Learning Lexical Semantic Relations using Lexical Analogies — Extended Abstract

Andy Chiu, Pascal Poupart, and Chrysanne DiMarco

David R. Cheriton School of Computer Science
University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
{pachiu,ppoupart,cdimarco}@uwaterloo.ca

1 Introduction

Linguistic ontologies, most notably WordNet [1], have been shown to be a valuable resource for a variety of natural language processing applications. Presently, linguistic ontologies are largely constructed by hand, which is both difficult and expensive. A central problem that demands an automated solution is the discovery and incorporation of *lexical semantic relations*, or semantic relations between concepts. Lexical semantic relations are the fundamental building blocks that allow words to be associated with each other and linked together to form cohesive text. Despite their importance, lexical semantic relations are severely underrepresented in current linguistic ontologies. As Morris and Hirst [2] point out, current linguistic ontologies only capture what they call *classical relations* — basically, WordNet relations such as hyponymy, hypernymy, troponymy, meronymy, antonymy, and synonymy. However, the majority of lexical semantic relations found in real-world text are in fact *non-classical* — for example, *positive-qualities* (*humbleness* and *kindness*), *cause-of* (*alcohol* and *drunk*), and *founder-of* (*Gate* and *Microsoft*) [2]. Manually populating linguistic ontologies with all instances of classical and non-classical relations is impractical as there are simply too many of them and there are as yet no systematic methods for even recognizing non-classical relations in an arbitrary text. Clearly, automation is needed.

In this research, we tackle the problem of automated learning of lexical semantic relations from text. We present an iterative algorithm in Sect. 2 that expands a small set of sample relation instances to a much larger set by making use of a dictionary of *lexical analogies*. In Sect. 3, we demonstrate a system that generates this dictionary automatically from text. The system builds lexical analogies by computing the similarities of the semantic relations between words, which we characterize by their dependency structures. The actual computation of similarity is carried out using a Vector Space Model augmented with Singular Value Decomposition. We give some promising preliminary experimental results in Sect. 4, and conclude with an outline of future work in Sect. 5.

2 Learning Semantic Relations Using Lexical Analogies

We refer to the semantic relation between a pair of words as the *underlying relation* of the word-pair. A *lexical analogy* $A = W_1 : W_2 :: W_3 : W_4$ is two

word-pairs (W_1, W_2) and (W_3, W_4) whose underlying relations are identical or similar, for example, *abbreviation:word::abstract:report*. (W_1, W_2) is called a *lexical analogue* of (W_3, W_4), and vice versa. We use the term *analogousness* to refer to their degree of similarity. Clearly, since similarity is subjective, analogousness is an application-dependent measure.

We propose that lexical analogies are the key for the systematic learning of lexical semantic relations. Suppose we are given that the semantic relation between W_1 and W_2 is R, and that (W_1, W_2) is a lexical analogue of (W_3, W_4). Then we can infer with high probability that the semantic relation between W_3 and W_4 is likely also R. Once we know this, we can apply the same inference to conclude that all lexical analogues of (W_3, W_4) are also likely to be related by R. This process continues until we run out of lexical analogies. Essentially, we are using lexical analogies as bridges through which relations can spread from one word-pair to another. This insight leads to our iterative algorithm of learning lexical semantic relations using lexical analogies.

Table 1 illustrates our algorithm in pseudo-code. The algorithm requires three inputs: a dictionary of lexical analogies A in which the lexical analogues of many word-pairs are listed, a set $L = \{L_1, ..., L_k\}$ of lexical semantic relations of interest, and a set E_{L_i} for each L_i that contains a small number of sample instances of the relation L_i. An example of the input is L=*{part-of}* and E_{L_1}=*{(finger, hand),(beak, bird)}*.

```
1.  for each L_i in L
2.    repeat
3.      pick a random subset S of E_{L_i} with |S| ≥ 2
4.      for each element in S, obtain its set of analogues from A
5.      take the intersection T of all the above sets
6.      add all elements in T to E_{L_i}
7.    until all possible subsets have been tried
```

Table 1. Algorithm for Learning Lexical Semantic Relations

The result of our algorithm is that each E_{L_i} is rapidly expanded from a small set of samples to a large set of instances by iteratively incorporating the lexical analogues of the samples. Clearly, the key to our algorithm is the dictionary of lexical analogies. In the next section, we present a system that builds this dictionary automatically by generating lexical analogies from text.

3 Generating Lexical Analogies from Text

The core of our analogy generation system is based on the use of a *dependency grammar* [3]. A dependency grammar specifies how words in a sentence are related and grouped, much like the familiar phrase structure grammar. However, instead of relating each word to the phrase to which it belongs, a dependency

grammar relates each word to the word it depends on syntactically. A dependency parse of a sentence produces a list of dependencies. For each dependency, the depending word is called the *dependent*, and the word depended on is called the *governor*. Each word can have multiple or no dependents, but must have exactly one governor, except for one word in the sentence called the *head word* which has no governor. A *dependency tree* organizes a dependency list by making each dependent a child of its governor, and the head word the root. A *dependency path* is an undirected path through a dependency tree, and a *dependency pattern* is a path with both ends replaced by slots. Fig. 1 shows the dependency structures for the sentence "*the council approved the new budget*".

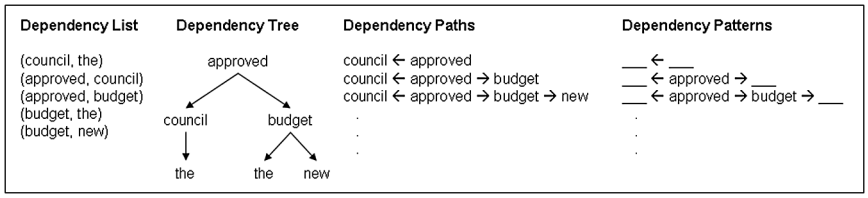

Fig. 1. Dependency Structures of "*the council approved the new budget*"

Our lexical analogy generation system is called GELATI, an abbreviation for **GE**neration of **L**exical **A**nalogies from **T**ext **I**nformation. Fig. 2 shows an overview of GELATI.

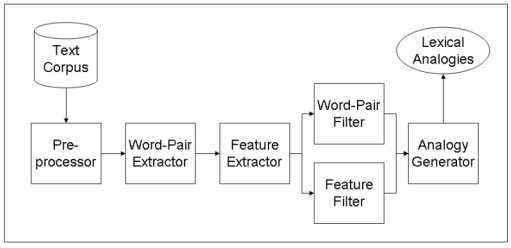

Fig. 2. Overview of GELATI

The input to GELATI is a corpus of text documents, which must be large enough for lexical analogies to occur repeatedly. The input first goes through a preprocessing stage, during which each input file is segmented into sentences and parsed into dependency tress. We use MxTerminator (Ratnaparkhi [4]) for segmentation, and Minipar (Lin [5]) for dependency parsing. Minipar also additionally performs word stemming and simple compound-noun recognition.

3.1 Extractors and Filters

The next components in GELATI's pipeline are the word-pair and the feature extractors. The word-pair extractor extracts a list of semantically related word-pairs from the input data. These word-pairs serve as the building blocks from which analogies are drawn. After extracting the word-pairs, ideally the next step would be to identify their underlying relations so their analogousness can be computed. Unfortunately, given just plain text data, such identification is extremely difficult. Therefore, GELATI instead uses the feature extractor to extract syntactic and semantic features about each word-pair, then uses these features as an indication to characterize the word-pair's underlying relation.

In our implementation of GELATI, we chose to merge the two extractors into one that performs extraction based on dependency patterns. The combined extractor is grounded in the hypothesis that highly syntactically related words also tend to be semantically related, and thus we can use the dependency structure of a sentence to approximate the semantic relations of its words. The extractor takes each dependency tree from the preprocessor and generates all possible dependency patterns from the tree. Each pattern is then tested against a set of constraints. If the pattern passes all constraints, the words that were in the pattern's slots are extracted as a word-pair, and the pattern itself is extracted as a feature for the word-pair. The constraints are (1) the pattern must span exactly three words; (2) both slots of the pattern must be nouns; and (3) the word between the slots must be a verb. These constraints are partially inspired by Lin's [6] work on discovering inference rules, and partially by the fact that they correspond to the most prominent construct in English to relate two words, namely the noun-verb-noun construct.

Continuing the example from Fig. 1, the extractor would extract the word-pair (*council, budget)*, as well as the dependency pattern "___ ← approved → ___", which becomes a feature of the word-pair.

Once all word-pairs and features have been extracted, they are filtered through two filtering components so that only the most relevant ones remain. The reason that filtering is done globally after extraction, instead of locally within each extraction component, is because the final result of extraction can provide important information for filtering. For example, suppose there is a feature that occurs with every word-pair. Clearly this feature is much too general to provide useful information for characterizing word-pairs, and hence should be filtered out. Such information, however, is only available after both extractors complete.

Currently GELATI uses a simple filtering scheme parameterized by four parameters, $K_{wp_{min}}$, $K_{wp_{min}}$, $K_{wp_{max}}$, $K_{f_{min}}$, and $K_{f_{max}}$. A word-pair is filtered out if it has less than $K_{wp_{min}}$ or more than $K_{wp_{max}}$ features, and a feature is filtered out if it is associated with less than $K_{f_{min}}$ or more than $K_{f_{max}}$ word-pairs. The optimal values for these parameters are determined through experiments.

3.2 Analogy Generator

The final component in GELATI's pipeline is the analogy generator, which produces a list of lexical analogies by associating word-pairs with high analogousness

as evidenced by their features. GELATI uses a Vector Space Model to perform this computation. Specifically, GELATI creates an F-dimensional vector for each word-pair, where F is the number of total features extracted. The i^{th} dimension corresponds to the i^{th} feature, and is set to 1 if that feature is associated with this word-pair, or 0 otherwise. Once the vectors are computed, the analogousness between any two word-pairs is simply the cosine measure of their vectors.

This straightforward implementation, however, fails to take into account the influence of other word-pairs in the extracted set. Consider, for example, three word-pairs WP_1, WP_2, and WP_3, such that WP_1 and WP_2 share many common features, as do WP_2 and WP_3. Clearly, in this case WP_1 and WP_2 are likely an analogy, as are WP_2 and WP_3. However, this implies that the underlying relation of all three word-pairs are similar, and hence WP_1 and WP_3 would also have a good chance of being an analogy regardless of how many features they share. This transitivity between word-pairs can be extended through any number of word-pairs, and a similar transitivity also applies to features. Consequently, the analogousness between any two word-pairs is ultimately influenced by all other word-pairs and features. The analogy generator must therefore consider the entire set of word-pairs and features together, instead of relying on just pair-wise comparisons.

We may observe that this is the same problem as a well-documented problem in Information Retrieval (IR), namely, relevant documents may not necessarily share many common words. A particular successful solution in the IR community is Latent Semantic Analysis (LSA) [7]. The intuition is that if the vector space is compressed into an optimal reduced dimension, the influence of other elements will be magnified so that elements that are truly relevant will be pushed closer together. This idea of dimension reduction has been shown to be useful in IR as well as a number of other applications including analogy comparison [8]. Hence we also adopt this technique. Specifically, the analogy generator first builds a large word-pair-by-feature matrix by concatenating the feature vectors of all word-pairs. It then applies Singular Value Decomposition to reduce the matrix to an empirically-determined optimal dimension of K_{dim}. Once SVD is completed, the generator uses the reduced vectors as the new feature vectors, and computes cosine measures as before.

4 Preliminary Experimental Result

Table 2 shows a small subset of lexical analogies generated by GELATI in a preliminary experimental run. The input for this experiment consists of about 1 gigabyte of text data from the TREC dataset [1]. The parameters were: $K_{wp_{min}} = 50$, $K_{wp_{max}} = \infty$, $K_{f_{min}} = 10$, $K_{f_{max}} = 100$, and $K_{dim} = 400$. A total of 8148 word-pairs and 10470 features was extracted. 3384 lexical analogies were generated, of which an estimated 40–60% are valid.

[1] http://trec.nist.gov/

Analogy Extracted	Underlying Relation
gorbachev:moscow::bush:washington	head-of
hostage:release::troops:withdrawal	safely-returns
legislature:law::city_council:ordinance	makes-and-revises
increase:rise::drop:decline	synonym
prosecutor:evidence::investigator:information	collects
article:newspaper::story:magazine	publication-in
problem:business::drought:farmer	causes-trouble-for
judge:request::court:claim	approves

Table 2. Extracted Lexical Analogies

5 Conclusion and Future Work

In this research we are developing methods that use machine learning techniques to systematically discover and learn lexical analogies and lexical semantic relations from text. There are a number of future research issues that we are planning to explore. First, we will implement the relation-learning algorithm outlined in Sect. 2 and investigate whether the initial sample set can be built automatically so as to fully automate the algorithm. Second, we will extend GELATI with alternative extractors and analogy generators. In particular, we plan to build extractors that can take advantage of hyperlinks, which often suggest strong semantic relatedness. We are also experimenting with a Bayesian alternative to SVD that allows dimension reduction to occur in a more principled manner.

References

1. Christiane Fellbaum, ed., *WordNet — An electronic lexical database*. MIT Press (1998)
2. Jane Morris and Graeme Hirst. Non-classical lexical semantic relations. In *Proceedings of HTL-NAACL Workshop on Computational Lexical Semantics* (2004)
3. Lucien Tesnière. *Éléments de syntaxe structurale*. Paris: Librairie C. Klincksieck (1959)
4. Jeffrey C. Reynar and Adwait Ratnaparkhi. A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the 5th Conference on Applied Natural Language Processing*, pp 16–19 (1997)
5. Dekang Lin. Principle-based parsing without overgeneration. In *Proceedings of the 31st Annual Meeting on ACL*, pp 112–120 (1993)
6. Dekang Lin and Patrick Pantel. Discovery of inference rules for question answering. *Natural Language Engineering*, 7(4):343–360 (2001)
7. Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41:391–407 (1990)
8. Peter Turney. Measuring semantic similarity by latent relational analysis. In *Proceedings of IJCAI'2005*, pp 1136–1141 (2005)