# Handling noisy training and testing data

**Don Blaheta**
Department of Computer Science
Brown University
dpb@cs.brown.edu

## Abstract

In the field of empirical natural language processing, researchers constantly deal with large amounts of marked-up data; whether the markup is done by the researcher or someone else, human nature dictates that it will have errors in it. This paper will more fully characterise the problem and discuss whether and when (and how) to correct the errors. The discussion is illustrated with specific examples involving function tagging in the Penn treebank.

## 1 Introduction: Errors

*Nobody's perfect.* A cliché, but in the field of empirical natural language processing, we know it to be true: on a daily basis, we work with large corpora created by, and often marked up by, humans. Fallible as ever, these humans have made errors. For the errors in content, be they spelling, syntax, or something else, we can hope to build more robust systems that will be able to handle them. But what of the errors in markup?

In this paper, we propose a system for cataloguing corpus errors, and discuss some strategies for dealing with them as a research community. Finally, we will present an example (function tagging) that demonstrates the appropriateness of our methods.

## 2 An error taxonomy

### 2.1 Type A: Detectable errors

The easiest errors, which we have dubbed "Type A", are those that can be automatically detected and fixed. These typically come up when there would be multiple reasonable ways of tagging a certain interesting situation: the markup guidelines arbitrarily choose one, and the human annotator unthinkingly uses the other.

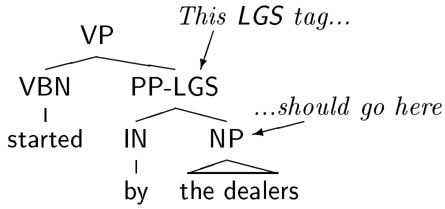


Figure 1: A function tag error of Type A

The canonical example of this sort of thing is the treebank's LGS tag, representing the "logical subject" of a passive construction. It makes a great deal of sense to put this tag on the NP object of the 'by' construction; it makes almost as much sense to tag the PP itself, especially since (given a choice) most other function tags are put there. The treebank guidelines specifically choose the former: "It attaches to the NP object of *by* and not to the PP node itself." (Bies et al., 1995) Nevertheless, in several cases the annotators put the tag on the PP, as shown in Figure 1. We can automatically correct this error by algorithmically removing the LGS tag from any such PP and adding it to the object thereof.

The unifying feature of all Type A errors is that the annotator's *intent* is still clear. In the LGS case, the annotator managed to clearly indicate the presence of a passive construction and its logical subject. Since the transformation from what was marked to what ought to have been marked is straightforward and algorithmic, we can easily apply this correction to all data.

### 2.2 Type B: Fixable errors

Next, we come to the Type B errors, those which are fixable but require human intervention at some point in the process. In theory, this category could include errors that could be found automatically but require a human to fix; this doesn't happen in practice, because if an error is sufficiently systematic that

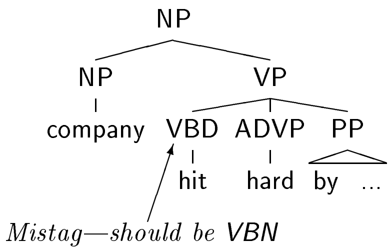


Figure 2: A part-of-speech error of Type $B_1$

an algorithm can detect it and be certain that it is in fact an error, it can usually be corrected with certainty as well. In practice, the instances of this class of error are all cases where the computer can't detect the error for certain. However, for all Type B errors, once detected, the correction that needs to be made is clear, at least to a human observer with access to the annotation guidelines.

Certain Type B errors are moderately easy to find. When annotators misunderstand a complicated markup guideline, they mismark in a somewhat predictable way. While not being totally systematically detectable, an algorithm can leverage these patterns to extract a list of tags or parses that *might* be incorrect, which a human can then examine. Some errors of this type (henceforth "Type $B_1$") include:

- VBD / VBN. Often the past tense form of a verb (VBD) and its past participle (VBN) have the same form, and thus annotators sometimes mistake one for the other, as in Figure 2. Some such cases are not detectable, which is why this is not Type A.[1]

- IN / RB / RP. There are specific tests and guidelines for telling these three things apart, but frequently a preposition (IN) is marked when an adverb (RB) or particle (PRT) would be more appropriate. If an IN is occurring somewhere other than under a PP, it is likely to be a mistag.

Occasionally, an extracted list of maybe-errors will be "perfect", containing only instances that are actually corpus errors. This happens when the pattern is a very good heuristic, though not *necessarily* valid (which is why the errors are Type $B_1$, and not Type A). When filing corrections for these, it is still best to annotate them individually, as the corrections may later be applied to an expanded or modified data set, for which the heuristic would no longer be perfect.

Other fixable errors are pretty much isolated. Within section 24 of the treebank, for instance, we have:

- the word 'long' tagged as an adjective (JJ) when clearly used as a verb (VB)

- the word 'that' parsed into a noun phrase instead of heading a subordinate clause, as in Figure 3

- a phrase headed by 'about', as in 'think about', tagged as a location (LOC)

These isolated errors (resulting, presumably, from a typo or a moment of inattention on the part of the annotator) are not in any way predictable, and can be found essentially only by examining the output of one's algorithm, analysing the "errors", and noticing that the treebank was incorrect, rather than (or in addition to) the algorithm. We will call these Type $B_2$.

### 2.3 Type C: Systematic inconsistency

Sometimes, there is a construction that the markup guidelines writers didn't think about, didn't write up, or weren't clear about. In these cases, annotators are left to rely on their own separate intuitions. This leaves us with markup that is inconsistent and therefore clearly partially in error, but with no obvious correction. There is really very little to be done about these, aside from noting them and perhaps controlling for them in the evaluation.

Some Type C errors in the treebank include:

- 'ago'. English's sole postposition seems to have given annotators some difficulty. Lacking a postposition tag, many tagged such occurrences of 'ago' as a preposition (IN); others used the adverb tag (RB) exclusively.[2] Since some occurrences really are adverbs, this just makes a big mess.

- ADVP-MNR. The MNR tag is meant to be applied to constituents denoting manner or instrument. Some annotators (but not all) seemed to decide that any adverbial phrase (ADVP) headed by an '-ly' word must get a MNR tag, applying it to words like 'suddenly', 'significantly', and 'clearly'.

[1] There is a *subclass* of this error which is Type A: when we find a VBD whose grandparent is a VP headed by a form of 'have', we can deterministically retag it as VBN.

[2] In particular, the annotators of sections 05, 09, 12, 17, 20, and 24 used IN sometimes, while the others tagged all occurrences of 'ago' as adverbs.

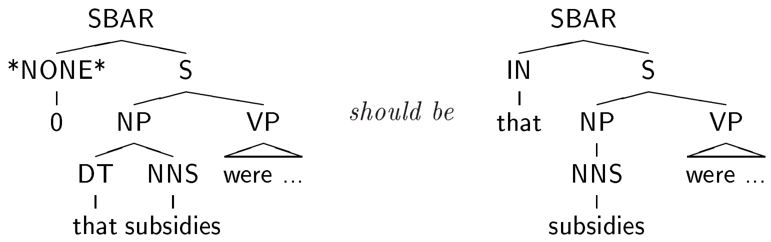


Figure 3: A parse error of Type $B_2$

The hallmark of a Type C error is that even what *ought* to be correct isn't always clear, and as a result, any plan to correct a group of Type C errors will have to first include discussion on what the correct markup guideline should be.

## 3 tsed

In order to effect these changes in some communicable way, we have implemented a program called `tsed`, by analogy with and inspired by the already prevalent `tgrep` search program.[3] It takes a search pattern and a replacement pattern, and after finding the constituent(s) that match the search pattern, modifies them and prints the result. For those already familiar with `tgrep` search syntax, this should be moderately intuitive.

To the basic pattern-matching syntax of `tgrep`, we have added a few extra restriction patterns (for specifying sentence number and head word), as well as a way of marking nodes for later reference in the replacement pattern (by simply wrapping a constituent in square brackets instead of parentheses).

The replacement syntax is somewhat more complicated, because wherever possible we want to be able to construct the new trees by reference to the old tree, in order to preserve modifiers and structure we may not know about when we write the pattern. For full details of the program's abilities, consult the program documentation, but here are the main ones:

- Relabelling. Constituents can be relabelled with no change to any of their modifiers or children.
- Tagging. A tag can be added to or removed from a constituent, without changing any modifiers or children.
- Reference. Constituents in the search pattern can be included by reference in the replacement pattern.
- Construction. New structure can be built by specifying it in the usual S-expression format, e.g. `(NP (NN snork))`. Usually used in combination with Reference patterns.

Along with `tsed` itself, we distribute a Perl program `wsjsed` to process treebank change scripts like the following:

```
{2429#0-b}<<EOF
NP $ [ADJP] > (VP / keep)      (S \0 \1)
NP <<, markets                 - SBJ
EOF
```

This script would make a batch modification to the zeroth sentence of the 29th file in section 24. The batch includes two corrections: the first matches a noun phrase (NP) whose sister is an ADJP and whose parent is a VP headed by the word 'keep'. The matched NP node is replaced by a (created) S node whose children will be that very NP and its sister ADJP. The second correction then finds an NP that ends in the word 'markets' and marks it with the SBJ function tag.

Distributing changes in this form is important for two reasons. First of all, by giving changes in their minimal, most general forms, they are small and easy to transmit, and easy to merge. Perhaps more importantly, since corpora are usually copyrighted and can only be used by paying a fee to the controlling body (usually LDC or ELDA), we need a way to distribute only the changes, in a form that is useless without having bought the original corpus. Scripts for `tsed`, or for `wsjsed`, serve this purpose.

These programs are available from our website.[4]

## 4 When to correct

Now that we have analysed the different types of errors that can occur and how to correct them, we can discuss when and whether to do so.

[3] `tgrep` was written by Richard Pito of the University of Pennsylvania, and comes with the treebank.

[4] `http://www.cs.brown.edu/~dpb/tsed/`

### 4.1 Training

In virtually all empirical NLP work, the training set is going to encompass the vast majority of the data. As such, it is usually impractical for a human (or even a whole lab of humans) to sit down and revise the training. Type A errors can be corrected easily enough, as can some Type $B_1$ errors whose heuristics have a high yield. Purely on grounds of practicality, though, it would be difficult to effect significant correction on a training set of any significant size (such as for the treebank).

Practicality aside, correcting the training set is a bad idea anyway. After expending an enormous effort to perfect one training set, the net result is just one correct training set. While it might make certain things easier and probably will improve the results of most algorithms, those improved results will not be valid for those same algorithms trained on other, non-perfect data; the vast majority of corpora will still be noisy. If a *user* of an algorithm, e.g. an application developer, chooses to perfect a training set to improve the results, that would be helpful, but it is important that researchers report results that are likely to be applicable more generally, to more than one training set. Furthermore, robustness to errors in the training, via smoothing or some other mechanism, will also make an algorithm robust to sparse data, the ever-present spectre that haunts nearly every problem in the field; thus eliminating all errors in the training ought not to have as much of an effect on a strong algorithm.

### 4.2 Testing

Testing data is another story, however. In terms of practicality, it is more feasible, as the test set is usually at least an order of magnitude smaller than the training. More important, though, is the issue of fairness. We need to continue using noisy training data in order to better model real-world use, but it is unfair and unreasonable to have noise in the gold standard[5], which causes an algorithm to be penalised where it is more correct than the human annotation.

As performance on various tasks improves, it becomes ever more important to be able to correct the testing data. A 'mere' 1% improvement on a result of 75% is not impressive, as it represents just a 4% reduction in apparent error, but the same 1% improvement on a result of 95% represents a 20% reduction in apparent error! In the end, a noisy gold standard sets an upper bound of less than 100% on performance, which is if nothing else counterintuitive.

[5] Sometimes more of a pyrite standard, really.

### 4.3 Ethical considerations

Of course, we cannot simply go about changing the corpus willy-nilly. We refer the reader to chapter 7 of David Magerman's thesis (1994) for a cogent discussion of why changing either the training or the testing data is a bad idea. However, we believe that there are now some changed circumstances that warrant a modification of this ethical dictum.

First, we are not allowed to look at testing data. How to correct it, then? An initial reaction might be to "promise" to forget everything seen while correcting the test corpus; this is not reasonable.

Another solution exists, however, which is nearly as good and doesn't raise any ethical questions. Many research groups already use yet another section, separate from both the training and testing, as a sort of development corpus.[6] When developing an algorithm, we must look at some output for debugging, preliminary evaluation, and parameter estimation; so this development section is used for testing until a piece of work is ready for publication, at which point the "true" test set is used. Since we are all reading this development output already anyway, there is no harm in reading it to perform corrections thereon. In publication, then, one can publish the results of an algorithm on both the unaltered and corrected versions of the development section, in addition to the results on the unaltered test section. We can then presume that a corrected version of the test corpus would result in a perceived error reduction comparable to that on the development corpus.

Another problem mentioned in that chapter is of a researcher quietly correcting a test corpus, and publishing results on the modified data (without even noting that it was modified). The solution to this is simple: any results on modified data will need to acknowledge that the data is modified (to be honest), and those modifications need to be made public (to facilitate comparisons by later researchers). For Type A errors fixed by a simple rule, it may be reasonable to publish them directly in the paper that gives the results.[7] For Type B errors, it would be more reasonable to simply publish them on a website, since there are bound to be a large number of them.[8]

[6] In the treebank, this is usually section 24.

[7] The rule we used to fix the LGS problem noted in section 2.1 is as follows:

```
{24*-bg}<<EOF
NP !- LGS > (PP - LGS)    - LGS
PP - LGS                  ! LGS
EOF
```

[8] The 235 corrections made to section 24 are available at http://www.cs.brown.edu/~dpb/tbfix/.

Finally, we would like to note that one of the reasons Magerman was ready to dismiss error in the testing was that the test data had "a consistency rate much higher than the accuracy rate of state-of-the-art parsers". This is no longer true.

### 4.4 Practical considerations

As multiple researchers each begin to impose their own corrections, there are several new issues that will come up. First of all, even should everyone publish their own corrections, and post comparisons to previous researchers' corrected results, there is some danger that a variety of different correction sets will exist concurrently. To some extent this can be mitigated if each researcher posted both their own corrections by themselves, and a full list of all corrections they used (including their own). Even so, from time to time these varied correction sets will need to be collected and merged for the whole community to use.

More difficult to deal with is the fact that, inevitably, there will be disputes as to what is correct. Sometimes these will be between the treebank version and a proposed correction; there will probably also be cases where multiple competing corrections are suggested. There really is no good systematic policy for dealing with this. Disputes will have to be handled on a case-by-case basis, and researchers should probably note any disputes to their corrections that they know of when publishing results, but beyond that it will have to be up to each researcher's personal sense of ethics.

In all cases, a search-and-replace pattern should be made as general as possible (without being too general, of course), so that it interacts well with other modifications. Various researchers are already working with (deterministically) different versions of corpora—with new tags added, or empty nodes removed, or some tags collapsed, for instance, not to mention other corrections already performed—and it would be a bad idea to distribute corrections that are specific to one version of these. When in doubt, one should favour the original form of the corpus, naturally.

The final issue is not a practical problem, but an observation: once a researcher publishes a correction set, any further corrections by other researchers are likely to decrease the results of the first researcher's algorithm, at least somewhat. This is due to the fact that that researcher is usually not going to notice corpus errors when the algorithm errs in the same way. This unfortunate consequence is inevitable, and hopefully will prove minor.

| | |
|---|---|
| Algorithm error | 44% |
| Parse error | 20% |
| Treebank error | 18% |
| Type C error | 13% |
| Dubious | 6% |

Table 1: Analysis of reported errors

## 5 Experimental results

As a sort of case study in the meta-algorithms presented in the previous sections, we will look at the problem of function tagging in the treebank. Blaheta and Charniak (2000) describe an algorithm for marking sentence constituents with function tags such as SBJ (for sentence subjects) and TMP (for temporal phrases). We trained this algorithm on sections 02–21 of the treebank and ran it on section 24 (the development corpus), then analysed the output.

First, we printed out every constituent with a function tag error. We then examined the sentence in which each occurred, and determined whether the error was in the algorithm or in the treebank, or elsewhere, as reported in Table 1. Of the errors we examined, less than half were due solely to an algorithmic failure in the function tagger itself. The next largest category was parse error: this function tagging algorithm requires parsed input, and in these cases, that input was incorrect and led the function tagger astray; had the tagger received the treebank parse, it would have given correct output. In just under a fifth of the reported "errors", the algorithm was correct and the treebank was definitely wrong. The remainder of cases we have identified either as Type C errors—wherein the tagger agreed with many training examples, but the "correct" tag agreed with many others—or at least "dubious", in the cases that weren't common enough to be systematic inconsistencies but where the guidelines did not clearly prefer the treebank tag over the tagger output, or vice versa.

Next, we compiled all the noted treebank errors and their corrections. The most common correction involved simply adding, removing, or changing a function tag to what the algorithm output (with a net effect of improving our score). However, it should be noted that when classifying reported errors, we examined their contexts, and in so doing discovered other sorts of treebank error. Mistags and misparses did not directly affect us; some function tag corrections actually decreased our score. All corrections were applied anyway, in the hope of cleaner evaluations for future researchers. In total, we made 235 corrections, including about 130 simple retags.

| | Grammatical tags | | | Form/function tags | | | Topicalisation tags | | |
|---|---|---|---|---|---|---|---|---|---|
| | P | R | F | P | R | F | P | R | F |
| Treebank | 96.37% | 95.04% | 95.70% | 81.61% | 76.44% | 78.94% | 96.74% | 94.68% | 95.70% |
| Fixed | 97.08% | 95.27% | 96.16% | 85.75% | 77.51% | 81.42% | 97.85% | 95.79% | 96.81% |
| False error | 19.56% | 4.64% | 10.70% | 22.51% | 4.54% | 11.78% | 34.05% | 20.86% | 25.81% |

Table 2: Function tagging results, adjusted for treebank error

Finally, we re-evaluated the algorithm's output on the corrected development corpus. Table 2 shows the resulting improvements.[9] Precision, recall, and F-measure are calculated as in (Blaheta and Charniak, 2000). The false error rate is simply the percent by which the error is reduced; in terms of the performance on the treebank version ($t$) and the fixed version ($f$),

$$\text{False error} = \frac{f - t}{1.0 - t} \times 100\%$$

This is the percentage of the reported errors that are due to treebank error.

The topicalisation result is nice, but since the TPC tag is fairly rare (121 occurrences in section 24), these numbers may not be robust. It is interesting, though, that the false error rate on the two major tag groups is so similar—roughly 20% in precision and 5% in recall for each, leading to 10% in F-measure. First of all, this parallelism strengthens our assertion that the false error rate, though calculated on a development corpus, can be presumed to apply equally to the test corpus, since it indicates that the human missed tag and mistag rates may be roughly constant. Second, the much higher improvement on precision indicates that the majority of treebank error (at least in the realm of function tagging) is due to human annotators forgetting a tag.

## 6 Conclusion

In this paper, we have given a new characterisation of the sorts of noise one finds in empirical NLP, and a roadmap for dealing with it in the future. For many of the problems in the field, the state of the art is now sufficiently advanced that evaluation error is becoming a significant factor in reported results; we show that it is correctable within the constraints of practicality and ethics.

Although our examples all came from the Penn treebank, the taxonomy presented is applicable to any corpus annotation project. As long as there are typographical errors, there will be Type B errors; and unclear or counterintuitive guidelines will forever engender Type A and Type C errors. Furthermore, we expect that the experimental improvement shown in Section 5 will be reflected in projects on other annotated corpora—perhaps to a lesser or greater degree, depending on the difficulty of the annotation task and the prior performance of the computer system.

An effect of the continuing improvement of the state of the art is that researchers will begin (or have begun) concentrating on specific subproblems, and will naturally report results on those subproblems. These subproblems are likely to involve the complicated cases, which are presumably also more subject to annotator error, and are certain to involve smaller test sets, thus increasing the performance effect of each individual misannotation. As the sizes of the subproblems decrease and their complexity increases, the ability to correct the evaluation corpus will become increasingly important.

[9] We did not run corrections on, nor do we show results for, Blaheta and Charniak's "misc" grouping, both because there were very many of them in the reported error list and because they are very frequently wrong in the treebank.

## References

Ann Bies, Mark Ferguson, Karen Katz, and Robert MacIntyre, 1995. *Bracketing Guidelines for Treebank II Style Penn Treebank Project*, January.

Don Blaheta and Eugene Charniak. 2000. Assigning function tags to parsed text. In *Proceedings of the 1st Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 234–240.

David M. Magerman. 1994. *Natural language parsing as statistical pattern recognition*. Ph.D. thesis, Stanford University, February.