Word Completion with Latent Semantic Analysis

Tristan Miller and Elisabeth Wolf
German Research Center for Artificial Intelligence (DFKI GmbH)
Postfach 20 80, 67608 Kaiserslautern, Germany
{tristan.miller, elisabeth.wolf}@dfki.de

Abstract

Current word completion tools rely mostly on statistical or syntactic knowledge. Can using semantic knowledge improve the completion task? We propose a language-independent word completion algorithm which uses latent semantic analysis (LSA) to model the semantic context of the word being typed. We find that a system using this algorithm alone achieves keystroke savings of 56% and a hit rate of 42%. This represents improvements of 4.3% and 12%, respectively, over existing approaches.

1. Introduction

Word completion is the task of predicting and automatically completing words that the user is in the process of typing. Such tools can prevent misspellings, help develop writing skills, and accelerate typing speed by saving keystrokes. (This last benefit is particularly important for users of keyboardless devices, such as mobile phones and PDAs, as well as for users with physical disabilities.) During typing, the user is offered a *prediction list* of words beginning with the letters, or *word prefix*, thus far typed. If the intended word is in the prediction list, the user can select it with a single keypress; otherwise, he continues typing until the word appears in the list or until he types the complete word.

The job of the word completion algorithm is to determine which words appear in the prediction list, the idea being to maximize the probability of presenting the user with the correct word. The earliest word completion algorithms used simple statistical methods, such as word or word-pair frequencies, to rank words in the prediction list. The frequencies are derived from a corpus of written text, though some systems dynamically update the frequency table to adapt to the user's writing style. More advanced systems incorporate syntactic data, such as part-of-speech tags and grammar rules, to avoid suggesting words which are grammatically incorrect in the given context.

However, even systems that combine statistical and syntactic data can suggest words that are *semantically* inappropriate. For instance, the writer of an essay on music who begins typing *Mende…* is far more likely to intend the completion to be *Mendelssohn* than *Mendel* or *Mendeleyev*, even though all three are proper nouns which may be equally statistically likely (in a unigram or bigram model, at least). There have been some recent attempts at incorporating semantic information into the completion task (e.g., [4]), but most of these require language-specific tools such as WordNet [5], and many operate only on words of a particular part of speech.

In this paper, we propose a word completion algorithm which aims to suggest words best fitting the semantic context. Semantic information is provided by latent semantic analysis [2], a language-neutral technique based on the vector-space model of information retrieval. A full explanation of the technique is beyond the scope of this paper, but in brief, LSA takes a term–document co-occurrence matrix and perturbs the values in such a way that the processed matrix captures the underlying transitivity relations among terms, allowing for identification of semantically similar documents which share few or no common terms withal. Likewise, terms may be compared by examining their vectors across documents. Terms may be judged semantically similar even though they never occur in the same document together.

2. Algorithm

To initialize the system, it is necessary to transform a text corpus into a reduced-dimensionality term–document matrix, where each "document" vector $d \in D$ of the matrix corresponds to a sentence of the corpus, and each "term" vector $t \in T$ is a unique word. (How to produce such a matrix is treated in detail in the available LSA literature.) Ideally the corpus should be large enough to contain any word the user is likely to type. Once the matrix is built, pairs of term or document vectors can be compared via the cosine coefficient, yielding a "semantic similarity" score in the range $[-1, 1]$.

Now, say the user is in the process of typing a word w with prefix $\mathrm{pre}(w)$. Then we define the *context* $C = \langle c_1, c_2, \ldots, c_{\ell-1}, c_\ell \rangle$ as the sequence of up to ℓ words immediately preceding w in the document. We refer to ℓ as the *context length*, though near the beginning of the document the actual length of the context, $|C|$, may be less than ℓ.

A *candidate word* $k \in K \subseteq T$ is any word whose prefix is the same as that of w—i.e., $K = \{k | k \in T \wedge \mathrm{pre}(k) = \mathrm{pre}(w)\}$. The best candidates comprise the *prediction list* $P \subseteq K$, which the user (or system) caps at a maximum length of m. Again, it is possible that $|P| < m$ if there are fewer than m known words with the given prefix.

We investigate two methods to populate P, both of which involve finding candidates with a high semantic similarity to the context. For the first method, *context as pseudo-document* (CAP), we convert C into a new document vector, d_C, which is the sum of all weighted term vectors t_i whose terms appear in the context. This new document vector then undergoes the same dimensional reduction as the rest of the matrix, a process known as *folding in* [2]. Following this, C can be compared to any term or document in the matrix with the cosine measure. For example, the semantic similarity of a candidate word k to the context is $\mathrm{sim}(k, C) = \cos(k, d_C)$. We therefore compute cosine scores for each possible k, and assign to P the m candidates with the highest scores.

In the second method, *sum of similarities* (SOS), instead of treating the context as a monolithic document vector, we compare the candidates to each word in the context individually. The similarity score for the context is thus the sum of similarity scores of each word in the context, possibly weighted with a monotonically non-increasing function f to give higher importance to words near the end of the context (i.e., near w):

$$\mathrm{sim}(k, C) = \sum_{i=1}^{|C|} \cos(k, c_i) \cdot f(|C| - i)\,.$$

We investigate three different weighting functions: a linear weight $f_1(x) = 1 - x \div |C|$, a hyperbolic curve $f_2(x) = (x+1)^{-1}$, and a linear step function $f_3(x)$. A similarity measure without weighting can be achieved by using a constant function $f_0(x) = 1$. (See Figure 1.)

2.1. Complexity analysis

The construction of the reduced-dimensionality matrix from the text takes about $O(|T| \cdot |D|^2)$ time, though fortunately need be done only once. Once constructed, terms and documents can be compared in time linear to the total number of each type of entity being compared—that is, $O(|T| \cdot |D|)$, $O(|T|^2)$, or $O(|D|^2)$.

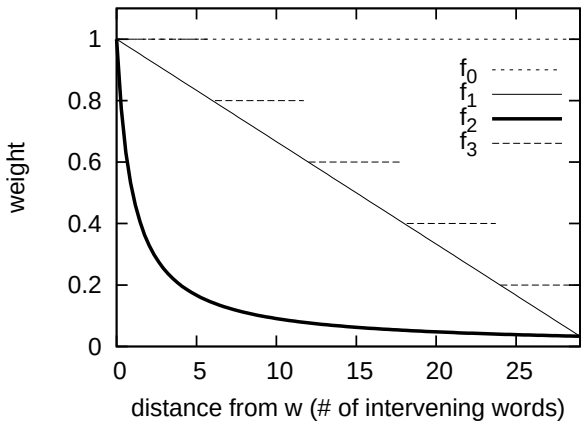

Figure 1. SOS weighting functions for $\ell = 30$

To determine the suitability of a single candidate in the CAP method, we need time linear to the number of terms in the context to build the document vector, plus time for one term–document comparison: $O(|C| + |T| \cdot |D|)$. To do so for all candidates, then, is an $O(|K| \cdot |C| + |K| \cdot |T| \cdot |D|)$ operation.

The calculation for SOS is more complicated, since a term–term comparison must be performed for each word in the context. This gives us $O(|K| \cdot |C| \cdot |T|^2)$, though various optimizations, such as memoization can be used to significantly cut down on running time. The use of any of the weighting functions we investigate has no significant impact on computational complexity.

3. Evaluation

We designed and implemented an automated simulation to test the variants of our word completion algorithm. The three-part system, illustrated in Figure 2, simulates a user who is assisted by a word completion tool.

The first component preprocesses the training corpus and generates the reduced-dimensionality term–document matrix. Our training corpus consists of 95% of the documents of the *Reuters-21578* news corpus [3], pre-processed to remove stop words (conjunctions, pronouns, etc.) and punctuation. The remaining 5% is used as the test corpus (W); the user simulation component reads documents from the test corpus and passes them character by character to the word completion component, which replies with a prediction list. Whenever the prediction list contains the word currently being "typed", the simulated user accepts the completion and skips ahead to the next word.

The simulation allows us to modify various parameters so that the optimal combination can be determined empirically; in this study we focus on just three. The first, obvi-

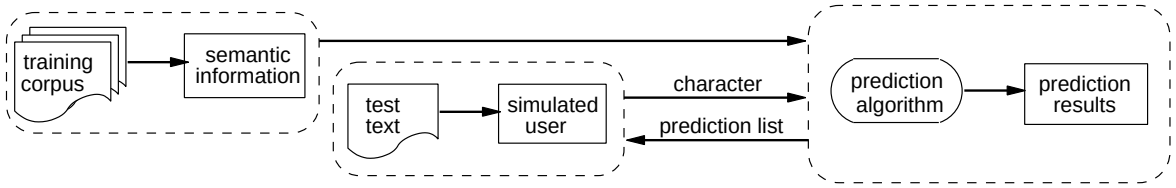

Figure 2. Simulation architecture

ously, is the prediction list population method (including the weighting function). The second is context length (ℓ), for which we tested values from 5 to 30 words. If the context is too short or too long, the topic it encapsulates may be too narrow or too broad to compare with the candidate words. The third parameter is the matrix dimension retention r, by which we mean the percentage of matrix dimensions retained during LSA's matrix recomposition. We investigate values of 0.4% to 8%. Our simulation also allows us to modify the prediction list length (m). The longer the prediction list, the higher the probability that the intended word appears therein, but longer lists require more time and effort from a human user. Therefore all tests were conducted with $m = 5$, which is a typical default for commercial completion systems.

Performance of the system is assessed with three standard measures:

Keystroke savings, the most important measure, is the percentage of keystrokes that the user saves by using the word completion utility:

$$\mathrm{KS} = 100 - \frac{100}{|W|} \cdot \sum_{w \in W} \frac{s_w + 1}{\mathrm{len}(w)} \,,$$

where $|W|$ is the number of words in the test corpus, s_w is the number of keystrokes used to type a given word w, $+1$ is the one additional keystroke to choose the appropriate word in the prediction list, and $\mathrm{len}(w)$ is the number of characters in w—i.e., the number of keystrokes that would have had to be typed without the word completion utility.

Hit rate refers to the percentage of keystrokes after which the intended word appears in the prediction list:

$$\mathrm{HR} = 100 \cdot \left[\sum_{w \in W} \mathrm{in}(w) \right] \div \sum_{w \in W} s_w \,,$$

where

$$\mathrm{in}(w) = \begin{cases} 1 & \text{if } w \in P \text{ after typing } s_w \text{ characters;} \\ 0 & \text{otherwise.} \end{cases}$$

Keystrokes until prediction is the mean number of keystrokes until the intended word appears in the prediction list or is completely typed:

$$\mathrm{KUP} = \frac{1}{|W|} \cdot \sum_{w \in W} s_w \,.$$

4. Results

We ran our simulation with the *context as pseudo-document* algorithm using various combinations of context length and matrix retention. The results for keystroke savings are summarized in Table 1; the boldface numbers mark the highest value for each context length. Irrespective of matrix retention, we see a gradual increase in keystroke savings across context length. However, no one value of matrix retention achieved the best keystroke savings across all context lengths. The values obtained for hit rate and keystrokes until prediction follow a similar trend, with results improving with greater context length but with no one matrix retention value as the winner.

matrix	context length (ℓ)				
retention (r)	5	10	15	20	30
0.4	**42.96**	**43.63**	**44.38**	44.68	45.16
2.0	41.94	43.40	44.31	**44.93**	**45.44**
4.0	41.01	42.75	43.73	44.32	45.32
8.0	39.37	41.62	42.92	43.75	44.78

Table 1. Keystroke savings: CAP

We ran the same test using the *sum of similarities* algorithm with no (i.e., constant) weighting. For this algorithm it is particularly important to find a good balance between performance and context length, because, as shown in §2.1, the similarity calculation is an order of magnitude more costly. Table 2 summarizes results for keystroke savings. This time keystroke savings correlates with context length only for $r \geq 4.0$. The best overall results (including those for hit rate and keystrokes until prediction) are obtained with $r = 8.0$.

To see whether the use of a weighting function could further improve the results for SOS, we reran the SOS simulation at $r = 8.0$ once with each of the three functions introduced in §2. The results for keystroke savings, shown

matrix	context length (ℓ)				
retention (r)	5	10	15	20	30
0.4	49.50	49.96	49.81	49.61	49.15
2.0	49.44	50.85	51.40	51.65	51.49
4.0	49.98	51.66	52.29	52.76	52.96
8.0	**51.61**	**53.24**	**54.11**	**54.54**	**55.05**

Table 2. Keystroke savings: SOS

in Table 3, show that any weighting function is a slight improvement over no weighting. The overall best combination of weighting and context length is the step function f_3 with $\ell = 30$. This yields an improvement of 0.98% over the best SOS without weighting. This combination is also the best for HR and KUP, with improvements of 1.8% and 1.7%, respectively.

weighting	context length (ℓ)				
function	5	10	15	20	30
f_0 (none)	51.61	53.24	54.11	54.54	55.05
f_1 (linear)	51.66	53.33	54.33	54.86	55.56
f_2 (hyperbolic)	**52.21**	**53.69**	54.45	54.87	55.35
f_3 (stepwise)	52.04	53.51	**54.49**	**55.07**	**55.59**

Table 3. Keystroke savings: weighted SOS

In summary, for all algorithms tested, the best results were obtained with a context length of 30. Table 4 shows all three performance measures for both algorithms—CAP at its optimal matrix retention of 2.0%, and SOS at its best of 8.0%. SOS greatly outperforms CAP in all three measures, with a 21% improvement in keystroke savings, a 34% increase in hit rate, and a 23% reduction in mean keystrokes until prediction. The addition of the stepwise linear term weighting to SOS increases these respective figures to 22%, 36%, and 25%.

algorithm	KS	HR	KUP
CAP ($r = 2.0$)	45.44	31.20	3.14
SOS + f_0 ($r = 8.0$)	55.05	41.66	2.41
SOS + f_3 ($r = 8.0$)	**55.59**	**42.39**	**2.37**

Table 4. CAP and SOS algorithms compared

How the best configuration of our LSA-based system compares to various third-party word completion systems (from [1]) is summarized in Table 5. The bigram predictor is statistics-based, the part-of-speech tag predictor is syntax based, and the other two methods combine statistical and syntactic knowledge. Our method outperforms all of these, with improvements of 4.3%, 12%, and 6.3%, over the best KS, HR, and KUP scores, respectively.

algorithm	KS	HR	KUP
bigram predictor	52.90	37.49	2.55
part-of-speech tag predictor	49.80	34.93	2.72
linear predictor	53.14	37.78	2.54
tags-and-words predictor	53.30	37.49	2.53
SOS + f_3	**55.59**	**42.39**	**2.37**

Table 5. SOS vs. third-party algorithms

5. Conclusion

Overall, our results show that semantic knowledge extracted through LSA offers a successful solution for word completion tasks. Our LSA-based method achieves significantly better keystroke savings, hit rate, and keystrokes until prediction than syntax- or statistics-based approaches.

There are many possibilities for future research into this approach. One is the integration of user-adaptive, syntactic, or statistical methods with our semantic system. (The use of a stemmed corpus as a surrogate for syntactic information is presented in [6].) We might also consider varying other experimental parameters, such as prediction list length, and investigating other performance measures, such as *accuracy* and *mean number of candidates* [1, p. 54]. Finally, we must eventually address the problem of integrating new terms into the matrix: past a certain point, the folding in of new terms may degrade the matrix's latent semantic structure, yet recomposing the entire matrix is a time-consuming process. A preliminary study of some of these avenues of inquiry is given in [6] and will form the basis of forthcoming papers.

References

[1] A. Fazly. The use of syntax in word completion utilities. Master's thesis, Department of Computer Science, University of Toronto, Jan. 2002.

[2] T. K. Landauer, P. W. Foltz, and D. Laham. An introduction to latent semantic analysis. *Discourse Processes*, 25(2&3):259–284, 1998.

[3] D. D. Lewis. *Reuters-21578 Text Categorization Test Collection Distribution 1.0 README File v1.3*, May 2004.

[4] J. Li and G. Hirst. Semantic knowledge in word completion. In *Proceedings of the 7th International ACM SIGACCESS Conference on Computers and Accessibility*, pages 121–128. ACM Press, Oct. 2005.

[5] G. A. Miller. WordNet: An on-line lexical database. *International Journal of Lexicography*, 3(4):235–244, 1990.

[6] E. Wolf. A semantic-based word completion utility using latent semantic analysis. Diplom-Informatik thesis, Department of Technical Sciences, University of Applied Sciences, Oldenburg/Ostfriesland/Wilhelmshaven, Emden, Oct. 2005.