Classifying Relationships Between Nouns

Jun Araki **Heeyoung Lee** **Erik Kuefler**

Abstract

In this paper we develop a system for classifying relationships between noun pairs. This is accomplished by finding the dependency paths between the nouns in the Reuters corpus and training a multi-class classifier based on the counts of the paths. We experiment with several classification algorithms (NB, kNN, SVM), feature selection techniques, and additional features. The resulting system achieves up to a 60% classification accuracy over four classes of word relationships using an SVM.

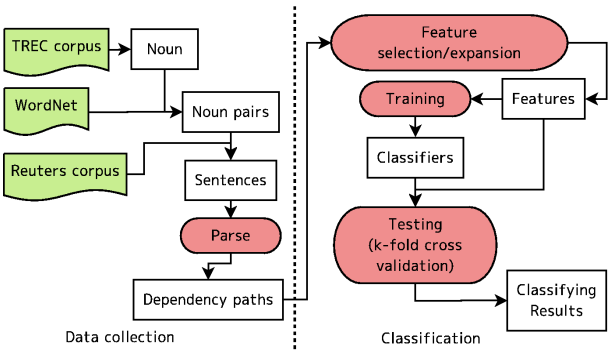

Figure 1: The data collection and classification process.

1 Introduction

As research in natural language processing shifts towards semantic understanding, it is becoming increasingly important to understand the relationships between words. One source of semantic information is WordNet, a lexical database which aggregates semantic relations over a huge number of words. However, constructing and maintaining such large-scale databases are extremely laborious. Therefore, there is much interest in the automatic extraction of semantic relations from text corpora. Such information is useful in topics including automatic thesaurus creation and question answering systems. In this paper, we develop a system for automatically classifying relationships between nouns based on their usage in a large corpus of text.

2 Related Work

Hearst developed a hypernymy detection method based on lexico-syntactic patterns, but did not apply it to other relations (Hearst, 1998). Girju et al. proposed a pattern-based meronym classifier that showed some improvement, but they focused on only a few hand-written patterns such as "NP1 of NP2" (Girju et al., 2003). Snow et al. built an automatic classifier for hypernymy relations that extracted lexico-syntactic patterns based on known hypernyms in WordNet (Snow et al., 2005). Using dependency paths, they generated a set of patterns to find novel hypernyms. We build off this work by generalizing from hypernym classifiers to multi-class classifiers capable of identifying several noun relationships.

3 Approach

Our approach consists of two phases: data collection and classification. In the data collection phase, we start by extracting nouns from a corpus and retrieving antonyms, meronyms, hyponyms, and hypernyms from WordNet for each noun (Section 3.1). For each pair retrieved, we find sentences in another corpus in which both words appear (Section 3.2.1). We then extract the dependency path[1] between the words in that sentence (Section 3.2.2). For each pair, we then count the number of times each type of dependency path occurs to generate a feature vector.

In the classification phase, we experiment with several classification algorithms and feature selection methods (Section 3.3) on the feature vectors resulting from the data collection phase. We consider a subset of the extracted vectors balanced by word class and use 10-fold cross-validation to train and evaluate a multi-class classifier. The resulting system is capable of classifying the relationship between an arbitrary word pair by examining its relationships in a corpus and running a classification algorithm on the results.

3.1 Word Relationship Extraction

In order to extract word relationships, we start with a seed set of nouns. To obtain this set, we use NLTK (Loper and Bird, 2002) to extract all nouns from the TREC-5 corpus (Voorhees and Harman, 1997) and retain the 3,000 most frequent nouns. We use a different corpus for this

[1] See (Snow et al., 2005) for a full description of dependency paths.

step so as to avoid over-fitting in our data, though we expect the results to be comparable.

For each candidate word, we then extract from WordNet the top antonym, meronym, hyponym, and hypernym for that word. This results in a large set of pairs of related words, labeled according to the type of their relationships.

3.2 Feature Extraction

Once a set of word pairs has been generated, the next step is to extract the features that will be used directly by the machine learning algorithm for each pair. This requires extracting sentences for the corpus (we use the Reuters RCV1 corpus (Lewis et al., 2004) for the remainder of this paper) and parsing and counting their dependency paths.

3.2.1 Sentence Extraction

The first step is to locate in the corpus every sentence that contains both words in the word pair. The large size of the Reuters corpus (nearly 4GB comprised of over 800,000 articles) makes this a non-trivial task, as performing a linear search for each word pair is prohibitively expensive. To alleviate this problem, we first construct an index that gives, for each word in the word pair list, the documents in which that word occurs. Since each article is short and many words are rare, this leads to an efficient representation (the resulting index is approximately 320 MB).

Given this index, we can efficiently extract sentences as follows. First, we fetch the document list for each word in the pair and take the intersection of those lists. We then perform a linear scan over the remaining documents, which can be done fairly efficiently as each individual document is short (usually under 10KB). Whenever we find a sentence containing both words, we add it to the list that will be processed in the next phase.

3.2.2 Sentence Parsing

We have so far gathered a list of sentences for each word pair. Next, we must parse each sentence to extract the dependency path between the word pair. To do this, we use the Stanford Parser (Klein and Manning, 2003), which, given a sentence, extracts a list of dependencies in that sentence in the form `dep_name(word1-index1, word2-index2)` (Figure 2).

Given this output, we wish to find the dependency path between the word pair. When the two words are directly dependent on each other (such as "president" and "USA" in Figure 2), we return this dependency directly. Otherwise, we perform a breadth-first search to find the shortest path between the two words. For example, in Figure 2 we would output "`nsubj-prep_of`" as the dependency path between "Obama" and "USA".

```
nsubj(president-3, Obama-1)
cop(president-3, is-2)
det(USA-6, the-5)
prep_of(president-3, USA-6)
```

Figure 2: *Output of the Stanford Parser for the sentence "Obama is president of the USA".*

This process is repeated for each extracted sentence, yielding a list of all dependency paths between the two words. All that remains in generating the feature vector is to assign a unique index to each path and to count the number of times that the path was encountered. We would expect to see many occurrences of short paths like `prep_of` and few of longer paths, e.g. `dep-advmod-dep-pobj`.

3.3 Feature Selection and Expansion

The process so far results in a set of feature vectors suitable for training and testing a classifier. However, we would not necessarily expect all of the extracted features to be useful, and there may be relevant factors of words not captured by dependency paths. Accordingly, we experiment with several feature selection and expansion techniques. We hope that feature selection will be able to remove irrelevant or noisy features, thereby improving our classifiers by reducing their variance. Conversely, adding new features should improve the flexibility of our classifiers, thereby decreasing their bias.

3.3.1 Feature Selection

One easy way to reduce the number of features is path shortening. There are many noun pairs having long and complex dependency paths. We examine three simple path shortening methods: taking only the first dependency, only the last dependency, and only the first and the last dependencies. For example, the dependency path `prep_by-dep-prep_to` shortens to `prep_by`, `prep_to`, or `prep_by-prep_to` respectively.

Another simple technique is removing the features that appears only once through the data. This technique could potentially be generalized to removing any feature that occurs fewer than N times; we experiment only with removing single-count features.

Finally, we use a χ^2 analysis to compute a rank for each feature for each class. We then retain a feature only if it appears in the top N features for any class (with N between 10 and 1,000).

3.3.2 Feature Expansion

In addition to experimenting with techniques for removing features, we also examine the effects of adding new features. The two additional features with which

we experiment are shared prefix and suffix length, which specify the number of characters that the words share at the beginning and end of the words, respectively. For example, the words "cooperation" and "competition" have a shared prefix length of 2 and a shared suffix length of 4. These features were motivated by the observation that antonyms in particular tend to share long common suffixes (e.g. "stability" vs. "instability"), so such features may be useful for discriminating antonyms in particular.

3.4 Classification

Once features have been chosen, we train and test on the resulting vectors using a variety of classification algorithms. The three we consider here are multinomial Naives Bayes (using Laplace smoothing and including variants such as Complement Naive Bayes (Rennie et al., 2003)), k-nearest neighbor, and support vector machines as implemented by libSVM (Chang and Lin, 2001). Each classifier is trained and evaluated using 10-fold cross-validation over a balanced subset of the extracted data.

4 Results

In this section we examine the results of our data collection and classification process. Section 4.1 examines the counts of each sort of relationship extracted at each step in the data collection process. Section 4.2 examines the accuracy of several classification algorithms using several feature selection techniques.

We focus on accuracy as our means of evaluating classifiers, as it is the most intuitive way to summarize a multi-class classification problem. Here, accuracy is defined as the number of correctly classified examples divided by the total number of examples. This gives us an aggregate score rather than a per-class score. We would expect an accuracy of 25% for purely random classification.

4.1 Data Collection

The number of features remaining after each step of the data collection process is shown in Table 1. The smallest class was antonyms with 60 examples and the largest was hypernyms with 762 examples. From this data, we created a balanced subset of 240 examples (60 per class) by taking the longest (and so least likely to be effected by noise) feature vectors from each class.

4.2 Classification and Feature Selection

Results for each classification and feature selection method are shown in Table 2. We found support vector machines with shared affix length features and no feature selection to be the most accurate classification method. We report first on the baseline classification results for each algorithm in Sections 4.2.1-4.2.3 and then examine the results of adding feature selection and expansion techniques in Sections 4.2.4 and 4.2.5.

	Pairs	Sentences	Paths
Antonyms	128	103	60
Meronyms	423	358	85
Hypernyms	1,658	1,599	762
Hyponyms	1,348	1,074	321

Table 1: *For each relationship, respectively, the total number of word pairs drawn from WordNet, the number of pairs which appear together in the Reuters corpus, and the number of words for which at least one dependency path could be extracted.*

4.2.1 Naive Bayes

Transformation and weight-normalization (Rennie et al., 2003) were found to hurt performance, so we report only the results of standard multinomial Naive Bayes (NB) and Complement Naive Bayes (CNB) in Table 2. We achieved a baseline accuracy of 42.9% using NB and a slight improvement of 45.0% using CNB.

	NB	CNB	kNN	SVM
Base	42.9%	45.0%	42.1%	57.1%
Thresholding	39.5%	43.6%	39.2%	52.4%
Shortening	44.6%	46.6%	44.2%	52.5%
χ^2	50.9%	51.3%	47.5%	53.9%
Expansion	52.9%*	52.9%*	48.3%	**60.8%**

*Expansion for NB and CNB includes χ^2 feature selection.

Table 2: *Accuracy of different classification techniques in the baseline case as well as after applying feature selection and expansion techniques.*

4.2.2 k-Nearest Neighbor

We experimented with kNN using $k = 3$ and $k = 5$. We observed that 5NN outperformed 3NN by a few percentages under every condition. For instance, we achieved a baseline accuracy of 41.7% for 3NN and 42.1% for 5NN. Thus, we report only the results of 5NN in Table 2.

4.2.3 Support Vector Machines

Our most accurate baseline results of 57.08% were achieved using a multi-class support vector machine (using the one-against-one method) with features scaled in the range [0, 1], a Gaussian kernel, a C value of 500,000, and a γ^2 value of .25.

We found [0,1] scaling to outperform both [-1, 1] scaling (which was better than no scaling), and C=500,000

[2] γ is a parameter to the Gaussian kernel, which is computed as $K(x, z) = \exp(-\gamma|x - z|^2)$.

and γ=.25 were found via grid search to outperform all other C and γ combinations. Polynomial kernels with degrees between 2 and 5 did not perform significantly better than chance. A linear kernel achieved an accuracy of only 34%. A sigmoid kernel performed relatively well (49%), but was still inferior to the Gaussian kernel (57%).

4.2.4 Feature Selection

For non-SVM classifiers, feature selection methods were useful in proportion to their complexity. Simple thresholding slightly harmed accuracy in all cases, whereas χ^2 gave a significant 5-8% boost in performance. Path shortening fell between these two methods with a minor 1-2% improvement in performance.

Table 3 shows a few of the highest-ranked features for each class according to χ^2. We saw that the most important paths tend to only contain one or two steps. Particularly important relationships were prepositional (`prep_for` and `prep_of`), conjunctional (`conj_and` and `conj_or`), and adjectival (`amod`).

	Anto.	Mero.	Hypo.	Hyper.
1	conj_and	prep_for-amod	amod	amod
2	conj_or	nsubjpass	nn	conj_and
3	nn	prep_of-amod	infmod	prep_of

Table 3: *The highest-ranked features for each class, according to χ^2.*

Though feature selection improves accuracy for other classification algorithms, it harms accuracy for SVMs: for each selection method examined, accuracy dropped from 57% to around 52%-53%.

4.2.5 Feature Expansion

Feature expansion via shared affix lengths was useful for all classifiers (note that feature selection harms the performance of SVM), leading to an improvement between 1% and 6% (even SVMs are improved by feature expansion). The SVM classifier resulting from feature expansion achieved an accuracy of 60.8%, which was the highest of any classifier examined.

Interestingly, feature expansion also greatly changed the optimal value of C in the SVM classifier, reducing it from 500,000 to 2. This implies that the optimal SVM without feature expansion approximates a hard-margin classifier, but that the optimal classifier with feature expansion is much more tolerant of noisy data.

5 Discussion

The results yield several interesting patterns. In Section 5.1 we examine the raw data that we collected. Section 5.2 looks at the results of feature selection and expansion techniques. Finally, Section 5.3 considers individual classification algorithms and insights gathered from them.

5.1 Data Collection

Even though the data is of reasonably high quality, the amount of data (Table 1) is limited by our smallest class (antonyms) due to the desire to train over a balanced data set. After balancing our data, we are left with 240 total training examples (60 per class). This seemed to be sufficient data to get useful results, but gathering more would be likely to improve classification accuracy. Based on our results, the best way to do so would appear to be finding a more comprehensive list of antonyms. Increasing the number of frequent words (currently 3000) or introducing some heuristics such as using antonym dictionaries into out system would likely improve its performance because it would have a balanced set of more features.

5.2 Feature Selection and Expansion

The results of feature selection as displayed in Table 2 are interesting for several reasons. We see that different techniques lead to different degrees of success: simple thresholding based on the number of times each feature occurs harms accuracy in all cases, whereas techniques that combine features by shortening paths are slightly helpful and χ^2 is very helpful. The first two results are interesting — it is not obvious that features occurring only once would have much explanatory power, nor is it clear that the first and last parts of the path are the most important, but focusing on them does appear to improve accuracy slightly. χ^2 is a standard technique known to be successful in many contexts, so its good performance is unsurprising. χ^2 generally did best when selecting around 40 features for each class (out of 2,422).

Another interesting fact is that these feature selection techniques only appear effective with less powerful classification algorithms (Naive Bayes and k-nearest neighbor). For support vector machines, performing any sort of feature selection harms performance, though the resulting classifier still outperforms the other techniques. SVMs seem powerful enough to select their own features without using a separate technique.

Though results from feature selection were mixed, the effect of adding a new feature (shared affix length) was unambiguously positive: accuracy was improved in all cases, including SVMs. The fact that adding features works better than removing features likely implies that classification errors are a result of high bias rather than high variance, meaning that the classification algorithms do not have enough information available to adequately distinguish among classes. It seems likely that finding additional useful features would further improve accuracy.

Finally, the results of the χ^2 analysis in Table 3 are interesting on their own, in that they reveal how word

pairs of a given class tend to be connected. For example, antonyms tend to be connected with the conjunctions 'and' and 'or'. This matches our intuitions of how antonyms appear in text.

5.3 Classification

Support vector machines proved to be the best classification algorithm by a wide margin, outperforming Naive Bayes and kNN regardless of the feature selection and expansion techniques used. Naive Bayes outperformed kNN, and was improved in most contexts by using the complement heuristic. These results are all consistent with those reported elsewhere.

Another interesting way to look at the results is via the confusion matrix, which is shown in Table 4. These results suggest that hypernyms can be identified with high accuracy compared to other classes. This may be due to having significantly more data available for hypernyms, which allows us to train and test over less noisy samples for that class.

Act \ Pred	Anto.	Mero.	Hypo.	Hyper.
Antonym	7	7	0	1
Meronym	4	8	2	1
Hyponym	0	1	6	8
Hypernym	0	0	1	14

Table 4: *Confusion matrix for the base SVM classifier over one fold of the data. Actual values are shown in rows and predicted values are shown in columns.*

One final way to examine performance on individual classes is to train one-vs-rest classifiers for each class of the data. To do so, we trained a classifier for each class by drawing 60 examples from that class and 20 from the remaining classes. The results are shown in Table 5. These results suggest that meronyms are the easiest to identify, achieving nearly 90% accuracy. Interestingly, these numbers don't correspond to the confusion matrix in Table 4. This may be because libSVM trains one-vs-one rather one-vs-rest classifiers.

Class	Anto.	Mero.	Hypo.	Hyper.
Accuracy	81.7%	89.2%	60.8%	67.5%

Table 5: *Per-class accuracy for one-vs-rest SVMs.*

6 Conclusion

We were able to extract dependency paths for many words pairs by examining the Reuters corpus. Using a support vector machine trained over these dependency paths with additional features based on affix length, we were able to achieve 60% accuracy classifying over four relationships (random classification would be 25%). Feature selection methods were useful for non-SVM classifiers, but did not improve SVM performance. These results suggest that our technique is useful for distinguishing among related words, and could likely be improved via the identification and addition of other useful features.

7 Future Work

Though our results are promising, there is still room for improvement. Our data set is fairly small, and so gathering more data from a larger corpus or a more comprehensive word list would likely improve results. Our experiments with affix lengths as a feature suggest that we could further improve accuracy by identifying relevant features apart from dependency paths. Finally, in order for this system to be useful in practice, it will likely have to be able to identify more than just the four relations we examined as well as explicitly indicate when no relation exists between a given word pair. We note that our multi-class classification method can be easily extended to identify more relationships, even between other parts of speech (e.g. verb-noun relationship).

References

Chih-Chung Chang and Chih-Jen Lin, 2001. *LIBSVM: a library for support vector machines*. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

R. Girju, A. Badulescu, and D. Moldovan. 2003. Learning Semantic Constraints for the Automatic Discovery of Part-Whole Relations. In *Proceedings of the Human Language Technology Conference (HLT)*.

Marti Hearst. 1998. Automated Discovery of WordNet Relations. In *An Electronic Lexical Database and Some of its Applications*. MIT Press.

Dan Klein and Christopher D. Manning. 2003. Accurate Unlexicalized Parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430.

David D. Lewis, Yiming Yang, Tony G. Rose, G. Dietterich, Fan Li, and Fan Li. 2004. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397.

Edward Loper and Steven Bird. 2002. NLTK: The natural language toolkit.

J.D. Rennie, L. Shih, J. Teevan, and D. Karger. 2003. Tackling the poor assumptions of naive bayes text classifiers. In *Machine Learning International Workshop and Conference*, volume 20, page 616.

Rion Snow, Daniel Jurafsky, and Andrew Y. Ng. 2005. Learning syntactic patterns for automatic hypernym discovery. In *Proceedings of NIPS 17*. MIT Press.

Ellen Voorhees and Donna Harman. 1997. Overview of the fifth text retrieval conference (trec-5). In *Proceedings of the sixth Text REtrieval Conference (TREC-6). (NIST Special Publication 500-240*.