N-Best Hidden Markov Model Supertagging to Improve Typing on an Ambiguous Keyboard

Saša Hasan[*] and **Karin Harbusch**
Computer Science Department
University of Koblenz-Landau
{hasan|harbusch}@informatik.uni-koblenz.de

Abstract

Ambiguous keyboards, i.e. several letters host on the same key (cf. telephone keyboard), produce candidate lists with words that match the entered code. These lists have to be disambiguated by the user for the intended word. Consequently, the primary goal is to order these candidates in a way that the most appropriate words are placed on top of the suggestion list for minimal selection costs or to postpone the attention to suggestions to a final editing step. This paper reports on promising results for this goal by inspecting the whole sentence on the basis of supertagging and lightweight dependency analysis.

1 Introduction

Watch-sized devices which lack space for a full keyboard, i.e. uniquely addressable keys, on the one hand, and on the other, Asian language typing or devices for speech- and motor-impaired people where not all letters can be addressed directly because so many buttons are not manageable in reasonable time make use of so-called *ambiguous* or *reduced* or *cluster* keyboards (for one of the earliest systems see (Witten, 1982)) — even only realized virtually on a screen (which does not make any difference for the following considerations). Typing with ambiguous keyboards, i.e. keyboards where several letters, symbols or numbers, respectively, host on one and the same button (cf. telephone keyboards), basically can be performed in two different manners. *Multi-tapping* as basic encoding method for short message sending (SMS) on cellular phones uniquely addresses a symbol by a predefined number of button hits in a row. Obviously, this method is cumbersome and time consuming.

As a consequence of an observation by (Witten, 1982), namely that in a dictionary with 24500 words only 8% are ambiguous if the respective button on a phone keyboard is pressed only once, predictive methods have emerged on the market. *Predictive* text entry devices (e.g., the product T9 by Tegic Communications for SMS typing (Kushler, 1998)) have been developed that reduce the number of keystrokes needed for entering a word by proposing possible candidates matching the current input. Moreover, the possible candidates for *completion* all match the already entered prefix of the word. By selecting one of the available suggestions (irrespective whether assuming prediction or completion mode here), the number of keypresses decreases but the overall time to enter the word is not necessarily reduced due to the cognitive load that emerges while scanning the suggested candidate list (see, e.g., (Horstmann Koester and Levine, 1994)). Consequently, the primary goal is to order these candidates in a way that the most appropriate words are placed on top of the suggestion list for minimal selection costs or to postpone the attention to suggestion lists to a final editing step. This paper reports on promising results for this goal by inspecting the whole sentence on the basis of supertagging and lightweight dependency analysis (LDA).

The paper is organized as follows. Section 2 gives a short overview on related work in the area of predictive typing. Section 3 presents the sentence-wise approach based on supertagging and LDA. The experiments and results achieved with these methods are summarized in Section 4. Section 5, finally, summarizes our approach and the conclusions we reached.

2 State of the art in predictive typing

Only for completeness, we mention *letter-wise* predictive systems here (see, e.g., LetterWise by (MacKenzie et al., 2001)) or the Reactive Keyboard by (Darragh and Witten, 1992). For *word-wise* systems, the easiest way to achieve appropriate suggestion lists is to sort the list according to word frequencies obtained from large corpora (see the

[*]The author is now affiliated with the Chair of Computer Science VI, RWTH Aachen University, Germany.

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 24-31.

proceedings of the EACL-workshop on language modeling for text entry methods (Harbusch et al., 2003) for an overview of recent n-gram systems). Tanaka et al. (2002) propose an adaptive language model utilizing prediction by partial match (PPM (Cleary and Witten, 1984), which actually originates from the information theory domain and deals with the problem of improving the compression rates of arithmetic coding) that lowers the entropy of a language model by maintaining a list of already seen contexts and its corresponding successors. In (Matiasek et al., 2002), a system based on word n-grams with additional part-of-speech information is outlined. Fazly and Hirst (2003) also impose part-of-speech information on prediction. Surprisingly, additional part-of-speech information hardly improves the prediction lists. Thus, other information sources have to be investigated.

The only approach that goes beyond a word-wise step-by-step disambiguation we are aware of is reported in (Rau and Skiena, 1996). Instead of permanently changing between two modes, i.e. a phase where a word is typed and a phase where it is disambiguated in a list of current suggestions, the user can solely concentrate on the process of text entry in a *sentence-wise* approach. Here, a telephone keypad that distributes the 26 letters and the blank character (word delimiter) on 9 keys serves as ambiguous keyboard (i.e. 3 letters are placed on one key at a time). The end of the sentence is marked unambiguously using the "#" key. Sentence disambiguation applies the Viterbi algorithm and involves word bigram probabilities and part-of-speech information extracted from the Brown Corpus. The results obtained by simulating the typing of various text samples with this framework look promising. For various domains, the percentage of correct words ranges from 92.86 to 97.67%. This is due to the relatively high number of keys and low number of ambiguous words, respectively.

A possible way of entering commonly used expressions such as "how are you" or "could you please open the door" fast is the use of *sentence compansion* (see e.g. in (Copestake, 1997; McCoy et al., 1998)). So, from the input "open door", the system would generate "could you please open the door". This *cogeneration* approach needs three knowledge sources, namely a grammar and a lexicon, statistical information about collocations, i.e. syntagmatically related words, and a set of templates. A thinkable drawback for the user might be that the expanded sentences sound monotonous by and by. In contrast, flexibility and individuality are valuable features of direct and unrestricted text entry systems. Thus, the motivation of typing on a sentence level is reasonable. The idea is to make use of the syntactic relations that exist in the sentence the user wants to express and exploit them to present more accurate candidate lists that allow for faster selection by moving likely matches to the top.

a g j l m q r w z ä	c f h k o s t u v x y ü ß	b d e i n p ö -
Button 1	Button 2	Button 3

Figure 1: The layout of the letter keys for German.

3 Sentence-wise predictive typing based on Supertagging

In this paper, we report on results for a sentence-wise text entry system with a highly reduced ambiguous keyboard containing only three letter keys and one command key (the setting results from needs of our disabled test subjects). Nevertheless, the presented system called UKO-II (Harbusch and Kühn, 2003) is adaptive with respect to the number of keys, i.e. the system can be tailored to any number of keys where the symbol distribution is matter of definition.

The distribution of the letters on the keys that is used in this work is language-specific by applying a genetic algorithm which optimizes the candidate lists' length and overall selection costs for a given lexicon. For German and English, the dictionaries are based on the CELEX lexical database (Baayen et al., 1995). The current keyboard layout of the letter keys for the German language is depicted in Figure 1. In contrast to the approach in (Rau and Skiena, 1996), the word delimiter (space) is coded unambiguously by entering the command mode.[1] So for example, in order to enter *guten Morgen* ("*good morning*") the user types the code sequence 1 2 2 3 3 ␣ 1 2 1 1 3 3. For the first code, there exist 48 possible words (*guten, außen, wohin, ...*), for the second, there are 30 entries (*wollen, morgen, Morgen, ...*). This small example already allows for a total of 1440 sentence hypotheses.

3.1 N-best Supertagger

The entire technique that is chosen to achieve our goal is based on *supertagging*, a procedure that associates so-called supertags, i.e. elementary trees in the grammar formalism of Lexicalized Tree-Adjoining Grammar (see, e.g., (Joshi and Schabes, 1997)) that code local dependencies, with the corresponding words of the sentence (see, e.g., (Bangalore and Joshi, 1999)). The core of the presented system is an n-best supertagger that is based on a second-order Hidden Markov Model (see (Bäcker and Harbusch, 2002)) and is able to find the n best sentence hypotheses for a sequence of coded words.

Let $t_1^N = t_1 t_2 \cdots t_N$ be a sequence of supertags for a sentence $w_1^N = w_1 w_2 \cdots w_N$. We are interested in the

[1] In command mode, the mapping of the letter keys is changed to commands like *delete last key* or *space*. Thus, the command button functions as a meta key and allows for hierarchical menu structures which are not further discussed here.

most probable tag sequence $\hat{t}_1^N$ which is defined by

$$\hat{t}_1^N = \text{argmax}_{t_1^N} P(t_1^N|w_1^N). \quad (1)$$

According to Bayes' law and additional assumptions that the words are independent of each other, the probability of a supertag sequence given a sentence, $P(t_1^N|w_1^N)$, can be rewritten as

$$P(t_1^N|w_1^N) \approx \prod_{i=1}^{N} P(t_i|t_{i-2}t_{i-1})P(w_i|t_i), \quad (2)$$

where *maximum likelihood estimation* (MLE) is used for the probabilities by relative frequencies derived from an annotated training set of supertagged sentences. For unknown events, Good-Turing discounting in combination with Katz's back-off is applied.

Usually, a dynamic programming technique (i.e. the Viterbi algorithm) finds the best supertag sequence of a sentence (cf. Equation 1) for a given HMM efficiently by storing the best local sequence probability and a backpointer to the predecessor state for each word position in the sentence. In order to find the n best paths through the HMM trellis, we have to allow the backpointer table to hold not only the best predecessor, but the n best predecessor states sorted by the corresponding log probability. Since we deal with trigrams in Equation 2, the states of the HMM have to be coded as supertag pairs, thus $P(t_i|t_{i-2}t_{i-1}) = P(\langle t_{i-1}, t_i\rangle|\langle t_{i-2}, t_{i-1}\rangle)$. This leads to the following recurrence formula for state probabilities at position k in the sentence $w_1^N, 1 \leq k \leq N$:

$$\delta_k(\langle t_{i-1}, t_i\rangle) = \max_{\langle t_{i-2}, t_{i-1}\rangle} \big[\delta_{k-1}(\langle t_{i-2}, t_{i-1}\rangle) \cdot P(\langle t_{i-1}, t_i\rangle|\langle t_{i-2}, t_{i-1}\rangle)\big] \cdot P(w_k|\langle t_{i-1}, t_i\rangle) \quad (3)$$

The values in the δ-table are used to build an additional table which yields the n best local hypothesis scores

$$\phi_k(s_j, s_i) = \delta_{k-1}(s_i)P(s_j|s_i)P(w_k|s_j) \quad (4)$$

for states $s_i = \langle t_{i-2}, t_{i-1}\rangle$ and $s_j = \langle t_{i-1}, t_i\rangle$. For each s_j, the number of predecessors s_i can be limited to n. The corresponding backpointers are stored in a table $\psi_k(s_j, m) = s_i$ where $m = 1$ denotes the best and $m = n$ the n^{th} predecessing state.

Now, after this forward-trellis step, a backward-tree search is applied in order to find the n most promising supertag sequences which are used to adjust the candidate lists and move likely matches to the top. The evaluation function $f(\langle t_{i-1}, t_i\rangle)$ that associates the current path cost with a state $\langle t_{i-1}, t_i\rangle$ can directly use the log probabilities from the forward-trellis step as a heuristic $h(\langle t_{i-1}, t_i\rangle)$. This approach leads to *greedy search*. An important note is that the heuristic is optimal since it actually returns the *exact* path costs to the goal. By also incorporating the backward partial path costs $g(\langle t_{i-1}, t_i\rangle)$ of the search process, i.e. $f = g + h$, we arrive at *A* search*. The resulting system is able to generate the n best supertag hypotheses for a given sentence. For a more detailed presentation of the system, see (Hasan, 2003).

3.2 Incorporating ambiguous codes

The starting point is an ambiguously coded word sequence typed with a reduced keyboard as introduced in the beginning of this section. Every code generates a list of words and every word has several supertags associated with it. A supertagger is used to find the most likely supertag sequence for the sentence and on the basis of this information, the candidate list becomes reordered such that the most likely words (which are the lexical anchors of the supertags) appear at the top. Due to the ambiguous coding, the number of supertags for a code (which corresponds to the supertags of all word expansions of a code) is so large that the best supertag sequence is not sufficient to improve the results significantly. Therefore, we use the n-best tree-trellis approach from Section 3.1 in order to produce more than one hypothesis. At this point, the code sequence of each sentence is associated with a list of the n best supertag sequences found by the supertagger.

Every word usually has several supertags, since the lexical items of an LTAG are almost always associated with several elementary structures that encode the various local dependencies of each word. And since every code expands to several matching words, the result is a set of supertag sets that form a trellis (cf. detailed view in Figure 2). This trellis is the basis for the tree-trellis search that finds the n best supertag hypotheses for a given sentence. Figure 2 also shows the different expansion steps for the sentence *ich habe ein kleines Problem* ("*I have a little problem*").

After typing the words of a sentence with the ambiguous keyboard, the code sequence is expanded and the candidate list is obtained according to the CELEX lexicon. After that, the possible supertags are looked up in the trained language model, i.e. all supertags that occurred in the training corpus with its corresponding lexical anchor are primed for the n-best tree-trellis search. The hypotheses that are returned by the search are then used to reorder the candidate lists. The effect is that likely words of the trained language model will move to the top of the match lists and improve the overall accuracy of the system.

3.3 Filtering ungrammatical hypotheses

In a second step, a *lightweight dependency analysis* (Bangalore, 2000) on the list of supertag hypotheses found by the n-best search is used as an additional knowledge source in order to determine likely chunks of the sentence. The dependencies coded in the elementary trees

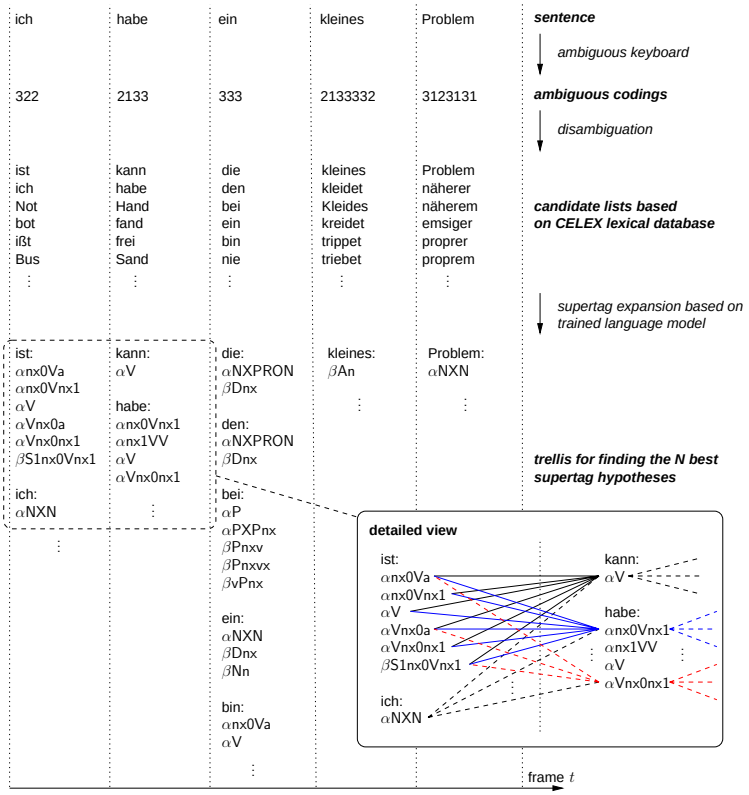


Figure 2: Coping with ambiguous words: disambiguation of coded words and the corresponding supertag expansion.

(supertags) can be used to actually derive a shallow parse of the sentence in linear time.[2] We use a *dependency coverage* criterion which determines how many dependency slots of each supertag are filled to the left and right of its lexical anchor. The hypotheses that have a maximum number of covered elements are used to adjust the final candidate lists, i.e. the supertag hypotheses that span the largest portion of the sentence and seem most "consistent" are moved to the top. This method is applied in order to discard hypotheses that have syntactic inconsistencies.

Figure 3 illustrates the rearrangements of the ambiguously typed sentence *ich habe ein kleines Problem* ("*I have a little problem*"). The three marked hypotheses all have a maximum coverage of 5, i.e. all supertags have their dependency slots filled, whereas the other hypotheses have coverages less than 5. One can see that local word probabilities would suggest *ist kann die kleines Problem* ("*is can the little problem*"). The information that is provided by the surviving hypotheses is used for additional final adjustments of the candidate lists to be presented to the user. We call this reordering process *match list boosting*, or shortly *boosting* (see Figure 3 for an example).

[2]We decided for LDA because it considers more syntactic knowledge than simple chunking techniques, while still being very efficient in comparison to full TAG parsing.

4 Evaluation

For an evaluation of the techniques presented in the previous section, the ambiguous typing of a sample text is simulated and processed with the n-best supertagger. As performance criteria, the *accuracy* and the *average rank* of the correct word are compared to the values obtained from the baseline approach using the word frequencies from the CELEX lexical database (approx. 320 000 word types). For this purpose, a lexicalized tree adjoining grammar is needed because of the lightweight dependency analysis performed in the last step of the n-best approach. The trigram HMM is directly trained on a corpus that is annotated with supertags.

For the experiments, the corpus and LTAG developed in (Bäcker and Harbusch, 2002) is used. It comprises 250 labeled sentences (approx. 2000 words), whereof 225 are used for training and 25 for testing. Since the corresponding LTAG is rather small, containing only 127 elementary trees (58 initial and 69 auxiliary trees), this directly impacts on the size of the trained HMM and the runtime of the LDA. Therefore, it was possible to run the n-best supertagger for up to 2000 hypotheses in an acceptable

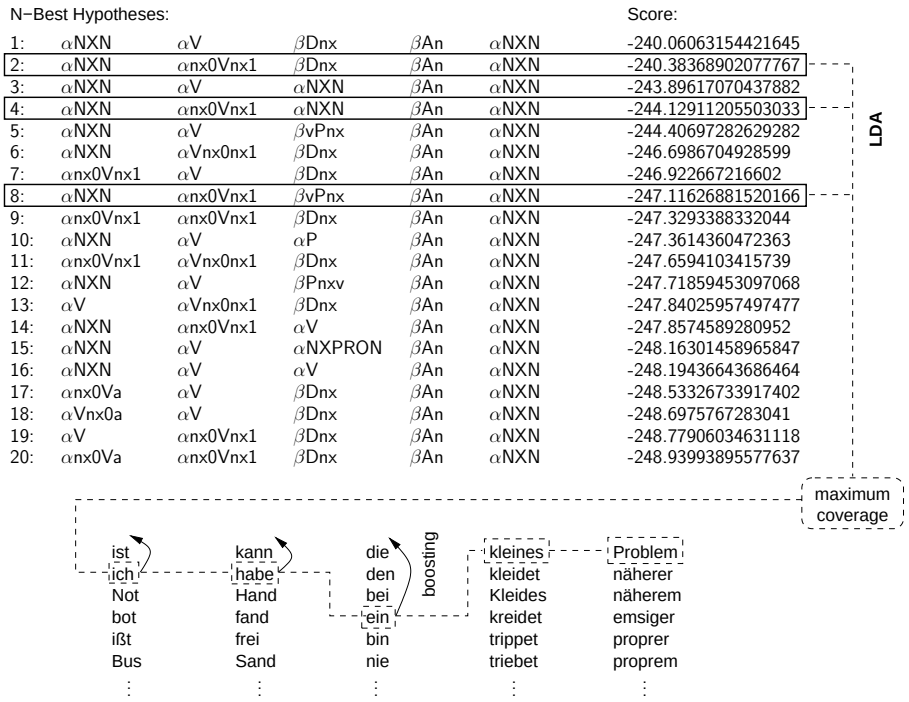


Figure 3: Example for rearranging word hypotheses according to the results of supertagging and LDA.

amount of time. On a 1.4GHz AMD Athlon, the evaluation of the reference test set needs approx. 10.58s for $n = 250$, i.e. 423ms per sentence. The adjustments of the match lists can therefore be performed in real-time for smaller values of n.

Coping with unknown words in ambiguous typing is a more complicated problem. If the word is not in the dictionary, it has to be disambiguated letter by letter for all the keys of the code. Since the primary goal was not to simulate a specific keyboard but to evaluate whole sentences with the n-best supertagging framework, the dictionary was patched by adding the unknown words with a zero-frequency and thus contained all words of the corpus.

4.1 Baseline

The baseline results are achieved with the simple unigram approach where the frequencies of the words that are stored in the lexicon order the candidate list in descending order, i.e. with highest frequency first. As evaluation criteria, the *accuracy of rank* r and the *average match position* is chosen. More formally, let

$$f_r(w|c) = \begin{cases} 1 & \text{if } w \in \text{matches}(c) \wedge \text{rank}(w) = r \\ 0 & \text{else} \end{cases} \tag{5}$$

be a binary function that returns 1 if a disambiguated target word w correctly occurs on the r^{th} position of the candidate list of its code c, which is given by $\text{matches}(c)$.

Reference test set evaluation, $\bar{r} = 3.02$					
	$r = 1$	$r = 2$	$r = 3$	$r = 4$	$r = 5$
$\text{acc}(r)$ [%]	50.26	28.04	5.29	7.41	1.59
$\text{cac}(r)$ [%]	50.26	78.30	83.59	91.00	92.59

Table 1: The baseline results of ambiguously typing the test corpus.

For a test corpus containing a total of N words, the accuracy of rank r for the given corpus can be computed as

$$\text{acc}(r) = \frac{\sum_w f_r(w|c)}{N}. \tag{6}$$

For a *cumulative accuracy*, i.e. where the target words appear within the first r ranks of the candidate lists, the single accuracy values are summed:

$$\text{cac}(r) = \sum_{i=1}^{r} \text{acc}(i). \tag{7}$$

The second evaluation measure is the *average rank* of words of the test corpus. It is simply computed by

$$\bar{r} = \frac{\sum_w \text{rank}(w)}{N}. \tag{8}$$

The results for the baseline are outlined in Table 1. Apparently, the unigram approach places approx. 50% of the target words on the first position of the candidate lists. 92.6% of the words appear within the first 5 ranks. The

Reference test set evaluation					(a)
Average for $n = 1, \ldots, 1000$, $\bar{r} = 2.18$					
	$r = 1$	$r = 2$	$r = 3$	$r = 4$	$r = 5$
$\mathrm{acc}(r)$ [%]	61.84	23.01	1.84	8.38	0.55
$\mathrm{cac}(r)$ [%]	61.84	84.85	86.69	95.07	95.62
Single best results					(b)
Overall best for $n = 592$, $\bar{r} = 2.11$					(b.1)
$\mathrm{acc}(r)$ [%]	66.67	18.52	1.59	8.47	0.53
$\mathrm{cac}(r)$ [%]	66.67	85.19	86.78	95.25	95.78
Best accuracy/time trade-off for $n = 250$, $\bar{r} = 2.16$					(b.2)
$\mathrm{acc}(r)$ [%]	61.90	22.75	2.12	8.47	0.53
$\mathrm{cac}(r)$ [%]	61.90	84.65	86.77	95.24	95.77
Trigram experiment					(c)
Average for $n = 1, \ldots, 1000$, $\bar{r} = 2.91$					
$\mathrm{acc}(r)$ [%]	60.01	19.64	4.09	6.59	2.26
$\mathrm{cac}(r)$ [%]	60.01	79.65	83.74	90.33	92.59
Upper bound experiment					(d)
Average for $n = 1, \ldots, 1000$, $\bar{r} = 2.11$					
$\mathrm{acc}(r)$ [%]	68.07	16.92	1.85	8.22	0.55
$\mathrm{cac}(r)$ [%]	68.07	84.99	86.84	95.07	95.62

Table 2: The improved results using the n-best supertagger/LDA system and additional experiments with trigrams and an upper bound.

rank expectation for the reference test set is 3, i.e. the user has to scroll two times on average before selecting the desired word.

4.2 N-best system

This section reports on the improvements obtained with the system using the n-best supertagger and additional LDA. The results show that the approach yields better rankings than the simple word-wise prediction method (baseline) and also outperforms a trigram language model. The overall results are shown in Table 2. The first part (a) shows the values computed for the reference test set, namely the average for the full evaluation runs with hypothesis sizes ranging from 1 to 1000. When comparing the values to those in Table 1, a significant improvement for the reference test set is visible. The cumulative accuracy of rank 1 raises by approx. 12%, i.e. 61.8% of the target words are now placed on top of the candidate lists. For the other ranks, the improvement is not as big as for rank 1, but there is still a significant increase. With the n-best approach, 95.6% are placed within the top 5 ranks, whereas the average rank drops down to 2.18. The overall best run of this evaluation session is given in (b.1). The maximum occurred for the hypothesis size $n = 592$, i.e. the 592 best supertag sequence hypotheses for the ambiguously coded sentences are used for adjusting the candidate lists. This result also shows that the biggest variation takes place for rank 1. The changes in cumulative accuracy for ranks ≥ 2 are very small for larger values of n. The graphs in Figure 4 give an overview on the differences between the n-best approach and the baseline and also show the slightly better performance of A* search when compared to greedy search.

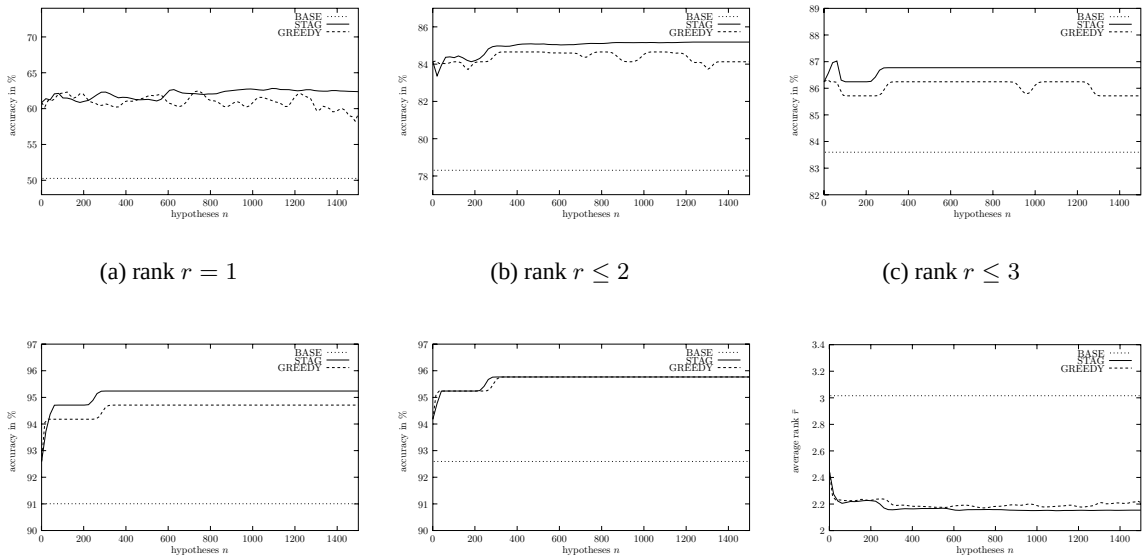
As can be seen in all graphs, enhancing the search from 1-best (Viterbi) to n-best has the largest effect for values of $n < 250$. After approx. 250 hypotheses, the results do not improve significantly, at least for higher cumulative ranks. In general, a hypothesis size of $n = 250$ (cf. Table 2 (b.2)) shows good results since the value for $\mathrm{cac}(5)$ does not increase any more for $n \geq 250$ and the computation time is quite fast.

Another method of evaluating the n-best supertagger is the possibility to look at the target words of the sentences that are typed ambiguously and use only the hypotheses that match closest for adjusting the candidate lists (cf. results in Table 2 (d)). Clearly, this procedure is illegal for an objective evaluation since we are already looking at the desired result we want to achieve, but nevertheless it gives an upper bound of what accuracy the n-best supertagger can theoretically reach by just picking the most promising hypotheses. The detailed evaluation graphs are given in Figure 5. As can be seen, the accuracy between the two approaches differs only for lower ranks (cf. (c)), while for higher ranks (cf. (d) and (e)), the graphs are nearly identical. This means that for the higher rank accuracy, the n-best supertagger already performs in an optimal way for the reference test set and it actually cannot get any better with this kind of training material. It is assumed that with a larger training corpus and thus better language model, the rankings can be further improved.

An interesting constellation is revealed in Figure 5 (a) where the trigram approach outperforms supertagging for lower hypothesis sizes considering rank 1, whereas the accuracy cannot compete for higher ranks (cf. (b)). This is possibly due to the sparseness of data, i.e. the few learned estimations lead to overproportionally many misclassifications for a small hypothesis search space. This claim has to be verified on the basis of more data.

5 Conclusion

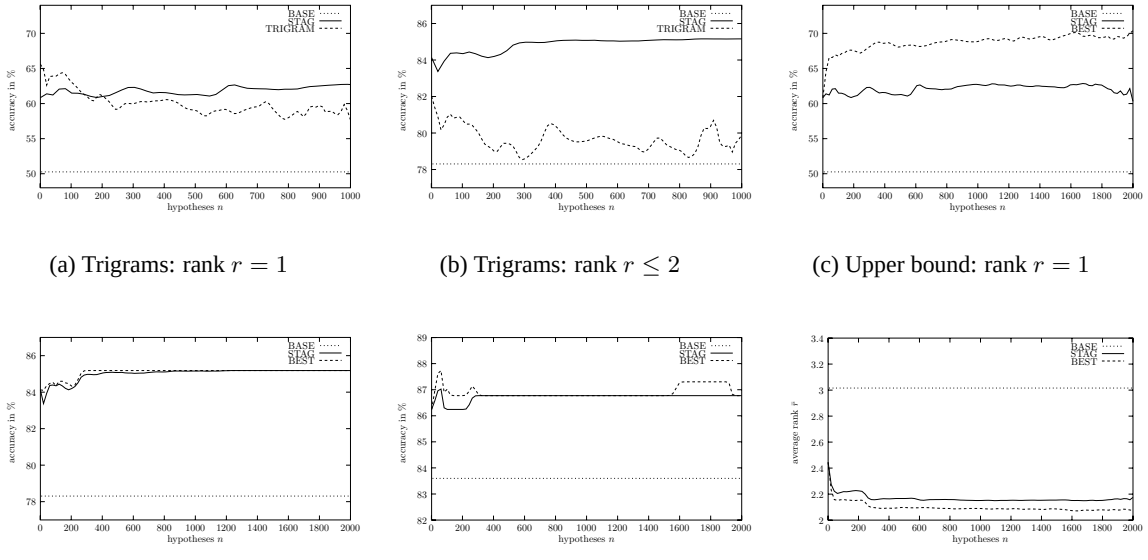
In this paper, we have presented a sentence-wise predictive typing method based on an n-best Hidden Markov Model Supertagger for typing with a four-button ambiguous keyboard. The main idea is to utilize the syntactic dependencies that exist between the words of a sentence and use this information to adjust the candidate lists such that more likely words appear at the top. Instead of being distracted by a list of proposals after every keypress, the user has to pay attention to the prediction list only at the end of the sentence. So the user can concentrate on what (s)he wants to express in the first phase, i.e. the ambiguous typing of the whole sentence, and disambiguate the target words from the candidate lists in a second phase. First evaluations show that users like this mode better than word-wise disambiguation. Further evaluations have



(a) rank $r = 1$ (b) rank $r \leq 2$ (c) rank $r \leq 3$

(d) rank $r \leq 4$ (e) rank $r \leq 5$ (f) average rank $\bar{r}$

Figure 4: Backward search: A* vs. Greedy, cumulative accuracy for ranks $r = 1, \ldots, 5$ and average match position $\bar{r}$. BASE refers to the unigram baseline, STAG/GREEDY is the supertagging+LDA approach with A* and Greedy search, respectively.



(a) Trigrams: rank $r = 1$ (b) Trigrams: rank $r \leq 2$ (c) Upper bound: rank $r = 1$

(d) Upper bound: rank $r \leq 2$ (e) Upper bound: rank $r \leq 3$ (f) Upper bound: average rank $\bar{r}$

Figure 5: Comparison of the baseline (unigram model, BASE), supertagging+LDA (STAG), a trigram approach (TRIGRAM) (Figures (a) and (b)) and an upper bound experiment (BEST) (Figures (c)–(f)).

to be carried out in the future to prove this claim.

As for future work, experiments should be carried out with an enhanced training corpus. Furthermore, a comparison has to be performed with the sentence-wise approach in (Rau and Skiena, 1996). Both systems have to deploy a 9-button keyboard, the LTAG underpinning the supertagging approach and the same lexicon. Under the current circumstances, a direct comparison is not possible.

As it is known (see, e.g., (Baayen, 1991)) that high frequency words often only differ in one letter, and thus, remain highly competitive in all syntactic approaches, we are going to add semantic features (taken from *WordNet* (Miller, 1995)) to the supertags. We expect that rearrangements in the prediction list according to semantic clusters will considerably improve the accuracy of predictions.

References

R. Harald Baayen, Richard Piepenbrock, and Léon Gulikers. 1995. The CELEX lexical database (release 2). CD-ROM, Linguistic Data Consortium, University of Pennsylvania, PA, USA.

R. Harald Baayen. 1991. A stochastic process for word frequency distributions. In *Proceedings of the 29th Meeting of the Association for Computational Linguistics*, pages 271–278, Berkeley, CA, USA.

Jens Bäcker and Karin Harbusch. 2002. Hidden Markov model-based supertagging in a user initiative dialogue system. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6)*, pages 269–278, Venice, Italy.

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265.

Srinivas Bangalore. 2000. A lightweight dependency analyzer for partial parsing. *Computational Linguistics*, 6(2):113–138.

John G. Cleary and Ian H. Witten. 1984. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402.

Ann Copestake. 1997. Augmented and alternative NLP techniques for Augmentative and Alternative Communication. In *Proceedings of the ACL workshop on Natural Language Processing for Communication Aids*, pages 37–42, Madrid, Spain. Association for Computational Linguistics.

John J. Darragh and Ian H. Witten. 1992. *The Reactive Keyboard*. Cambridge Series on Human-Computer Interaction. Cambridge University Press.

Afsaneh Fazly and Graeme Hirst. 2003. Testing the efficacy of part-of-speech information in word completion. In (Harbusch et al., 2003), pages 9–16.

Karin Harbusch and Michael Kühn. 2003. Towards an adaptive communication aid with text input from ambiguous keyboards. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL03), Conference Companion*, pages 207–210, Budapest, Hungary.

Karin Harbusch, Michael Kühn, and Harald Trost, editors. 2003. *Proceedings of the Workshop on Language Modeling for Text Entry Methods*, Budapest, Hungary. Association for Computational Linguistics.

Saša Hasan. 2003. *N-Best Hidden Markov Model Supertagging for Typing with Ambiguous Keyboards*. Diploma thesis, Computer Science Department, University of Koblenz-Landau, Koblenz, Germany.

Heidi Horstmann Koester and Simon P. Levine. 1994. Modeling the speed of text entry with a word prediction interface. *IEEE Transactions on Rehabilitation Engineering*, 2(3):177–187.

Aravind K. Joshi and Yves Schabes. 1997. Tree Adjoining Grammars. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 69–214. Springer, Berlin, Germany.

Cliff Kushler. 1998. AAC using a reduced keyboard. In *Proceedings of the Technology and Persons with Disabilities Conference 1998*. Online document available at `http://www.csun.edu/cod/conf/1998/proceedings/csun98_140.htm`.

I. Scott MacKenzie, Hedy Kober, Derek Smith, Terry Jones, and Eugene Skepner. 2001. LetterWise: Prefix-based disabiguation for mobile text input. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology (UIST 2001)*, pages 111–120, Orlando, FL, USA.

Johannes Matiasek, Marco Baroni, and Harald Trost. 2002. FASTY — A multilingual approach to text prediction. In K. Miesenberger, J. Klaus, and W. Zagler, editors, *Computers Helping People with Special Needs — Proceedings of the 8th International Conference ICCHP 2002, Linz, Austria*, volume 2398 of *Lecture Notes in Computer Science*, pages 243–250, Berlin, Germany. Springer.

Kathleen F. McCoy, Christopher A. Pennington, and Arlene Luberoff Badman. 1998. Compansion: From research prototype to practical integration. *Natural Language Engineering*, 4(1):73–95.

George A. Miller. 1995. WordNet: A lexical database for English. *Communications of the ACM*, 38(11):39–41.

Harald Rau and Steven S. Skiena. 1996. Dialing for documents: An experiment in information theory. *Journal of Visual Languages and Computing*, 7:79–95.

Kumiko Tanaka-Ishii, Yusuke Inutsuka, and Masato Takeichi. 2002. Entering text with a four-button device. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING '02)*, pages 988–994, Taipei, Taiwan.

Ian H. Witten. 1982. *Principles of Computer Speech*. Academic Press, London, UK.