Correcting Dependency Annotation Errors

Markus Dickinson
Indiana University
Bloomington, IN, USA
md7@indiana.edu

Abstract

Building on work detecting errors in dependency annotation, we set out to correct local dependency errors. To do this, we outline the properties of annotation errors that make the task challenging and their existence problematic for learning. For the task, we define a feature-based model that explicitly accounts for non-relations between words, and then use ambiguities from one model to constrain a second, more relaxed model. In this way, we are successfully able to correct many errors, in a way which is potentially applicable to dependency parsing more generally.

1 Introduction and Motivation

Annotation error detection has been explored for part-of-speech (POS), syntactic constituency, semantic role, and syntactic dependency annotation (see Boyd et al., 2008, and references therein). Such work is extremely useful, given the harmfulness of annotation errors for training, including the learning of noise (e.g., Hogan, 2007; Habash et al., 2007), and for evaluation (e.g., Padro and Marquez, 1998). But little work has been done to show the full impact of errors, or what types of cases are the most damaging, important since noise can sometimes be overcome (cf. Osborne, 2002). Likewise, it is not clear how to learn from consistently misannotated data; studies often only note the presence of errors or eliminate them from evaluation (e.g., Hogan, 2007), and a previous attempt at correction was limited to POS annotation (Dickinson, 2006). By moving from annotation error detection to error correction, we can more fully elucidate ways in which noise can be overcome and ways it cannot.

We thus explore annotation error correction and its feasibility for dependency annotation, a form of annotation that provides argument relations among words and is useful for training and testing dependency parsers (e.g., Nivre, 2006; McDonald and Pereira, 2006). A recent innovation in dependency parsing, relevant here, is to use the predictions made by one model to refine another (Nivre and McDonald, 2008; Torres Martins et al., 2008). This general notion can be employed here, as different models of the data have different predictions about whch parts are erroneous and can highlight the contributions of different features. Using differences that complement one another, we can begin to sort accurate from inaccurate patterns, by integrating models in such a way as to learn the true patterns and not the errors. Although we focus on dependency annotation, the methods are potentially applicable for different types of annotation, given that they are based on the similar data representations (see sections 2.1 and 3.2).

In order to examine the effects of errors and to refine one model with another's information, we need to isolate the problematic cases. The data representation must therefore be such that it clearly allows for the specific identification of errors between words. Thus, we explore relatively simple models of the data, emphasizing small substructures (see section 3.2). This simple modeling is not always rich enough for full dependency parsing, but different models can reveal conflicting information and are generally useful as part of a larger system. Graph-based models of dependency parsing (e.g., McDonald et al., 2006), for example, rely on breaking parsing down into decisions about smaller substructures, and focusing on pairs of words has been used for domain adaptation (Chen et al., 2008) and in memory-based parsing (Canisius et al., 2006). Exploring annotation error correction in this way can provide insights into more general uses of the annotation, just as previous work on correction for POS annotation (Dickinson, 2006) led to a way to improve POS tagging (Dickinson, 2007).

After describing previous work on error detection and correction in section 2, we outline in section 3 how we model the data, focusing on individual relations between pairs of words. In section 4, we illustrate the difficulties of error correction and show how simple combinations of local features perform poorly. Based on the idea that ambiguities from strict, lexical models can constrain more general POS models, we see improvement in error correction in section 5.

2 Background

2.1 Error detection

We base our method of error correction on a form of error detection for dependency annotation (Boyd et al., 2008). The variation n-gram approach was developed for constituency-based treebanks (Dickinson and Meurers, 2003, 2005) and it detects strings which occur multiple times in the corpus with varying annotation, the so-called *variation nuclei*. For example, the variation nucleus *next Tuesday* occurs three times in the Wall Street Journal portion of the Penn Treebank (Taylor et al., 2003), twice labeled as NP and once as PP (Dickinson and Meurers, 2003).

Every variation detected in the annotation of a nucleus is classified as either an annotation error or as a genuine ambiguity. The basic heuristic for detecting errors requires one word of recurring context on each side of the nucleus. The nucleus with its repeated surrounding context is referred to as a *variation n-gram*. While the original proposal expanded the context as far as possible given the repeated n-gram, using only the immediately surrounding words as context is sufficient for detecting errors with high precision (Boyd et al., 2008). This "shortest" context heuristic receives some support from research on first language acquisition (Mintz, 2006) and unsupervised grammar induction (Klein and Manning, 2002).

The approach can detect both bracketing and labeling errors in constituency annotation, and we already saw a labeling error for *next Tuesday*. As an example of a bracketing error, the variation nucleus *last month* occurs within the NP *its biggest jolt last month* once with the label NP and once as a non-constituent, which in the algorithm is handled through a special label NIL.

The method for detecting annotation errors can be extended to discontinuous constituency annotation (Dickinson and Meurers, 2005), making it applicable to dependency annotation, where words in a relation can be arbitrarily far apart. Specifically, Boyd et al. (2008) adapt the method by treating dependency pairs as variation nuclei, and they include NIL elements for pairs of words not annotated as a relation. The method is successful at detecting annotation errors in corpora for three different languages, with precisions of 93% for Swedish, 60% for Czech, and 48% for German.[1]

2.2 Error correction

Correcting POS annotation errors can be done by applying a POS tagger and altering the input POS tags (Dickinson, 2006). Namely, ambiguity class information (e.g., IN/RB/RP) is added to each corpus position for training, creating complex ambiguity tags, such as <IN/RB/RP,IN>. While this results in successful correction, it is not clear how it applies to annotation which is not positional and uses NIL labels. However, ambiguity class information is relevant when there is a choice between labels; we return to this in section 5.

3 Modeling the data

3.1 The data

For our data set, we use the written portion (sections P and G) of the Swedish Talbanken05 treebank (Nivre et al., 2006), a reconstruction of the Talbanken76 corpus (Einarsson, 1976) The written data of Talbanken05 consists of 11,431 sentences with 197,123 tokens, annotated using 69 types of dependency relations.

This is a small sample, but it matches the data used for error detection, which results in 634 shortest non-fringe variation n-grams, corresponding to 2490 tokens. From a subset of 210 nuclei (917 tokens), hand-evaluation reveals error detection precision to be 93% (195/210), with 274 (of the 917) corpus positions in need of correction (Boyd et al., 2008). This means that 643 positions do not need to be corrected, setting a baseline of 70.1% (643/917) for error correction.[2] Following Dickinson (2006), we train our models on the entire corpus, explicitly including NIL relations (see

[1]The German experiment uses a more relaxed heuristic; precision is likely higher with the shortest context heuristic.

[2]Detection and correction precision are different measurements: for detection, it is the percentage of variation nuclei types where at least one is incorrect; for correction, it is the percentage of corpus tokens with the true (corrected) label.

section 3.2); we train on the original annotation, but not the corrections.

3.2 Individual relations

Annotation error correction involves overcoming noise in the corpus, in order to learn the true patterns underlying the data. This is a slightly different goal from that of general dependency parsing methods, which often integrate a variety of features in making decisions about dependency relations (cf., e.g., Nivre, 2006; McDonald and Pereira, 2006). Instead of maximizing a feature model to improve parsing, we isolate individual pieces of information (e.g., context POS tags), thereby being able to pinpoint, for example, when non-local information is needed for particular types of relations and pointing to cases where pieces of information conflict (cf. also McDonald and Nivre, 2007).

To support this isolation of information, we use dependency pairs as the basic unit of analysis and assign a dependency label to each word pair. Following Boyd et al. (2008), we add *L* or *R* to the label to indicate which word is the head, the left (L) or the right (R). This is tantamount to handling pairs of words as single entries in a "lexicon" and provides a natural way to talk of ambiguities. Breaking the representation down into strings whch receive a label also makes the method applicable to other annotation types (e.g., Dickinson and Meurers, 2005).

A major issue in generating a lexicon is how to handle pairs of words which are not dependencies. We follow Boyd et al. (2008) and generate NIL labels for those pairs of words which also occur as a true labeled relation. In other words, only word pairs which can be relations can also be NILs. For every sentence, then, when we produce feature lists (see section 3.3), we produce them for all word pairs that are related or could potentially be related, but not those which have never been observed as a dependency pair. This selection of NIL items works because there are no unknown words. We use the method in Dickinson and Meurers (2005) to efficiently calculate the NIL tokens.

Focusing on word pairs and not attempting to build a a whole dependency graph allows us to explore the relations between different kinds of features, and it has the potential benefit of not relying on possibly erroneous sister relations. From the perspective of error correction, we cannot assume that information from the other relations in the sentence is reliable.[3] This representation also fits nicely with previous work, both in error detection (see section 2.1) and in dependency parsing (e.g., Canisius et al., 2006; Chen et al., 2008). Most directly, Canisius et al. (2006) integrate such a representation into a memory-based dependency parser, treating each pair individually, with words and POS tags as features.

3.3 Method of learning

We employ memory-based learning (MBL) for correction. MBL stores all corpus instances as vectors of features, and given a new instance, the task of the classifier is to find the most similar cases in memory to deduce the best class. Given the previous discussion of the goals of correcting errors, what seems to be needed is a way to find patterns which do not fully generalize because of noise appearing in very similar cases in the corpus. As Zavrel et al. (1997, p. 137) state about the advantages of MBL:

> Because language-processing tasks typically can only be described as a complex interaction of regularities, subregularities and (families of) exceptions, storing all empirical data as potentially useful in analogical extrapolation works better than extracting the main regularities and forgetting the individual examples (Daelemans, 1996).

By storing all corpus examples, as MBL does, both correct and incorrect data is maintained, allowing us to pinpoint the effect of errors on training. For our experiments, we use TiMBL, version 6.1 (Daelemans et al., 2007), with the default settings. We use the default overlap metric, as this maintains a direct connection to majority-based correction. We could run TiMBL with different values of k, as this should lead to better feature integration. However, this is difficult to explore without development data, and initial experiments with higher k values were not promising (see section 4.2).

To fully correct every error, one could also experiment with a real dependency parser in the future, in order to look beyond the immediate context and to account for interactions between rela-

[3]We use POS information, which is also prone to errors, but on a different level of annotation. Still, this has its problems, as discussed in section 4.1.

tions. The approach to correction pursued here, however, isolates problems for assigning dependency structures, highlighting the effectiveness of different features within the same local domain. Initial experiments with a dependency parser were again not promising (see section 4.2).

3.4 Integrating features

When using features for individual relations, we have different options for integrating them. On the one hand, one can simply additively combine features into a larger vector for training, as described in section 4.2. On the other hand, one can use one set of features to constrain another set, as described in section 5. Pulling apart the features commonly employed in dependency parsing can help indicate the contributions each has on the classification.

This general idea is akin to the notion of classifier stacking, and in the realm of dependency parsing, Nivre and McDonald (2008) successfully stack classifiers to improve parsing by "allow[ing] a model to learn relative to the predictions of the other" (p. 951). The output from one classifier is used as a feature in the next one (see also Torres Martins et al., 2008). Nivre and McDonald (2008) use different kinds of learning paradigms, but the general idea can be carried over to a situation using the same learning mechanism. Instead of focusing on what one learning algorithm informs another about, we ask what one set of more or less informative features can inform another set about, as described in section 5.1.

4 Performing error correction

4.1 Challenges

The task of automatic error correction in some sense seems straightforward, in that there are no unknown words. Furthermore, we are looking at identical recurring words, which should for the most part have consistent annotation. But it is precisely this similarity of local contexts that makes the correction task challenging.

Given that variations contain sets of corpus positions with differing labels, it is tempting to take the error detection output and use a heuristic of "majority rules" for the correction cases, i.e., correct the cases to the majority label. When using only information from the word sequence, this runs into problems quickly, however, in that there are many non-majority labels which are correct. Some of these non-majority cases pattern in uniform ways and are thus more correctable; others are less tractable in being corrected, as they behave in non-uniform and often non-local ways. Exploring the differences will highlight what can and cannot be easily corrected, underscoring the difficulties in training from erroneous annotation.

Uniform non-majority cases The first problem with correction to the majority label is an issue of coverage: a large number of variations are ties between two different labels. Out of 634 shortest non-fringe variation nuclei, 342 (53.94%) have no majority label; for the corresponding 2490 tokens, 749 (30.08%) have no majority tag.

The variation *är väg* ('is way'), for example, appears twice with the same local context shown in (1),[4] once incorrectly labeled as OO-L (other object [head on the left]) and once correctly as SP-L (subjective predicative complement). To distinguish these two, more information is necessary than the exact sequence of words. In this case, for example, looking at the POS categories of the nuclei could potentially lead to accurate correction: AV NN is SP-L 1032 times and OO-L 32 times (AV = the verb "vara" (be), NN = other noun). While some ties might require non-local information, we can see that local—but more general—information could accurately break this tie.

(1)

kärlekens	väg	**är/AV**	en	lång	**väg/NN**	och
love's	way	is	a	long	way	and
...						
...						

Secondly, in a surprising number of cases where there is a majority tag (122 out of the 917 tokens we have a correction for), a non-majority label is actually correct. For the example in (2), the string *institution kvarleva* ('institution remnant') varies between CC-L (sister of first conjunct in binary branching analysis of coordination) and AN-L (apposition).[5] CC-L appears 5 times and AN-L 3 times, but the CC-L cases are incorrect and need to be changed to AN-L.

(2)

en	föråldrad	**institution/NN**	,/IK	en/EN
an	obsolete	institution	,	a
kvarleva/NN	från	1800-talets		
remnant	from	the 1800s		

[4]We put variation nuclei in bold and underline the immediately surrounding context.

[5]Note that CC is a category introduced in the conversion from the 1976 to the 2005 corpus.

Other cases with a non-majority label have other problems. In example (3), for instance, the string *under hägnet* ('under protection') varies in this context between HD-L (other head, 3 cases) and PA-L (complement of preposition, 5 cases), where the PA-L cases need to be corrected to HD-L. Both of these categories are new, so part of the issue here could be in the consistency of the conversion.

(3) fria liv **under/PR hägnet/ID|NN**
free life under the protection
av/ID|PR ett en gång givet löfte
of a one time given promise

The additional problem is that there are other, correlated errors in the analysis, as shown in figure 1. In the case of the correct HD analysis, both *hägnet* and *av* are POS-annotated as ID (part of idiom (multi-word unit)) and are HD dependents of *under*, indicating that the three words make up an idiom. The PA analysis is a non-idiomatic analysis, with *hägnet* as NN.

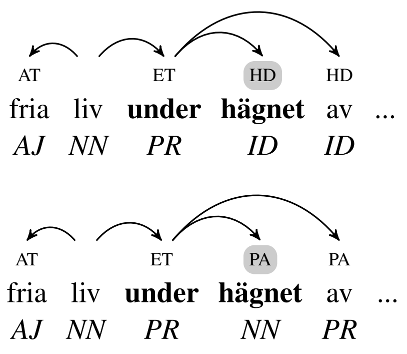
AT
ET
HD
HD
fria liv **under** **hägnet** av ...
AJ NN PR ID ID
AT
ET
PA
PA
fria liv **under** **hägnet** av ...
AJ NN PR NN PR

Figure 1: Erroneous POS & dependency variation

Significantly, *hägnet* only appears 10 times in the corpus, all with *under* as its head, 5 times HD-L and 5 times PA-L. We will not focus explicitly on correcting these types of cases, but the example serves to emphasize the necessity of correction at all levels of annotation.

Non-uniform non-majority cases All of the above cases have in common that whatever change is needed, it needs to be done for all positions in a variation. But this is not sound, as error detection precision is not 100%. Thus, there are variations which clearly must not change.

For example, in (4), there is legitimate variation between PA-L (4a) and HD-L (4b), stemming from the fact that one case is non-idiomatic, and the other is idiomatic, despite having identical local context. In these examples, at least the POS labels are different. Note, though, that in (4) we need to trust the POS labels to overcome the similarity of text, and in (3) we need to distrust them.[6]

(4) a. **Med/PR** andra **ord/NN** en
with other words an
ändamålsenlig ...
appropriate

b. **Med/AB** andra **ord/ID** en form av
with other words a form of
prostitution .
prostitution

Without non-local information, some legitimate variations are virtually irresolvable. Consider (5), for instance: here, we find variation between SS-R (other subject), as in (5a), and FS-R (dummy subject), as in (5b). Crucially, the POS tags are the same, and the context is the same. What differentiates these cases is that *går* has a different set of dependents in the two sentences, as shown in figure 2; to use this information would require us to trust the rest of the dependency structure or to use a dependency parser which accurately derives the structural differences.

(5) a. **Det/PO går/VV** bara inte ihop .
it goes just not together
'It just doesn't add up.'

b. **Det/PO går/VV** bara inte att hålla
it goes just not to hold
ihop ...
together ...

4.2 Using local information

While some variations require non-local information, we have seen that some cases are correctable simply with different kinds of local information (cf. (1)). In this paper, we will not attempt to directly cover non-local cases or cases with POS annotation problems, instead trying to improve the integration of different pieces of local information.

In our experiments, we trained simple models of the original corpus using TiMBL (see section 3.3) and then tested on the same corpus. The models we use include words (W) and/or tags (T) for nucleus and/or context positions, where context here

[6]Rerunning the experiments in the paper by first running a POS tagger showed slight degradations in precision.

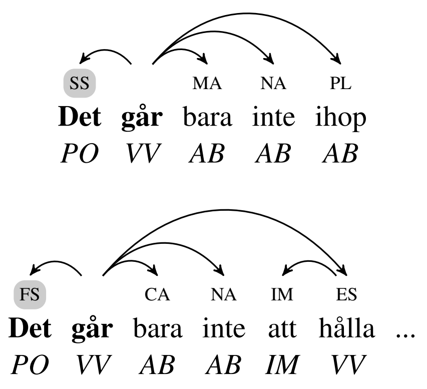

Figure 2: Correct dependency variation

refers only to the immediately surrounding words. These are outlined in table 1, for different models of the nucleus (*Nuc.*) and the context (*Con.*). For instance, the model 6 representation of example (6) (=(1)) consists of all the underlined words and tags.

(6) kärlekens väg/NN **är/AV** en/EN lång/AJ **väg/NN** och/++ man gör oklokt ...

In table 1, we report the precision figures for different models on the 917 positions we have corrections for. We report the correction precision for positions the classifier changed the label of (*Changed*), and the overall correction precision (*Overall*). We also report the precision TiMBL has for the whole corpus, with respect to the original tags (instead of the corrected tags).

#	Nuc.	Con.	TiMBL	Changed	Overall
1	W	-	86.6%	34.0%	62.5%
2	W, T	-	88.1%	35.9%	64.8%
3	W	W	99.8%	50.3%	72.7%
4	W	W, T	99.9%	52.6%	**73.5%**
5	W, T	W	99.9%	50.8%	72.4%
6	W, T	W, T	99.9%	51.2%	72.6%
7	T	-	73.4%	20.1%	49.5%
8	T	T	92.7%	50.2%	73.2%

Table 1: The models tested

We can draw a few conclusions from these results. First, all models using contexual information perform essentially the same—approximately 50% on changed positions and 73% overall. When not generalizing to new data, simply adding features (i.e., words or tags) to the model is less important than the sheer presence of context. This is true even for some higher values of k: model 6, for example, has only 73.2% and 72.1% overall precision for $k = 2$ and $k = 3$, respectively.

Secondly, these results confirm that the task is difficult, even for a corpus with relatively high error detection precision (see section 2.1). Despite high similarity of context (e.g., model 6), the best results are only around 73%, and this is given a baseline (no changes) of 70%. While a more expansive set of features would help, there are other problems here, as the method appears to be overtraining. There is no question that we are learning the "correct" patterns, i.e., 99.9% similarity to the benchmark in the best cases. The problem is that, for error correction, we have to overcome noise in the data. Training and testing with the dependency parser MaltParser (Nivre et al., 2007, default settings) is no better, with 72.1% overall precision (despite a labeled attachment score of 98.3%).

Recall in this light that there are variations for which the non-majority label is the correct one; attempting to get a non-majority label correct using a strict lexical model does not work. To be able not to learn the erroneous patterns requires a more general model. Interestingly, a more general model—e.g., treating the corpus as a sequence of tags (model 8)—results in equally good correction, without being a good overall fit to the corpus data (only 92.7%). This model, too, learns noise, as it misses cases that the lexical models get correct. Simply combining the features does not help (cf. model 6); what we need is to use information from both stricter and looser models in a way that allows general patterns to emerge without overgeneralizing.

5 Model combination

Given the discussion in section 4.1 surrounding examples (1)-(5), it is clear that the information needed for correction is sometimes within the immediate context, although that information is needed, however, is often different. Consider the more general models, 7 and 8, which only use POS tag information. While sometimes this general information is effective, at times it is dramatically incorrect. For example, for (7), the original (incorrect) relation between *finna* and *erbjuda* is CC-L; the model 7 classifier selects OO-L as the correct tag; model 8 selects NIL; and the correct label is +F-L (coordination at main clause level).

(7) försöker **finna/VV** ett lämpligt arbete i
try to find a suitable job in
öppna marknaden eller **erbjuda/VV** andra
open market or to offer other
arbetsmöjligheter .
work possibilities

The original variation for the nucleus *finna erbjuda* ('find offer') is between CC-L and +F-L, but when represented as the POS tags VV VV (other verb), there are 42 possible labels, with OO-L being the most frequent. This allows for too much confusion. If model 7 had more restrictions on the set of allowable tags, it could make a more sensible choice and, in this case, select the correct label.

5.1 Using ambiguity classes

Previous error correction work (Dickinson, 2006) used ambiguity classes for POS annotation, and this is precisely the type of information we need to constrain the label to one which we know is relevant to the current case. Here, we investigate ambiguity class information derived from one model integrated into another model.

There are at least two main ways we can use ambiguity classes in our models. The first is what we have just been describing: an ambiguity class can serve as a constraint on the set of possible outcomes for the system. If the correct label is in the ambiguity class (as it usually is for error correction), this constraining can do no worse than the original model. The other way to use an ambiguity class is as a feature in the model. The success of this approach depends on whether or not each ambiguity class patterns in its own way, i.e., defines a sub-regularity within a feature set.

5.2 Experiment details

We consider two different feature models, those containing only tags (models 7 and 8), and add to these ambiguity classes derived from two other models, those containing only words (models 1 and 3). To correct the labels, we need models which do not strictly adhere to the corpus, and the tag-based models are best at this (see the *TiMBL* results in table 1). The ambiguity classes, however, must be fairly constrained, and the word-based models do this best (cf. example (7)).

5.2.1 Ambiguity classes as constraints

As described in section 5.1, we can use ambiguity classes to constrain the output of a model. Specifically, we take models 7 and 8 and constrain each selected tag to be one which is within the ambiguity class of a lexical model, either 1 or 3. That is, if the TiMBL-determined label is not in the ambiguity class, we select the most likely tag of the ones which are. If no majority label can be decided from this restricted set, we fall back to the TiMBL-selected tag. In (7), for instance, if we use model 7, the TiMBL tag is OO-L, but model 3's ambiguity class restricts this to either CC-L or +F-L. For the representation VV VV, the label CC-L appears 315 times and +F-L 544 times, so +F-L is correctly selected.[7]

The results are given in table 2, which can be compared to the the original models 7 and 8 in table 1, i.e., total precisions of 49.5% and 73.2%, respectively. With these simple constraints, model 8 now outperforms any other model (75.5%), and model 7 begins to approach all the models that use contextual information (68.8%).

#	AC	Changed	Total
7	1	28.5% (114/400)	57.4% (526/917)
7	3	45.9% (138/301)	68.8% (631/917)
8	1	54.0% (142/263)	74.8% (686/917)
8	3	56.7% (144/254)	75.5% (692/917)

Table 2: Constraining TiMBL with ACs

5.2.2 Ambiguity classes as features

Ambiguity classes from one model can also be used as features for another (see section 5.1); in this case, ambiguity class information from lexical models (1 and 3) is used as a feature for POS tag models (7 and 8). The results are given in table 3, where we can see dramatically improved performance from the original models (cf. table 1) and generally improved performance over using ambiguity classes as constraints (cf. table 2).

#	AC	Changed	Total
7	1	33.2% (122/368)	61.9% (568/917)
7	3	50.2% (131/261)	72.1% (661/917)
8	1	59.0% (148/251)	**76.4%** (701/917)
8	3	55.1% (130/236)	73.6% (675/917)

Table 3: TiMBL with ACs as features

If we compare the two results for model 7 (61.9% vs. 72.1%) and then the two results for model 8 (76.4% vs. 73.6%), we observe that the

[7]Even if CC-L had been selected here, the choice is significantly better than OO-L.

better use of ambiguity classes integrates contextual and non-contextual features. Model 7 (POS, no context) with model 3 ambiguity classes (lexical, with context) is better than using ambiguity classes derived from a non-contextual model. For model 8, on the other hand, which uses contextual POS features, using the ambiguity class without context (model 1) does better. In some ways, this combination of model 8 with model 1 ambiguity classes makes the most sense: ambiguity classes are derived from a lexicon, and for dependency annotation, a lexicon can be treated as a set of pairs of words. It is also noteworthy that model 7, despite not using context directly, achieves comparable results to all the previous models using context, once appropriate ambiguity classes are employed.

5.2.3 Both methods

Given that the results of ambiguity classes as features are better than that of constraining, we can now easily combine both methodologies, by constraining the output from section 5.2.2 with the ambiguity class tags. The results are given in table 4; as we can see, all results are a slight improvement over using ambiguity classes as features without constraining the output (table 3). Using only local context, the best model here is 3.2% points better than the best original model, representing an improvement in correction.

#	AC	Changed	Total
7	1	33.5% (123/367)	62.2% (570/917)
7	3	55.8% (139/249)	74.1% (679/917)
8	1	59.6% (149/250)	**76.7%** (703/917)
8	3	57.1% (133/233)	74.3% (681/917)

Table 4: TiMBL w/ ACs as features & constraints

6 Summary and Outlook

After outlining the challenges of error correction, we have shown how to integrate information from different models of dependency annotation in order to perform annotation error correction. By using ambiguity classes from lexical models, both as features and as constraints on the final output, we saw improvements in POS models that were able to overcome noise, without using non-local information.

A first step in further validating these methods is to correct other dependency corpora; this is limited, of course, by the amount of corpora with corrected data available. Secondly, because this work is based on features and using ambiguity classes, it can in principle be applied to other types of annotation, e.g., syntactic constituency annotation and semantic role annotation. In this light, it is interesting to note the connection to annotation error detection: the work here is in some sense an extension of the variation n-gram method. Whether it can be employed as an error detection system on its own requires future work.

Another way in which this work can be extended is to explore how these representations and integration of features can be used for dependency parsing. There are several issues to work out, however, in making insights from this work more general. First, it is not clear that pairs of words are sufficiently general to treat them as a lexicon, when one is parsing new data. Secondly, we have explicit representations for word pairs not annotated as a dependency relation (i.e., NILs), and these are constrained by looking at those which are the same words as real relations. Again, one would have to determine which pairs of words need NIL representations in new data.

Acknowledgements

Thanks to Yvonne Samuelsson for help with the Swedish examples; to Joakim Nivre, Mattias Nilsson, and Eva Pettersson for the evaluation data for Talbanken05; and to the three anonymous reviewers for their insightful comments.

References

Boyd, Adriane, Markus Dickinson and Detmar Meurers (2008). On Detecting Errors in Dependency Treebanks. *Research on Language and Computation* 6(2), 113–137.

Canisius, Sander, Toine Bogers, Antal van den Bosch, Jeroen Geertzen and Erik Tjong Kim Sang (2006). Dependency parsing by inference over high-recall dependency predictions. In *Proceedings of CoNLL-X*. New York.

Chen, Wenliang, Youzheng Wu and Hitoshi Isahara (2008). Learning Reliable Information for Dependency Parsing Adaptation. In *Proceedings of Coling 2008*. Manchester.

Daelemans, Walter (1996). Abstraction Considered Harmful: Lazy Learning of Language Processing. In *Proceedings of the 6th Belgian-Dutch Conference on Machine Learning*. Maastricht, The Netherlands.

Daelemans, Walter, Jakub Zavrel, Ko Van der Sloot and Antal Van den Bosch (2007). *TiMBL: Tilburg Memory Based Learner, version 6.1, Reference Guide*. Tech. rep., ILK Research Group. ILK Research Group Technical Report Series no. 07-07.

Dickinson, Markus (2006). From Detecting Errors to Automatically Correcting Them. In *Proceedings of EACL-06*. Trento, Italy.

Dickinson, Markus (2007). Determining Ambiguity Classes for Part-of-Speech Tagging. In *Proceedings of RANLP-07*. Borovets, Bulgaria.

Dickinson, Markus and W. Detmar Meurers (2003). Detecting Inconsistencies in Treebanks. In *Proceedings of TLT-03*. Växjö, Sweden.

Dickinson, Markus and W. Detmar Meurers (2005). Detecting Errors in Discontinuous Structural Annotation. In *Proceedings of ACL-05*.

Einarsson, Jan (1976). *Talbankens skriftsprøakskonkordans*. Tech. rep., Lund University, Dept. of Scandinavian Languages.

Habash, Nizar, Ryan Gabbard, Owen Rambow, Seth Kulick and Mitch Marcus (2007). Determining Case in Arabic: Learning Complex Linguistic Behavior Requires Complex Linguistic Features. In *Proceedings of EMNLP-07*.

Hogan, Deirdre (2007). Coordinate Noun Phrase Disambiguation in a Generative Parsing Model. In *Proceedings of ACL-07*. Prague.

Klein, Dan and Christopher D. Manning (2002). A Generative Constituent-Context Model for Improved Grammar Induction. In *Proceedings of ACL-02*. Philadelphia, PA.

McDonald, Ryan, Kevin Lerman and Fernando Pereira (2006). Multilingual Dependency Analysis with a Two-Stage Discriminative Parser. In *Proceedings of CoNLL-X*. New York City.

McDonald, Ryan and Joakim Nivre (2007). Characterizing the Errors of Data-Driven Dependency Parsing Models. In *Proceedings of EMNLP-CoNLL-07*. Prague, pp. 122–131.

McDonald, Ryan and Fernando Pereira (2006). Online learning of approximate dependency parsing algorithms. In *Proceedings of EACL-06*. Trento.

Mintz, Toben H. (2006). Finding the verbs: distributional cues to categories available to young learners. In K. Hirsh-Pasek and R. M. Golinkoff (eds.), *Action Meets Word: How Children Learn Verbs*, New York: Oxford University Press, pp. 31–63.

Nivre, Joakim (2006). *Inductive Dependency Parsing*. Berlin: Springer.

Nivre, Joakim, Johan Hall, Jens Nilsson, Atanas Chanev, Gulsen Eryigit, Sandra Kubler, Svetoslav Marinov and Erwin Marsi (2007). Malt-Parser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering* 13(2), 95–135.

Nivre, Joakim and Ryan McDonald (2008). Integrating Graph-Based and Transition-Based Dependency Parsers. In *Proceedings of ACL-08: HLT*. Columbus, OH.

Nivre, Joakim, Jens Nilsson and Johan Hall (2006). Talbanken05: A Swedish Treebank with Phrase Structure and Dependency Annotation. In *Proceedings of LREC-06*. Genoa, Italy.

Osborne, Miles (2002). Shallow Parsing using Noisy and Non-Stationary Training Material. In *JMLR Special Issue on Machine Learning Approaches to Shallow Parsing*, vol. 2, pp. 695–719.

Padro, Lluis and Lluis Marquez (1998). On the Evaluation and Comparison of Taggers: the Effect of Noise in Testing Corpora. In *Proceedings of ACL-COLING-98*. San Francisco, CA.

Taylor, Ann, Mitchell Marcus and Beatrice Santorini (2003). The Penn Treebank: An Overview. In Anne Abeillé (ed.), *Treebanks: Building and using syntactically annotated corpora*, Dordrecht: Kluwer, chap. 1, pp. 5–22.

Torres Martins, André Filipe, Dipanjan Das, Noah A. Smith and Eric P. Xing (2008). Stacking Dependency Parsers. In *Proceedings of EMNLP-08*. Honolulu, Hawaii, pp. 157–166.

Zavrel, Jakub, Walter Daelemans and Jorn Veensta (1997). Resolving PP attachment Ambiguities with Memory-Based Learning. In *Proceedings of CoNLL-97*. Madrid.