Computational Linguistics in the Netherlands Journal 1 (2011) 79-94 Submitted 7/2011; Published 12/2011

Effects of Context and Recency in Scaled Word Completion

Antal van den Bosch A.VANDENBOSCH@LET.RU.NL
Centre for Language Studies
Radboud University Nijmegen

Abstract

The commonly accepted method for fast and efficient word completion is storage and retrieval of character n-grams in tries. We perform learning curve experiments to measure the scaling performance of the trie approach, and present three extensions. First, we extend the trie to store characters of previous words. Second, we extend the trie to the double task of completing the current word and predicting the next word. Third, we augment the trie with a recent word buffer to account for the fact that recently used words have a high chance of recurring. Learning curve experiments on English and Dutch newspaper texts show that (1) storing the characters of previous words yields an increasing and substantial improvement over the baseline with more data, also when compared to a word-based text completion baseline; (2) simultaneously predicting the next word provides an additional small improvement; and (3) the initially large contribution of a recency model diminishes when the trie is trained on more background training data.

1. Introduction

Word completion is a basic technology for text entry as well as an important component in augmentative language technology tools. Its main aim is to reduce the number of keystrokes needed during text entry by correctly suggesting the completion of the word currently being entered, as soon as possible, so that the word does not have to be completely keyed in. One strategy is to generate a suggestion at the word's unicity point, the point at which the word is the only word available in the algorithm's internal word model (e.g. a list of words) that fits the string of characters keyed in so far. The word completion algorithm may also venture to suggest a completion even before the word's unicity point is reached—or even before its first character is typed. From the moment of suggestion (typically displayed in a designated part of the device's screen), the user is able to click an "Accept" button to accord and enact the suggestion. Although word completion algorithms are used in many devices with some success, current word completion systems remain imperfect, and are vulnerable to at least the following two factors.

The first factor that hampers all basic word completion systems is that they will not suggest words that are not in their on-board word list. This problem can be overcome by including more words in the construction of the algorithm, in the hope of having a better coverage of new, unseen text. In the case of morphologically complex languages that allow affixation or compounding, generation rules might be applied to counter the unknown word problem (Trost et al. 2005); we will not concern us with this issue here.

A second issue that applies to the simpler kind of word completion systems based on efficient lookup in a word list, is that they reset after every space or punctuation mark; they do not take into account the previous sequence of characters or words. Yet, a classic and fundamental insight is that characters or words preceding the current word may contribute information that would enable suggesting the current word earlier than its unicity point, in some cases even immediately (Shannon 1948). Using information from the previously entered text may also help in picking one particular suggested word over alternative words with the same initial characters, but which are less likely given the previous word.

The latter issue has been widely studied, yielding promising and relatively successful methods that usually trade in speed and memory usage for above-baseline keystroke savings; we provide an overview of the main findings in Section 2. The main contribution of this paper is the perspective of the learning curve experiment. The amount of text available for training a word or text completion algorithm is likely correlated with keystroke savings, but what is the actual relation? We know from Lesher et al. (1999) that this relation is positive, but their learning curves extend to two million words of training data. To complement Lesher *et al*'s findings, we ask what happens with more data than that: for instance, does the curve flatten? We report that we indeed find performance improvements with more training data up to a certain amount of training data, after which we observe some flattening in the performance of a simple baseline approach. In contrast, a number of extensions to the baseline (in total we compare the baseline to five extensions, all suggested by the existing literature) continue to increase their performance when trained on more data in a log-linear fashion, i.e., with every n-fold increase in the amount of training data, performance increases by a roughly constant amount.

After our review of related work, we introduce in Section 3 a simple character-based word completion algorithm based on the normal prefix trie, and a recent-keystrokes variant which we select as the baseline. After introducing our training and testing data and our experimental setup in Section 4, we introduce four extensions to the baseline model. We compare the baseline and its four extensions in learning curve experiments, of which the results are given in Section 5. We discuss our findings and phrase our conclusions in Section 6.

2. Related Work

The approach we focus on in this paper aims at reducing the amount of typing necessary to enter a text by using automatic predictive typing aids. Tools of this type try to predict the completion of the current text as it is being keyed in, and are available for input modalities ranging from full keyboards to the more restricted keypads of mobile phones. For instance, the popular T9 algorithm and several variants were designed specifically to offer word completion on the standard 12-key keypad layout on the majority of mobile phones used in the last decade (Grover et al. 1998, MacKenzie et al. 2001). However, a growing number of mobile devices sport full QWERTY keyboards, either soft keyboards (i.e. on-screen keyboards that can be utilized through touch or pen input), or hard keyboards just as regular desktop computers and laptops have. In this paper we focus on predictive text entry using QWERTY keyboards, i.e. one-letter-per-key keyboards.

Predictive text entry algorithms use context information to anticipate what block of characters (letters, n-grams, syllables, words, or entire phrases) a person is going to write next (Garay-Vitoria and Abascal 2006), lowering the overall cognitive load of text entry, even though the additional interaction with the predictive text entry software introduces a load of its own (Horstmann Koester and Levine 1996). The block size the typing aid tries to predict influences the potential savings in terms of time and keystrokes. However, predicting n-grams (Goodman et al. 2002) or sub-word units such as syllables or morphemes can result in increased cognitive load for the user who has to check and approve the predictions. Because words (whitespace-delimited sequences of letters) are more easily recognizable and thus reduce the cognitive load required, and because with whitespace-delimited strings the space bar can play a double role as a whitespace and an “Accept” button, words are most widely used as predicted blocks (Garay-Vitoria and Abascal 2006). We take this approach here as well, and thus focus on word completion.

Word completion was adopted early on as a topic in the development of augmentative and rehabilitation technologies, where it has been established as a generally effective tool for text entry (Horstmann Koester and Levine 1996). An early version of a simple wordlist-based system was *PAL* (Swiffin et al. 1985), a portable device in which high-frequency words are suggested that partially match the characters keyed in thus far. Extensions were proposed that exploited conditional probabilities between successive words (Bentrup 1987) and syntactic parsing (VanDyke 1991).

Another more technical strand of work then emerged that largely followed the same development path as the previous augmentative technology strand: a context-insensitive wordlist-based system is augmented by using statistically or linguistically more sophisticated context of the text typed in thus far. Syntactic information, which can produce constraints on the likely syntactic category of the next word to be completed based on a partial parse of the left context, is used in by Wood (1996) and Garay-Vitoria and González-Abascal (1997). Carlberger et al. (1997) use part-of-speech tagging in their *Profet* system. Cagigas (2001) combines grammatical knowledge and part-of-speech tag statistics, as do Matiasek et al. (2002) with their *FASTY* system, and Fazly and Hirst (2003). Discussing the use of part-of-speech tagging likelihoods to constrain and augment word completion, Fazly and Hirst say, when observing that part-of-speech tagging information hardly contributes to the performance as compared to the context-sensitive statistical component in their model, “The relatively small improvement is possibly because word-bigram probabilities implicitly capture tag-bigram probabilities to a considerable extent as well.” (Fazly and Hirst 2003). It is for this reason that we do not make use of part-of-speech tagging in our experiments.

Several studies focus on augmenting text completion purely with statistical word bigram or word trigram models. A key publication is that of Lesher et al. (1999), in which the relation is investigated between word completion success and the order of the word n-gram model underlying the word completion, as well as with the amount of training material. Their results show that in their experimental setting, word trigram models contribute 7.5% additional keystroke savings over a unigram model (i.e. the word list baseline). They also report on learning curve experiments with increasing training set sizes and constant test material, and report increasingly higher keystroke savings with more training data. In the study described in this article, training set size and context beyond the single current word are among the factors we also vary in our experimental matrix. Other examples of word completion work purely driven by statistical word n-gram models are the *WordQ* system of Nantais and colleagues (Nantais et al. 2001, Shein et al. 2001), and the system presented by How and Kan (2005). Note that our work concerns word completion based on character-n-

gram models, although we do present a comparison against word prediction through a word n-gram language model.

Copestake (1997) combines context-sensitive part-of-speech tagging information with a recency model. Information in recent text is also exploited by Darragh et al. (1990), who use a dynamically updated tree-based memory structure to cache n-grams. Suggestions are sorted by preference to the longest, most frequent substrings. Tanaka-Ishii (2007) compares four language models for predictive text entry: two simple models based on frequency and recency counts, a model based on co-occurrence between words, and an adaptive n-gram model that takes into account the probability distribution of words previously used by the user as well. The adaptive n-gram model performs best.

Although a 2003 EACL workshop[1] showcased a considerable breadth and quantity of research in the text completion field at that point in time, there seems to have been somewhat of a lull in the research since then, while the industrial application field of efficient text entry methods in increasingly advanced devices is only becoming wider. As it stands, it may already be the widest-spread piece of language technology in the world, shipped with nearly every one of the several billions of mobile phones in use worldwide.

3. A baseline word completion algorithm and three extensions

Arguably, the two goals of a word completion algorithm are (i) to store a wide-coverage word or language model that it can access rapidly to match input character sequences, and (ii) that it needs to be able to say at the earliest possible point of a character sequence being entered, with a success rate that should be as high as possible, to which word the sequence can be completed.

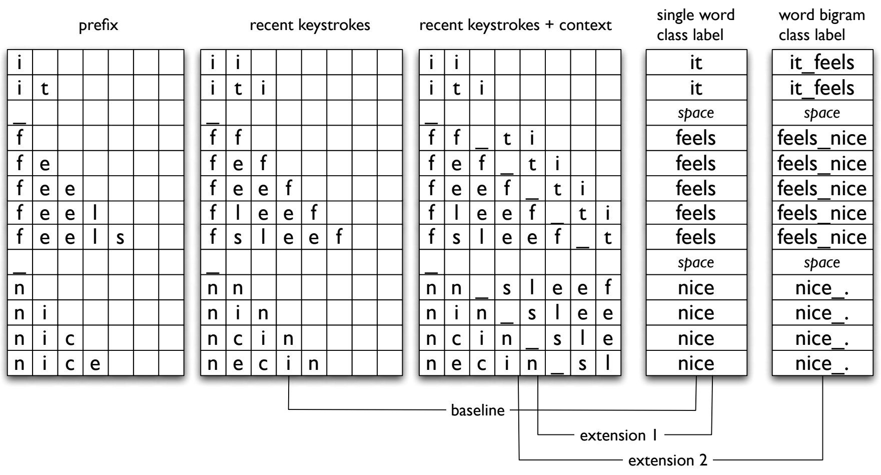
This task can be phrased as a classification problem in which an input sequence of entered keys is mapped to the complete intended word that is currently being keyed in. Suppose that a person intends to key in the text "it feels nice" on a normal QWERTY keyboard. The first task is to predict "it" after seeing the "i", however unreasonable this may seem. After having seen "it" and the space bar, the next task is to suggest "feels"; either immediately, or as soon as possible in the sequence of that word being keyed in: after "f", after "fe", etcetera. Its classification task thus can be made explicit as thirteen classification instances depicted at the left-hand side of Figure 1, labeled "prefix".

We first introduce tries as the data structure of choice, and describe the algorithm to generate them and use them during processing. We then introduce our baseline word completion algorithm, and describe five extensions to this baseline.

3.1 Tries

To store instances such as the ones in Figure 1 and convert them into an efficient data structure from which word completions can be quickly retrieved, *tries* are ideally suited (Knuth 1973). In our study, we employ the IGTree decision tree algorithm (Daelemans et al. 1997), which implements trie compression and retrieval. In the first compression phase, a list of words (e.g., a lexicon or a word list from a corpus) is compressed by IGTree into a trie

1. The EACL 2003 Workshop on Language Modeling for Text Entry Methods, Budapest, Hungary, URL: `http://www.uni-koblenz.de/~compling/eaclws2003/`.



prefix	recent keystrokes	recent keystrokes + context	single word class label	word bigram class label
i	i i	i i	it	it_feels
i t	i t i	i t i	it	it_feels
_	_	_	*space*	*space*
f	f f	f f _ t i	feels	feels_nice
f e	f e f	f e f _ t i	feels	feels_nice
f e e	f e e f	f e e f _ t i	feels	feels_nice
f e e l	f l e e f	f l e e f _ t i	feels	feels_nice
f e e l s	f s l e e f	f s l e e f _ t	feels	feels_nice
_	_	_	*space*	*space*
n	n n	n n _ s l e e f	nice	nice_.
n i	n i n	n i n _ s l e e	nice	nice_.
n i c	n c i n	n c i n _ s l e	nice	nice_.
n i c e	n e c i n	n e c i n _ s l	nice	nice_.

Figure 1: Example classification instances derived from the sentence "it feels nice" for the prefix trie (left), the recent-keystrokes trie (middle), and the recent-keystrokes trie with context (right).

structure. To do this, first a one-time ordering of features is computed, where the features are the positions in the character buffer. The ordering of the features is determined by their information gain ratio with respect to predicting the word completion. Subsequently, a root node is created, which represents the most likely word when no key is pressed yet. This root node fans out to a first layer of nodes through arcs, where the arcs represent all possible keystrokes. Each first-layer node represents the most likely word given a single keystroke, and connects to second-layer nodes by arcs that denote all possible keystrokes given the first keystroke. A node becomes an end node if the arc leading to that node uniquely identifies a single word (i.e. the word's unicity point is reached), even if the arc is not the last character of the word. This algorithm is recursively applied until all instances are represented as paths in the trie.

The normal prefix trie retrieval process matches new incoming instances to paths in the trie by simply traversing the trie level by level, one arc per level, according to every new character in the input. At any point during the traversal, the trie is able to generate the most likely word as a completion suggestion given the path so far, provided that frequency information is stored on the trie's nodes. Starting with the root node, it deterministically takes the arc representing the first (leftmost) character, and continues following matching arcs in the ordering of the features, during which it may either (i) encounter a non-ending node with a matching arc pointing deeper into the trie, at which the current sequence of keystrokes may still be completed into more than a single word, it may suggest the most likely word so far (in case of frequency ties, the first word encountered in the training set is presented), or (ii) it may fail to find a matching arc from the last visited node, from which it can be concluded that the current word has not been seen in training, or finally (iii) it may encounter an end node, at which point the full word completion is emitted as the only known output given that sequence of keystrokes.

As a side note, trie-based retrieval in character n-gram paths according to a fixed ordering (in our case, ordered by information gain) is equivalent to Katz back-off smoothing in character n-gram language models (Zavrel and Daelemans 1997).

In the larger context of the text application, this trie is embedded in a real-time wrapper that reads each incoming keystroke, updates the character buffer (shifting it leftward with each keystroke, erasing it after each space), sends the buffer to the trie, catches the prediction emitted by the trie, and presents this to the user, who may then press a special "Accept" key (or the space bar, as implemented in many text completion interfaces) to accept the trie's suggestion.

3.2 The baseline algorithm: Recent-keystroke tries

The classic prefix trie, compressing the types of instances depicted on the left-hand side of Figure 1, is a data structure that offers a left-to-right letter-by-letter access to words in the word list. When following the keystrokes belonging to a word, the trie simply needs to be traversed from the top down. An alternative is to order the keystrokes from most recent to less recent, i.e. from right to left; examples are given in the middle of Figure 1. As each new user keystroke changes the history buffer by one position, this trie needs to be re-accessed with each keystroke. The resulting trie is different from a normal prefix trie, but has the same function; in practice, it turns out it has virtually the same performance as well, provided that this "reverse" trie also keeps the first letter in the trie (the first column of the examples in the second column of Figure 1 labeled "recent keystrokes"), to prevent it from suggesting words with a different starting letter.

We use this "recent keystrokes" trie as our baseline word completion system for the reason given in the next subsection.

3.3 Extension 1: Adding previous word context

The advantage of the recent-keystrokes trie over a normal prefix trie is that it can represent keystrokes before the current word in their natural order, simply by continuing to represent all characters typed so far from right to left, from the current word down to the previously complete word, the word before that, etc., up to some fixed buffer limit. We adopt a limit of 15 characters, which is typically wide enough to hold at least one previous word besides the keyed-in characters of the current word. We explicitly keep spaces and punctuation marks in the buffer. Examples generated this way can be found in the third column in Figure 1 labeled "recent keystrokes + context".

As these tries do not compress a word list but rather an extended character n-gram model over text, they can be expected to be larger than the baseline "recent keystrokes" tries.

3.4 Extension 2: Predicting the next word as well

If the previous word is available through a character n-gram window, and the currently keyed-in word is already becoming visible, then it is conceivable to extend the prediction to the word following the current word to be completed. In other words, the task could be extended from a unigram prediction task to a bigram prediction task. If the suggestion

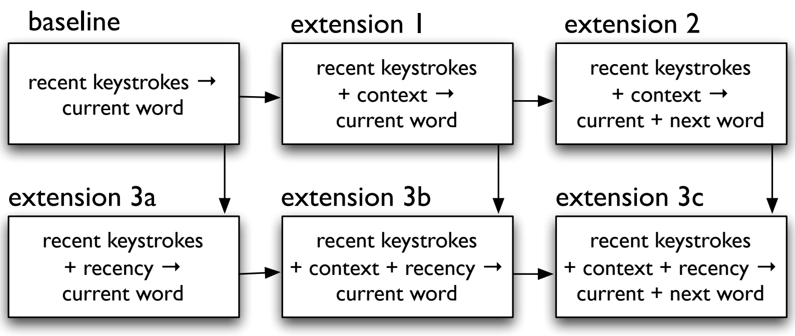

baseline
recent keystrokes → current word
extension 1
recent keystrokes + context → current word
extension 2
recent keystrokes + context → current + next word
extension 3a
recent keystrokes + recency → current word
extension 3b
recent keystrokes + context + recency → current word
extension 3c
recent keystrokes + context + recency → current + next word

Figure 2: Graphical representations of the baseline and the extensions with their interrelations. Arrows depict the "extends" relation.

for the current word is accorded by the user, this could be immediately followed by a suggestion for the second part of the predicted bigram. If that was the word the user was indeed intending to write, then all characters of that word can be saved; only one "Accept" action (one keystroke) is needed. We introduce this extension as a second variant of the baseline. In Figure 1 this extension uses the same input as the first "recent keystrokes + context" input features, but now predicting bigrams of words. Tries of this type can be expected to be larger than the tries of the first extension, as more information (longer paths) will be needed to disambiguate between word bigrams with the same to be completed left-hand side word, but different right-hand side words.

3.5 Extension 3: Adding a recency model

Following Copestake (1997) and Tanaka-Ishii (2007), the third extension we implemented is a recency model. While our second extension extends only the first extension, the third extension can be applied to the baseline system and the first and second extensions. We illustrate these interrelations in Figure 2 for clarity.

The recency model is a small continuously updated memory buffer of the last m words seen in the text so far. With every new word, elements in the buffer shift one place backwards, dropping the element at the back of the memory, and inserting the word that was just keyed in or completed. The recency model operates in parallel to the trie, and checks if the sequence of characters currently keyed in uniquely matches one of the words in the buffer. If the recency model finds a match, its suggestion is presented instead of the trie's suggestion. The motivation for the recency model is that words tend to recur burstily once they have appeared in a text, due to their association with the text's topic; the probability of words recurring in a short stretch of text is far higher than the Poisson distribution based on their frequency in a large corpus would suggest (Church 2000). A recency model may therefore provide useful guesses complementary to those of the generic trie, which has no statistical information on the text being processed.

As the recency model can be applied to the baseline model as well as the two extensions described before, we have three variants of the third extension: 3a, 3b, and 3c, as visualized in Figure 2.

3.6 Classification and storage complexity

Retrieval of classification information in a trie has a favorable upper-bound complexity of $O(f)$, where f is the number of features and hence the depth of the trie. Storage on the other hand is linearly bounded by $O(n)$, where n is the number of examples. In practice, a trie may compress the training examples by a large margin.

In the baseline experiment and in all extensions, f remains constant at character buffer size 15. In practice we do expect to find differences in storage space needed for the different extensions. Extension 1, adding previous word context, will increase the average branching factor of the tries built. Extension 2, predicting two words instead of one, will also likely increase path length and branching. Despite this we expect to find relatively little effect of these potentially large storage differences on the speed of classification.

In contrast to our fixed f, n is varied in learning curve experiments, as detailed in the next section.

4. Experimental setup

We performed experiments on comparable Dutch and English data. Dutch has a productive compounding morphology, allowing for long, relatively infrequent words. Dutch therefore has more unknown words than English (Ordelman et al. 2001), but once a long compound is predicted correctly during the first letters, many keystrokes can be saved—roughly equivalent to what could be saved in extension 2 applied to English, when predicting two English words at the same time that in Dutch would be single words.

As our Dutch text corpus for training and testing, we used a Dutch newspaper's article archive for the year 1998, comprising a total of 128 million words. We split this corpus into a training corpus of up to the first 30 million words, and a disjoint test set of the final 100,000 words. The test set thus contains articles describing events which occurred at a later point in time, and not immediately following the articles in the training set. For English we made use of the Reuters RCV1 corpus (Lewis et al. 2004), of which we also took the final 100,000 words as test data, and up to 30 million words from the start of the corpus as training data.

4.1 Experimental design and evaluation

Keeping the test sets constant for each language, we grow the amount of training material in a pseudo-exponential series, starting at 1,000 words, ending currently at 30 million. At each step in the curve we perform a full IGTree experiment in which we generate a trie, and process the test sets with the trie. With each experiment we measure the percentage of keystrokes saved: the proportion of the total number of keystrokes needed to generate the entire test text when all correct suggestions of the system are effectuated as soon as they are presented, over the total number of keystrokes needed when the full test text would

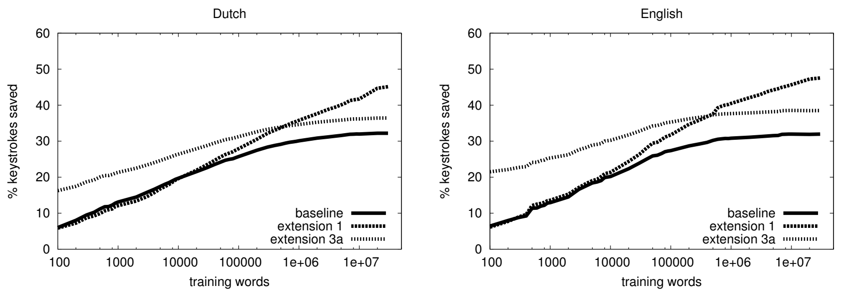


Figure 3: Learning curves on Dutch (left) and English (right) data of the baseline system (the "recent keystrokes" trie), extension 1 (the "recent keystrokes + context" trie), and extension 3a (adding a recency model to the baseline system).

be keyed in character by character. Pressing the "Accept" button is also counted as a keystroke.

Counter to Lesher et al. (1999), Fazly and Hirst (2003), and several other approaches mentioned earlier, we do not consider ranked n-best lists of completion suggestions, as in many devices and circumstances it is inefficient or impossible to present these suggestion lists. Inspecting a list of suggestions also poses a larger cognitive load than checking a single suggestion, and furthermore it is unclear how scrolling, browsing and selecting items from this list should be counted in terms of keystrokes. One could adopt the reasoning that selecting the highest-ranked suggestion costs one keystroke, selecting number two costs two keystrokes: (1) go one step down the list, (2) accept the suggestion, etcetera. Both Lesher *et al.* and Fazly and Hirst report keystroke saving results with $n > 1$ ($n = 10$ and $n = 5$, respectively) but do not take the ranking of the correct suggestion into account. We choose to work with $n = 1$.

5. Results

As a first analysis, we compare the baseline system against its two direct variants, extension 2 (the "recent keystrokes + context" trie) and extension 3a (adding a recency model), both for the Dutch and the English data. Figure 3 displays the two learning curve graphs.

The graphs of the two languages are quite similar. The keystroke savings of the baseline system increase at a reasonably steady log-linear pace until about a training set of 100,000 words, after which the increase in performance decreases with more data, and even appears to flatten. The extension 1 system, adding previous-word context to the input, however, continues its approximately log-linear growth, and gains marked improvements over the baseline from 100,000 words of training onwards. Second, we observe that adding a recency model to the baseline system boosts keystroke savings considerably when the trie is trained on small amounts of data, but the surplus effect of the recency model diminishes gradually, and the curve flattens just as the baseline curve does.

Figure 4 compares extension 1 (already shown in Figure 3) and 2, and compares the two extensions with their recency-model variants (3b and 3c, respectively). The graphs

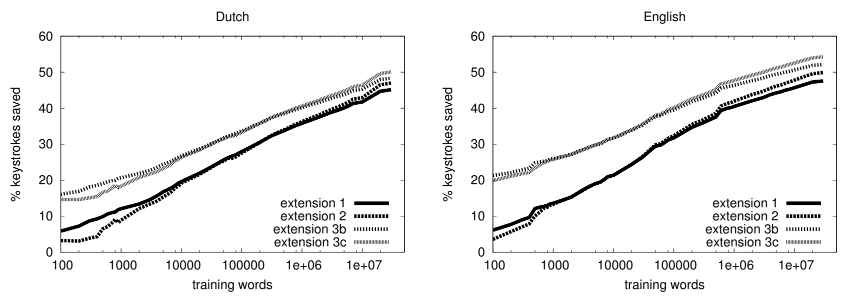


Figure 4: Learning curves on Dutch (left) and English (right) data of extension 1 (the "recent keystrokes + context" trie), extension 2 (extension 1 plus predicting the following word), extension 3b (adding a recency model to extension 1), and extension 3c (adding a recency model to extension 2).

for Dutch and English look similar again, with somewhat higher scores observed for English. Compared to extension 1, predicting the next word as well (extension 2) leads to slightly better keystroke savings only at the largest training set sizes; at best, the surplus of predicting the extra word is 2.3% for English and 1.8% for Dutch).

Adding a recency model to the two extensions also leads to enhanced scores when trained on small amounts of data, but the surplus of the recency model decreases with more training data. Extension 3c represents the aggregate of all available information; its curves combine, as can be seen in the graphs of both languages, the (diminishing) effect of the recency model with the (increasing) effect of previous-word context in the input and the (modest) effect of predicting the extra next word. Extension 3 produced the highest keystroke savings: 50.1% on Dutch and 54.0% on English, both at 30 million training words.

As pointed out, all systems compared here are based on character n-grams; earlier we referred to related work on predicting the next word based on an input of previous words (Nantais et al. 2001, Shein et al. 2001, How and Kan 2005), i.e. based on word n-grams. We trained and tested a memory-based word trigram language model (Van den Bosch 2006) on the same data. The model predicts the next word on the basis of the two previously typed words. If the predicted word is correct, all characters of the entire word (plus a whitespace character) are saved. The disadvantage of this system is that it cannot suggest the completion of a word as it is keyed in, as it needs to wait for this word to be entered completely to be able to predict the next word.

Figure 5 compares the learning curve of this word trigram-based model versus our character-based baseline system for Dutch. It is clear that the word-based model produces considerably lower key savings. An analysis shows that this can be attributed largely to the fact that the word prediction is particularly successful in predicting short function words, on which few keystrokes can be saved. Trained on 30 million words, the trigram language model is able to predict 20.4% of all next words correctly, while this only accounts for 14.0% keystrokes saved. Yet, the word-based learning curve does not flatten, which is in accordance with earlier observations (Van den Bosch 2006).

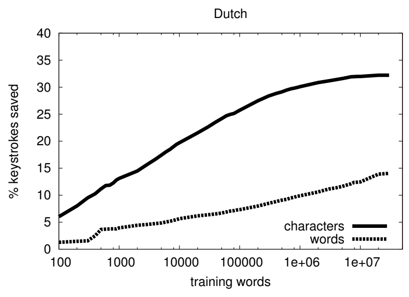


Figure 5: Learning curves on Dutch data of the (character n-gram-based) baseline system and a word-based trigram language model.

As stated earlier, adding character context to the input (extension 1) and predicting the next word as well (extension 2) can be expected to produce a larger memory footprint due to a larger branching factor in the trie, and longer paths stored in the trie. The left-hand graph in Figure 6 displays the number of trie nodes needed at the various training set sizes by the three systems trained on Dutch data. The y-axis is also logarithmic. The numbers of trie nodes needed by both extensions stand in a roughly linear relation with the number of training instances (recall the linear $O(n)$ worst case complexity); largely mimicking the results shown earlier in Figures 3 and 4, the curve of the baseline system flattens earlier and more than the other two curves[2]. Comparing the curves of the two extensions, the second extension appears to continue a near-linear trajectory. As the memory footprint of the baseline flattens more, the difference between the trie sizes of the extensions and the baseline grows to more than one order of magnitude.

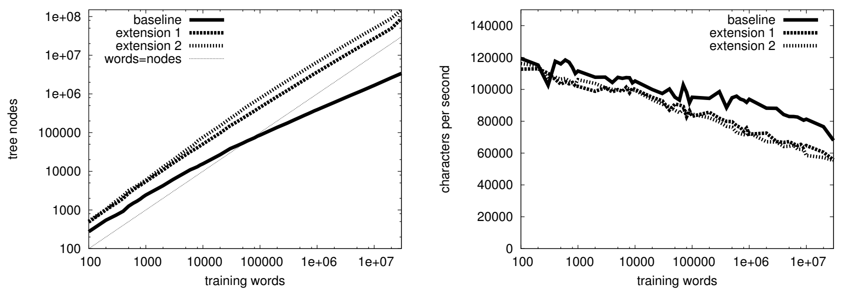


Figure 6: Left: Memory usage of the baseline system versus the first and second extensions, in terms of the number of trie nodes. Extension 1 extends the input character buffer to cover previous words as well; extension 2 adds the prediction of the next word. Right: Speed of the three word completion variants, in terms of the number of characters processed per second.

2. The current implementation of IGTree uses 40 bytes per trie node; the 604,285-node trie of the baseline system trained on 2 million words costs just over 23 MB of memory.

Besides their increased memory footprint, for practical purposes it would be good to know that the extensions are not exceptionally slower than the baseline, as the algorithm has to at least be fast enough to follow real time keystrokes, the average speed of which has been estimated at 0.12 seconds per keystroke for a good typist, and 0.28 seconds per keystroke for untrained typists (Kieras and John 1994). The worst-case complexity ($O(f)$, f being the number of features) would predict no substantial loss in speed as f remains constant in our experiments.

The right-hand graph of Figure 6 displays the speed of the three systems using the same logarithmic learning curve axis, measured on the Dutch test set in uninterrupted classification mode, on a state-of-the-art computing server. Single measurements were made per training set size, hence the occasional spike, which may be due to uncontrollable task attribution behavior of the computer's kernel. The figure shows that the speed of the three systems, despite their different memory footprints, is indeed similar. The speed of all three systems declines from about 110 thousand classifications (keystrokes) per second to around 55 thousand classifications per second for the two extensions, and around 70 thousand classifications per second for the baseline system, at 30 million words of training data.

As a final analysis, in order to gain some insight into the performance of the recency-based model in isolation and not as a companion to the trie-based word completion system, we performed a series of experiments varying the size of the memory buffer m in steps of 5, again on Dutch data. In our experiments we arbitrarily set $m = 300$, as this seemed a reasonable approximation of the average length of a newspaper article, in which words can be expected to recur burstily because of the topical coherence of the text seen so far. The results visualized in Figure 7 show that our estimate is reasonable, with 14.5% keypresses saved at $m = 300$, a point where the curve has flattened considerably and little is to be gained from further increase.

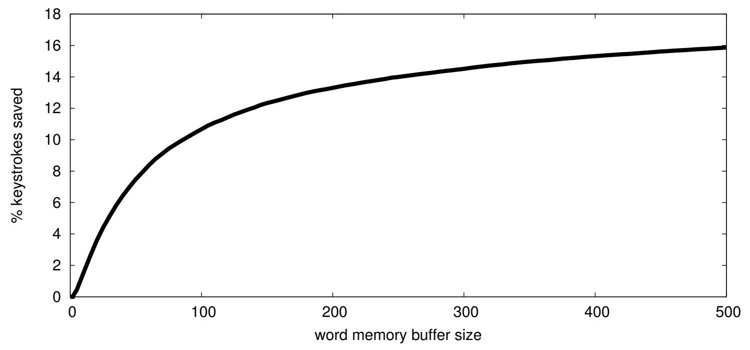


Figure 7: Percentage of keystrokes saved using the recency model in isolation, with an increasing recent-word memory buffer up to 500 words in size.

6. Discussion

Given the common wisdom on the topic of word completion, our goal in this paper was to investigate the scaling behavior of a baseline trie-based word completion system as well as that of two extensions. The first extension adds characters of previous words to the input; the second extension adds the next word to the completion output, turning it into a bigram predictor. The baseline and its two extensions were also augmented with a recency model. First, we focused on performance in terms of keystroke savings on an unseen text. We performed experiments on English and Dutch data from the news domain, and plotted learning curves based on a pseudo-incrementally increasing training set regime. In agreement with the observation made by Lesher et al. (1999), the learning curves show initial log-linear increases, but we observe flattening curves for the context-insensitive baseline system, reaching about 32% character savings in both languages at 30 million words of training data.

Extending the character input buffer to cover previous words as well leads to an increasing surplus in saved keystrokes of up to 15% in our experiments. This is in line with, but a larger effect than the 7.5% surplus reported by (Lesher et al. 1999) at a training set maximum of two million words; we do find a similar 7% increase with Dutch on that same training set size (but an 11% increase with English). The surplus increases because the baseline learning curve flattens more with more data than the curve of extension 1. Extending this further by predicting not only the current word but at the same time also the next word yields an additional small surplus of about 2% (results on the two languages differ slightly, but show the same trends and differences overall).

Finally, adding a recency model to the three systems produces an additional boost in performance, but this effect is not sustained; rather, the added effect of the recency model decreases as the systems are trained on more data. The initial boost of about 11% decreases to about 4% in our experiments. Completing words that recently occurred is a sensible idea, but with a completion system trained on more data the recency model's contribution will be increasingly less complementary.

In general the learning curve results indicate that more training data leads to increased performance of context-sensitive trie-based word completion systems; we logged keystroke savings of 50% (Dutch) to 54% (English). We also have gathered indications on whether and how much our learning curves flatten, and thus to what extent the performance could still be boosted by adding more data. The early study of Lesher et al. (1999) reports a non-trivial rate of increase up to two million training words. Using a 10-best list, they report average keystroke savings over seven different test sets of about 54%. More recently, Fazly and Hirst (2003) measured the performance of their best system on two different training set sizes, 5.6 million and 81 million words from the British National Corpus. They report relatively limited improvements on two differently-sized test sets: an increase from about 51% to 53% (recall that they draw the correct suggestion out of a ranked 5-best list). They characterize the 2% increase as "only a small improvement", suggesting a more serious flattening than we observed so far.

Despite the positive effects on performing the task, more training data also leads to larger tries; the baseline system produces lean and fast tries, while the extended systems yield tries that are at least an order of magnitude larger. Still, as our complexity analysis predicted, the effectiveness of the trie data structure leads to high processing speeds that are only about 21% slower for the extensions as compared to the baseline.

There are several obvious paths to take in future work, the first one being the continuation of the learning curves. A second extension that the literature suggests (cf. Section 2) is to use part-of-speech tagging or syntactic information, but we also quoted Fazly and Hirst's hypothesis on why the influence of this information is limited in their experiments (Fazly and Hirst 2003). Garay-Vitoria and González-Abascal (1997) report similar observations with using statistical grammatical information. The experiment nonetheless remains interesting when seen through the instrument of learning curves, as the influence of explicit linguistic information may be larger with smaller training set sizes, as shown earlier by Van den Bosch and Buchholz (2002).

Another direction of future research with a considerable practical value would be to train and test word completion systems on personalized collections of documents or social media messages (such as email or tweets), and perform matrices of experiments with in-domain/out-of-domain and generic/personalized data in all combinations of training and test set types. These dimensions are likely to have a major impact on the percentages of keystrokes saved, but we would expect the relative learning curve effects to persist.

References

Bentrup, J. A. (1987), Exploiting word frequencies and their sequential dependencies., *Proceedings of the 10th Annual Conference on Rehabilitation Technology*, RESNA, pp. 121–123.

Cagigas, S. E. Palazuelos (2001), *Contribution to Word Prediction in Spanish and its Integration in Technical Aids for People With Physical Disabilities*, PhD thesis, Universidad Politechnica de Madrid, Madrid, Spain.

Carlberger, A., J. Carlberger, T. Magnuson, S. Hunnicutt, S. E. Palazuelos Cagigas, and S. Aguilera Navarro (1997), Profet, a new generation of word prediction: An evaluation study, *ACL Workshop on Natural Language Processing for Communication Aids*, Madrid, Spain, pp. 23–28.

Church, K. W. (2000), Empirical estimates of adaptation: The chance of two Noriegas is closer to $p/2$ than p^2, *Proceedings of the 18th Conference on Computational Linguistics*, Vol. 1, pp. 180–186.

Copestake, A. (1997), Augmented and alternative nlp techniques for augmented and alternative nlp techniques for augmentative and alternative communication, *Proceedings of the ACL workshop on Natural Language Processing for Communication Aids*, Madrid, Spain, pp. 37–42.

Daelemans, W., A. Van den Bosch, and A. Weijters (1997), IGTree: using trees for compression and classification in lazy learning algorithms, *Artificial Intelligence Review* **11**, pp. 407–423.

Darragh, J., I. Witten, and M. James (1990), The reactive keyboard: A predictive typing aid, *Computer* **23** (11), pp. 41–49, IEEE Computer Society, Los Alamitos, CA, USA.

Fazly, A. and G. Hirst (2003), Testing the efficacy of part-of-speech information in word completion, *Proceedings of the 2003 EACL Workshop on Language Modeling for Text Entry Methods*, pp. 9–16.

Garay-Vitoria, N. and J. González-Abascal (1997), Intelligent word-prediction to enhance text input rate, *Proceedings of the 2nd International Conference on Intelligent User Interfaces*, pp. 241–244.

Garay-Vitoria, Nestor and Julio Abascal (2006), Text Prediction Systems: A Survey, *Universal Access in the Information Society* **4** (3), pp. 188–203, Springer.

Goodman, J., G. Venolia, K. Steury, and C. Parker (2002), Language modeling for soft keyboards, *IUI '02: Proceedings of the 7th International Conference on Intelligent User Interfaces*, ACM, New York, NY, USA, pp. 194–195.

Grover, D., M. King, and C. Kushler (1998), Reduced keyboard disambiguating computer.

Horstmann Koester, H. and S. Levine (1996), Effect of a word prediction feature on user performance, *Augmentative and Alternative Communication* **12**, pp. 155–168.

How, Yijue and Min-Yen Kan (2005), Optimizing Predictive Text Entry for Short Message Service on Mobile Phones, *in* Smith, M. J. and G. Salvendy, editors, *HCII '05: Proceedings of the The 11th International Conference on Human-Computer Interaction*, Lawrence Erlbaum Associates, Las Vegas, NV.

Kieras, D. and B. John (1994), The GOMS family of analysis techniques: Tools for design and evaluation, *Technical Report CMU-HCII-94-106*, Carnegie Mellon University.

Knuth, D. E. (1973), *The Art of Computer Programming*, Vol. 3: Sorting and Searching, Addison-Wesley, Reading, MA.

Lesher, G. W., B. J. Moulton, and D. J. Higginbotham (1999), Effects of ngram order and training text size on word prediction, *Proceedings of the Annual Conference of the RESNA*.

Lewis, D.D., Y. Yang, T.G. Rose, and F. Li (2004), RCV1: A new benchmark collection for text categorization research, *Journal of Machine Learning Research* **5**, pp. 361–397.

MacKenzie, I. S., H. Kober, D. Smith, T. Jones, and E. Skepner (2001), LetterWise: Prefix-Based Disambiguation For Mobile Text Input, *UIST '01: Proceedings of the 14th annual ACM Symposium on User Interface Software and Technology*, ACM, New York, NY, USA, pp. 111–120.

Matiasek, J., M. Baroni, and H. Trost (2002), FASTY: A multi-lingual approach to text prediction, *Computers Helping People With Special Needs*, Springer Verlag, Berlin, Germany, pp. 165–176.

Nantais, T., F. Shein, and M. Johansson (2001), Efficacy of the word prediction algorithm in wordq, *Proceedings of the 24th Annual Conference on Technology and Disability*, RESNA.

Ordelman, R., A. Van Hessen, and F. De Jong (2001), Lexicon optimization for dutch speech recognition in spoken document retrieval, *Proceedings of Eurospeech 2001.*

Shannon, C. E. (1948), A mathematical theory of communication, *Bell Systems Technical Journal* **27**, pp. 623–656.

Shein, F., T. Nantais, R. Nishiyama, C. Tam, and P. Marshall (2001), Word cueing for persons with writing difficulties: Wordq, *Proceedings of CSUN 16th Annual Conference on Technology for Persons with Disabilities.*

Swiffin, A. L., J. A. Pickering, J. L. Arnott, and A. F. Newell (1985), PAL: An effort efficient portable communication aid and keyboard emulator, *Proceedings of the 8th Annual Conference on Rehahilitation Technology*, RESNA, pp. 197–199.

Tanaka-Ishii, Kumiko (2007), Word-based Predictive Text Entry using Adaptive Language Models, *Natural Language Engineering* **13** (1), pp. 51–74, Cambridge University Press, New York, NY, USA.

Trost, H., J. Matiasek, and M. Baroni (2005), The language component of the FASTY text prediction system, *Applied Artificial Intelligence* **19** (8), pp. 743–781.

Van den Bosch, A. (2006), Scalable classification-based word prediction and confusible correction, *Traitement Automatique des Langues* **46** (2), pp. 39–63.

Van den Bosch, A. and S. Buchholz (2002), Shallow parsing on the basis of words only: A case study, *Proceedings of the 40th Meeting of the Association for Computational Linguistics*, pp. 433–440.

VanDyke, J. A. (1991), A syntactic predictor to enhance communication for disabled users, *Technical Report 92-03*, Dept. of Computer and Information Sciences, University of Delaware.

Wood, M. (1996), *Syntactic pre-processing in single-word prediction for disabled people*, PhD thesis, University of Bristol.

Zavrel, J. and W. Daelemans (1997), Memory-based learning: Using similarity for smoothing, *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pp. 436–443.