Sentence Completion

Korinna Grabski
Otto-von-Guericke-Universität
FIN/IWS
Universitätsplatz 2
39106 Magdeburg, Germany
kgrabski@iws.cs.uni-magdeburg.de

Tobias Scheffer
Humboldt-Universität zu Berlin
Department of Computer Science
Unter den Linden 6
10099 Berlin, Germany
scheffer@informatik.hu-berlin.de

ABSTRACT

We discuss a retrieval model in which the task is to complete a sentence, given an initial fragment, and given an application specific document collection. This model is motivated by administrative and call center environments, in which users have to write documents with a certain repetitiveness. We formulate the problem setting and discuss appropriate performance metrics. We present an index-based retrieval algorithm and a cluster-based approach, and evaluate our algorithms using collections of emails that have been written by two distinct service centers.

Categories and Subject Descriptors

H.3.3 [**Information search and retrieval**]: Retrieval models; H.5.2 [**User interfaces**]: Natural language

General Terms

Algorithms, Experimentation

Keywords

Retrieval models, indexing methods, applications

1. INTRODUCTION

Information retrieval is concerned with the construction of methods that satisfy a user's *information need* [3]. Often, but not *necessarily*, the user has to explicitly encode the information need in a query. Here, we discuss a distinct retrieval model in which the information need consists of the remaining part of a sentence – typically part of a document – which the user is just beginning to enter. The *query* thus consists of the initial sentence fragment that has already been entered, possibly accompanied by preceding sentences. In order to be able to retrieve the sentence, we are given a domain specific collection of documents, written by the same user, or group of users.

SIGIR'04, July 25–29, 2004, Sheffield, South Yorkshire, UK.

This setting is motivated by processes such as answering questions sent by email in a call center, or writing letters in an administrative environment. Analyzing call center data, we found that outbound messages contain many similar, often semantically equivalent sentences. However, templates and maintained lists with answers to frequently asked questions are not widely used in call center environments; possibly because of lack of the administrative structures and commitment required to maintain such templates. Our approach is therefore to integrate retrieval functionality into the human computer interface.

Our paper is organized as follows. We review related work in Section 2. In Section 3, we discuss the problem setting and appropriate performance metrics for sentence completion methods. We present a retrieval algorithm in Section 4 and a clustering approach in Section 5. In Section 6, we conduct experiments with data collections from two call centers. Section 7 concludes.

2. RELATED WORK

Darragh and Witten [4] have developed an "interactive keyboard" that uses the sequence of past keystrokes to predict the most likely succeeding keystrokes. Clearly, in an unconstrained application context, keystrokes can only be predicted with limited accuracy. By contrast, in the very specific context of entering URLs, completion predictions are commonly provided by web browsers [6].

Predicting the next action of a user is a problem that has been studied in the area of human computer interface design. Motoda and Yoshida [16] and Davison and Hirsch [5] developed a Unix shell which predicts the command stubs that a user is most likely to enter, given the current history of entered commands. Korvemaker and Greiner [12] have developed this idea into a system which predicts whole command lines. The Unix command prediction problem has also been addressed by Jacobs and Blockeel who infer macros from frequent command sequences [8] and predict the next command using variable memory Markov models [9, 10].

Similar approaches have been applied to predict the link that a user browsing the web is likely to follow, and to predict user actions in other domains [14, 7]. Often, the goal of such user models is to detect anomalies and fraud.

While these previous studies have inspired our work, they focus on predicting one of a given set of possible actions, given a trace of actions. By contrast, the problem which we discuss here, possesses a new dimension. In the problem setting at hand, the "action" which is to be predicted is the remaining part of a sentence in natural language. However,

we assume an environment with a relatively narrow distribution over possible texts – such as a call center – and we are given a (typically very large) application specific corpus. Hence, we are facing an *information retrieval* problem.

A possible approach for this problem might be to learn a language model, and to construct the most likely sentence, given its initial fragment. However, as we know from speech recognition and machine translation, constructing syntactically and semantically proper sentences automatically is a very hard problem (the decoding problem). We therefore consider sentence completion to be a problem that is most naturally and most effectively addressed by information retrieval methods.

An information retrieval approach to sentence completion involves finding, in a corpus, the sentence which is most similar to a given initial fragment. Several approaches to solving this problem are possible. Without index structure, the problem could be solved by approximate string matching [18, 13]. This solution can be sped up by using inverted index files [15, 19, 1, 2]; here, the occurrences of the query terms are merged and approximate string matching is performed on the corresponding text positions.

Our solution extends this approach in two respects. Firstly, given a sentence fragment of length m, we search for sentences whose m initial words are most similar to the query fragment in the vector space model (rather that differing in at most k terms). Secondly, we present a pruning algorithm that interleaves accessing of indexed sentences with pruning of sentences which need not be accessed because they cannot be more similar to the query fragment than the best sentence currently found.

Processing nearest neighbor queries with vector space similarity and index structures has been studied in the context of biological sequence data [17]. Kahveci and Singh [11] discussed a wavelet based index structure for strings.

3. PROBLEM SETTING

The framework which we propose differs in small but significant ways from the standard retrieval models. We are given a domain specific collection of documents written by a user (or group of users). The user's *information need* is (the remaining part of) a sentence which he or she intends to write and is beginning to enter. The query is given by an initial sentence fragment. This information need intrinsically refers to the user's intention; it is therefore clear that the problem cannot be solved in all cases.

Our postulate, however, is that this framework is interesting because in specific environments the domain specific document collection does in fact provide the information required to satisfy the user's information need in many cases. We pose the sentence completion problem as follows.

PROBLEM SETTING 1. *We are given a domain specific document collection. Furthermore given an initial document fragment, the sentence completion problem is to identify the remaining part of the sentence that the user currently intends to write.*

This problem setting can naturally be generalized along several dimensions. Firstly, it seems reasonable to consider additional attributes of the current communication process, such as other documents that are currently visible on the user's display. Secondly, instead of demanding that the current sentence be completed entirely, it may often be more natural to predict a string up to the point where the uncertainty about the remaining words of the sentence increases.

The algorithms which we present in this paper address a more specialized version of the sentence completion problem. We consider only the initial fragment of the current sentence as input, disregarding all preceding sentences and inter-sentential relationships.

What is the proper metric for evaluating the performance of sentence completion methods? Our definition of the problem setting refers to the user's intention. Hence, a perfect evaluation metric has to refer to the user's reaction – acceptance or rejection – to proposed sentence completions. At any time, a sentence completion system can, but need not, produce an output. This output is a single sentence. The user interface can, for instance, display the retrieved completion in a bubble and insert the text when the user presses the "tab" key. When a completion is proposed which the user accepts, he or she benefits, whereas a false proposal may unnecessarily distract the user's attention.

A typical sentence completion method will calculate a confidence measure for the completion of a given query sentence. A completion is proposed when the confidence exceeds some threshold value. We characterize the performance of a sentence completion algorithm with respect to a given test collection in terms of two quantities: *precision* and *recall*. By varying the confidence threshold, we obtain a *precision-recall curve* that characterizes the completion method.

For a given initial fragment for which the confidence threshold is exceeded and an algorithm therefore produces a completion, the precision is 1 if the completion is accepted by the user, 0 otherwise. The precision of a collection is averaged over all initial fragments. Hence, the *precision* of an algorithm for a test collection is the number of completions accepted by the user over the number of all proposed completions (Equation 1).

For a given initial fragment, a recall of 1 is achieved when a completion is produced which the user accepts. If the user rejects the produced completion, or if no completion is produced, recall for that initial fragment is zero. For the whole collection, the recall is obtained by averaging over all sentences, and over all initial fragments of each sentence. Hence, the *recall* of an algorithm with respect to a test collection is the number of accepted completions over the number of all queries (Equation 2). Note that our definition of recall does not refer to the set of sentences that are retrievable for a given initial fragment.

$$Precision = \frac{\text{accepted completions}}{\text{predicted completions}} \quad (1)$$

$$Recall = \frac{\text{accepted completions}}{\text{queries (initial word sequences)}} \quad (2)$$

Whether it is more natural to average precision and recall over all initial strings, or over all initial sequences of entire words, depends on the user interface; we consider initial sequences of entire words.

In order to obtain a performance measure that can be calculated without any "human in the loop", we make a simplifying assumption: we assume that the user will accept any sentence that is semantically equivalent to the sentence which the user intends to write. We can thus measure the performance using a test collection; all sentences in this collection have to be manually grouped into classes of seman-

tically equivalent sentences. A single instance of a sentence completion problem can be simulated by drawing a target sentence from the test collection, and using an initial fragment as query. When the system's confidence exceeds the threshold and the retrieved sentence lies within the same equivalence class as the target sentence, this is counted as an accepted completion.

4. RETRIEVAL ALGORITHM

Our general approach is to search for the sentence whose initial words are most similar to the given initial fragment in vector space representation; ties are broken in favor of the more frequent sentence.

For each training sentence d_j and each length ℓ, we first calculate a TF-IDF representation of the first ℓ words: $f^{\ell}_{i,j} = normalize(TF(t_i, d_j, \ell) \times IDF(t_i))$. The inverse document frequency refers to the number of *sentences* containing term t_i. The similarity between two vectors is defined by the cosine measure. When comparing a sentence fragment of length ℓ to a complete sentence, the respective TF-IDF representation of the first ℓ words of the sentence is used. The easiest way to find the best fitting sentence is to sequentially compare each sentence with the fragment. However, this has a runtime of $O(n)$.

Therefore, we develop an indexing algorithm that still ensures finding the best sentence but typically has a sub-linear behavior. This algorithm is based on an inverse index structure (see Figure 1) that lists, for each term, the sentences in which the term occurs (the postings). The postings lists are sorted according to a relation "$<$" that is defined on sentence pairs as follows: $s_1 < s_2$ if s_1 appears in the document collection more frequently than s_2, or if both appear equally frequent and s_1 is alphabetically smaller than s_2. s_1 and s_2 are never lexicographically identical beacuse sentence instances are mapped to an entity with a frequency count. This ensures a canonical ordering of the sentences in the postings lists.

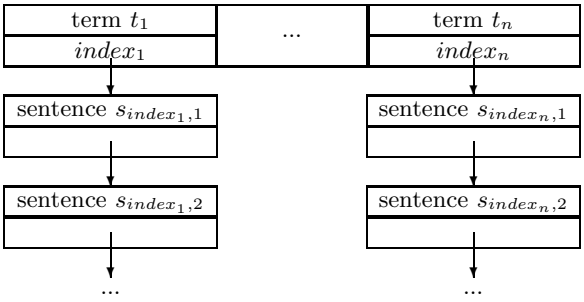

Figure 1: Inverted index structure over the data.

When searching for the best sentence, it is only necessary to look through the lists that belong to the terms that occur in the fragment, as at least one correct term should be in the best sentence.

A further reduction can be achieved by utilizing an upper bound on the similarity – which is this algorithm's main novel aspect. Since we have normalized TF-IDF vectors, we can calculate and bound the similarity between f and s as in Equation (3).

$$sim(s, f) = \sum_{j=1}^{n} s_j f_j = \sum_{j:s_j \neq 0 \wedge f_j \neq 0} s_j f_j \leq \sum_{j:f_j \neq 0} f_j \quad (3)$$

When looking through the postings lists, we will eventually reach the end of each one. If the index list $index_j$ is empty, all remaining sentences do not include the respective word. This means, the respective s_j's are 0. Therefore, we can bound the similarity of each of the remaining sentences and the fragment as in Equation 4.

$$sim(s, f) \leq \sum_{j:index_j \neq \emptyset} f_j \quad (4)$$

If this bound is smaller than the best similarity already found, the algorithm can be stopped, because there is no better sentence left to find. Otherwise, the first sentence in the index has to be examined in greater detail. Before it is necessary to look at the specific vector, a tighter upper bound on the similarity between this sentence and the fragment can be calculated for the smallest (with respect to "$<$") first sentence s over all postings lists. Recall that we have sorted the postings according to our canonical ordering "$<$". When s is the smallest sentence among the first sentences of all postings lists but s is not the first sentence of some postings list $index_j$, then we know that t_j does not occur in s (otherwise, since s is the smallest sentence according to "$<$", it would be the first element of t_j's postings list). Equation 5 bounds the similarity of f and s by summing over the terms whose first posting is s.

$$(\forall index_j : s < s_{index_j,1}) \Rightarrow sim(s, f) \leq \sum_{j:s=s_{index_j,1}} f_j \quad (5)$$

If this estimate is smaller than the best similarity already found, the sentence can be skipped and the algorithm continues with the next sentence. Only otherwise, the real similarity has to be calculated.

The algorithm can also terminate once a sentence with greatest possible similarity of one has been found, because the sentences are sorted by decreasing frequency. The complete retrieval algorithm is displayed in Table 1.

THEOREM 1. *The algorithm in Table 1 accepts an initial fragment (query) q of length ℓ and an indexed collection with TF-IDF vectors of initial fragments of length ℓ and returns the most similar and secondly most frequent sentence $s = \text{argmax}_{s \in S_{ms}} freq(s)$ with $S_{ms} = \{s | \text{argmax } sim(f, s^{\ell})\} = \{s | \text{argmax} \langle f, s^{\ell} \rangle\}$, where f is the vector representation of q and s^{ℓ} refers to the ℓ initial words of s.*

Proof (sketch). The proof can be split into two parts:

1. The algorithm always stores the best sentence s_{best} seen so far.
2. When the algorithm terminates, there is no better sentence than s_{best} in the remaining sentences.

First, we want to show (1). In the first iteration of the loop, the first sentence in the index structure becomes the best sentence s_{best}. This follows from the initialization of the best similarity with -1. As the smallest possible value for the similarity is zero, the first sentence is always better than -1.

Table 1: Retrieval algorithm.

Input: Inverted index, sentence fragment q of length ℓ.

1. Construct an inverted index for the query by copying the sentence lists for the terms in q from the global inverted index structure. **Let** $index_1, \ldots, index_\ell$ be the postings lists for terms $t_1, \ldots, t_\ell$.
2. **Let** f be the TF-IDF representation of q.
3. **Let** s_{best} be the best sentence found so far. **Set** $s_{best} = NULL$.
4. **Let** sim_{best} be the similarity between f and s_{best}. **Set** $sim_{best} = -1$.
5. **Do Until** break
   (a) **Determine** upper bound $sim_{UBall} = \sum_{j:index_j \neq \emptyset} f_j$ for all sentences in the index structure.
   (b) **Determine** the first sentence in the index structure s_{first}, which is the smallest first sentence over all postings lists according to the canonical sorting criterion "<" ($s_{first} = \min_j\{s^\ell_{index_j,1}\}$).
   (c) **If** $s_{first} = NULL$ or $sim_{UBall} \leq sim_{best}$ **Then** break.
   (d) **Determine** upper bound $sim_{UBfirst} = \sum_{j:s_{first}=s^\ell_{index_j,1}} f_j$ for s_{first}.
   (e) **If** $sim_{UBfirst} > sim_{best}$ **Then**
      i. **Determine** true similarity $sim_{first} = sim(s_{first}, f)$.
      ii. **If** $sim_{first} > sim_{best}$ **Then**
         A. **Set** $s_{best} = s_{first}$, $sim_{best} = sim_{first}$.
         B. **If** $sim_{best} = 1$ **Then** break.
   (f) **Move** pointers to next element in all index lists that include s_{first}, thereby removing all postings of s_{first} from the index. Let $s_{index_j,1}$ be the new first posting of term t_j.

Return s_{best}.

According to Equation 3, the similarity between f and *any* indexed sentence can be bounded by $sim_{UBall} = \sum_{j:index_j \neq \emptyset} f_j$. When $sim_{best} \geq sim_{UBall}$, then no remaining sentence contains sufficiently many query terms to outperform s_{best}. When $sim_{best} = sim_{UBall}$, then there may be another sentence equally similar to s_{best}. However, as our canonical ordering prefers frequent sentences, there can be no equally similar sentence more frequent than s_{best}.

Equation 5 bounds the similarity for the smallest indexed sentence. Therefore, if $sim_{UBfirst} < sim_{best}$, then s_{first} is strictly worse than s_{best}. If $sim_{best} = sim_{UBfirst}$ then we have already found a more frequent sentence s_{best} which is at least as good as s_{first}; in both cases, s_{first} does not need to be investigated. Only if both criteria do not exclude the current sentence, the real similarity has to be determined. If it is higher than the best similarity, the current sentence is better and will be stored. Otherwise, the best sentence is not changed. This proves that the algorithm always stores the best fitting sentence.

Now, we show (2). There are three situations, in which the algorithm terminates. In the first case, the index structure is empty. This implies that the best sentence has already been found because all sentences that have not been ruled out have been checked. In the second case, $sim_{UBall} \leq sim_{best}$ is valid. In this case, all sentences remaining in the index structure are worse than the best sentence found so far by definition of sim_{UBall}. In the third case, the similarity between the best sentence and the fragment is one. In this case, the algorithm is allowed to terminate, because one is the highest possible similarity and the sentence has the highest possible frequency of all sentences with a similarity of one because of the sorting criterion "<" (frequency sorting). This proves that the algorithm only terminates if there is no better sentence than s_{best} in the remaining sentences. This completes the proof. ∎

The worst case runtime is still $O(n)$, since there is the possibility that the index includes all sentences and the best one is the last. However, this case is very rare. The best case has a time complexity of $O(1)$. It occurs when already the first sentence has a similarity of 1. Nevertheless, the most interesting question is how this algorithm behaves on practical collections. We will explore this issue in section 6.

5. DATA COMPRESSION BY CLUSTERING

The algorithm presented in the last section has to store vector space representations of the whole training data and access the most similar sentence in order to retrieve it. An inverted index improves access time but leaves us with the problem of having to store and access a potentially huge amount of data.

When trying to address this problem, one can exploit the fact that rare sentences are less likely to contribute to predictions than clusters of frequently occurring similar sentences. Hence, removing these elements from the training data appears to be a reasonable heuristic.

In the following, we describe the clustering algorithm for finding groups of semantically equivalent sentences which we use in our experiments. Our problem cannot easily be solved with an agglomerative hierarchical clustering method – such as the matrix update algorithm – because the distance matrix has a memory requirement of $O(n^2)$ which is prohibitive for any reasonably large collection.

Since we have no prior knowledge on the number of clusters, we run the EM algorithm with mixtures of two Gaussian models recursively. Each data element is assigned to the cluster with higher likelihood, and the clustering is started recursively. We end the recursion when the number of instances in a cluster falls below a threshold, or the variance within the cluster falls below a variance threshold.

The Gaussians that describe the clusters are characterized by a mean vector and a scalar variance. We refrain from characterizing the Gaussians with their covariance matrix since this would be quadratic in the dictionary length.

The result of the clustering algorithm is a tree of clusters. In the leaf nodes, it contains the groups of – ideally semantically equivalent – sentences. The similarity of the sentences is quantified by the variance. When the variance of a cluster falls below a certain threshold, it is assumed to

only contain semantically equivalent sentences. The data is reduced by pruning all leaves with less than a minimum number of sentences (in our experiments, 10).

The tree can also be used to access the data more quickly. When predicting a sentence, the algorithm first decides for a cluster by greedily moving through the tree from the root until a leaf node is reached. Then it chooses a sentence by the presented algorithm from the data in this cluster.

Table 2 shows characteristic sentences (translated from German into English) from the ten largest clusters found in collections of emails sent by the service centers of an online-provider of education services (left hand side) and a large online shop (right hand side). The clusters contain variations of sentences most of which we would intuitively expect to be used frequently by service centers. Surprisingly, the salutation lines for two individuals ("Dear Mr. X", "Dear Mr. Y") appear in the set of most frequent sentences. However, a closer look at the data reveals that, in the period in which the data were recorded, these customers did in fact ask substantially many questions.

6. EMPIRICAL STUDIES

In this section, we want to investigate the relative benefits of the retrieval, and the clustering approach to sentence completion. We also want to shed light on the question whether sentence completion can have a significant beneficial impact in practical applications.

We use two collections: collection A has been provided by an online education provider and contains emails that have been sent to students – or prospective students – by the service center, in response to inquiries. Collection B has been provided by a large online shop and contains emails that have been sent in reply to customer requests. Both collections have about the same size, which is around 10000 sentences. They differ in the distribution over text; collection A includes more diverse sentences than collection B.

Note that both data sets contain only answer messages, not the corresponding questions. It is intuitively clear that the questions contain some information about the sentences that will most likely occur in the answer. However, the algorithms that we compare base their prediction only on the initial fragment of the sentence that the user is currently entering, and we assume the perspective of the service center that writes the answer messages.

We manually partition the collection of sentences into equivalence classes. Each element of an equivalence class can reasonably be exchanged for each other element without changing the semantics of the message. For the experiments, each data collection was split at random into a training set (75%), and a test set (25%).

The indexing and clustering algorithms are provided with the training part. In order to measure precision and recall, we iterate over all sentences x of the test set and, for each sentence, over all possible sequences that contain the first k words, for all possible values of k. Each such initial sequence is counted as a query. We use the initial sequence as input to the studied algorithm; the algorithm returns either no prediction, or else one single completion prediction.

Each initial word sequence is counted as a query. When the similarity between query fragment and most similar sentence exceeds the threshold parameter, the algorithm produces a completion prediction. If prediction y is returned, we check whether the actual sentence x and the prediction y lie in the same equivalence class. If this is the case, we count a correct prediction; if x and y lie in distinct classes, we count an incorrect prediction. We count the *precision* of a method as the number of correct predictions over the number of both, correct and incorrect prediction. *Recall* is calculated as the number of correct predictions over the number of queries.

Figure 2 shows the precision for collection A for the retrieval algorithm using our index structure algorithm and for the algorithm extended with clustering, depending on the initial fragment length. Both algorithms are investigated for two different confidence (similarity) threshold values. Only if the similarity is greater or equal than 0.6, or 1.0 respectively, a prediction is made. Figure 3 shows the corresponding recall, again depending on the fragment length. Figures 4 and 5 show the same information for collection B.

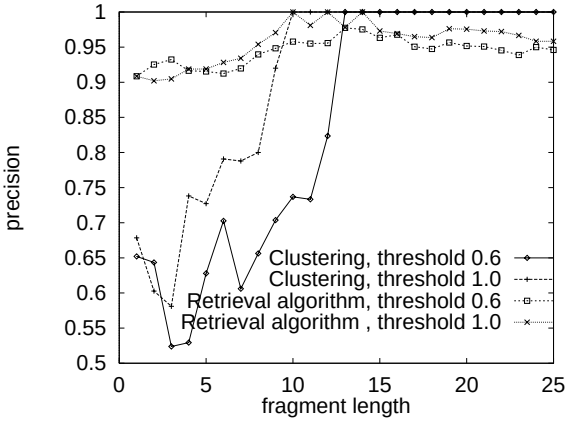

Figure 2: Precision for collection A

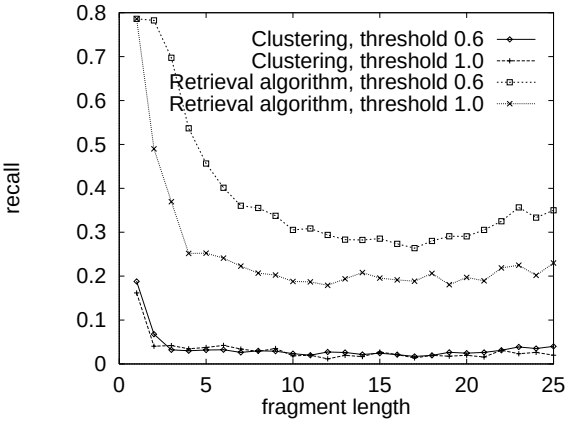

Figure 3: Recall for collection A

For collection A, the retrieval algorithm always has a high precision of above 0.9. Adding the clustering reduces the precision gained with shorter initial fragments. However, it raises precision for longer initial fragments. In collection B, there is no real difference between both algorithms. This is probably due to the lower diversity in this collection. In all runs, a pruning threshold of 10 sentences was used.

The recall depends on the threshold value used for the

Table 2: Characteristic sentences of the identified clusters.

Online education provider	Online shop
Please do not hesitate to ask further questions.	(lines from the signature file)
Please let me know if you have any further questions.	I would like to answer your question as follows.
I hope that you find this answer helpful.	Please contact our toll-free hotline ...
Please let me answer your question as follows.	Please apologize the mistake in your confirmation mail.
In addition, I would like to answer your question as follows.	We apologize for the inconvenience.
Please contact our consultation hotline ...	Please apologize our mistake.
(15 minutes installation support)	Please apologize the delay.
Dear Mr. X,	Please apologize the incomplete delivery.
Dear Mr. Y,	Please follow the link "Create new account".
Dear Sirs,	I have shipped replacement free of charge.

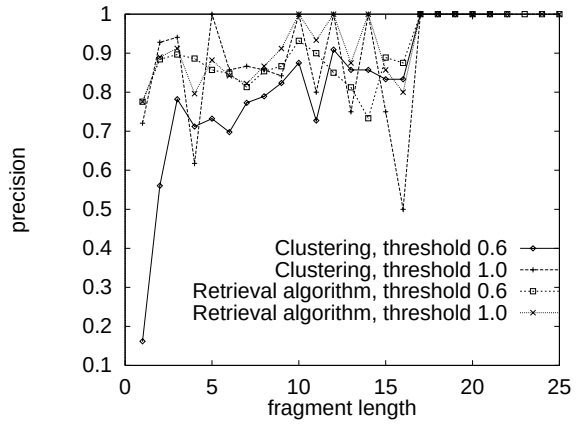

Figure 4: Precision for collection B

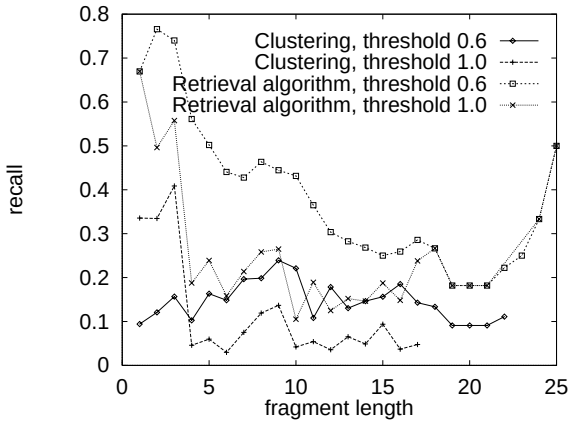

Figure 5: Recall for collection B

similarity. It is also smaller for the clustering algorithm, as the stored data is reduced. In collection A, clustering dramatically reduces the recall. For collection B, recall is only reduced by 10%. This is, again, caused by the greater diversity of collection A. On average, recall is lower for longer initial fragments. Although longer initial fragments result in fewer false completions (higher precision), the diversity among the sentences increases and the chance of a sufficiently similar sentence being available becomes smaller.

The time that is needed to make a prediction is also a very important factor. If the user is typing faster than the system is retrieving proposals, then it is useless. Figure 6 compares prediction times for a sequential search to the index-based search, and the index-based search with clustering extension. Figure 7 shows the same information measured for the second data collection.

We can clearly see that our index-based retrieval algorithm is substantially faster than the sequential algorithm. With increasing number of sentences in the model, the gap grows further; with as few as 7000 sentences, index-based search is already 5 times faster. As the sentences in collection A are more diverse, data compression by clustering additionally reduces prediction times notably. For collection B, the overhead produced when manipulating the clusters is larger than the gain, so the index algorithm behaves a little bit better. However, the difference is small.

7. CONCLUSION AND FUTURE WORK

We discussed a retrieval model in which the user expresses his or her information need by beginning to enter a sentence; the retrieval task is to complete the sentence in a way that the user accepts. A possible implementation could display the proposed completion in a bubble at the cursor position and insert the proposed completion when the user confirms it, *e.g.,* by pressing the "tab" key. Applications of the proposed retrieval model include the problem of writing letters or other documents in an administrative environment, or answering emails in a service center. Rather than a concluding solution, this paper provides a starting point for discussion of the sentence completion problem.

In our model, the query consists of the initial part of the sentence, rather than, for instance, the most characteristic part of a sentence or a special phrase that uniquely identifies a sentence. The benefit of this approach is that the user does not have to adapt his or her own behavior towards the system; in addition, the result of a failure to predict the correct sentence completion is only that the user has to complete entering the current sentence manually.

We have conducted case studies with collections of emails from two service centers, comparing a purely retrieval-based to a clustering method. For both applications, we can conclude that a substantial fraction of sentences can be completed successfully – after three initial words, we obtain up to 90% precision and 60% recall. This indicates that this problem setting may have a considerable economic potential. Clearly, such high values can only be achieved in envi-

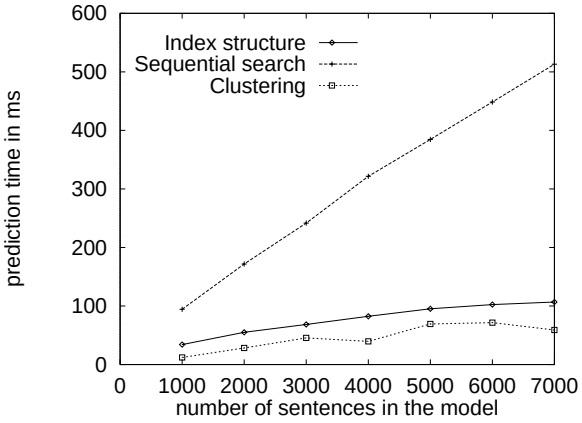

Figure 6: Prediction times for collection A

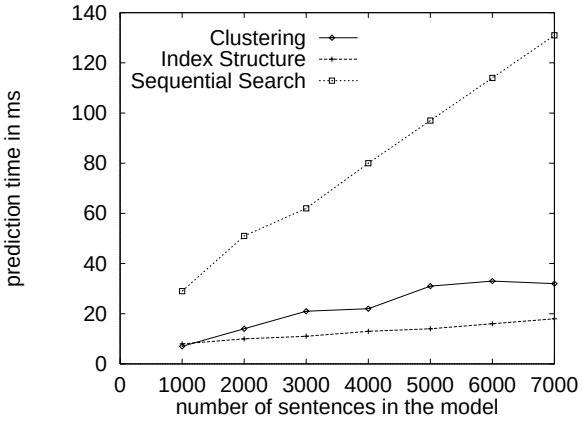

Figure 7: Prediction times for collection B

ronments with a relatively limited discourse area.

Comparing the retrieval to the clustering approach we can conclude that the retrieval method, on average, has higher precision and recall. The indexing method allows data access in typically sub-linear time. The clustering approach, however, reduces the data effectively. Further studies should investigate whether clustering becomes more interesting for very large document collections.

Our current solution ignores inter-sentential relationships as well as inter-document relationships (*e.g.,* between a question and the sentences in the corresponding answer). It appears intuitively clear that more accurate predictions can be obtained by an algorithm which takes these important attributes into account. Also, we will investigate on methods that may predict some succeeding words, but not necessarily the complete remainder of the current sentence. For instance, it may be possible to predict that the string "Your order" proceeds as "will be shipped on", but it may not be possible to predict whether the final word is "Monday." or "Tuesday."

Our evaluation has focused on the probability of predicting a sentence that lies within the same equivalence class as the sentence which a user is entering. An ultimate evaluation metric has to refer to the actual user benefit, and has to measure potential time savings as well as potential annoyance caused by incorrect completion predictions. We will address these issues in our future work.

Acknowledgment

This work was supported by the German Science Foundation DFG under grants SCHE 540/10-1 and NU 131/1-1.

8. REFERENCES

[1] Marcio Araujo, Gonzalo Navarro, and Nivio Ziviani. Large text searching allowing errors. In *Proceedings of the South American Workshop on String Processing*, 1997.

[2] R. Baeza-Yates and G. Navarro. Faster approximate string matching. *Algorithmica*, 23(2):127–158, 1999.

[3] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.

[4] J. Darragh and I. Witten. *The Reactive Keyboard*. Cambridge University Press, 1992.

[5] B. Davison and H. Hirsch. Predicting sequences of user actions. In *Proceedings of the AAAI/ICML Workshop on Predicting the Future: AI Approaches to Time Series Analysis*, 1998.

[6] M. Debevc, B. Meyer, and R. Svecko. An adaptive short list for documents on the world wide web. In *Proceedings of the International Conference on Intelligent User Interfaces*, 1997.

[7] T. Fawcett and F. Provost. Activity monitoring: Noticing interesting changes in behavior. In *Proceedings on the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1999.

[8] N. Jacobs and H. Blockeel. The learning shell: automatec macro induction. In *Proceedings of the International Conference on User Modelling*, 2001.

[9] N. Jacobs and H. Blockeel. Sequence prediction with mixed order Markov chains. In *Proceedings of the Belgian/Dutch Conference on Artificial Intelligence*, 2003.

[10] N. Jacobs and H. Blockeel. User modelling with sequential data. In *Proceedings of the HCI International*, 2003.

[11] T. Kahveci and A. Singh. Efficient index structures for string databases. In *Proceedings of the International Conference on Very Large Databases*, pages 351–360, 2001.

[12] B. Korvemaker and R. Greiner. Predicting Unix command lines: adjusting to user patterns. In *Proceedings of the National Conference on Artificial Intelligence*, 2000.

[13] G. Landau and U. Vishkin. Fast string matching with k differences. *Journal of Conputer Systems Science*, 37:63–78, 1988.

[14] T. Lane. Hidden markov models for human/computer interface modeling. In *Proceedings of the IJCAI-99 Workshop on Learning About Users*, 1999.

[15] A. Moffat and T. Bell. In situ generation of compressed inverted files. *Journal of the American Society for Information Science*, 46(7):537–550, 1995.

[16] H. Motoda and K. Yoshida. Machine learning techniques to make computers easier to use. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.

[17] O. Ozturk and H. Ferhatosmanoglu. Effective indexing and filtering for similarity search in large biosequence databases. In *Proceedings of the IEEE International Symposium on Bioinformatics and Bioengineering*, 2003.

[18] P. Sellers. The theory and computation of evolutionary distances: pattern recognition. *Journal of Algorithms*, 1:359–373, 1980.

[19] S. Wu and U. Manber. Fast text searching allowing errors. *Communications of the ACM*, 35(10):83–91, 1992.