baseline, as it is used in commercial dictionary-based products.

Part-of-speech models offer an interesting alternative to lexical models as the number of parts of speech does not exceed 100 tags in most languages. The possible number of bigrams is then at most 10,000 and much less in practice. We defined the part-of-speech model score, s_{POS} as

$$P(t|t_1, t_2, \dots, t_i),$$

where t_i is the part of speech of w_i and t, the part of speech of the candidate word w. We used a 5-gram approximation of this probability with a simple back-off model:

$$s_{POS} = \begin{cases} P(t|t_{i-3}, \dots, t_i) & \text{if } C(t_{i-3}, ..., t_i) \neq 0 \\ P(t|t_{i-2}, \dots, t_i) & \text{if } C(t_{i-2}, ..., t_i) \neq 0 \\ \vdots & \\ P(t), & \text{otherwise} \end{cases}$$

We used the Granska tagger (Carlberger and Kann, 1999) to carry out the part-of-speech annotation of the word sequence.

3 Semantic Affinity

Because of their arbitrary length, language models miss possible relations between words that are semantically connected in a sentence but within a distance greater than one, two, or three words apart, the practical length of most n-grams models. Li and Hirst (2005) introduced the semantic relatedness between two words to measure such relations within a sentence. They defined it as

$$SemR(w_i, w_j) = \frac{C(w_i, w_j)}{C(w_i)C(w_j)},$$

where $C(w_i, w_j)$ is the number of times the words w_i and w_j co-occur in a sentence in the corpus,

and $C(w_i)$ is the count of word w_i in the corpus. The relation is symmetrical, i.e.

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

The estimated semantic affinity of a word w is defined as:

$$SemA(w|H) = \sum_{w_j \in H} SemR(w, w_j),$$

where H is the context of the word w. In our case, H consists of words to the left of the current word.

Gong et al. (2008) used a similar model in a predictive text application with a slight modification to the $SemR$ function:

$$SemR(w_i, w_j) = \frac{C(stem(w_i), stem(w_j))}{C(stem(w_j))},$$

where the $stem(w)$ function removes suffixes from words. We refined this model further and we replaced the stemming function with a real lemmatization.

4 Dependency Parsing

Dependency syntax (Tesnière, 1966) has attracted a considerable interest in the recent years, spurred by the availability of data-driven parsers as well as annotated data in multiple languages including Arabic, Chinese, Czech, English, German, Japanese, Portuguese, or Spanish (Buchholz and Marsi, 2006; Nivre et al., 2007). We used this syntactic formalism because of its availability in many languages.

4.1 Parser Implementation

There are two main classes of data-driven dependency parsers: graph-based (McDonald and Pereira, 2006) and transition-based (Nivre, 2003). We selected Nivre's parser because of its implementation simplicity, small memory footprint, and linear time complexity. Parsing is always achieved in at most $2n - 1$ actions, where n is the length of the sentence. Both types of parser can be combined, see Zhang and Clark (2008) for a discussion.

Nivre's parser is an extension to the shift–reduce algorithm that creates a projective and acyclic graph. It uses a stack, a list of input words, and builds a set of arcs representing the graph of dependencies. The parser uses two operations in addition to shift and reduce, left-arc and right-arc:

- $Shift$ pushes the next input word onto the stack.
- $Reduce$ pops the top of the stack with the condition that the corresponding word has a head.
- $LeftArc$ adds an arc from the next input word to the top of the stack and pops it.
- $RightArc$ adds an arc from the top of the stack to the next input word and pushes the input word on the stack.

Table 1 shows the start and final parser states as well as the four transitions and their conditions and Algorithm 1 describes the parsing algorithm.

4.2 Features

At each step of the parsing procedure, the parser turns to a guide to decide on which transition to apply among the set $\{LeftArc, RightArc, Shift, Reduce\}$. We implemented this guide as a four-class classifier that uses features it extracts from the parser state. The features consist of words and their parts of speech in the stack, in the queue, and in the partial graph resulting from what has been parsed so far. The classifier is based on a linear logistic regression function that evaluates the transition probabilities from the features and predicts the next one.

In the learning phase, we extracted a data set of feature vectors using the gold-standard parsing procedure (Algorithm 2) that we applied to Talbanken corpus of Swedish text (Einarsson, 1976; Nilsson et al., 2005). Each vector being labeled with one of the four possible transitions. We trained the classifiers using the LIBLINEAR implementation (Fan et al., 2008) of logistic regression.

However, classes are not always separable using linear classifiers. We combined single features as pairs or triples. This emulates to some extent quadratic kernels used in support vector machines, while preserving the speed of the linear models. Table 2 shows the complete feature set to predict the transitions. A feature is defined by

- A source: S for stack and Q for the queue;
- An offset: 0 for the top of the stack and first in the queue; 1 and 2 for levels down in the stack or to the right in the queue;

Name	Action	Condition
Initialization	$\langle nil, W, \emptyset \rangle$	
Termination	$\langle S, nil, A \rangle$	
$LeftArc$	$\langle n\|S, n'\|Q, A \rangle \rightarrow \langle S, n'\|Q, A \cup \{\langle n', n \rangle\} \rangle$	$\neg\exists n'', \langle n, n'' \rangle \in A$
$RightArc$	$\langle n\|S, n'\|Q, A \rangle \rightarrow \langle n'\|n\|S, Q, A \cup \langle n, n' \rangle \rangle$	$\neg\exists n'', \langle n', n'' \rangle \in A$
$Reduce$	$\langle n\|S, Q, A \rangle \rightarrow \langle S, Q, A \rangle$	$\exists n', \langle n, n' \rangle \in A$
$Shift$	$\langle S, n\|Q, A \rangle \rightarrow \langle n\|S, Q, A \rangle$	

Table 1: Parser transitions. W is the original input sentence, A is the dependency graph, S is the stack, and Q is the queue. The triplet $\langle S, Q, A \rangle$ represents the parser state. n, n', and n'' are lexical tokens. The pair $\langle n', n \rangle$ represents an arc from the head n' to the dependent n.

- Possible applications of the function head, H, leftmost child, LC, or righmost child, RC;
- The value: word, w, or POS tag, t, at the specified position.

Queue	Q0w
	Q1w
	Q0t
	Q1t
	Q0tQ0w
	Q0tQ1t
	Q1wQ1t
	Q0tQ1tQ2t
	Q0wQ1tQ2t
Stack	S0t
	S0w
	S0tS0w
	S0tS1t
Stack/Queue	S0wQ0w
	Q0tS0t
	Q1tS0t
	Q0tS1t
	Q1tS1t
	S0tQ0tQ1t
	S0tQ0wQ0t
Partial Graph	S0HtS0tQ0t
	Q0LCtS0tQ0t
	Q0LCtS0tQ0w
	S0RCtS0tQ0t
	S0RCtS0tQ0w

Table 2: Feature model for predicting parser actions with combined features.

4.3 Calculating Graph Probabilities

Nivre (2006) showed that every terminating transition sequence $A_1^m = (a_1, ..., a_m)$ applied to a sentence $W_1^n = (w_1, ..., w_n)$ defines exactly one parse tree G. We approximated the probability $P(G|W_1^n)$ of a dependency graph G as $P(A_1^m|W_1^n)$ and we estimated the probability of G as the product of the transition probabilities, so that

$$\begin{aligned} P_{Parse}(G|W_1^n) &= P(A_1^m|W_1^n) \\ &= \textstyle\prod_{k=1}^{m} P(a_k|A_1^{k-1}, W_1^{\phi(k-1)}), \end{aligned}$$

where a_k is member of the set $\{LeftArc, RightArc, Shift, Reduce\}$ and $\phi(k)$ corresponds to the index of the current word at transition k.

We finally approximated the term $A_1^{k-1}, W_1^{\phi(k-1)}$ to the feature set and computed probability estimates using the logistic regression output.

4.4 Beam Search

We extended Nivre's parser with a beam search to mitigate error propagation that occurs with a deterministic parser (Johansson and Nugues, 2006). We maintained N parser states in parallel and we applied all the possible transitions to each state. We scored each transition action and we ranked the states with the product of the action's probabilities leading to this state. Algorithm 3 outlines beam search with a diameter of N.

An alternative to training parser transitions using local features is to use an online learning algorithm (Johansson and Nugues, 2007; Zhang and Clark, 2008). The classifiers are then computed over the graph that has already been built instead of considering the probability of a single transition.

4.5 Evaluation

We evaluated our dependency parser separately from the rest of the application and Table 3 shows the results. We optimized our parameter selection for the unlabeled attachment score (UAS). This explains the relatively high difference with the labeled attachment score (LAS): about –8.6.

Table 3 also shows the highest scores obtained on the same Talbanken corpus of Swedish text (Einarsson, 1976; Nilsson et al., 2005) in the CoNLL-X evaluation (Buchholz and Marsi, 2006): 89.58 for unlabeled attachments (Corston-Oliver and Aue, 2006) and 84.58 for labeled attachments (Nivre et al., 2006). CoNLL-X systems were optimized for the LAS category.

The figures we reached were about 1.10% below those reported in CONLL-X for the UAS category. However our results are not directly comparable as the parsers or the classifiers in CONLL-X have either a higher complexity or are more time-consuming. We chose linear classifiers over kernel machines as it was essential to our application to run on mobile devices with limited resources in both CPU power and memory size.

This paper			CONLL-X	
Beam width	LAS	UAS	LAS	UAS
1	79.45	88.05	84.58	89.54
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: Parse results on the Swedish Talbanken corpus obtained for this paper as well as the best reported results in CONLL-X on the same corpus (Buchholz and Marsi, 2006).

5 Dependencies to Predict the Next Word

We built a syntactic score to measure the grammatical relevance of a candidate word w in the current context, that is the word sequence so far $w_1, ..., w_i$. We defined it as the weighted sum of three terms: the score of the partial graph resulting from the analysis of the words to the left of the candidate word and the scores of the link from w to its head, $h(w)$, using their lexical forms and their parts of speech:

$$\begin{aligned} s_{DepSyn}(w) = \lambda_1 P_{Parse}(G(w)|w_1, ..., w_i, w) + \\ \lambda_2 P_{Link}(w, h(w)) + \\ \lambda_3 P_{Link}(POS(w), POS(h(w))), \end{aligned}$$

where $G(w)$ is the partial graph representing the word sequence $w_1, ..., w_i, w$. The P_{Link} terms are intended to give an extra-weight to the probability of an association between the predicted word and a possible head to the left of it. They hint at the strength of the ties between w and the words before it.

We used the transition probabilities described in Sect. 4.3 to compute the score of the partial graph, yielding

$$P_{Parse}(G(w)|w_1, ..., w_i, w) = \prod_{k=1}^{j} P(a_k),$$

where $a_1, ..., a_j$ is the sequence of transition actions producing $G(w)$ and $P(a_k)$, the probability output of transition k given by the logistic regression engine.

The last two terms $P_{Link}(w, h(w))$ and $P_{Link}(POS(w), POS(h(w)))$ are computed from counts in the training corpus using maximum likelihood estimates:

$$P_{Link}(w, h(w)) = \frac{C(Link(w, h(w)) + 1}{\sum_{w_l \in PW} C(Link(w_l, h(w_l)))} + |PW|$$

and

$$P_{Link}(POS(w), POS(h(w))) = \frac{C(Link(POS(w), POS(h(w)))) + 1}{\sum_{w_l \in PW} C(Link(POS(w_l), h(POS(w_l))))} + |PW|,$$

where $PW = match(\mathbf{ks}^i)$, is the set of predicted words for the current key sequence.

If the current word w has not been assigned a head yet, we default $h(w)$ to the root of the graph and $POS(h(w))$ to the $ROOT$ value.

6 Experiments and Results

6.1 Experimental Setup

Figure 2 shows an overview of the three stages to produce and evaluate our models: training,

tuning, and testing. Ideally, we would have trained the classifiers on a corpus matching a text entry application. However, as there is no large available SMS corpus in Swedish, we used the Stockholm-Umeå corpus (SUC) (Ejerhed and Källgren, 1997). SUC is balanced and the largest available POS-tagged corpus in Swedish with more than 1 million words.

We parsed the corpus and we divided it randomly into a training set (80%), a development set (10%), and a test set (10%). The training set was used to gather statistics on word n-grams, POS n-grams, collocations, lemma frequencies, dependent/head relations. We discarded hapaxes: relations and sequences occurring only once. We used lemmas instead of stems in the semantic relatedness score, $SemR$, because stemming is less appropriate in Swedish than in English.

We used the development set to find optimal weights for the scoring functions, resulting in the lowest KSPC. We ran an exhaustive search using all possible linear combinations with increments of 0.1, except for two functions, where this was too coarse. We used 0.01 then.

We applied the resulting linear combinations of scoring functions to the test set. We first compared the frequency-based disambiguation acting as a baseline to linear combinations involving or not involving syntax, but always excluding bigrams. Table 4 shows the most significant combinations. We then compared a set of other combinations with the bigram model. They are shown in Table 6.

6.2 Metrics

We redefined the KSPC metric of MacKenzie (2002), since the number of characters needed to input a word is now dependent on the word's left context in the sentence. Let $S = (w_1, \ldots, w_n) \in L$ be a sentence in the test corpus. The KSPC for the test corpus then becomes

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

where $KS(w|LContext)$ is the number of key strokes needed to enter a word in a given context, $LContext(w,S)$ 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

$$DA = \frac{\sum_{S \in L} \sum_{w \in S} PredHit(w|LContext(w,S))}{\#w},$$

where $PredHit(w|Context) = 1$ if w is the top prediction and 0 otherwise, and $\#w$, the total number of words in L. A good DA means that the user can more often simply accept the default proposed word instead of navigating the prediction list for the desired word.

As scoring tokens, we chose to keep the ones that actually have the ability to differentiate the models, i.e. we did not count the KSPC and DA for words that were not in the dictionary. Neither did we count white spaces, nor the punctuation marks.

All our measures are without word or phrase completion. This means that the lower-limit figure for $KSPC$ is 1.

6.3 Results

As all the KSPC figures are close to 1, we computed the error reduction rate (ERR), i.e. the reduction in the number of extra keystrokes needed beyond one. We carried out all the optimizations considering KSPC, but we can observe that KSPC ERR and DA ERR strongly correlate.

Table 5 shows the results with scoring functions using the word frequencies. The columns include KSPC and DA together with KSPC ERR and DA ERR compared with the baseline. Table 7 shows the respective results when using a bigram-based disambiguation instead of just frequency. The ERR is still compared to the word frequency baseline but attention should also be drawn on the relative increases: how much the new models can improve bigram-based disambiguation.

7 Discussion

We can observe from the results that a model based on dependency grammars improves the prediction considerably. The $DepSyn$ model is actually the most effective one when applied together with the frequency counts. Furthermore, the improvements from the POS, $SemA$, and $DepSyn$ model are almost disjunct, as the combined model improvement matches the sum of their respective individual contributions.

The 4.2% ERR observed when adding the $SemA$ model is consistent with the result from

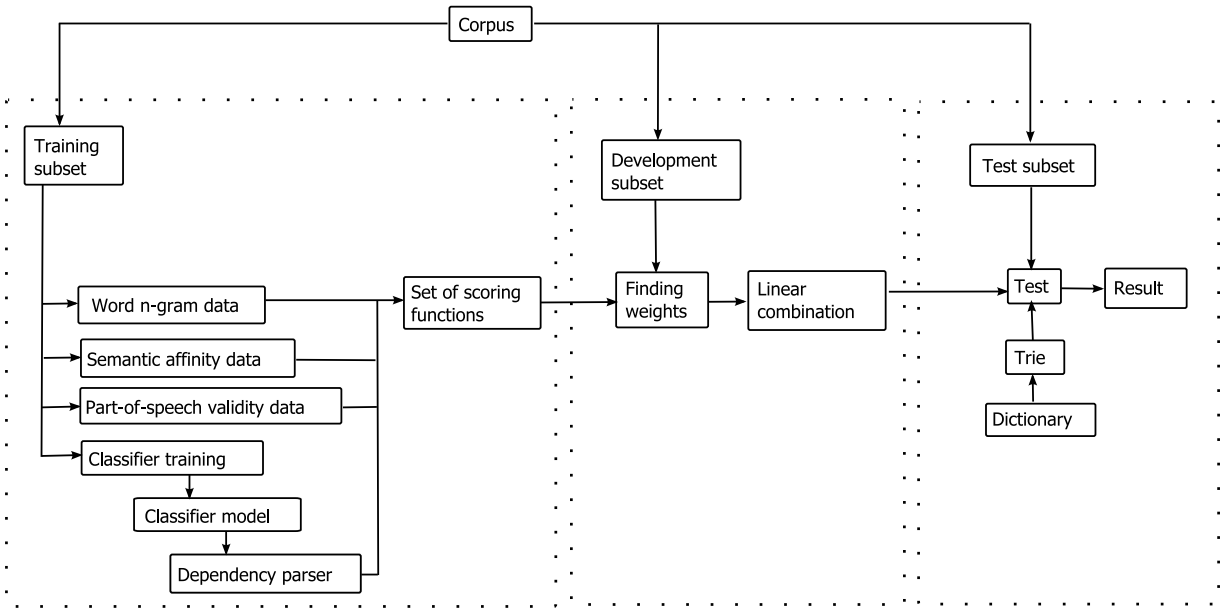

Figure 2: System architecture, where the set of scoring functions is $S = \{s_{LM}, s_{SemA}, s_{POS}, s_{DepSyn}\}$ and the linear combination is $= \sum_{s \in S} \lambda_s \cdot s(w)$.

Gong et al. (2008), where a 4.6% ERR was found. On the other hand, the POS model only contributed 4.7% ERR in our case, whereas Gong et al. (2008) observed 12.6%. One possible explanation for this is that they clustered related POS tags into 19 groups reducing the sparseness problem. By performing this grouping, we 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.

The bigram language model has the largest impact on the performance. The ERR for bigrams alone is higher than all the other models combined. Still, the other models have the ability to contribute on top of the bigram model. For example, the POS model increases the ERR by about 5% both when using bigram- and frequency-based disambiguation, suggesting that this information is not captured by the bigrams. On the other hand, $DepSyn$ increases the ERR by a more modest 3% when using bigrams instead of 7% with word frequencies. This is likely due to the fact that about half of the dependency links only stretch to the next preceding or succeeding word in the corpus.

The most effective combination of models are the bigrams together with the POS sequence and the dependency structure, both embedding syntactic information. With this combination, we were able to reduce the number of erroneous disambiguations as well as extra keystrokes by almost one third.

8 Further Work

SMS texting, which is the target of our system, is more verbal than the genres gathered in the Stockholm-Umeå corpus. The language models of a final application would then change considerably from the ones we extracted from the SUC. A further work would be to collect a SMS corpus and replicate the experiments: retrain the models and obtain the corresponding performance figures.

Moreover, we carried out our implementation and simulations on desktop computers. The POS model has an estimated size of 700KB (Gong et al., 2008). The P_{Parse} term of the $DepSyn$ model can be made as small as the feature model. We expect the optimized size of this model to be under 100KB in an embedded environment. The size of the lexical variant of P_{Link} is comparable to the bigram model. This could however be remedied by using the probability of the action that constructed this last link. The computational power required by LIBLINEAR is certainly within the reach of modern hand-held devices. However, a prototype simulation with real hardware conditions would

be needed to prove an implementability on mobile devices.

Finally, a user might perceive subtle differences in the presentation of the words compared with that of popular commercial products. Gutowitz (2003) noted the reluctance to single-tap input methods because of their "unpredictable" behavior. Introducing syntax-based disambiguation could increase this perception. A next step would be to carry out usability studies and assess this element.

References

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York City.

Johan Carlberger and Viggo Kann. 1999. Implementing an efficient part-of-speech tagger. *Software – Practice and Experience*, 29(2):815–832.

Simon Corston-Oliver and Anthony Aue. 2006. Dependency parsing with reference to slovene, spanish and swedish. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 196–200, New York City, June.

Jan Einarsson. 1976. Talbankens skriftspråkskonkordans. Technical report, Lund University, Institutionen för nordiska språk, Lund.

Eva Ejerhed and Gunnel Källgren. 1997. Stockholm Umeå Corpus version 1.0, SUC 1.0.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

Jun Gong, Peter Tarasewich, and I. Scott MacKenzie. 2008. 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, Lund, Sweden.

Dale L. Grover, Martin T. King, and Clifford A. Kushler. 1998. Reduced keyboard disambiguating computer. U.S. Patent no. 5,818,437.

Ebba Gustavii and Eva Pettersson. 2003. A Swedish grammar for word prediction. Technical report, Department of Linguistics, Uppsala University.

Howard Gutowitz. 2003. Barriers to adoption of dictionary-based text-entry methods; a field study. In *Proceedings of the Workshop on Language Modeling for Text Entry Systems (EACL 2003)*, pages 33–41, Budapest.

Jan Haestrup. 2001. Communication terminal having a predictive editor application. U.S. Patent no. 6,223,059.

Jon Hasselgren, Erik Montnemery, Pierre Nugues, and Markus Svensson. 2003. HMS: A predictive text entry method using bigrams. In *Proceedings of the Workshop on Language Modeling for Text Entry Methods (EACL 2003)*, pages 43–49, Budapest.

Richard Johansson and Pierre Nugues. 2006. Investigating multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CONLL-X)*, pages 206–210, New York.

Richard Johansson and Pierre Nugues. 2007. Incremental dependency parsing using online learning. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL*, pages 1134–1138, Prague, June 28-30.

Jianhua Li and Graeme Hirst. 2005. Semantic knowledge in word completion. In *Assets '05: Proceedings of the 7th international ACM SIGACCESS conference on Computers and accessibility*, pages 121–128, Baltimore.

I. Scott MacKenzie, Hedy Kober, Derek Smith, Terry Jones, and Eugene Skepner. 2001. LetterWise: Prefix-based disambiguation for mobile text input. In *14th Annual ACM Symposium on User Interface Software and Technology*, Orlando, Florida.

I. Scott MacKenzie. 2002. KSPC (keystrokes per character) as a characteristic of text entry techniques. In *Proceedings of the Fourth International Symposium on Human Computer Interaction with Mobile Devices*, pages 195–210, Heidelberg, Germany.

Johannes Matiasek, Marco Baroni, and Harald Trost. 2002. 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.

Johannes Matiasek. 2006. 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, Dagstuhl, Germany.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 81–88, Trento.

Jens Nilsson, Johan Hall, and Joakim Nivre. 2005. MAMBA meets TIGER: Reconstructing a Swedish treebank from antiquity. In *Proceedings of the NODALIDA Special Session on Treebanks*, Joensuu, Finland.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülsen Eryigit, and Svetoslav Marinov. 2006. 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.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932, Prague.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160, Nancy.

Joakim Nivre. 2006. *Inductive Dependency Parsing*. Springer, Dordrecht, The Netherlands.

Claude Elwood Shannon. 1951. Prediction and entropy of printed English. *The Bell System Technical Journal*, pages 50–64, January.

K. Sundarkantham and S. Mercy Shalinie. 2007. Word predictor using natural language grammar induction technique. *Journal of Theoretical and Applied Information Technology*, 3:1–8.

Lucien Tesnière. 1966. *Éléments de syntaxe structurale*. Klincksieck, Paris, 2e edition.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 562–571, Hawaii, October 25–27.

Algorithm 1 Nivre's algorithm.

$Queue \Leftarrow W$
$Stack \Leftarrow nil$
while $\neg Queue.isEmpty()$ **do**
  $features \Leftarrow ExtractFeatures()$
  $action \Leftarrow guide.Predict(features)$
  if $action = RightArc \wedge canRightArc()$ **then**
    $RightArc()$
  else if $action = LeftArc \wedge canLeftArc()$ **then**
    $LeftArc$
  else if $action = Reduce \wedge canReduce()$ **then**
    $Reduce()$
  else
    $Shift()$
  end if
end while
$return(A)$

Algorithm 2 Reference parsing.

$Queue \Leftarrow W$
$Stack \Leftarrow nil$
while $\neg Queue.isEmpty()$ **do**
  $x \Leftarrow ExtractFeatures()$
  if $\langle Stack.peek(), Queue.get(0)\rangle \in A \wedge canRightArc()$ **then**
    $t \Leftarrow RightArc$
  else if $\langle Queue.get(0), Stack.peek()\rangle \in A \wedge canLeftArc()$ **then**
    $t \Leftarrow LeftArc$
  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**
    $t \Leftarrow Reduce$
  else
    $t \Leftarrow Shift$
  end if
  store training example $\langle x, t\rangle$
end while

Algorithm 3 Beam parse.

$Agenda.add(InititalParserState)$
while $\neg done$ **do**
  for $parserState \in Agenda$ **do**
    $Output.add(parserState.doLeftArc())$
    $Output.add(parserState.doRightArc())$
    $Output.add(parserState.doReduce())$
    $Output.add(parserState.doShift())$
  end for
  $Sort(Output)$
  $Clear(Agenda)$
  Take N best parse trees from $Output$ and put in $Agenda$.
end while
$Return$ best item in $Agenda$.

Configuration	Scoring model	$DepSyn$ weights
F1 baseline	$1 \times LM1$ (Word frequencies)	–
F2	$0.9 \times LM1 + 0.1 \times POS$	–
F3	$0.7 \times LM1 + 0.3 \times SemA$	–
F4	$0.6 \times LM1 + 0.4 \times DepSyn$	(0.3, 0.7, 0.0)
F5	$0.6 \times LM1 + 0.1 \times POS + 0.3 \times DepSyn$	(0.0 1.0 0.0)
F6	$0.5 \times LM1 + 0.2 \times SemA + 0.3 \times DepSyn$	(0.2 0.7 0.1)
F7	$0.4 \times LM1 + 0.1 \times POS + 0.3 \times DepSyn + 0.2 \times SemA$	(0.2, 0.8, 0.0)

Table 4: The different combinations of scoring models using frequency-based disambiguation as a baseline. The $DepSyn$ weight triples corresponds to $(\lambda_1, \lambda_2, \lambda_3)$ in Sect. 5.

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

Table 5: Results for the disambiguation based on word frequencies together with the semantic and syntactic models.

Configuration	Scoring model	Bigram weights	$DepSyn$ weights
B1	$1 \times LM2$ (Bigram frequencies)	(0.9, 0.1)	–
B2	$0.9 \times LM2 + 0.1 \times POS$	(0.8, 0.2)	–
B3	$0.95 \times LM2 + 0.05 \times SemA$	(0.8, 0.2)	–
B4	$0.9 \times LM2 + 0.1 \times DepSyn$	(0.8, 0.2)	(0.2, 0.8, 0.0)
B5	$0.8 \times LM2 + 0.1 \times POS + 0.1 \times SemA$	(0.8, 0.2)	–
B6	$0.81 \times LM2 + 0.08 \times POS + 0.11 \times DepSyn$	(0.8, 0.2)	(0.2, 0.8, 0.0)

Table 6: The different combinations of scoring models using bigram-based disambiguation as baseline. In addition to the $DepSyn$ weights, this table also shows the language model interpolation weights, β_1 and β_2 described in Sect. 2.2.

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

Table 7: Results for the disambiguation based on bigrams plus the semantic and syntactical models. The error reduction rate is relative to the word frequency baseline.