An Efficient Python Module for Lexical Distributional Similarity

Brian Tomasik
Department of Computer Science
Swarthmore College
Swarthmore, PA 19081
`btomasi1@swarthmore.edu`

Dougal Sutherland
Department of Computer Science
Swarthmore College
Swarthmore, PA 19081
`dsuther1@swarthmore.edu`

Abstract

We implement the algorithm of (Rychly and Kilgarriff, 2007) for computing distributional similarity among words based on shared context relations in a manner designed to scale to billion-word corpora.

1 Introduction

Several NLP tasks require knowledge of the *distributional similarity* of words in a particular large corpus. Two words are distributionally similar if one could be substituted for the other in a sentence with a high probability of preserving the plausibility of seeing the sentence in real text (Weeds and Weir, 2005, p. 442). For instance, we expect "apple" and "banana" to be distributionally similar because they can both substitute for x in phrases like "eat the x," "slice the x," "fresh x," "rotten x," and "x-flavored."

A list of distributionally similar words can be used for a number of tasks. One example is *cooccurrence smoothing* in n-gram language modeling, in which unseen n-grams can be given higher probability if they contain words that are similar to those of seen n-grams (Essen and Steinbiss, 1992; Dagan et al., 1993; Dagan et al., 1999). Even more common is the use of distributional similarity to approximate semantic similarity, based on the so-called *distributional hypothesis* (Harris, 1954), the assumption that, to quote John Firth, "You shall know a word by the company it keeps" (Firth, 1957). (Curran, 2004, sec. 1.4) surveys a number of applications of semantic similarity using cooccurrence information, including parse selection, collocation identification, sentiment classification, and query expansion in information retrieval and question answering.

The most straightforward use of similarity information is for *thesaurus construction*. While manually constructed tools like Roget's thesaurus and WordNet can also be used for identifying synonyms in standard English, such resources are not available for all languages, nor can they be easily adapted to the technical terminology of a particular domain (Jurafsky and Martin, 2008, p. 658).

(Curran and Moens, 2002) find that the quality of automatically extracted thesauri continues to improve as input-text size increases into the hundreds of millions of words. As a result, (Gorman and Curran, 2006) investigate a number of approximation algorithms for scaling to large corpora. (Rychly and Kilgarriff, 2007) subsequently proposed a much faster algorithm requiring essentially no approximations, and this paper describes our implementation of this latter approach.

Section 2 of this paper describes the general algorithmic challenge of problem, section 3 describes the specifics of our implementation, section 4 illustrates our output, and section 5 describes ways in which one could augment our package and more systematically evaluate the results.

2 Algorithm Complexity

As is standard in the distributional-similarity literature, (Gorman and Curran, 2006) base their algorithm on *context relations* of the form (w, r, w'). For instance, from "bake the cake," they would extract (`cake`, `dir-obj`, `bake`). The pair (r, w') is called an *attribute* by (Gorman and Curran, 2006) and a *feature* by other authors (Lin, 1998a; Weeds and Weir, 2005). For each attribute, we can list the number of times a given word has it (see Table 1).

Table 1: Example context vectors for two words.

Attribute	`cake`	`cookies`
(`dir-obj`, `bake`)	5	3
(`dir-obj`, `eat`)	4	9
(`subject`, `swam`)	0	0
(`modified-by`, `yummy`)	2	1
...	...	...

A number of measures have been proposed for computing the similarity of two words as a function of their attribute counts (Curran, 2004; Weeds and Weir, 2005), but they are in general $\mathcal{O}(m)$, where m is the number of attributes. Given n total words, a naïve nearest-neighbor algorithm that compares each word to each other word is $\mathcal{O}(mn^2)$.

(Gorman and Curran, 2006) evaluate several approximation schemes, including dimensionality reduction and randomization techniques to reduce the context-vector size m and data structures to approximate the $\mathcal{O}(n^2)$ nearest-neighbor search. They find that Spatial Approximation Sample Hierarchy (SASH) (Houle and Sakuma, 2005) performed almost as well at matching synonyms from a gold standard as doing the full $\mathcal{O}(n^2)$ computation but took an order of magnitude less time.

However, Rychlý and Kilgarriff (Rychly and Kilgarriff, 2007) point out that most of the entries of Table 1 are zeros, since most words don't appear in most contexts. So rather than fixing two columns and looking at the (small number of) rows they share, one can fix a row and look at all of the columns with nonzero entries. This number is generally small (usually less than 1000), and in the rare case when it's larger than $p = 10,000$, the authors skip it, since extremely common contexts are unlikely to be informative anyway. The worst-case run-time is thus $\mathcal{O}(mp^2)$ with $p \ll n$.

For each pair of words with nonzero counts in a given row, Rychlý and Kilgarriff increment partial sums of the similarity between the words. Storing a full word-by-word similarity matrix in RAM becomes impossible for large corpora, so the authors suggest two alternatives:

1. *External-memory sorting*: Write the partial-sum increments out to a file to which an external-memory mergesort can later be applied.

2. *In-memory partitioning*: Build the similarity matrix in RAM, but store only the rows corresponding to words in a selected range (e.g., words 1 through 99, 100 through 199, etc.). Write out each partial matrix to a file and then start over with the next range of words.

Applying their algorithm to large corpora of up to 1.9 billion words, Rychlý and Kilgarriff keep computation time below 2 hours.

3 Implementation Details

We implement the algorithm of Rychlý and Kilgarriff in Python as a module for the Natural Language Toolkit (NLTK) (Bird and Loper, 2004).

3.1 Relation Extraction

Our module assumes the existence of input context relations (w, r, w') that will be supplied by the user. NLTK currently lacks a module for extracting such relations from a corpus, so we use Lin's Minipar (Lin, 1998b), which is only available in the form of Linux and Windows binaries.

Minipar relations come with associated parts of speech (POS)—e.g., "buy V:obj:N car"—so our program gives the user the option, enabled by default, to attach the POS to the word—e.g., "buy(V)" and "car(N)"—so that thesaurus results will be POS-specific, as are standard thesauri like Roget's. To ensure that our input relations are clean and semantically meaningful, we restrict consideration to those in which both w and w' are nouns, verbs, or adjectives; however, this is a configurable parameter in our program.

3.2 Similarity Functions

(Curran, 2004, chapter 4) presents a number of similarity functions, each consisting of a component *weight* function and *measure* function, explained further below.

For a given context c, say (`dir-obj`, `bake`), and word w, say, `cake`, the weight function $\text{wgt}(w, c)$

indicates how useful or interesting it is that w has context c. For instance, one might use a TF-IDF score, which gives more credit to more common (w, c) relations but discounts contexts that are very common in general. After evaluation, (Curran, 2004, p. 85) finds that a *t-test* weight performs best against gold-standard thesauri:

$$\text{wgt}_{\text{ttest}}(w, c) = \frac{p(c, w) - p(c)p(w)}{\sqrt{p(c)p(w)}}, \qquad (1)$$

where $p(\cdot)$ is the maximum-likelihood probability estimate based on frequency counts. We use this as our default weight function,[1] but our program could easily extend to other weight functions computable from the values $p(c)$, $p(w)$, $p(c, w)$, and the total number of relations N, including all of those shown in Table 1 of (Weeds and Weir, 2005, p. 446).

A measure function is a way of combining weights for two words w_i and w_j over all of their contexts. Words that share large weights in many contexts are taken to be more distributionally similar. (Curran, 2004, p. 84) found best performance from two measures:

$$\text{meas}_{\text{dice-mod}}(w_i, w_j) = \frac{\sum_{c \in C_{ij}} 2 \min\left(\text{wgt}(w_i, c), \text{wgt}(w_j, c)\right)}{\sum_{c \in C_{ij}} \text{wgt}(w_i, c) + \text{wgt}(w_j, c)} \qquad (2)$$

and

$$\text{meas}_{\text{jaccard}}(w_i, w_j) = \frac{\sum_{c \in C_{ij}} \min\left(\text{wgt}(w_i, c), \text{wgt}(w_j, c)\right)}{\sum_{c \in C_{ij}} \max\left(\text{wgt}(w_i, c), \text{wgt}(w_j, c)\right)}, \qquad (3)$$

where

$$C_{ij} := \{c \;:\; (w_i, c) \text{ and } (w_j, c) \text{ both exist}\}. \qquad (4)$$

[1]According to (Jurafsky and Martin, 2008, p. 662) and references cited therein, it is standard to bound weights from below by 0, since negative weights tend to be unreliable unless the input corpora are sufficiently large. (Curran, 2004, p. 85-86) echoes this finding. As a result, our program defaults to a weight function that is actually the max of the quantity in (1) and 0.

Both of these involve two sums over contexts (one in the numerator and one in the denominator), so in the algorithm of Rychlý and Kilgarriff, we maintain two separate partial sums and divide them at the end. Some of the other measures considered in (Curran, 2004, sec. 4.2), such as cosine and Lin, only sum over C_{ij} in the numerator and have marginal sums over contexts for each word separately in the denominator. Our program doesn't currently allow for such measures, but making changes to do so would be straightforward.

3.3 Memory

Our program is intended to support both the external-memory sorting and in-memory partitioning methods. However, our implementation of the external-memory multiway mergesort currently contains a relatively major stumbling block, which prevents our algorithm from fulfilling its original goal—namely, it does not deal gracefully with operating system limits on the number of open files. This can mean that when processing a large dataset, it will simply crash after some time in analyzing the data. Some more careful attention to how many files are being created and more recursive merges would alleviate this problem.

It would also be desirable to implement some form of heuristic which could distinguish between the cases in which a partitioned in-memory sort would be appropriate, as those tend to be significantly faster, and when an external-memory sort is required.

4 Evaluation and Results

4.1 Measures

In (Curran, 2004, sec. 2.2), distributional similarity is evaluated with a "gold standard" list of 300 common nouns along with the union of their synonyms from four major thesauri. For each of the nouns, Curran returns up to 200 candidate synonyms and computes two measures: a direct-match score (the percent of returned synonyms present in the gold standard) and an inverse-rank score (e.g., if words 1, 4, and 199 appear in the gold standard, the score is $\frac{1}{1} + \frac{1}{4} + \frac{1}{199} = 1.255$).

While Curran lists the words that he evaluates, he isn't able to reproduce the entire set of gold-standard entries for those words. Since manually reconstructing those entries would be very difficult, we chose a different gold-standard: The synset lists in WordNet. We take the true synonyms of a word with a given part of speech to be the union of all other words in any of its synsets. By using WordNet, we can generate a gold-standard synonym list for every word in our thesaurus that also appears in WordNet, not just for 70 high-frequency nouns.

4.2 Corpus

Our input consists of the 1,200,000 context relations extracted from the "CE" subset of the British National Corpus (BNC). We have also extracted 80,500,000 relations from the entire A-K subset of the BNC, but we haven't had time to run our algorithm on the latter.

4.3 Results

From the thesaurus generated from the "CE" relations, 4,880 of the words appear in WordNet. The average direct-match score is only 0.07%, and the average inverse-rank score is 0.003. These compare poorly against, say, the results in (Gorman and Curran, 2006), which showed typical direct-match scores of 2-5% and inverse-rank scores between 0.76 and 1.71. However, our results are not directly comparable, because we evaluate all of the words we can, not just 300 frequent nouns, and we use a more limited thesaurus, as WordNet has fewer entries than, say, Roget's (Weeds and Weir, 2005, p. 462).

Following is a sample entry from our thesaurus for the adjective "loving." All of these words had similarity scores of 1.0:

> abstruse, accessible, accomplished, ambitious, astonishing, basic, beautiful, certainly, cogent, consistently, cynical, definitely, disgusting, eloquent, exclusive, extraordinary, famous, ferocious, fervent, gratifying, gruelling, gutsy, hated, historic, humiliating, incriminating, intractable, levelheaded, long-drawn-out, memorable, modern, notably, passionate, photographed, popular, potent, powerful, protected, rational, readily, recent, romantic, sacred, scrumptious, serious, shocking, successful, sustained, unchildlike, unimpressed, unusual, urgent, valued, vehement, viable.

Many of them bear little similarity to "loving," and the fact that they had perfect scores shows that our training corpus is currently too small.

5 Future Work

(Curran, 2004) and (Weeds and Weir, 2005) present a variety of weight and measure functions beyond those we have implemented. While ours are the ones that (Curran, 2004), our module would ideally provide the other options as well.

In addition, we have only tested our program on a very limited data set, far smaller than those to which it is designed to scale. We would like to try it on a larger portion of the BNC, as well as perhaps the 1B-word Oxford English Corpus and the 1.9B-word Itwac on which (Rychly and Kilgarriff, 2007) ran their algorithm.

Finally, it would be interesting to examine other similarity measures. If we had access to Curran's gold-standard list, we could directly compare against his results (something which (Rychly and Kilgarriff, 2007) did not try themselves). And there are many evaluation techniques in addition to the gold-standard approach, such as correlating distributional-similarity measures against WordNet's hierarchical measures (Weeds and Weir, 2005, sec. 5.1).

Acknowledgements

We thank Rich Wicentowski for assistance extracting context relations using Minipar and Steven Bird for guidance on fitting our module into NLTK.

References

S. Bird and E. Loper. 2004. NLTK: The Natural Language Toolkit. In *Proc. Association for Computational Linguistics*, pages 214–217.

J.R. Curran and M. Moens. 2002. Scaling context space. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 7–12.

J.R. Curran. 2004. *From Distributional to Semantic Similarity*. Ph.D. thesis.

I. Dagan, S. Marcus, and S. Markovitch. 1993. Contextual word similarity and estimation from sparse data. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, pages 164–171. Association for Computational Linguistics Morristown, NJ, USA.

I. Dagan, L. Lee, and F.C.N. Pereira. 1999. Similarity-Based Models of Word Cooccurrence Probabilities. *Machine Learning*, 34(1):43–69.

U. Essen and V. Steinbiss. 1992. Cooccurrence smoothing for stochastic language modeling. In *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, volume 1.

J.R. Firth. 1957. A synopsis of linguistic theory 1930-1955. *Studies in Linguistic Analysis*, pages 1–32.

J. Gorman and J.R. Curran. 2006. Scaling distributional similarity to large corpora. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL*, pages 361–368. Association for Computational Linguistics Morristown, NJ, USA.

Z. Harris. 1954. Distributional structure. *Word*, 10(23):146–162.

M.E. Houle and J. Sakuma. 2005. Fast Approximate Similarity Search in Extremely High-Dimensional Data Sets. In *Proceedings of the International Conference on Data Engineering*, volume 21, page 619. IEEE Computer Society Press; 1998.

D. Jurafsky and J.H. Martin. 2008. *Speech And Language Processing*. Prentice Hall.

D. Lin. 1998a. An information-theoretic definition of similarity. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 296–304.

D. Lin. 1998b. Minipar.

P. Rychly and A. Kilgarriff. 2007. An efficient algorithm for building a distributional thesaurus (and other Sketch Engine developments). In *Annual Meeting-Association for Computational Linguistics*, volume 45, page 2.

J. Weeds and D. Weir. 2005. Co-occurrence Retrieval: A Flexible Framework for Lexical Distributional Similarity. *Computational Linguistics*, 31(4):439–475.