Evolving new lexical association measures using genetic programming

Jan Šnajder   **Bojana Dalbelo Bašić**   **Saša Petrović**   **Ivan Sikirić**
Faculty of Electrical Engineering and Computing, University of Zagreb
Unska 3, Zagreb, Croatia
{jan.snajder, bojana.dalbelo, sasa.petrovic, ivan.sikiric}@fer.hr

Abstract

Automatic extraction of collocations from large corpora has been the focus of many research efforts. Most approaches concentrate on improving and combining known lexical association measures. In this paper, we describe a genetic programming approach for evolving new association measures, which is not limited to any specific language, corpus, or type of collocation. Our preliminary experimental results show that the evolved measures outperform three known association measures.

1 Introduction

A *collocation* is an expression consisting of two or more words that correspond to some conventional way of saying things (Manning and Schütze, 1999). Related to the term collocation is the term n-*gram*, which is used to denote any sequence of *n* words. There are many possible applications of collocations: automatic language generation, word sense disambiguation, improving text categorization, information retrieval, etc. As different applications require different types of collocations that are often not found in dictionaries, automatic extraction of collocations from large textual corpora has been the focus of much research in the last decade; see, for example, (Pecina and Schlesinger, 2006; Evert and Krenn, 2005).

Automatic extraction of collocations is usually performed by employing *lexical association measures* (AMs) to indicate how strongly the words comprising an *n*-gram are associated. However, the use of lexical AMs for the purpose of collocation extraction has reached a plateau; recent research in this field has focused on combining the existing AMs in the hope of improving the results (Pecina and Schlesinger, 2006). In this paper, we propose an approach for deriving new AMs for collocation extraction based on genetic programming. A similar approach has been usefully applied in text mining (Atkinson-Abutridy et al., 2004) as well as in information retrieval (Gordon et al., 2006).

Genetic programming is an evolutionary computational technique designed to mimic the process of natural evolution for the purpose of solving complex optimization problems by stochastically searching through the whole space of possible solutions (Koza, 1992). The search begins from an arbitrary seed of possible solutions (the initial population), which are then improved (evolved) through many iterations by employing the operations of selection, crossover, and mutation. The process is repeated until a termination criterion is met, which is generally defined by the goodness of the best solution or the expiration of a time limit.

2 Genetic programming of AMs

2.1 AM representation

In genetic programming, possible solutions (in our case lexical AMs) are mathematical expressions represented by a tree structure (Koza, 1992). The leaves of the tree can be constants, or statistical or linguistic information about an *n*-gram. A constant can be any real number in an arbitrarily chosen interval; our experiments have shown that variation of this interval does not affect the performance. One special constant that we use is the number of words in the corpus. The statistical information about an *n*-gram can be the frequency of any part of the *n*-gram. For example, for a trigram *abc* the statistical information can be the frequency $f(abc)$ of the whole trigram, frequencies $f(ab)$ and $f(bc)$ of the digrams, and the frequencies of individual words $f(a)$, $f(b)$, and $f(c)$. The linguistic information about an *n*-gram is the part-of-speech (POS) of any one of its words.

Inner nodes in the tree are operators. The binary operators are addition, subtraction, multiplication, and division. We also use one unary operator, the natural logarithm, and one ternary operator, the IF-THEN-ELSE operator. The IF-THEN-ELSE node has three descendant nodes: the left descendant is the condition in the form "i-th word of the *n*-gram has a POS tag T," and the other two descendants are operators or constants. If the condition is true, then the subexpression corresponding to the middle descendant is evaluated, otherwise the subexpression corresponding to the right descendant is evaluated.

The postfix expression of an AM can be obtained by traversing its tree representation in postorder. Figure 1 shows the representation of the Dice coefficient using our representation.

2.2 Genetic operators

The crossover operator combines two parent solutions into a new solution. We defined the crossover operator as follows: from each of the two parents, one node is chosen randomly, excluding any nodes that represent the condition of the IF-THEN-ELSE operator. A new solution is obtained by replacing the subtree of the chosen node of the first parent with the subtree of the chosen node of the second parent. This method of defining the crossover operator is the same as the one described in (Gordon et al., 2006).

The mutation operator introduces new "genetic material" into a population by randomly changing a solution. In our case, the mutation operator can do one of two things: either remove a randomly selected inner node (with probability of 25%), or insert an inner node at a random position in the tree (with probability of 75%). If a node is being removed from the tree, one of its descendants (randomly chosen) takes its place. An exception to this rule is the IF-THEN-ELSE operator, which cannot be replaced by its condition node. If a node is being inserted, a randomly created operator node replaces an existing node that then becomes a descendant of the new node. If the inserted node is not a unary operator, the required number of random leaves is created.

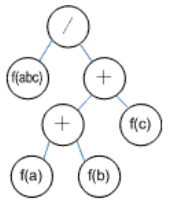
The selection operator is used to copy the best solutions into the next iteration. The goodness of the solution is determined by the *fitness function*, which assigns to each solution a number indicating how good that particular solution actually is. We measure the goodness of an AM in terms of its F_1 score, obtained from the precision and recall computed on a random sample consisting of 100 positive *n*-grams (those considered collocations) and 100 negative *n*-grams (non-collocations). These *n*-grams are ranked according to the AM value assigned to them, after which we compute the precision and recall by considering first n best-ranked *n*-grams as positives and the rest as negatives, repeating this for each n between 1 and 200. The best F_1 score is then taken as the AM's goodness.

Using the previous definition of the fitness function, preliminary experiments showed that solutions soon become very complex in terms of number of nodes in the tree (namely, on the order of tens of thousands). This is a problem both in terms of space and time efficiency; allowing unlimited growth of the tree quickly consumes all computational resources. Also, it is questionable whether the performance benefits from the increased size of the solution. Thus, we modified the fitness function to also take into account the size of the tree (that is, the less nodes a tree has, the better). Favoring shorter solutions at the expense of some loss in performance is known as *parsimony*, and it has already been successfully used in genetic programming (Koza, 1992). Therefore, the final formula for the fitness function we used incorporates the parsimony factor and is given by

$$\mathit{fitness}(j) = F_1(j) + \eta \frac{L_{max} - L(j)}{L_{max}}, \qquad (1)$$

where $F_1(j)$ is the F_1 score (ranging from 0 to 1) of the solution j, η is the parsimony factor, L_{max} is the maximal size (measured in number of nodes), and $L(j)$ is the size of solution j. By varying η we can control how much loss of performance we will tolerate in order to get smaller, more elegant solutions.

Genetic programming algorithms usually iterate until a *termination criterion* is met. In our case, the algorithm terminates when a certain number, k, of iterations has passed without an improvement in the



$$Dice(a,b,c) = \frac{f(abc)}{f(a)+f(b)+f(c)}$$

Figure 1: Dice coefficient for digrams represented by tree

results. To prevent the overfitting problem, we measure this improvement on another sample (*validation sample*) that also consists of 100 collocations and 100 non-collocations.

3 Preliminary results

3.1 Experimental setting

We use the previously described genetic programming approach to evolve AMs for extracting collocations consisting of three words from a corpus of 7008 Croatian legislative documents. Prior to this, words from the corpus were lemmatized and POS tagged. Conjunctions, propositions, pronouns, interjections, and particles were treated as stop-words and tagged with a POS tag X. *N*-grams starting or ending with a stopword, or containing a verb, were filtered out. For evaluation purposes we had a human expert annotate 200 collocations and 200 non-collocations, divided into the evaluation and validation sample. We considered an *n*-gram to be a collocation if it is a compound noun, terminological expression, or a proper name. Note that we could have adopted any other definition of a collocation, since this definition is implicit in the samples provided.

In our experiments, we varied a number of genetic programming parameters. The size of the initial population varied between 50 and 50 thousand randomly generated solutions. To examine the effects of including some known AMs on the performance, the following AMs had a 50% chance of being included in the initial population: pointwise mutual information (Church and Hanks, 1990), the Dice coefficient, and the heuristic measure defined in (Petrović et al., 2006):

$$H(a,b,c) = \begin{cases} 2\log\frac{f(abc)}{f(a)f(c)} & \text{if } POS(b) = X, \\ \log\frac{f(abc)}{f(a)f(b)f(c)} & \text{otherwise.} \end{cases}$$

For the selection operator we used the well-known *three-tournament selection*. The probability of mutation was chosen from the interval $[0.0001, 0.3]$, and the parsimony factor η from the interval $[0, 0.05]$, thereby allowing a maximum of 5% loss of F_1 in favor of smaller solutions. The maximal size of the tree in nodes was chosen from the interval $[20, 1000]$. After the F_1 score for the validation sample began dropping, the algorithm would continue for another k iterations before stopping. The parameter k was chosen from the interval $[10^4, 10^7]$. The experiments were run with 800 different random combinations of the aforementioned parameters.

3.2 Results

Around 20% of the evolved measures (that is, the solutions that remained after the algorithm terminated) achieved F_1 scores of over 80% on both the evaluation and validation samples. This proportion was 13% in the case when the initial population did not include any known AMs, and 23% in the case when it did, thus indicating that including known AMs in the initial population is beneficial. The overall best solution had 205 nodes and achieved an F_1 score of 88.4%. In search of more elegant AMs, we singled out solutions that had less than 30 nodes. Among these, a solution that consisted of 13 nodes achieved the highest F_1. This measure is given by

$$M_{13}(a,b,c) = \begin{cases} -0.423\frac{f(a)f(c)}{f^2(abc)} & \text{if } POS(b) = X, \\ 1 - \frac{f(b)}{f(abc)} & \text{otherwise.} \end{cases}$$

The association measure M_{13} is particularly interesting because it takes into account whether the middle word in a trigram is a stopword (denoted by the POS tag X). This supports the claim laid out in (Petrović et al., 2006) that the trigrams containing stopwords (e.g., *cure for cancer*) should be treated differently, in that the frequency of the stopword should be ignored. It is important to note that the aforementioned measure H was not included in the initial population from which M_{13} evolved. It is also worthwhile noting that in such populations, out of 100 best evolved measures, all but four of them featured a condition identical to that of M_{13} ($POS(b) = X$). In other words, the majority of the measures evolved this condition completely independently, without H being included in the initial population.

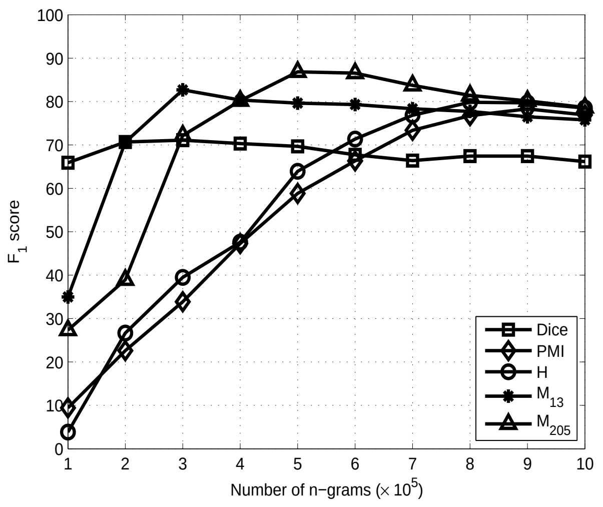


Figure 2: Comparison of association measures on a corpus of 7008 Croatian documents

Figure 2 shows the comparison of AMs in terms of their F_1 score obtained on the corpus of 7008 documents. The x axis shows the number of n best ranked *n*-grams that are considered positives (we show only the range of n in which all the AMs achieve their maximum F_1; all measures tend to perform similarly with increasing n). The maximum F_1 score is achieved if we take 5×10^5 *n*-grams ranked best by the M_{205} measure. From Fig. 2 we can see that the evolved AMs M_{13} and M_{205} outperformed the other three considered AMs. For example, collocations *kosilica za travu* (*lawn mower*) and *digitalna obrada podataka* (*digital data processing*) were ranked at the 22th and 34th percentile according to Dice, whereas they were ranked at the 97th and 87th percentile according to M_{13}.

4 Conclusion

In this paper we described a genetic programming approach for evolving new lexical association measures in order to extract collocations.

The evolved association measure will perform at least as good as any other AM included in the initial population. However, the evolved association measure may be a complex expression that defies interpretation, in which case it may be treated as a black-box suitable for the specific task of collocation extraction. Our approach only requires an evaluation sample, thus it is not limited to any specific type of collocation, language or corpus.

The preliminary experiments, conducted on a corpus of Croatian documents, showed that the best evolved measures outperformed other considered association measures. Also, most of the best evolved association measures took into account the linguistic information about an *n*-gram (the POS of the individual words).

As part of future work, we intend to apply our approach to corpora in other languages and compare the results with existing collocation extraction systems. We also intend to apply our approach to collocations consisting of more than three words, and to experiment with additional linguistic features.

Acknowledgments

This work has been supported by the Government of Republic of Croatia, and Government of Flanders under the grant No. 036-1300646-1986 and KRO/009/06.

References

John Atkinson-Abutridy, Chris Mellish, and Stuart Aitken. 2004. Combining information extraction with genetic algorithms for text mining. *IEEE Intelligent Systems*, 19(3):22–30.

Kenneth W. Church and Patrick Hanks. 1990. Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1):22–29.

Stephan Evert and Brigitte Krenn. 2005. Using small random samples for the manual evaluation of statistical evaluation measures. *Computer Speech and Language*, 19(4):450–466.

Michael Gordon, Weiguo Fan, and Praveen Pathak. 2006. Adaptive web search: Evolving a program that finds information. *IEEE Intelligent Systems*, 21(5):72–77.

John R. Koza. 1992. *Genetic programming: On the programming of computers by means of natural selection*. MIT Press.

Christopher Manning and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA.

Pavel Pecina and Pavel Schlesinger. 2006. Combining association measures for collocation extraction. In *Proc. of the COLING/ACL 2006*, pages 651–658.

Saša Petrović, Jan Šnajder, Bojana Dalbelo Bašić, and Mladen Kolar. 2006. Comparison of collocation extraction measures for document indexing. *J. of Computing and Information Technology*, 14(4):321–327.