Complex, Corpus-Driven, Syntactic Features for Word Sense Disambiguation

Ari Chanen and Jon Patrick
Sydney Language Technology Research Group
School of Information Technologies
University of Sydney
Sydney, Australia, 2006
{ari,jonpat}@it.usyd.edu.au

Abstract

Although syntactic features offer more specific information about the context surrounding a target word in a Word Sense Disambiguation (WSD) task, in general, they have not distinguished themselves much above positional features such as bag-of-words. In this paper we offer two methods for increasing the recall rate when using syntactic features on the WSD task by: 1) using an algorithm for discovering in the corpus every possible syntactic feature involving a target word, and 2) using wildcards in place of the lemmas in the templates of the syntactic features. In the best experimental results on the SENSEVAL-2 data we achieved an F-measure of 53.1% which is well above the mean F-measure performance of official SENSEVAL-2 entries, of 44.2%. These results are encouraging considering that only one kind of feature is used and only a simple Support Vector Machine (SVM) running with the defaults is used for the machine learning.

1 Introduction: Syntactic Features

The best features for machine learning classification are the ones that have the most discriminatory power, for the task at hand. This paper will be discussing the use of **syntactic features** (SF's) in **word sense disambiguation** (WSD.) In WSD, the task is to choose (or classify) the correct sense of the **target word** (the word whose sense is to be disambiguated) given the surrounding text. One type of feature that is commonly used in WSD classification systems is called **bag-of-words**. Bag-of-words features are rather information poor, only specifying the presence or absence of words in the target word's context. SF's, on the other hand, are much richer in information. Not only do syntactic features have information on the presence of words in the context but they also include information about the syntactic relationships that hold between the target word and context words in the same sentence as the target.

In order to use SF's, a syntactic parser is needed that produces a parse tree for every training corpus sentence. The parse tree gives the syntactic relationships between words in each sentence. Connexor parser (Järvinen and Tapanainen, 1997) was used to annotate the data with syntactic relationships in the research presented here. In this research, a SF is defined as a connected group of words from a parse tree that must include the target word. The SF includes information on each of its words, the syntactic relationships between them, and information on how each word relates to the others in the tree hierarchy. The SF word information includes the word, lemma, and **part-of-speech** (POS) for each word.

The use of syntactic features in WSD might seem to be a more effective discriminatory feature compared to a information poor feature like bag-of-words because of the potential for SF's to offer more specific information about how the sense of

Proceedings of the Australasian Language Technology Workshop 2004
Macquarie University, Sydney, December 8th, 2004.

Accepted after full paper review

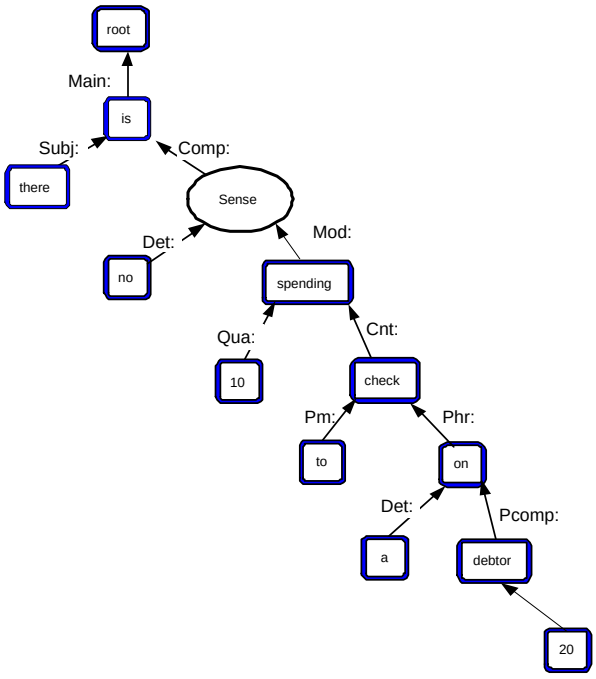

Figure 1: Conexor parser tree for "**There is** no **sense** spending 10 to check on a 20 debtor."

a target word relies not only on the words around it, but also on the information about the syntactic relationships that hold between words. Alas, the fact that SF's contain more detail about the set of words in a specific SF makes it less likely for a given SF to occurred as frequently as the corresponding set of bag-of-words features, thus leading to a general problem with SF's of having lower recall. The issue is data sparseness not whether or not syntactic features have potential as a powerful NLP feature. Rather, the question is how can the strength of syntactic features be boosted. In this paper, we explore two ways to help syntactic features live up to their promise by:

1. Developing an algorithm for finding syntactic features in the sentence that surrounds the target-word. More specifically, the algorithm identifies **all** syntactic features that 1) involve the target word 2) contain a number of syntactic links that is less than or equal to a fixed maximum.
2. Allowing for abstract features. Syntactic features are made up of member elements of various types. The term "abstract" here is being used in the sense of being opposite of concrete. One type of member is the lemma element. A method has been devised to exhaustively enumerate all possible features where one or two wildcards have replaced original lemma elements. Any feature where a lemma has been abstracted to a wildcard is defined as abstract.

Both of these methods are shown experimentally to be effective in boosting recall. The specifics of the methods will be discussed in section 4 and the experimental results will be discussed in section 5.

2 The WSD Task

2.1 General Description

One of the reasons that human language is far from trivial to process is that many words in the lexicon hold different meanings depending on their context. For instance, the word "sense" has five senses as a noun and four senses as a verb according to WordNet 2.0. Two of the five noun senses are exemplified in the following two sentences:

1. There is a pleasing **sense** of justice about the observation. (Here "sense" means "general conscious awareness" – WordNet 2.0)
2. There is no conceivable **sense** in going to the opposite extreme. (Here "sense" means "sound practical judgement" – WordNet 2.0)

The sense of a particular instance of a word in a text can only be determined by the surrounding context.

In the SENSEVAL competitions, teams of researchers build word sense classifiers. The teams are all given the same training examples of the same set of words. The task is to build one classifier for each word that classifies an instance of that word in context as one of its possible senses. The training examples consist of the target-word, along with the surrounding context which is typically several sentences. SENSEVAL is a supervised learning task so all of the training examples supplied by the SENSEVAL organisers come with a label in the form of a sense-tag that nails down the sense of the target-word to one of the senses listed in WordNet.

Proceedings of the Australasian Language Technology Workshop 2004
Macquarie University, Sydney, December 8th, 2004.

Accepted after full paper review

Different research groups try to outperform the other groups by using different and hopefully superior methods. There are, at least, five basic areas in which the groups may differ 1) training data used 2) enrichment of the training data, if any 3) kinds of features extracted for the machine-learning process 4) method of selecting the best features 5) machine-learning algorithm or combination of algorithms used.

2.2 Data Enhancements by Parsing

The SENSEVAL-2 and SEMCOR sense-tagged corpora were used as the training data in this research. This data was enriched by extracting syntactic information using the Connexor parser. It is difficult to extract reliable syntactic information without first processing the data with a good parser. Connexor is a dependency parser, as opposed to a constituency parser. Any type of syntactic parser that produces a hierarchical sentence tree could be used with the syntactic feature extraction methods used in this work.

2.3 Feature Selection Method

The method used for feature selection follows (Ng and Lee, 1996). Three steps are used on all the features found by a given feature-extractor to filter them down to the selected, final set of features used in the machine learner. According to this paper, each feature must meet the following conditions to be selected:

1. The feature f must have a feature count of at least FC_{min}.
2. The conditional probability for some sense i given the feature f must be greater than a predetermined probability. $P(i \mid f) \geq CP_{min}$.

Condition one is enhanced by allowing the minimum count to depend on the abstraction level of the feature where the **abstraction level** is defined as the number of wildcards in a syntactic feature.

2.4 Motivation

The SENSEVAL-2 papers indicate that no one came up with a single "magic bullet" idea that put them out in front of the crowd, rather the teams that did best were better able to combine known ideas and better able to make small adjustments in the application of these ideas. This is one of reasons this study concentrates on understanding the problems in-depth and improving a single type of feature rather than combining many features and using many different machine learners.

2.5 WSD Learning Infrastructure

Our WSD system is built on an extensible framework for feature extraction and feature vector construction. All of the experiments reported on here were done using this in-house system. Due to limited space, the system will not be described here but in (Bell and Patrick 2004.)

3 Syntactic Features and WSD

3.1 Dekan Lin

(Lin, 2000) describes the only other WSD system we are aware of that makes use of syntactic features alone. Lin's system discovered all syntactic features in the corpus which inspired the current systems principle of only using syntactic features discoved automatically in the corpus. Lin's syntactic features are less inclusive, and less complex then those described in this paper. See section for further comparisons. He used a nearest neighbour (NN) algorithm to choose the best sense of the word. Despite having simpler features, his system showed better performance on the same task. In the conclusion to this paper , we will speculate as to why.

3.2 David Yarowsky et al.

(Yarowsky et al., 2001) describes the system that did best among all competing supervised-learning systems at SENSEVAL-2. This system is not directly comparable because they used five types of features and a more complex, voting scheme machine-learner. Nevertheless, it is instructive to contrast the Yarowsky's system syntactic features with those being described here. Their system identifies a closed set of syntactic feature types first (e.g. verb/obj) and then can only extract those types from the corpus.

3.3 David Fernandez-Amoros

(Férnandez-Amorós, 2004) is also not directly comparable because he uses unsupervised learning. Again though, a comparison can be made be-

Accepted after full paper review

tween his syntactic features and those being described here. He first parsed all the WordNet glosses. He looked for parts of the parse trees that contained WSD target-words and used these subtrees as patterns for that target-word. He also used wildcards in place of pronouns and content words, like the current research. He uses transformations on these sub-tree patterns in a further attempts to increase recall. In spirit, his research and that described here are similar however he was not able to achieve the same amount of automation in both identifying syntactic tree topologies and in generating wildcard features. Also, his base-syntactic patterns were limited to those found in the WordNet glosses for a given target-word.

4 Methods

The SF's, described here, were originally inspired by (Lin, 2000), however, a single one of Lin's SF's is not capable of capturing all the different topologies of subtrees involving the target word. Lin chose a limited yet easy to calculate set of syntactic features that involve the target word. Specifically, he extracted features that always started with the target word and included all the dependency links and words that would be touched on the way to any word in the sentence. Because his features never branch but rather are a string of words connected by dependency relationships we call them **linear syntactic features** (LSF's.) In his syntactic features he included the lemma and POS of the words in the feature and the dependency relationships that are on the links in-between the words. We have followed his lead in including this information in the syntactic features.

Figure 1 shows the Connexor dependency parse of a random (and somewhat awkward) sentence from the SENSEVAL-2 data.

An example of a two link feature in this sentence would be one that started at the target word "sense" and then goes to the word "spending" by following the "mod" link and then finally on to "check" by following the "cnt" link.

From Lin's description of his features, even some linear features would not be extracted. For instance, features where the target word is in the middle of a linear path from one word to another in this sentence would not be extracted because his features must start with the target word.

links	ATSF	running total	LSF	running total
1	3	3	3	3
2	6	9	3	6
3	10	19	2	8
4	15	34	1	9
5	21	55	1	10
6	27	82	0	10
7	30	112	0	10
8	26	138	0	10
9	16	154	0	10
10	6	160	0	10
11	1	161	0	10

Table 3: Comparing the number of LSF's vs. ATSF's found in the sentence: "There is no sense spending 10 to check on a 20 debtor." (from the SENSEVAL-2 corpus)

4.1 All-topologies Syntactic Features (ATSF's)

A version of Lin's LSF's was first implemented. Those types of SF's were automatically extracted from the enhanced corpus documents. It was obvious when looking at a dependency parse tree that while many features were identified there were many more potential features that the LSF feature-extractor was not able to catch. Thus, a more all-inclusive class of syntactic feature which we have named **all-topology syntactic features** (ATSF's), was developed.

The major motivating factor behind seeking to extract ATSF's was that it seemed they would be more abundant than LSF's. The basic idea is that **any** subtree of a sentence parse tree with up to and including a maximum number of dependency links could be potentially useful as a feature. Referring to figure 1, one example of a feature that is not a LSF would be one that involves the words **(is, no, sense, spending)**. The links involved in that feature could not be placed in a line. To give an idea of how many more ATSF's there might be compared to LSF's, table 3 shows the feature counts in the example parse tree. The feature count is broken down into groups of features

Accepted after full paper review

that have the same number of dependency links in them. In each of our experiments, a parameter sets the maximum number of links that a feature can have in that experiment. The best performance to date comes from a maximum of three links. Table 3 shows that there are five times as many three link ATSF's as three link LSF's. In fact, in this example sentence, there are no LSF's that have more than five links and this sentence is typical.

4.1.1 Canonical Form and Representation

The same subtree could be represented in many different ways so a canonical form needs to be defined. ATSF's are defined as nested elements were each element has the basic form:

{**[***DRWP***]::***lemma=POS [children]*}

Where *DRWP* stands for *dependency relationship with parent*. Out of the words in a feature, the word that is topmost in the parse tree being represented is placed first in the feature set. The *DRWP* of that top-most word is deleted. The rest of the dependency relationships further down the tree are represented in the children element of the top tree item. The children of a feature or sub-feature are sorted in alphabetical order by first the POS, then DRWP and finally by lemma.

For example, the feature involving the words **(is, no, sense, spending)** is rendered as:

{::be=v{comp::sense=n{det::no=det}{mod::spending=ing}}}

4.1.2 Algorithm for Identifying ATSF's

In (Férnandez-Amorós, 2004), the author called the problem of systematically identifying all syntactic features "challenging" and said that for lack of time he was not able to come up with a solution yet. We also found it challenging but were able to come up with a divide-and-conquer/dynamic programming solution which is presented in outline form here.

The basic idea is to define a recursive function whose job it is to identify all possible parse tree topologies that can be formed with a constant number of links where all topologies must involve a target tree node and may involve any or all of a group of neighbour nodes and their children. Let us call the function **gen-all-topologies**. It returns a list of features of all topologies. Its arguments are:

target-ID The unique identifier of the target node. This would usually be the ID of the token node for a word.

links The function returns only syntactic features with this many links.

neighbour-IDS The neighbours of the target node which can be used to form the features. Notice, this is usually not all of the neighbours of the **target-ID**

Inside **gen-all-topologies** there is a loop that assigns a variable **links-to-first-neighbour** values from zero to the value of the argument **links**. For each iteration in this loop we try different splits of the **links** between the first neighbour in the list and the target node[1] Here are the two sub-recursive calls:

```
feature-list-1 =
gen-all-topologies(
   first(neighbour-IDS),
   links-to-first-neighbour,
   neighbours*(first(neighbour-IDS))

feature-list-2 =
gen-all-topologies(
   target-ID,
   links - links-to-first-neighbour,
   rest(neighbour-IDS))
```

first and **rest** get the first element and the rest of the elements of a list, respectively. **neighbours*** gets all of the neighbours of a node except for the target node. Once these two sub-recursive calls have returned we do a cross product of the two lists meaning that each member of a list must be combined with each of the features on the other list yielding a number features equal to the product of the sizes of the two lists of features. Determing how to combine two features into a bigger feature has a straightforward solution.

gen-all-topologies is called n times, where the **links** arguments ranges from 1 to n which will obtain all features for a sentence with from 1 to n links in the features.

The implementation of **gen-all-topologies** makes use of dynamic programming techniques,

[1]Only on the top-level call is the target node actually the target word for the WSD problem.

Accepted after full paper review

as some of these sub-recursive calls will be called more than once with the same arguments. Therefore, the returned features from each call are saved and simply used again if a call to **gen-all-topologies** with the same arguments is repeated. In one test, 35% of the calls were able to get the results from the dynamic programming results table. In practice, it seems that the feature extraction algorithm is fairly fast, even when extracting features with as many as five links.

4.1.3 Adding Abstraction to Improve Recall

Experiments show that the WSD system using ATSF's outperform the mean F-measure of 44.2% (see the results table 2, second to last row.) The best recall is 50.1%. The syntactic feature-extractor was extended to first extract the same features as before and, in addition, derive additional abstract features where a "*" or wildcard might take the place of a (non-targetword) lemma. It is important to do the bookkeeping that keeps track of how many literal features make up an abstract feature so when it comes time for feature selection we know the count and the conditional probability with which that abstract feature supports given senses. Table 1 and its caption give details of a real example from the training data.

The addition of wildcard features can make an especially noticeable difference when a sense of a word goes from having zero features when wildcards are not used to having one or more features with wildcard use. Table 1 shows such an example.

4.1.4 Minimum Abstraction Support

Table 1 shows an example of an abstract feature that has five literal features mapped to it. However, many abstract features are only spawned by a single literal feature from the training data (single-support abstract features.) At best, such abstraction features do not add new information and at worst they may add noise. Therefore, by default, abstract features with only one supporting literal feature have been removed. This aspect of the system is called over **abstraction protection** (OAP.)

4.1.5 Feature Selection Strategies Based on Abstraction Level

It could be argued, that constructing abstract features comes with the risk of overgeneralization. One way to control this risk is by use of OAP. Another way to control this risk is in the feature selection process. Section 2.3 specifies that one of the conditions that a feature must meet to be selected is that its count must be greater than FC_{min}. With the addition of abstract features, the system now allows for different values of FC_{min} based on the level of abstraction in a feature. If the feature has n wildcards then that feature must have at least a minimum count of $FC[\mathrm{n}]_{min}$ to not be eliminated.

5 Experiments and Results

All results discussed below are listed in table 2.

Abstract Features: The experimental results back the importance of abstraction. The results table is divided into three horizontal sections based on the number of wildcards (0, 1, or 2) used in the experiments. The F-measure of every 2-wildcard experiment is greater than the the F-measure of every 1-wildcard experiment just as the 1-wildcard's are greater than the 0-wildcard's. This seems like strong evidence that abstraction is invaluable tool for increasing both precision and recall.

OAP: The use of OAP is supported experimentally as the system does slightly worse when single-support abstract features are not removed. Experiment number 13 has the exact same parameters as the best performing experiment 14, but in 13 OAP is off while in 14 it is on. Experiment 13 does slightly worse than 14 probably because of the extra noise.

Basing FC_{Min} on abstraction level: Experiments 12 and 14 have all parameters exactly the same except $FC[1]_{min}$ and they come up with different results lending weight to the proposition that such control could be useful. Further experiments need to be run to determine the scope of variation in results as a result of different settings of $FC[\mathrm{n}]_{min}$ for different values of n.

Accepted after full paper review

Best performance: Experiment 14 performed best.

Again, Lin's system is one of the few SENSEVAL systems that only uses syntactic features and thus should be quite comparable with our system. Lin only gave his results in the course-grained scale. The scores in table 2 are all in terms of the fine-grained scale. Therefore, Lin's results are not included in table 2. His most comparable experiment achieved a coarse-grained F-score of 67%. The best coarse-grained F-score, of the system described here, was 61.2%.

6 Conclusion

The Yarowsky et al. WSD system achieved the highest official score with an F-measure of 64.2%. Their system used six types of features and a voting-scheme machine-learner that used five base machine-learners. Given that the system described here is using only a single type of feature, syntactic, and a single type of machine-learner, SVM, coming within 11.1% of the top score is quite respectable.

Lin's system, that used LSF's, performed better then the ATSF's despite our expectations to the contrary. One reason that ATSF's might not have outperformed Lin's features could be because Lin is using a nearest neighbor (NN) learner and Lin may be able to compose many simpler features to build up a similar picture to a fewer number of the more complex ATSF's. If this is the case, then ATSF's would not seem to offer any advantages over LSF's. The fact that Lin's system did significantly better then this system might say something about the use of nearest-neighbor , compared to SVM's. Lin's system built up a case library and thus did not forget any data quirks. This might be important in an area like WSD , where there is not a lot of supervised training data available, at this point.

There are some advantages to the ATSF's representation of the data. If one thinks of a feature as representing properties of the data then ATSF's can represent such properties more compactly. Several of Lin's features might be required to represent the same data property as one ATSF. Especially where it is important for humans to interpret the features culled from the data, the ATSF representation might be more efficient for humans to deal with.

Acknowledgements

The word sense disambiguation architecture was jointly constructed with David Bell. We would like to thank the Capital Markets CRC and the University of Sydney for financial supported and everyone in the Sydney Language Technology Research Group for their support.

References

David F´ernandez-Amor´os. 2004. Wsd based on mutual information and syntactic patterns. In Rada Mihalcea and Phil Edmonds, editors, *Senseval-3: Third International Workshop on the Evaluation of Systems for the Semantic Analysis of Text*, pages 117–120, Barcelona, Spain, July. Association for Computational Linguistics.

Timo Järvinen and Pasi Tapanainen. 1997. A dependency parser for english. Technical Report TR-1, Department of General Linguistics, University of Helsinki, Finland.

David D. Lewis. 1992. An evaluation of phrasal and clustered representations on a text categorization task. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 37–50. ACM Press.

Dekang Lin. 2000. Word sense disambiguation with a similarity-smoothed case library.

Hwee Tou Ng and Hian Beng Lee. 1996. Integrating multiple knowledge sources to disambiguate word sense: An exemplar-based approach. In Arivind Joshi and Martha Palmer, editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 40–47, San Francisco. Morgan Kaufmann Publishers.

Fabrizio Sebastiani. 2002. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47.

David Yarowsky, Silviu Cucerzan, Radu Florian, Charles Schafer, and Richard Wicentowski. 2001. The johns hopkins SENSEVAL2 system descriptions.

D. Yarowsky. 2000. Hierarchical decision lists for word sense disambiguation.

Accepted after full paper review

Feature Type	**ATSF**	**example phrase**	**sense of "blind"**
literal	{::bit=n {attr::blind=a} {mod::of=prep}}	blind bit of	[unassignable]
literal	{::fear=n {attr::blind=a} {mod::of=prep}}	blind fear of	irrational
literal	{::force=n {attr::blind=a} {mod::of=prep}}	blind force of	irrational
literal	{::hatred=n {attr::blind=a} {mod::of=prep}}	blind hatred of	irrational
literal	{::pursuit=n {attr::blind=a} {mod::of=prep}}	blind pursuit of	irrational
abstract	{::*=n {attr::blind=a} {mod::of=prep}}	blind * of	irrational

Table 1: The first five rows above hold literal features from the training data for the word "blind." These 5 literal features did form one abstract feature, shown in the last row. The first example was not observed with the same sense as the other literal features. The occurrence count of the abstract feature that was formed from the literal features is five and since four out of five of the senses of the literal feature, are of the same sense (the sense of "blind" as being *irrational*), the conditional probability that this abstract feature supports that sense is 0.80. Under the feature selection parameter settings of the experiment that had the best performance, the minimum count for a one-wildcard feature was 4 and the conditional probability cut-off was 66%. Therefore, the abstract feature shown above would have been selected.

	Parameters							**Results**		
Experiment#	**links**	***'s**	$FC[\mathbf{n}]_{min}$			CP_{min}	**OAP**	**Precision**	**Recall**	**F-measure**
			0	**1**	**2**					
1	1	0	3	-	-	75	-	0.505	0.482	0.493
2	2	0	3	-	-	75	-	0.516	0.501	0.508
3	3	0	3	-	-	75	-	0.515	0.500	0.507
4	4	0	3	-	-	75	-	0.515	0.500	0.507
5	4	0	3	-	-	80	-	0.511	0.492	0.501
6	5	0	3	-	-	75	-	0.515	0.500	0.507
7	2	1	3	4	-	75	yes	0.530	0.514	0.522
8	3	1	3	4	-	75	yes	0.527	0.511	0.519
9	4	1	3	4	-	75	yes	0.528	0.512	0.520
10	5	1	3	4	-	75	yes	0.528	0.512	0.520
11	3	2	3	4	4	51	yes	0.533	0.517	0.525
12	3	2	3	3	4	66	yes	0.538	0.521	0.529
13	3	2	3	4	4	66	no	0.539	0.522	0.530
14	3	2	3	4	4	66	yes	0.539	0.523	**0.531**
15	3	2	3	4	4	75	yes	0.536	0.520	0.528
16	4	2	4	4	4	75	yes	0.533	0.517	0.525
17	4	2	3	4	4	75	yes	0.534	0.518	0.526
18	5	2	3	4	4	75	yes	0.534	0.518	0.526
SENSEVAL-2 competition baseline:								0.476	0.476	0.476
SENSEVAL-2 competition mean:								0.459	0.425	0.442
SENSEVAL-2 competition best:								0.642	0.642	0.642

Table 2: Experimental results

Accepted after full paper review