Web-Scale Features for Full-Scale Parsing

Mohit Bansal and **Dan Klein**
Computer Science Division
University of California, Berkeley
{mbansal, klein}@cs.berkeley.edu

Abstract

Counts from large corpora (like the web) can be powerful syntactic cues. Past work has used web counts to help resolve isolated ambiguities, such as binary noun-verb PP attachments and noun compound bracketings. In this work, we first present a method for generating web count features that address the full range of syntactic attachments. These features encode both surface evidence of lexical affinities as well as paraphrase-based cues to syntactic structure. We then integrate our features into full-scale dependency and constituent parsers. We show relative error reductions of 7.0% over the second-order dependency parser of McDonald and Pereira (2006), 9.2% over the constituent parser of Petrov et al. (2006), and 3.4% over a non-local constituent reranker.

1 Introduction

Current state-of-the art syntactic parsers have achieved accuracies in the range of 90% F1 on the Penn Treebank, but a range of errors remain. From a dependency viewpoint, structural errors can be cast as incorrect attachments, even for constituent (phrase-structure) parsers. For example, in the Berkeley parser (Petrov et al., 2006), about 20% of the errors are prepositional phrase attachment errors as in Figure 1, where a preposition-headed (IN) phrase was assigned an incorrect parent in the implied dependency tree. Here, the Berkeley parser (solid blue edges) incorrectly attaches *from debt* to the noun phrase *$ 30 billion* whereas the correct attachment (dashed gold edges) is to the verb *raising*. However, there are a range of error types, as shown in Figure 2. Here, (a) is a non-canonical PP

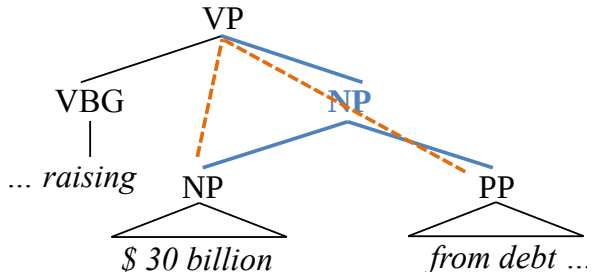

Figure 1: A PP attachment error in the parse output of the Berkeley parser (on Penn Treebank). Guess edges are in solid blue, gold edges are in dashed gold and edges common in guess and gold parses are in black.

attachment ambiguity where *by yesterday afternoon* should attach to *had already*, (b) is an NP-internal ambiguity where *half a* should attach to *dozen* and not to *newspapers*, and (c) is an adverb attachment ambiguity, where *just* should modify *fine* and not the verb *'s*.

Resolving many of these errors requires information that is simply not present in the approximately 1M words on which the parser was trained. One way to access more information is to exploit surface counts from large corpora like the web (Volk, 2001; Lapata and Keller, 2004). For example, the phrase *raising from* is much more frequent on the Web than *$ x billion from*. While this 'affinity' is only a surface correlation, Volk (2001) showed that comparing such counts can often correctly resolve tricky PP attachments. This basic idea has led to a good deal of successful work on disambiguating isolated, binary PP attachments. For example, Nakov and Hearst (2005b) showed that looking for *paraphrase* counts can further improve PP resolution. In this case, the existence of reworded phrases like *raising it from* on the Web also imply a verbal at-

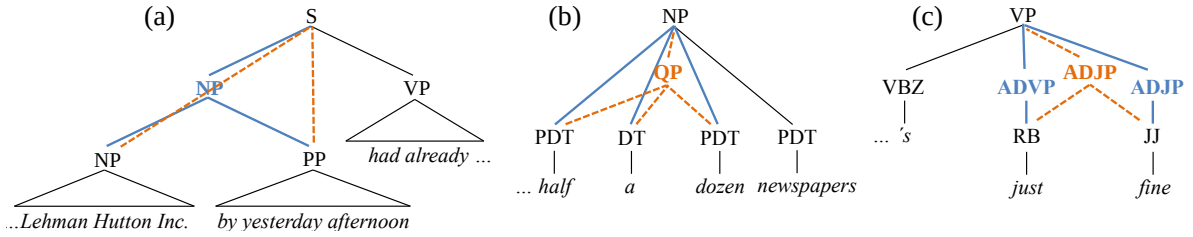

Figure 2: Different kinds of attachment errors in the parse output of the Berkeley parser (on Penn Treebank). Guess edges are in solid blue, gold edges are in dashed gold and edges common in guess and gold parses are in black.

tachment. Still other work has exploited Web counts for other isolated ambiguities, such as NP coordination (Nakov and Hearst, 2005b) and noun-sequence bracketing (Nakov and Hearst, 2005a; Pitler et al., 2010). For example, in (b), *half dozen* is more frequent than *half newspapers*.

In this paper, we show how to apply these ideas to all attachments in full-scale parsing. Doing so requires three main issues to be addressed. First, we show how features can be generated for arbitrary head-argument configurations. Affinity features are relatively straightforward, but paraphrase features, which have been hand-developed in the past, are more complex. Second, we integrate our features into full-scale parsing systems. For dependency parsing, we augment the features in the second-order parser of McDonald and Pereira (2006). For constituent parsing, we rerank the output of the Berkeley parser (Petrov et al., 2006). Third, past systems have usually gotten their counts from web search APIs, which does not scale to quadratically-many attachments in each sentence. Instead, we consider how to efficiently mine the Google n-grams corpus.

Given the success of Web counts for isolated ambiguities, there is relatively little previous research in this direction. The most similar work is Pitler et al. (2010), which use Web-scale n-gram counts for multi-way noun bracketing decisions, though that work considers only sequences of nouns and uses only affinity-based web features. Yates et al. (2006) use Web counts to filter out certain 'semantically bad' parses from extraction candidate sets but are not concerned with distinguishing amongst top parses. In an important contrast, Koo et al. (2008) smooth the sparseness of lexical features in a discriminative dependency parser by using cluster-based word-senses as intermediate abstractions in addition to POS tags (also see Finkel et al. (2008)). Their work also gives a way to tap into corpora beyond the training data, through cluster membership rather than explicit corpus counts and paraphrases.

This work uses a large web-scale corpus (Google n-grams) to compute features for the full parsing task. To show end-to-end effectiveness, we incorporate our features into state-of-the-art dependency and constituent parsers. For the dependency case, we can integrate them into the dynamic programming of a base parser; we use the discriminatively-trained MST dependency parser (McDonald et al., 2005; McDonald and Pereira, 2006). Our first-order web-features give 7.0% relative error reduction over the second-order dependency baseline of McDonald and Pereira (2006). For constituent parsing, we use a reranking framework (Charniak and Johnson, 2005; Collins and Koo, 2005; Collins, 2000) and show 9.2% relative error reduction over the Berkeley parser baseline. In the same framework, we also achieve 3.4% error reduction over the non-local syntactic features used in Huang (2008). Our web-scale features reduce errors for a range of attachment types. Finally, we present an analysis of influential features. We not only reproduce features suggested in previous work but also discover a range of new ones.

2 Web-count Features

Structural errors in the output of state-of-the-art parsers, constituent or dependency, can be viewed as attachment errors, examples of which are Figure 1 and Figure 2.[1] One way to address attachment errors is through features which factor over head-argument

[1]For constituent parsers, there can be minor tree variations which can result in the same set of induced dependencies, but these are rare in comparison.

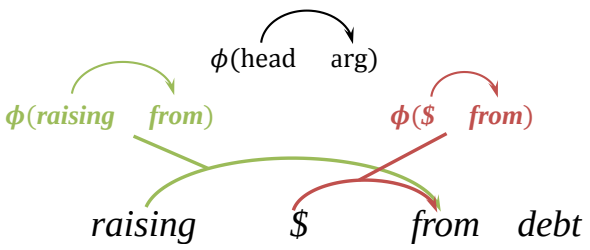

Figure 3: Features factored over head-argument pairs.

pairs, as is standard in the dependency parsing literature (see Figure 3). Here, we discuss which web-count based features $\phi(h, a)$ should fire over a given head-argument pair (we consider the words h and a to be indexed, and so features can be sensitive to their order and distance, as is also standard).

2.1 Affinity Features

Affinity statistics, such as lexical co-occurrence counts from large corpora, have been used previously for resolving individual attachments at least as far back as Lauer (1995) for noun-compound bracketing, and later for PP attachment (Volk, 2001; Lapata and Keller, 2004) and coordination ambiguity (Nakov and Hearst, 2005b). The approach of Lauer (1995), for example, would be to take an ambiguous noun sequence like *hydrogen ion exchange* and compare the various counts (or associated conditional probabilities) of n-grams like *hydrogen ion* and *hydrogen exchange*. The attachment with the greater score is chosen. More recently, Pitler et al. (2010) use web-scale n-grams to compute similar association statistics for longer sequences of nouns.

Our *affinity features* closely follow this basic idea of association statistics. However, because a real parser will not have access to gold-standard knowledge of the competing attachment sites (see Atterer and Schutze (2007)'s criticism of previous work), we must instead compute features for all possible head-argument pairs from our web corpus. Moreover, when there are only two competing attachment options, one can do things like directly compare two count-based heuristics and choose the larger. Integration into a parser requires features to be functions of single attachments, not pairwise comparisons between alternatives. A learning algorithm can then weight features so that they compare appropriately across parses.

We employ a collection of affinity features of varying specificity. The basic feature is the core adjacency count feature ADJ, which fires for all (h, a) pairs. What is specific to a particular (h, a) is the value of the feature, not its identity. For example, in a naive approach, the value of the ADJ feature might be the count of the query issued to the web-corpus – the 2-gram $q = ha$ or $q = ah$ depending on the order of h and a in the sentence. However, it turns out that there are several problems with this approach. First, rather than a single all-purpose feature like ADJ, the utility of such query counts will vary according to aspects like the parts-of-speech of h and a (because a high adjacency count is not equally informative for all kinds of attachments). Hence, we add more refined affinity features that are specific to each pair of POS tags, i.e. ADJ $\wedge$ POS(h) $\wedge$ POS(a). The values of these POS-specific features, however, are still derived from the same queries as before. Second, using real-valued features did not work as well as binning the query-counts (we used $b = floor(\log_r(\text{count})/5) * 5)$ and then firing indicator features ADJ $\wedge$ POS(h) $\wedge$ POS(a) $\wedge$ b for values of b defined by the query count. Adding still more complex features, we conjoin to the preceding features the order of the words h and a as they occur in the sentence, and the (binned) distance between them. For features which mark distances, wildcards ($\star$) are used in the query $q = h \star a$, where the number of wildcards allowed in the query is proportional to the binned distance between h and a in the sentence. Finally, we also include unigram variants of the above features, which are sensitive to only one of the head or argument. For all features used, we add cumulative variants where indicators are fired for all count bins b' up to query count bin b.

2.2 Paraphrase Features

In addition to measuring counts of the words present in the sentence, there exist clever ways in which paraphrases and other accidental indicators can help resolve specific ambiguities, some of which are discussed in Nakov and Hearst (2005a), Nakov and Hearst (2005b). For example, finding attestations of *eat : spaghetti with sauce* suggests a nominal attachment in *Jean ate spaghetti with sauce*. As another example, one clue that the example in Figure 1 is

a verbal attachment is that the proform paraphrase *raising it from* is commonly attested. Similarly, the attestation of *be noun prep* suggests nominal attachment.

These paraphrase features hint at the correct attachment decision by looking for web n-grams with special contexts that reveal syntax superficially. Again, while effective in their isolated disambiguation tasks, past work has been limited by both the range of attachments considered and the need to intuit these special contexts. For instance, frequency of the pattern *The noun prep* suggests noun attachment and of the pattern *verb adverb prep* suggests verb attachment for the preposition in the phrase *verb noun prep*, but these features were not in the manually brainstormed list.

In this work, we automatically generate a large number of paraphrase-style features for arbitrary attachment ambiguities. To induce our list of features, we first mine useful context words. We take each (correct) training dependency relation (h, a) and consider web n-grams of the form cha, hca, and hac. Aggregating over all h and a (of a given POS pair), we determine which context words c are most frequent in each position. For example, for h = *raising* and a = *from* (see Figure 1), we look at web n-grams of the form *raising c from* and see that one of the most frequent values of c on the web turns out to be the word *it*.

Once we have collected context words (for each position p in {BEFORE, MIDDLE, AFTER}), we turn each context word c into a collection of features of the form PARA $\wedge$ POS(h) $\wedge$ POS(a) $\wedge$ c $\wedge$ p $\wedge$ dir, where dir is the linear order of the attachment in the sentence. Note that h and a are head and argument words and so actually occur in the sentence, but c is a context word that generally does not. For such features, the queries that determine their values are then of the form cha, hca, and so on. Continuing the previous example, if the test set has a possible attachment of two words like h = *lowering* and a = *with*, we will fire a feature PARA $\wedge$ VBG $\wedge$ IN $\wedge$ *it* $\wedge$ MIDDLE $\wedge$ $\rightarrow$ with value (indicator bins) set according to the results of the query *lowering it with*. The idea is that if frequent occurrences of *raising it from* indicated a correct attachment between *raising* and *from*, frequent occurrences of *lowering it with* will indicate the correctness of an attachment between *lowering* and *with*. Finally, to handle the cases where no induced context word is helpful, we also construct abstracted versions of these paraphrase features where the context words c are collapsed to their parts-of-speech POS(c), obtained using a unigram-tagger trained on the parser training set. As discussed in Section 5, the top features learned by our learning algorithm duplicate the hand-crafted configurations used in previous work (Nakov and Hearst, 2005b) but also add numerous others, and, of course, apply to many more attachment types.

3 Working with Web n-Grams

Previous approaches have generally used search engines to collect count statistics (Lapata and Keller, 2004; Nakov and Hearst, 2005b; Nakov and Hearst, 2008). Lapata and Keller (2004) uses the number of page hits as the web-count of the queried n-gram (which is problematic according to Kilgarriff (2007)). Nakov and Hearst (2008) post-processes the first 1000 result snippets. One challenge with this approach is that an external search API is now embedded into the parser, raising issues of both speed and daily query limits, especially if all possible attachments trigger queries. Such methods also create a dependence on the quality and post-processing of the search results, limitations of the query process (for instance, search engines can ignore punctuation (Nakov and Hearst, 2005b)).

Rather than working through a search API (or scraper), we use an offline web corpus – the Google n-gram corpus (Brants and Franz, 2006) – which contains English n-grams (n = 1 to 5) and their observed frequency counts, generated from nearly 1 trillion word tokens and 95 billion sentences. This corpus allows us to efficiently access huge amounts of web-derived information in a compressed way, though in the process it limits us to local queries. In particular, we only use counts of n-grams of the form $x \star y$ where the gap length is ≤ 3.

Our system requires the counts from a large collection of these n-gram queries (around 4.5 million). The most basic queries are counts of head-argument pairs in contiguous *h a* and gapped *h* $\star$ *a* configurations.[2] Here, we describe how we process queries

[2]Paraphrase features give situations where we query $\star$ *h a*

of the form (q_1, q_2) with some number of wildcards in between. We first collect all such queries over all trees in preprocessing (so a new test set requires a new query-extraction phase). Next, we exploit a simple but efficient trie-based hashing algorithm to efficiently answer all of them in one pass over the n-grams corpus.

Consider Figure 4, which illustrates the data structure which holds our queries. We first create a trie of the queries in the form of a nested hashmap. The key of the outer hashmap is the first word q_1 of the query. The entry for q_1 points to an inner hashmap whose key is the final word q_2 of the query bigram. The values of the inner map is an array of 4 counts, to accumulate each of $(q_1 q_2)$, $(q_1 \star q_2)$, $(q_1 \star\star q_2)$, and $(q_1 \star\star\star q_2)$, respectively. We use k-grams to collect counts of $(q_1 ... q_2)$ with gap length = $k-2$, i.e. 2-grams to get $count(q_1 q_2)$, 3-grams to get $count(q_1 \star q_2)$ and so on.

With this representation of our collection of queries, we go through the web n-grams (n = 2 to 5) one by one. For an n-gram $w_1 ... w_n$, if the first n-gram word w_1 doesn't occur in the outer hashmap, we move on. If it does match (say $\bar{q}_1 = w_1$), then we look into the inner map for $\bar{q}_1$ and check for the final word w_n. If we have a match, we increment the appropriate query's result value.

In similar ways, we also mine the most frequent words that occur before, in between and after the head and argument query pairs. For example, to collect mid words, we go through the 3-grams $w_1 w_2 w_3$; if w_1 matches $\bar{q}_1$ in the outer hashmap and w_3 occurs in the inner hashmap for $\bar{q}_1$, then we store w_2 and the count of the 3-gram. After the sweep, we sort the context words in decreasing order of count. We also collect unigram counts of the head and argument words by sweeping over the unigrams once.

In this way, our work is linear in the size of the n-gram corpus, but essentially constant in the number of queries. Of course, if the number of queries is expected to be small, such as for a one-off parse of a single sentence, other solutions might be more appropriate; in our case, a large-batch setting, the number of queries was such that this formulation was chosen. Our main experiments (with no parallelization) took 115 minutes to sweep over the 3.8 billion

and *h a* $\star$; these are handled similarly.

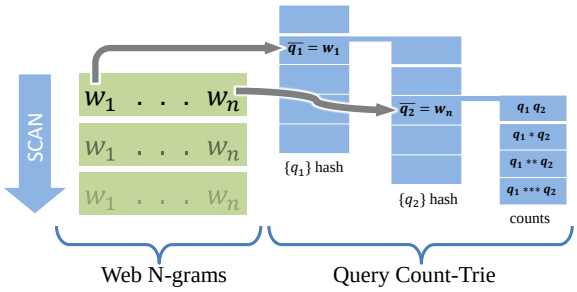

Figure 4: Trie-based nested hashmap for collecting ngram web-counts of queries.

n-grams (n = 1 to 5) to compute the answers to 4.5 million queries, much less than the time required to train the baseline parsers.

4 Parsing Experiments

Our features are designed to be used in full-sentence parsing rather than for limited decisions about isolated ambiguities. We first integrate our features into a dependency parser, where the integration is more natural and pushes all the way into the underlying dynamic program. We then add them to a constituent parser in a reranking approach. We also verify that our features contribute on top of standard reranking features.[3]

4.1 Dependency Parsing

For dependency parsing, we use the discriminatively-trained MSTParser[4], an implementation of first and second order MST parsing models of McDonald et al. (2005) and McDonald and Pereira (2006). We use the standard splits of Penn Treebank into training (sections 2-21), development (section 22) and test (section 23). We used the 'pennconverter'[5] tool to convert Penn trees from constituent format to dependency format. Following Koo et al. (2008), we used the MXPOST tagger (Ratnaparkhi, 1996) trained on the full training data to provide part-of-speech tags for the development

[3] All reported experiments are run on *all* sentences, i.e. without any length limit.

[4] http://sourceforge.net/projects/mstparser

[5] This supersedes 'Penn2Malt' and is available at http://nlp.cs.lth.se/software/treebank_converter. We follow its recommendation to patch WSJ data with NP bracketing by Vadas and Curran (2007).

	Order 2	+ Web features	% Error Redn.
Dev (sec 22)	92.1	**92.7**	7.6%
Test (sec 23)	91.4	**92.0**	7.0%

Table 1: UAS results for English WSJ dependency parsing. Dev is WSJ section 22 (all sentences) and Test is WSJ section 23 (all sentences). The order 2 baseline represents McDonald and Pereira (2006).

	$k = 1$	$k = 2$	$k = 10$	$k = 25$	$k = 50$	$k = 100$
Dev	90.6	92.3	95.1	95.8	96.2	96.5
Test	90.2	91.8	94.7	95.6	96.1	96.4

Table 2: Oracle F1-scores for k-best lists output by Berkeley parser for English WSJ parsing (Dev is section 22 and Test is section 23, all lengths).

and the test set, and we used 10-way jackknifing to generate tags for the training set.

We added our first-order Web-scale features to the MSTParser system to evaluate improvement over the results of McDonald and Pereira (2006).[6] Table 1 shows unlabeled attachments scores (UAS) for their second-order projective parser and the improved numbers resulting from the addition of our Web-scale features. Our first-order web-scale features show significant improvement even over their non-local *second*-order features.[7] Additionally, our web-scale features are at least an order of magnitude fewer in number than even their first-order base features.

4.2 Constituent Parsing

We also evaluate the utility of web-scale features on top of a state-of-the-art constituent parser – the Berkeley parser (Petrov et al., 2006), an unlexicalized phrase-structure parser. Because the underlying parser does not factor along lexical attachments, we instead adopt the discriminative reranking framework, where we generate the top-k candidates from the baseline system and then rerank this k-best list using (generally non-local) features.

Our baseline system is the Berkeley parser, from which we obtain k-best lists for the development set (WSJ section 22) and test set (WSJ section 23) using a grammar trained on all the training data (WSJ sections 2-21).[8] To get k-best lists for the training set, we use 3-fold jackknifing where we train a grammar on 2 folds to get parses for the third fold.[9] The oracle scores of the k-best lists (for different values of k) for the development and test sets are shown in Table 2. Based on these results, we used 50-best lists in our experiments. For discriminative learning, we used the averaged perceptron (Collins, 2002; Huang, 2008).

Our core feature is the log conditional likelihood of the underlying parser.[10] All other features are indicator features. First, we add all the Web-scale features as defined above. These features alone achieve a 9.2% relative error reduction. The affinity and paraphrase features contribute about two-fifths and three-fifths of this improvement, respectively. Next, we rerank with only the features (both local and non-local) from Huang (2008), a simplified merge of Charniak and Johnson (2005) and Collins (2000) (here *configurational*). These features alone achieve around the same improvements over the baseline as our web-scale features, even though they are highly non-local and extensive. Finally, we rerank with both our Web-scale features and the configurational features. When combined, our web-scale features give a further error reduction of 3.4% over the configurational reranker (and a combined error reduction of 12.2%). All results are shown in Table 3.[11]

5 Analysis

Table 4 shows error counts and relative reductions that our web features provide over the 2nd-order dependency baseline. While we do see substantial gains for classic PP (IN) attachment cases, we see equal or greater error reductions for a range of attachment types. Further, Table 5 shows how the to-

[6] Their README specifies 'training-k:5 iters:10 loss-type:nopunc decode-type:proj', which we used for all final experiments; we used the faster 'training-k:1 iters:5' setting for most development experiments.

[7] Work such as Smith and Eisner (2008), Martins et al. (2009), Koo and Collins (2010) has been exploring more non-local features for dependency parsing. It will be interesting to see how these features interact with our web features.

[8] Settings: 6 iterations of split and merge with smoothing.

[9] Default: we ran the Berkeley parser in its default 'fast' mode; the output k-best lists are ordered by max-rule-score.

[10] This is output by the flag -confidence. Note that baseline results with just this feature are slightly worse than 1-best results because the k-best lists are generated by max-rule-score. We report both numbers in Table 3.

[11] We follow Collins (1999) for head rules.

Parsing Model	Dev (sec 22) F1	Dev (sec 22) EX	Test (sec 23) F1	Test (sec 23) EX
Baseline (1-best)	90.6	39.4	90.2	37.3
log $p(t\|w)$	90.4	38.9	89.9	37.3
+ Web features	91.6	42.5	91.1	40.6
+ Configurational features	91.8	43.8	91.1	40.6
+ Web + Configurational	**92.1**	**44.0**	**91.4**	**41.4**

Table 3: Parsing results for reranking 50-best lists of Berkeley parser (Dev is WSJ section 22 and Test is WSJ section 23, all lengths).

Arg Tag	# Attach	Baseline	This Work	% ER
NN	5725	5387	5429	12.4
NNP	4043	3780	3804	9.1
IN	4026	3416	3490	12.1
DT	3511	3424	3429	5.8
NNS	2504	2319	2348	15.7
JJ	2472	2310	2329	11.7
CD	1845	1739	1738	-0.9
VBD	1705	1571	1580	6.7
RB	1308	1097	1100	1.4
CC	1000	855	854	-0.7
VB	983	940	945	11.6
TO	868	761	776	14.0
VBN	850	776	786	13.5
VBZ	705	633	629	-5.6
PRP	612	603	606	33.3

Table 4: Error reduction for attachments of various child (argument) categories. The columns depict the tag, its total attachments as argument, number of correct ones in baseline (McDonald and Pereira, 2006) and this work, and the relative error reduction. Results are for dependency parsing on the dev set for *iters:5,training-k:1*.

tal errors break down by gold head. For example, the 12.1% total error reduction for attachments of an IN argument (which includes PPs as well as complementized SBARs) includes many errors where the gold attachments are to both noun and verb heads. Similarly, for an NN-headed argument, the major corrections are for attachments to noun and verb heads, which includes both object-attachment ambiguities and coordination ambiguities.

We next investigate the features that were given high weight by our learning algorithm (in the constituent parsing case). We first threshold features by a minimum training count of 400 to focus on frequently-firing ones (recall that our features are not bilexical indicators and so are quite a bit more

Arg Tag	% Error Redn for Various Parent Tags
NN	IN: 18, NN: 23, VB: 30, NNP:20, VBN: 33
IN	NN: 11, VBD: 11, NNS: 20, VB:18, VBG: 23
NNS	IN: 9, VBD: 29, VBP: 21, VB:15, CC: 33

Table 5: Error reduction for each type of parent attachment for a given child in Table 4.

POS_{head}	POS_{arg}	Example (*head, arg*)
RB	IN	*back → into*
NN	IN	*review → of*
NN	DT	*The ← rate*
NNP	IN	*Regulation → of*
VB	NN	*limit → access*
VBD	NN	*government ← cleared*
NNP	NNP	*Dean ← Inc*
NN	TO	*ability → to*
JJ	IN	*active → for*
NNS	TO	*reasons → to*
IN	NN	*under → pressure*
NNS	IN	*reports → on*
NN	NNP	*Warner ← studio*
NNS	JJ	*few ← plants*

Table 6: The highest-weight features (thresholded at a count of 400) of the affinity schema. We list only the head and argument POS and the direction (arrow from head to arg). We omit features involving punctuation.

frequent). We then sort them by descending (signed) weight.

Table 6 shows which affinity features received the highest weights, as well as examples of training set attachments for which the feature fired (for concreteness), suppressing both features involving punctuation and the features' count and distance bins. With the standard caveats that interpreting feature weights in isolation is always to be taken for what it is, the first feature (RB→IN) indicates that high counts for an adverb occurring adjacent to a preposition (like *back into the spotlight*) is a useful indicator that the adverb actually modifies that preposition. The second row (NN→IN) indicates that whether a preposition is appropriate to attach to a noun is well captured by how often that preposition follows that noun. The fifth row (VB→NN) indicates that when considering an NP as the object of a verb, it is a good sign if that NP's head frequently occurs immediately following that verb. All of these features essentially state cases where local surface counts are good indi-

POS_{head}	mid-word	POS_{arg}	Example (*head, arg*)
VBN	*this*	IN	*learned, from*
VB	*this*	IN	*publish, in*
VBG	*him*	IN	*using, as*
VBG	*them*	IN	*joining, in*
VBD	*directly*	IN	*converted, into*
VBD	*held*	IN	*was, in*
VBN	*jointly*	IN	*offered, by*
VBZ	*it*	IN	*passes, in*
VBG	*only*	IN	*consisting, of*
VBN	*primarily*	IN	*developed, for*
VB	*us*	IN	*exempt, from*
VBG	*this*	IN	*using, as*
VBD	*more*	IN	*looked, like*
VB	*here*	IN	*stay, for*
VBN	*themselves*	IN	*launched, into*
VBG	*down*	IN	*lying, on*

Table 7: The highest-weight features (thresholded at a count of 400) of the mid-word schema for a verb head and preposition argument (with head on left of argument).

bfr-word	POS_{head}	POS_{arg}	Example (*head, arg*)
second	NN	IN	*season, in*
The	NN	IN	*role, of*
strong	NN	IN	*background, in*
our	NNS	IN	*representatives, in*
any	NNS	IN	*rights, against*
A	NN	IN	*review, of*
:	NNS	IN	*Results, in*
three	NNS	IN	*years, in*
In	NN	IN	*return, for*
no	NN	IN	*argument, about*
current	NN	IN	*head, of*
no	NNS	IN	*plans, for*
public	NN	IN	*appearance, at*
from	NNS	IN	*sales, of*
net	NN	IN	*revenue, of*
,	NNS	IN	*names, of*
you	NN	IN	*leave, in*
have	NN	IN	*time, for*
some	NN	IN	*money, for*
annual	NNS	IN	*reports, on*

Table 8: The highest-weight features (thresholded at a count of 400) of the before-word schema for a noun head and preposition argument (with head on left of argument).

cators of (possibly non-adjacent) attachments.

A subset of paraphrase features, which in the automatically-extracted case don't really correspond to paraphrases at all, are shown in Table 7. Here we show features for verbal heads and IN arguments. The mid-words m which rank highly are those where the occurrence of hma as an n-gram is a good indicator that a attaches to h (m of course does not have to actually occur in the sentence). Interestingly, the top such features capture exactly the intuition from Nakov and Hearst (2005b), namely that if the verb h and the preposition a occur with a pronoun in between, we have evidence that a attaches to h (it certainly can't attach to the pronoun). However, we also see other indicators that the preposition is selected for by the verb, such as adverbs like *directly*.

As another example of known useful features being learned automatically, Table 8 shows the previous-context-word paraphrase features for a noun head and preposition argument (N $\rightarrow$ IN). Nakov and Hearst (2005b) suggested that the attestation of *be* N IN is a good indicator of attachment to the noun (the IN cannot generally attach to forms of auxiliaries). One such feature occurs on this top list – for the context word *have* – and others occur farther down. We also find their surface marker / punctuation cues of *:* and *,* preceding the noun. However, we additionally find other cues, most notably that if the N IN sequence occurs following a capitalized determiner, it tends to indicate a nominal attachment (in the n-gram, the preposition cannot attach leftward to anything else because of the beginning of the sentence).

In Table 9, we see the top-weight paraphrase features that had a conjunction as a middle-word cue. These features essentially say that if two heads w_1 and w_2 occur in the direct coordination n-gram w_1 *and* w_2, then they are good heads to coordinate (coordination unfortunately looks the same as complementation or modification to a basic dependency model). These features are relevant to a range of coordination ambiguities.

Finally, Table 10 depicts the high-weight, high-count general paraphrase-cue features for arbitrary head and argument categories, with those shown in previous tables suppressed. Again, many interpretable features appear. For example, the top entry (*the* JJ NNS) shows that when considering attaching an adjective a to a noun h, it is a good sign if the

POS_{head}	mid-CC	POS_{arg}	Example (*head, arg*)
NNS	*and*	NNS	*purchases, sales*
VB	*and*	VB	*buy, sell*
NN	*and*	NN	*president, officer*
NN	*and*	NNS	*public, media*
VBD	*and*	VBD	*said, added*
VBZ	*and*	VBZ	*makes, distributes*
JJ	*and*	JJ	*deep, lasting*
IN	*and*	IN	*before, during*
VBD	*and*	RB	*named, now*
VBP	*and*	VBP	*offer, need*

Table 9: The highest-weight features (thresholded at a count of 400) of the mid-word schema where the mid-word was a conjunction. For variety, for a given head-argument POS pair, we only list features corresponding to the *and* conjunction and $h \rightarrow a$ direction.

POS_h	POS_a	mid/bfr-word	Example (*h, a*)
NNS	JJ	b = *the*	*other* $\leftarrow$ *things*
NN	NN	m = -	*auto* $\leftarrow$ *maker*
VBD	NN	m = *this*	*adopted* $\rightarrow$ *plan*
NNS	NN	b = *of*	*computer* $\leftarrow$ *products*
NN	DT	m = *current*	*the* $\leftarrow$ *proposal*
VBG	IN	b = *of*	*going* $\rightarrow$ *into*
NNS	IN	m = ”	*clusters* $\rightarrow$ *of*
IN	NN	m = *your*	*In* $\rightarrow$ *review*
TO	VB	b = *used*	*to* $\rightarrow$ *ease*
VBZ	NN	m = *that*	*issue* $\leftarrow$ *has*
IN	NNS	m = *two*	*than* $\rightarrow$ *minutes*
IN	NN	b = *used*	*as* $\rightarrow$ *tool*
IN	VBD	m = *they*	*since* $\rightarrow$ *were*
VB	TO	b = *will*	*fail* $\rightarrow$ *to*

Table 10: The high-weight high-count (thresholded at a count of 2000) general features of the mid and before paraphrase schema (examples show head and arg in linear order with arrow from head to arg).

trigram *the a h* is frequent – in that trigram, the adjective attaches to the noun. The second entry (NN - NN) shows that one noun is a good modifier of another if they frequently appear together hyphenated (another punctuation-based cue mentioned in previous work on noun bracketing, see Nakov and Hearst (2005a)). While they were motivated on separate grounds, these features can also compensate for inapplicability of the affinity features. For example, the third entry (VBD *this* NN) is a case where even if the head (a VBD like *adopted*) actually selects strongly for the argument (a NN like *plan*), the bigram *adopted plan* may not be as frequent as expected, because it requires a determiner in its minimal analogous form *adopted the plan*.

6 Conclusion

Web features are a way to bring evidence from a large unlabeled corpus to bear on hard disambiguation decisions that are not easily resolvable based on limited parser training data. Our approach allows revealing features to be mined for the entire range of attachment types and then aggregated and balanced in a full parsing setting. Our results show that these web features resolve ambiguities not correctly handled by current state-of-the-art systems.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful suggestions. This research is supported by BBN under DARPA contract HR0011-06-C-0022.

References

M. Atterer and H. Schutze. 2007. Prepositional phrase attachment without oracles. *Computational Linguistics*, 33(4):469476.

Thorsten Brants and Alex Franz. 2006. The Google Web 1T 5-gram corpus version 1.1. *LDC2006T13*.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of ACL*.

Michael Collins and Terry Koo. 2005. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–70.

Michael Collins. 1999. Head-Driven Statistical Models for Natural Language Parsing. *Ph.D. thesis, University of Pennsylvania, Philadelphia*.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proceedings of ICML*.

Michael Collins. 2002. Discriminative training methods for Hidden Markov Models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*.

Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. 2008. Efficient, feature-based, conditional random field parsing. In *Proceedings of ACL*.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL*.

Adam Kilgarriff. 2007. Googleology is bad science. *Computational Linguistics*, 33(1).

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of ACL*.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of ACL*.

Mirella Lapata and Frank Keller. 2004. The Web as a baseline: Evaluating the performance of unsupervised Web-based models for a range of NLP tasks. In *Proceedings of HLT-NAACL*.

M. Lauer. 1995. Corpus statistics meet the noun compound: some empirical results. In *Proceedings of ACL*.

André F. T. Martins, Noah A. Smith, and Eric P. Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proceedings of ACL-IJCNLP*.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of EACL*.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of ACL*.

Preslav Nakov and Marti Hearst. 2005a. Search engine statistics beyond the n-gram: Application to noun compound bracketing. In *Proceedings of CoNLL*.

Preslav Nakov and Marti Hearst. 2005b. Using the web as an implicit training set: Application to structural ambiguity resolution. In *Proceedings of EMNLP*.

Preslav Nakov and Marti Hearst. 2008. Solving relational similarity problems using the web as a corpus. In *Proceedings of ACL*.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning Accurate, Compact, and Interpretable Tree Annotation. In *Proceedings of COLING-ACL*.

Emily Pitler, Shane Bergsma, Dekang Lin, , and Kenneth Church. 2010. Using web-scale n-grams to improve base NP parsing performance. In *Proceedings of COLING*.

Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Proceedings of EMNLP*.

David A. Smith and Jason Eisner. 2008. Dependency parsing by belief propagation. In *Proceedings of EMNLP*.

David Vadas and James R. Curran. 2007. Adding noun phrase structure to the Penn Treebank. In *Proceedings of ACL*.

Martin Volk. 2001. Exploiting the WWW as a corpus to resolve PP attachment ambiguities. In *Proceedings of Corpus Linguistics*.

Alexander Yates, Stefan Schoenmackers, and Oren Etzioni. 2006. Detecting parser errors using web-based semantic filters. In *Proceedings of EMNLP*.