Enhanced Suffix Arrays as Language Models: Virtual *k*-testable Languages

Herman Stehouwer and Menno van Zaanen

TiCC, Tilburg University, Tilburg, The Netherlands
{J.H.Stehouwer, M.M.vanZaanen}@uvt.nl

Abstract. In this article, we propose the use of suffix arrays to efficiently implement n-gram language models with practically unlimited size n. This approach, which is used with synchronous back-off, allows us to distinguish between alternative sequences using large contexts. We also show that we can build this kind of models with additional information for each symbol, such as part-of-speech tags and dependency information.

The approach can also be viewed as a collection of virtual k-testable automata. Once built, we can directly access the results of any k-testable automaton generated from the input training data. Synchronous back-off automatically identifies the k-testable automaton with the largest feasible k. We have used this approach in several classification tasks.

1 Introduction

When writing texts, people often use spelling checkers to reduce the number of mistakes in their texts. Many spelling checkers concentrate on non-word errors. However, there are also types of errors in which words are correct, but used incorrectly in context. These errors are called *contextual errors* and are much harder to recognize than non-word errors.

In this paper, we describe a novel approach, which is based on suffix arrays, which are sorted arrays containing all suffixes of a collection of sequences, to store the models. This approach can be used to make decisions about alternative corrections of contextual errors. The use of suffix arrays allows us to use large, potentially enriched n-grams and as such can be seen as an extension to more conventional n-gram models. The underlying assumption of the language model is that using more (precise) information pertaining to the decision is better [3].

The approach can also be seen as a collection of k-testable automata that we can access using by using a single query. As De Higuera states in [4] choosing the right size k is a crucial issue. When k is too small over-generalization will occur, conversely too large k leads to models that might not generalize enough. The approach described here automatically chooses the largest k applicable to the situation.

2 Approach

To select the best sequence out of a set of alternative sequences, such as in the problem of contextual errors in text, we consider all possible alternatives and use a model to select the most likely sequence. The sequence with the highest probability is selected as the correct form.

The language model we use here is based on unbounded size n-grams. The probability of a sequence is computed by multiplying the probabilities of the n-gram for each position in the sequence.

$$P_{seq} = \prod_{w \in seq} P_{LM}(w|w_{-1} \dots w_{-n})$$

Considering that the probabilities are extracted from the training data, when using n-grams with very large n, data sparseness is an issue. Long sequences may simply not occur in the data, even though the sequence is correct, leading to a probability of zero, even though the correct probability should be non-zero (albeit small).

To reduce the impact of data sparseness, we can use techniques such as smoothing [2], which redistributes probability mass to estimate the probability of previously unseen word sequences[1] or back-off, where probabilities of lower order n-grams are used to approximate the probability of the larger n-gram.

In this article, we use the synchronous back-off method [6] to deal with data sparseness. This method analyzes n-grams of the same size for each of the alternative sequence in parallel. If all n-grams have zero probability, the method backs off to $n-1$-grams. This continues until at least one n-gram for an alternative has a non-zero probability. This implements the idea that, assuming the training data is sufficient, if a probability is zero the n-gram combination is not in the language. Effectively, this method selects the largest, usable n-grams automatically.

Probabilities of all n-grams (from the training data) of all sizes are stored in an enhanced suffix array. A suffix array is a flat data-structure containing an implicit suffix tree structure [1]. A suffix tree is a trie-based data structure [5, pp. 492] that stores all suffixes of a sequence in such a way that a suffix (and similarly an infix) can be found in linear time in the length of the suffix. All suffixes occupy a single path from the root of the suffix tree to a leaf. Construction of the data structure only needs to be performed once.

Due to the way suffix arrays are constructed, we can efficiently find the number of occurrences of subsequences (used as n-grams) of the training data. Starting from the entire suffix array we can quickly identify the interval(s) that pertain to the particular n-gram query. The interval specifies exactly the number of occurrences of the subsequence in the training data. Effectively, this means that we can find the largest non-zero n-gram efficiently.

[1] In this paper we do not employ smoothing or interpolation methods as they modify the probabilities of all alternatives equally and hence will not affect the ordering of alternative sequences.

3 Suffix arrays as Collections of *k*-testable Machines

An enhanced suffix array extends a regular suffix array with a data-structure allowing for the implicit access of the longest-common-prefix (lcp) intervals [1]. An lcp interval represents a virtual node in the implicit suffix trie. A simple enhanced suffix-array with its corresponding implicit suffix-trie is shown in Figure 1 as an example.

We can view a suffix array as a virtual DFA in which each state is described by a set of lcp-intervals over the suffix array. This view allows us to determine (by the size of the interval) the number of valid sequences that terminated in each state. If there is no valid path in the DFA for the queried sequence it results in an empty state and the sequence is rejected by the learned grammar.

Since the suffix array stores the n-grams of all sizes n, this comes down to a collection of k-testable machines with $k = 1 \ldots |T|$ (with T the training data). Querying with length k automatically results in using a k-testable machine.

There is an interesting property of the n-gram suffix array approach, which separates it from collections of regular k-testable machine DFAs. All the states on the suffix array are accepting states. Rejection of a sequence only happens when the query cannot be found in the training data at all. The system also does not support negative training examples, only positive ones.

To enhance the system, we have generalized a state to be described by a set of lcp intervals. This allows for the supports of single position wildcards. In practice, wildcards allow for the integration of additional information. By interleaving the symbol sequences with the additional symbols, we can incorporate for instance, long range information, such as dependency information and local, less specific features such as part-of-speech tags. Using wildcards, we can construct queries that either use such additional information on one or more positions or not.

i	suffix	lcp	S[suffix]
0	2	0	aaacatat$
1	3	2	aacatat$
2	0	1	acaaacatat$
3	4	3	acatat$
4	6	1	atat$
5	8	2	at$
6	1	0	caaacatat$
7	5	2	catat$
8	7	0	tat$
9	9	1	t$
10	10	0	$

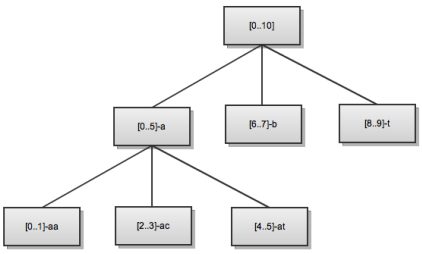

Fig. 1. An enhanced suffix array on the string S= acaaacatat on the left, and its corresponding lcp-interval tree on the right. From [1].

To evaluate the approach, we ran experiments on three contextual error problems from the natural language domain, namely confusible disambiguation, verb and noun agreement and adjective ordering. The synchronous back-off method automatically selects the k-testable machine that has the right amount of specificity for selecting between the alternative sequences. These experiments where run with a simple words-only approach and also with part-of-speech tags. The experiments show that the approach is feasible and efficient.

When trained on the first 675 thousand sequences of the British National Corpus building the enhanced suffix array takes 2.3 minutes on average. These sequences contain about 27 million tokens. When loaded into memory the enhanced suffix array uses roughly 500 megabytes. We ran speed-tests using 10.000 randomly selected sequences of length 10. The system has an average runtime off 10.2 minutes over tens of runs, with as extremes 8.1 and 12.1 minutes. This means that we can expect the enhanced suffix array to process around 1200 queries per minute. All tests where run on a 2GHz opteron system with 32GB of main memory. The suffix array process is single-threaded.

4 Conclusion and Future Work

We have proposed a novel approach which implements a collection of k-testable automata using an enhanced suffix-array. This approach describes automata that have no explicit reject states and do not require (or support) negative examples during training. Nevertheless, this approach allows for an efficient implementation of many concurrent k-testable machines of various k using suffix arrays.

The implementation will be applied as a practical system in the context of text correction, allowing additional linguistic information to be added when needed. In this context, the effectiveness of the additional information in combination with the limitations of k-testable languages still needs to be evaluated.

References

1. Abouelhoda, M.I., Kurtz, S., Ohlebusch, E.: Replacing suffix trees with enhanced suffix arrays. Journal of Discrete Algorithms 2(1), 53–86 (2004)
2. Chen, S., Goodman, J.: An empirical study of smoothing techniques for language modelling. In: Proceedings of the 34th Annual Meeting of the ACL. pp. 310–318. ACL (June 1996)
3. Daelemans, W., Van den Bosch, A., Zavrel, J.: Forgetting exceptions is harmful in language learning. Machine Learning, Special issue on Natural Language Learning 34, 11–41 (1999)
4. de la Higuera, C.: Grammatial Inference, Learning Automata and Grammars. Cambridge University Press (2010)
5. Knuth, D.E.: The art of computer programming, vol. 3: Sorting and searching. Addison-Wesley, Reading, MA (1973)
6. Stehouwer, H., Van den Bosch, A.: Putting the t where it belongs: Solving a confusion problem in Dutch. In: Verberne, S., van Halteren, H., Coppen, P.A. (eds.) Computational Linguistics in the Netherlands 2007: Selected Papers from the 18th CLIN Meeting. pp. 21–36. Nijmegen, The Netherlands (2009)