A Dependency-based Analysis of Treebank Annotation Errors

Katri Haverinen,[1,3] Filip Ginter,[3] Veronika Laippala,[2] Samuel Kohonen,[3]
Timo Viljanen,[3] Jenna Nyblom[3] and Tapio Salakoski[1,3]

[1]Turku Centre for Computer Science (TUCS)
[2]Department of French studies
[3]Department of Information Technology
20014 University of Turku, Finland
first.last@utu.fi

Abstract

In this paper, we investigate errors in syntax annotation with the Turku Dependency Treebank, a recently published treebank of Finnish, as study material. This treebank uses the Stanford Dependency scheme as its syntax representation, and its published data contains all data created in the full double annotation as well as timing information, both of which are necessary for this study.

First, we examine which syntactic structures are the most error-prone for human annotators, and compare these results to those of a baseline automatic parser. We find that annotation decisions involving highly semantic distinctions, as well as certain morphological ambiguities, are especially difficult for both human annotators and the parser. Second, we train an automatic system that offers for inspection sentences ordered by their likelihood of containing errors. We find that the system achieves a performance that is clearly superior to the random baseline: for instance, by inspecting 10% of all sentences ordered by our system, it is possible to weed out 25% of errors.

1 Introduction

In the field of natural language processing (NLP), human-annotated training data is of crucial importance, regardless of the specific task. The creation of this data requires a large amount of resources, and the data quality affects applications. Thus it is important to ensure that first, the quality of the data is as sufficiently high for the desired purpose, and second, that the amount of expensive manual work is kept to a reasonable amount. Considering the importance of manual annotation for NLP, studies on different aspects of the annotation process are of great interest.

This work strives to examine the difficulty of syntax annotation in the context of Finnish. Our primary objective is to study human annotation and the errors in it, so as to make observations beneficial for future treebanking efforts. As dependency representations have been argued to be a good choice for the purposes of evaluating the correctness of an analysis as well as the general intuitiveness of evaluation measures (see, for instance, the work of Lin (1998) and Clegg and Shepherd (2007)), and as there exists a recently published, dependency-based treebank for Finnish, also this study uses dependency-based evaluation.

Our experiments are twofold. First, we conduct an experiment to find which phenomena and constructions are especially error-prone for human annotators. We also compare human errors to those of an automatic baseline parser. Second, as a practical contribution, we build an automatic system that orders annotated sentences in such a way that those sentences most likely to contain errors are presented for inspection first.

The difficulty of annotation is not a heavily studied subject, but there has been some previous work. For instance, Tomanek and Hahn (2010) have studied the difficulty of annotating named entities by measuring annotation time. They found that cost per annotated unit is not uniform, and thus suggested that this finding could be used to improve models for active learning (Cohn et al., 1996), the goal of which is to select for annotation those examples that are expected to most benefit an existing machine learning system. Tomanek et

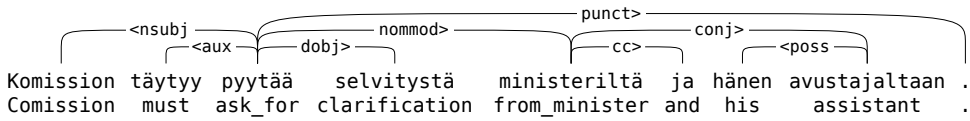

Figure 1: The Stanford Dependency scheme. The sentence can be translated as *The commission must ask for clarification from the minister and his assistant*.

al. (2010) have conducted a follow-up study using eye-tracking data, and found that annotation time and accuracy depend on both the syntactic and semantic complexity of the annotation unit.

Dligach et al. (2010) have studied annotation costs in the context of word sense disambiguation and concluded that for data annotated solely for machine learning purposes, single-annotating a large amount of data appears to be preferable over double-annotating a smaller amount of data. On the level of discourse annotation, Zikánová et al. (2010) have examined typical disagreements between annotators in the context of discourse connectives and their scopes, and on the level of syntax, Dickinson (2010) has studied the possibilities of finding errors in automatic parses in the context of producing *parsebanks*.

However, studies in the context of manual syntax annotation in particular have been rare. One reason for this may be that data which would enable such studies is not generally available. Many treebanks, such as the well-known Penn Treebank (Marcus et al., 1993), are single-annotated, after an initial annotator training period, and thus agreement of the annotators cannot be measured across the whole treebank. Also, timing data for the annotation process is usually not recorded and made available.

2 Data: The Turku Dependency Treebank

In our experiments, we use the first Finnish treebank, the Turku Dependency Treebank (TDT) by Haverinen et al. (2010). TDT is a treebanking effort still in progress, and the new version used in this work is a superset of the recent second release of the treebank and consists of 7,076 sentences (100,073 tokens). Approximately 10%[1] of this data is not used in our experiments, except for parser parameter optimization as described below, and this portion of the data will be held secret for the purpose of possible future parser comparisons and scientific challenges. The remaining 90% of TDT, the portion that was used in this work, consists of 6,375 sentences (89,766 tokens). This data will be made available at the address `http://bionlp.utu.fi/`.

The annotation scheme of the treebank is a slightly modified version of the well-known Stanford Dependency (SD) scheme (de Marneffe and Manning, 2008a; de Marneffe and Manning, 2008b). The annotation in TDT is based on the *basic* variant of the scheme, in which the analyses are trees of dependencies. In total, the scheme version of Haverinen et al. contains 45 different dependency types, whereas the original scheme version contains 54 types. The scheme modifications include both omissions of types where the corresponding phenomenon does not occur in Finnish, and additions where a phenomenon has not been accounted for in the original SD scheme. Figure 1 illustrates the usage of the SD scheme on a Finnish sentence. In this paper, we only discuss those aspects of the SD scheme that are relevant for the current study. For further details of the scheme, we refer the reader to the annotation manual by de Marneffe and Manning (2008a), and for changes made during the annotation process of TDT, the paper by Haverinen et al. (2009).

The Turku Dependency Treebank is exceptional in the sense that the whole treebank has been created using *full double annotation*, where each sentence is first independently annotated by two different annotators, and all differences are then jointly resolved. This results in a single analysis that is called the *merged* annotation. Afterward, the treebank data is subjected to consistency checks, the purpose of which is to ensure that the final release of the treebank, called the *final* annotation, consists of analyses that are updated to conform to the newest annotation decisions. Consistency checks are needed, as some decisions may need revision when the annotation team comes across new examples, and thus the annotation scheme undergoes slight changes.

The treebank also contains the morphologi-

[1] 10% on the level of full text documents

cal analyses of two Finnish morphology tools by Lingsoft Ltd., FinTWOL and FinCG (Koskenniemi, 1983; Karlsson, 1990).[2] Out of these, FinTWOL gives each token all of its possible morphological readings, and FinCG disambiguates between these readings. When unable to fully disambiguate, FinCG can select multiple readings.

In addition to the actual treebank — the *final* annotations — TDT releases contain the *individual* annotations of each annotator, two per sentence, and the *merged* annotations. In addition, the documents include a full edit history with millisecond-resolution timestamps.

In total five different annotators have taken part in the annotation of TDT. The annotators have backgrounds including PhD and Master's students in computer science and linguistics, and also their prior knowledge of linguistics varies substantially.

Our experiments have been conducted against the *merged* annotations, not the *final* annotations of the treebank. This is because we want to avoid penalizing an annotator for a decision that was correct at annotation time but has later become outdated. In addition, the numbers of tokens and sentences in the individually annotated documents and in the final treebank documents do not necessarily match, as possible sentence splitting and tokenization issues are corrected at the consistency fix stage of the annotation process. The only exception to this strategy of comparing *individual* annotations against the *merged* annotation is the experiment detailed in Section 4, where an annotator re-annotated some of the treebank sentences, to estimate the quality of the *final* annotation.

For experiments where a baseline parser was needed, we used the MaltParser[3] (Nivre et al., 2007). Of the treebank documents, 10% were used for parameter optimization and excluded from the experiments. The remaining portion of the treebank was parsed using *ten-fold crossvalidation*, meaning that 90% of the data was used to train a parser and the remaining 10% was then parsed with it, and this process was repeated ten times in order to parse the whole data (disregarding the parameter optimization set) while ensuring that the training and testing data do not overlap.

[2] `http://www.lingsoft.fi`

[3] `http://www.maltparser.org/`

3 Error-prone constructions

As the first part of our study, we have examined the numbers of different errors by the human annotators as well as the baseline parser. In these experiments, all dependencies that remain unmatched between the *merged* annotation (henceforth discussed as *gold standard, GS*) and the *individual* annotation (human or automatic), are considered errors. In our measurements, we have used the standard F_1-score, defined as $F_1 = \frac{2PR}{P+R}$, where *P* stands for precision and *R* stands for recall. Precision, in turn, is the proportion of correctly annotated dependencies out of all dependencies present in the *individual* annotation, and recall is the proportion of correctly annotated dependencies out of all dependencies present in the gold standard. In some experiments, we also use the *labeled attachment score (LAS)*, the proportion of tokens with the correct governor and dependency type.

In addition to the measurements described below, we have also studied the overall annotator performance on the different *sections*[4] of TDT, in order to find how genre affects the agreement. However, the differences found were small, and it appears that the annotator performance on different genres is similar to the overall performance.

3.1 Most difficult dependency types

In our first set of measurements, we examined which dependency types were the most difficult for the human annotators and the baseline parser. This was done by calculating an F_1-score for each of the dependency types, and the types with the lowest F_1-scores were considered the most difficult ones. Only those dependency types that occur in the gold standard at least 150 times were considered, in order to avoid taking into account types that may have extremely low F_1-scores, but which are also very rare, meaning that their being incorrect hardly affects the overall treebank at all. Table 1 shows the ten most difficult types for the annotators, as well as for the baseline parser.[5]

From this table it can be seen that several of the most difficult dependency types for human annotators represent a complement of the verb. The annotation scheme of the treebank contains sev-

[4] Current sections include Wikipedia and Wikinews texts, articles from a university online magazine and from student-magazines, blog entries, EU text and grammar examples.

[5] In this experiment, we have disregarded the small single-annotated proportion of TDT constructed in the very beginning of the annotation process in so called *trial annotations*.

Human					Parser				
type	**P**	**R**	**F**	**freq.**	**type**	**P**	**R**	**F**	**freq.**
iccomp	68.8	70.9	69.8	261 (0.3%)	parataxis	24.2	8.2	12.3	280 (0.4%)
parataxis	69.9	71.6	70.7	280 (0.4%)	advcl	34.9	39.2	36.9	982 (1.3%)
acomp	74.1	70.5	72.2	154 (0.2%)	appos	41.3	38.5	39.8	658 (0.9%)
compar	77.0	71.6	74.2	178 (0.2%)	compar	62.4	35.3	45.2	178 (0.2%)
dep	85.8	69.4	76.7	291 (0.4%)	acomp	53.2	43.5	47.9	154 (0.2%)
advcl	79.2	79.1	79.2	982 (1.3%)	rcmod	49.7	48.2	49.0	897 (1.2%)
auxpass	84.9	75.7	80.0	282 (0.4%)	ccomp	57.2	49.2	52.9	835 (1.1%)
ccomp	82.2	79.4	80.8	835 (1.1%)	iccomp	64.4	47.9	54.9	261 (0.3%)
appos	81.7	80.2	81.0	658 (0.9%)	name	50.7	68.0	58.1	1,925 (2.5%)
gobj	88.6	77.4	82.6	579 (0.8%)	conj	61.7	62.9	62.3	4,041 (5.3%)
overall	89.9	89.1	89.5	76,693 (100%)	**overall**	71.4	70.2	70.8	76,693 (100%)

Table 1: The ten hardest dependency types for the human annotators and the parser. The standard F_1-score was calculated for each dependency type separately, considering only those types that occur in the gold standard at least 150 times. This table presents the ten dependency types with the lowest F_1-scores. For each type is given its precision, recall and F_1-score, and its frequency in the gold standard.

eral different types for these complements, such as clausal complement (*ccomp*) and infinite clausal complement (*iccomp*), as well as a clausal complement with external subject (*xcomp*). Distinguishing these types, especially *ccomp* and *iccomp*, is often challenging, as the distinction depends on only the form of the complement verb. Adjectival complements (*acomp*) likely fall victim to the difficulty of assessing whether a sentence element is a complement. The attachment of sentence elements can also be a source of difficulty. For instance, in word order variations of an example like *The man in the brown coat came into the train* it may be difficult to determine whether *in the brown coat* should modify *man* or *came into the train*. In these cases, the analysis in the treebank follows rules similar to those used in the Prague Dependency Treebank (Hajič, 1998), where in *The man in the brown coat came into the train* there is considered to be a *man in the brown coat*, but in *The man came into the train in a brown coat* the coming into the train happened while wearing a brown coat. These rules, however, are easy to overlook especially in fast-paced annotation. Adverbial clause modifiers (*advcl*), non-complement subordinate clauses, are an example of a phenomenon where the difficulty of annotation may be partly due to attachment issues and partly the difficulty of distinguishing complements and modifiers.

The dependency type *parataxis* is used to mark two different phenomena: direct speech and certain types of implicit clausal coordination, for instance, clauses combined using a semicolon. Especially the latter use can be difficult due to the phenomenon being closely related to coordination. Comparative structures (marked with the

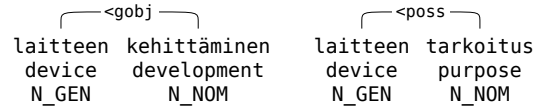

Figure 2: Genitive objects (left) and other genitive modifiers (right). The examples can be translated as *the development of the device* and *the purpose of the device*, respectively. The word *laitteen* (genitive form of *device*) is a genitive attribute of the noun in both examples, but on the left, the noun *kehittäminen* has been derived from the corresponding verb *kehittää* (*to develop*), and the device acts as the object of the developing. Direct derivations of a verb are morphologically marked in the treebank, but other verb-related nouns are not.

type *compar*), in turn, are often elliptical, and it may be unclear what is being compared with what.

Passive auxiliaries (*auxpass*) may suffer from the annotator simply forgetting them, as there is also a more general dependency type for auxiliaries (*aux*). In some cases drawing the line between passives and other subjectless expressions[6] may be difficult. In addition, some passive participles can also be interpreted as adjectives, and thus clauses containing these participles can be read as copular. Another mistake that is easily made out of carelessness is that of mistaking genitive objects (*gobj*) for more general genitive modifiers (*poss*). On the other hand, the distinction of genitive objects and general genitive modifiers is also highly semantic in nature. For an illustration of genitive objects in the SD scheme, see Figure 2.

[6]such as the *zeroth person*, *nollapersoona* (Hakulinen et al., 2004, §1347)

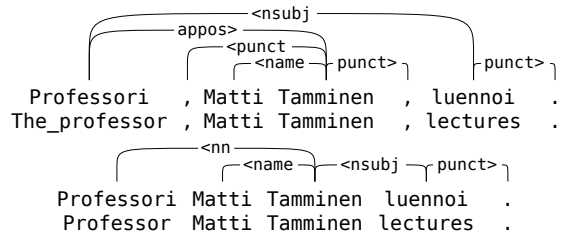

Figure 3: Appositions (top) and appellation modifiers (bottom). The examples can be translated as *The professor, Matti Tamminen, lectures* and *Professor Matti Tamminen lectures*, respectively. The key difference between the examples is that the apposition structure includes commas, while the one with the appellation modifier does not.

Another difficult phenomenon seen in Table 1 is the apposition (*appos*). Appositions are often hard to distinguish from nominal modifiers (*nommod*) due to the semantic requirement that an apposition should have the same referent as its head word. In addition, the annotation scheme distinguishes between appositions and appellation modifiers (marked with the type *nn* alongside with noun compound modifiers), where the distinction usually depends on small details such as the inflection forms of the words involved or the presence or absence of punctuation. Figure 3 illustrates appositions and appellation modifiers in the Finnish-specific version of the SD scheme. Finally, the most generic dependency type *dep* (dependent) is also among the most difficult types. This type is meant for cases where no other, more specific type applies, and in the treebank, it is mostly used for idiomatic two-word expressions.

The most difficult dependency types for the automatic parser are in some respects similar compared to humans, although there are differences as well. Like human annotators, the parser had difficulties with different clausal complements and modifiers (types *ccomp*, *advcl* and *iccomp*), and unlike humans, it also scored low on relative clause modifiers (*rcmod*). Appositions were also clearly difficult for the parser, which is understandable due to the semantic distinctions involved. Another two types that were difficult for the parser but not particularly for humans, were *conj* (coordinated element, see Figure 1) and *name*. With coordinations, it is difficult for a parser to decide which sentence element is coordinated with which, and additionally, for instance an apposition structure may seem coordination-like without any semantic information. The closely related *parataxis* was also especially difficult for the parser. The low F_1-score of the type *name*, which is used for multi-word named entities, has to do, at least partly, with lack of morphological information. Many of the words that are marked with *name* in the training material are unknown to the morphological analyzer, and thus the parser is eager to mark unknown words as multi-word named entities. The overall F_1-score of the parser is 70.8%, compared to the overall human performance of 89.5%.

3.2 Dependency type confusions

Seeing that for many of the most difficult dependency types, the potential explanation seemed to include a possible confusion with another type, we have investigated this matter further. We have calculated the numbers of those errors where the governor is correct, but where the dependency type is wrong, that is, where a dependency type has been replaced by another type. Table 2 shows the five most common type confusions for all five annotators as well as the parser. In total, approximately 32.4% of all erroneous dependencies assigned by annotators only had an incorrect dependency type.

The confusion errors can be divided into several different classes. One error type that can be seen from the table are errors arising from both morphological and semantic closeness of two phenomena. For instance, a common type confusion for nearly all annotators was that of confusing the types *nommod (nominal modifier)* and *dobj (direct object)*. The distinction between nominal modifiers and direct objects is based on both structure and morphology; objects are complements of the verb that can only take certain cases of the Finnish case system (Hakulinen et al., 2004, §925). It is likely that the semantic closeness of objects and certain nominal modifiers misled annotators. In addition, some measures of amount take the same cases as objects and closely resemble them. A nominal modifier like this is called an *object-cased amount adverbial*[7] (Hakulinen et al., 2004, §972).

Also a second confusion seemed to be affected by morphological and semantic closeness. This confusion occured particularly for Annotators 2 and 4, who notably confused subjects and objects on occasion. For other annotators this confusion occured as well, but not as frequently. Subjects

[7] *objektin sijainen määrän adverbiaali (OSMA)*

Annotator 1			Annotator 2			Annotator 3		
GS type	**annot. type**	**fr. (%)**	**GS type**	**annot. type**	**fr. (%)**	**GS type**	**annot. type**	**fr. (%)**
advmod	nommod	5.6	dobj	nommod	6.8	advmod	nommod	6.8
dobj	nommod	3.7	gobj	poss	5.5	dobj	nommod	5.7
auxpass	aux	3.0	nsubj	dobj	4.8	nommod	dobj	4.2
gobj	poss	2.9	advmod	nommod	4.4	advmod	advcl	3.0
nommod	advmod	2.6	nommod	dobj	4.3	nommod	appos	3.0
Annotator 4			**Annotator 5**			**Parser**		
GS type	**annot. type**	**fr. (%)**	**GS type**	**annot. type**	**fr. (%)**	**GS type**	**annot. type**	**fr. (%)**
dobj	nommod	11.5	dobj	nommod	7.1	nommod	dobj	5.5
nommod	dobj	6.0	acomp	nommod	7.1	gobj	poss	5.4
gobj	poss	5.4	partmod	advcl	5.4	partmod	amod	4.8
nsubj	nommod	5.1	appos	conj	5.4	nsubj	dobj	4.1
nsubj	dobj	4.0	nommod	dobj	5.4	dobj	nommod	4.0

Table 2: The five most common dependency type confusions for each annotator and the parser. For each confusion is given the gold standard dependency type (GS type) and the type suggested by the annotator (annot. type), as well as the frequency of the confusion, out of all type confusions by the annotator/parser.

and objects may at first seem like a surprising confusion pair, but actually, due to several reasons these two can rather easily be confused in Finnish, especially when annotating quickly. First, both subject and object use the same cases of the Finnish case system: the nominative, the partitive, and the genitive. Second, Finnish is a free word-order language, and thus the word-order does not necessarily reveal the role of a word. Also, certain verbs that are passive-like in nature, but in fact take a subject and not an object, so called *derived passives* [8] (Hakulinen et al., 2004, §1344), further add to the misleading characters of subjects and objects. In the majority of cases, it is not difficult to decide which of the two analyses is correct in the annotation scheme, once the disagreement is brought into attention, but rather it is the case that annotators are easily misled by the similar properties of these two sentence elements.

A second error type seen in the table is a confusion that is based on a difficult morphological distinction. The distinction between nominal (*nommod*) and adverbial modifiers (*advmod*) was, for several annotators, among the most difficult ones. It is not always clear whether a word should be analyzed as an adverb or rather as an inflected noun, as it is possible for many adverbs to inflect in certain cases, similarly to nouns. For instance, the Finnish word *pääasiassa* (*mainly*) could be analyzed as an adverb, or it could be seen as an inflected form of the noun *pääasia* (*main thing*).

One unexpected type of confusion errors was typical for Annotator 3 in particular. These errors are not due to linguistic similarity, but are simply typographical errors. The annotator has confused adverb modifiers (*advmod*) with adverbial clause modifiers (*advcl*), which are linguistically rather easily distinguishable, but in the annotation software user interface, the shortcut key for *advmod* is capital *V*, while the shortcut key for *advcl* is non-capital *v*. Similarly, this annotator has confused also other dependency types where the respective shortcut keys were capital and non-capital versions of the same letter, but these were not as frequent. Annotator 1 also used the shortcut keys of the annotation user interface and made some typographical errors, although not frequently enough to appear among the five most common type confusions. An example of such an error by Annotator 1 is the confusion of subjects (*nsubj*) and adjectival modifiers (*amod*). The explanation for this otherwise peculiar error is that the shortcut key for *nsubj* is *s* and the one for *amod* is *a*, which are adjacent on a regular Finnish keyboard.

The automatic parser also displayed confusion errors in its output (approximately 16.3% of all erroneous dependencies), involving many of the same semantic distinctions that were difficult for humans, such as genitive objects versus other genitive modifiers and nominal modifiers versus direct objects. Notably the confusion of subjects and objects was also present. Also one morphological distinction was among the most difficult ones for the parser: participal versus adjectival modifiers, where the distinction is, in fact, between participles and adjectives. The same confusion was present for human annotators, but not among the five most common ones. As an example, consider the Finnish word *tunnettu* (*well-known*). It could be a form of the verb *tuntea* (*to know*), but on the other hand, it can be given the comparative and

[8] *johdospassiivi*

superlative forms, which are typical of adjectives. The only type of confusions that did not, naturally, occur for the parser were the typographical errors.

3.3 Correlation of human and parser errors

An interesting question to study is whether the annotator and parser errors correlate with respect to their position in the sentence. Such correlation would indicate that certain structures are in some sense "universally difficult", regardless of whether the annotator is human or machine. This correlation is easy to analyze on the level of tokens: a token is deemed correct if its governor and dependency type are correct. Since we have two independent human annotations for each sentence, we take the union of the individual annotators' errors, thus defining a token as correct only if it was correctly analyzed by both of the annotators. In this experiment, we can only take a sentence into account, if it has both human analyses available. This is the case for a total of 82,244 tokens not used for parser optimization, as a small portion of TDT has, in the very beginning of the annotation process, been constructed in so called *trial annotations*, where a single annotator has annotated the sentence and it has then been jointly inspected (Haverinen et al., 2009).

The results are shown in Table 3. We find that 35.9% (8,677/24,152) of parser errors co-occur with human errors, whereas only a 18.9% (15,548/82,244) co-occurrence, corresponding to the human error-rate, would be expected by chance. Similarly, we find that 55.8% (8,677/15,548) of human errors co-occur with parser errors, whereas only a 29.3% (24,152/82,244) co-occurrence, corresponding to the parser error-rate, would be expected by chance. We can thus conclude that there is a notable positive association between human and parser errors, strongly statistically significant with $p \ll 0.001$ (Pearson's chi-square test on Table 3).

		human	
		error	correct
parser	error	8,677	15,475
	correct	6,871	51,221

Table 3: Token-level correlation between human and parser errors.

4 Correctness of double-annotated data

As part of our investigation on the number of errors by human annotators, we have conducted a small-scale experiment on the correctness of the final treebank annotation. We sampled a random set of 100 sentences from the *final* annotations of the treebank and assigned them to an annotator who had not annotated them previously. This annotator then independently re-annotated these sentences, and the resulting annotation was compared to the previously existing *final* annotation in a regular meeting between all the annotators.

Effectively, we thus gained a set of triple-annotated sentences. The *final* annotation of the corresponding portion of the treebank was compared against these triple-annotated sentences, and thus we gained an estimate of the error-rate of the *final* annotation in the treebank. The LAS for the final treebank annotation against the triple-annotated sample as gold standard was 98.1%, which means that the minimum error-rate of the *final* annotation is 1.9%. This is a lower bound, as it is possible (although unlikely) that further errors go unnoticed because three annotators have given a sentence the same, erroneous analysis.

We thus find that *full double annotation* is an efficient way to produce annotation of high quality. The triple annotation agreement of 98.1% together with the original inter-annotator agreement of 89.6% in LAS (89.5% in $F_1 - score$) implies that approximately 82% ((98.1-89.6)/(100-89.6)) of errors remaining in the single annotated documents can be weeded out using double annotation.

5 Automated recognition of annotation errors

While full double annotation produces high-quality results, as shown in the previous section, it is undoubtedly a resource-intensive approach to annotation. In many cases, particularly when building large treebanks, a compromise between single and double annotation will be necessary. Under such a compromise annotation strategy, only some proportion of sentences would be double annotated or otherwise carefully inspected for errors, while the rest would remain single-annotated. If we were to select these sentences randomly, we would expect to correct the same proportion of annotation errors present in the treebank, assuming that the errors are approximately evenly distributed throughout the treebank. Thus,

for example, by randomly selecting 25% of the sentences for double annotation we would expect to visit 25% of annotation errors present in the treebank. The necessary effort would naturally decrease if we used a strategy that is better than random at selecting sentences which contain annotation errors. In the following, we investigate a machine-learning based method which, given a single-annotated sentence, assigns each token a score that reflects the likelihood of that token being an annotation error, i.e., not having the correct governor and dependency type in the tree.

We approach the problem as a supervised binary classification task where incorrectly annotated tokens are the *positive class* and correctly annotated tokens are the *negative class*. Training data for the classifier can be obtained from the individual annotators' trees, by comparing them against the *merged* trees resulting from the double annotation protocol. If for any token its governor or dependency type do not match those in the merged tree, this token is considered an annotation error (a positive instance), otherwise it is considered correct (a negative instance). Since the average LAS of our annotators is about 90%, the training data contains about 10% positive instances and 90% negative instances, a considerably disbalanced distribution.

The features that represent the tokens in classification are as follows:

Annotator The annotator who produced the tree.

Morphology/POS The lemma, POS, and morphological tags given for all possible morphological readings of the word (prefixed by "cg_" if the reading was selected by the FinCG tagger). The number of possible morphological readings of the word, and the number of readings selected by FinCG.

Dependency Whether the token acts as a dependent, the dependency type, and all morphology/POS features of the governor, given both separately and in combination with the dependency type. The same features are also generated for all dependents of the token under consideration.

We split the available data randomly into a training set (50%), a parameter estimation set (25%), and a test set (25%). The split is performed on the level of documents, so that all instances generated from both annotations of a single document are always placed together in one of the three sets. This prevents any possible leak of information between data used for training and that used for testing. As the classifier, we use the support vector machine (SVM)[9] (Joachims and Yu, 2009) with the radial basis kernel. We select the C and γ parameters by a wide grid search on the parameter estimation set. To account for the pronounced disbalance in the positive and negative class distribution, we use the standard *area under ROC curve (AUC)* performance measure, which is not sensitive to class distribution, and is thus preferred in this case over the usual F_1 or accuracy. We use AUC as both the SVM loss function and the performance criterion to select the best parameter combination.

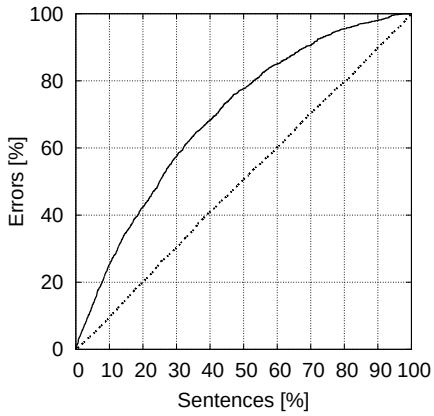

Figure 4: Proportion of annotation errors recovered. The full line represents the machine-learning based ordering of sentences, while the dashed line represents a baseline obtained by ordering the same sentences randomly.

To evaluate the accuracy of the classification, and its practical impact on annotation, we first calculate for each sentence the maximum of the classification scores over all of its tokens and then order the sentences in descending order by this value. The results on the test set are shown in Figure 4. The classifier is notably better than the random baseline: the first 10% of the sentences contain 25% of all annotation errors, and the first 25% of the sentences contain 50% of all annotation errors. These differences are large enough to provide a notable decrease in annotation effort. For instance, the effort to correct 50% of annotation errors is halved: only 25% of all sentences need to be double-annotated, instead of the 50% random baseline. For a treebank of 10,000 sentences, this

[9] Implemented in the *SVM^{perf}* package available at `http://www.joachims.org`

would mean that 2,500 sentences less would need to be double annotated, a notable reduction of effort. Here it should be noted that the classification problem is relatively difficult: we are asking a classifier to recognize human mistakes, at a task at which the humans are 90% correct to start with.

We have also investigated, as an additional feature for the classification, the time spent by the annotator to insert all dependencies for the given token (governor and all dependents), including dependencies that are removed or relabeled in the course of the annotation. Our hypothesis was that those parts of the sentence on which the annotator spent an unusually long time are more difficult to analyze, and thus prone to error as well. This experiment is possible since the treebank contains annotation history data with millisecond-resolution timestamps. However, a substantial part of the treebank is annotated so that of the two individual annotations for each sentence, one is constructed from scratch with all dependencies inserted manually, while the other is constructed on top of the output of a parser, with the annotator correcting the parser output (Haverinen et al., 2010). Complete timing data can naturally be extracted only in the former case, amounting to 119,117 tokens.

We further normalize the annotation timing data to account for the different baseline annotation speeds of the annotators, as well as for the simple fact that the annotation of a token with more dependencies takes longer to complete. We first divide the annotation time of each token by the number of its dependencies in the completed tree and then, for each sentence separately, subtract from each time the mean and divide by standard deviation of the times in that particular sentence. Thus normalized annotation times were then included as a feature in the classification. However, no measurable gain in the performance of the classifier could be observed.

To investigate the correlation between annotation speed and annotation accuracy further, we define a token as "slow" if the time it took to complete is more than one standard deviation above the mean[10] time in the given sentence (we first divide by the number of the token's dependencies, as previously). We then correlate the correctness and speed of annotation in a contingency table (Table 4). We find that incorrectly annotated tokens are overrepresented among "slow" tokens (13.4%), compared to the rest of the tokens (9.6%), as per our original hypothesis. This positive association is strongly statistically significant ($p \ll 0.001$, Pearson's chi-square test on Table 4). While this observation is of some interest, the magnitude of the difference is likely too small for practical applications and annotation times do not seem to provide new information — on top of the features listed above — to a classifier predicting incorrectly annotated tokens.

	correct	incorrect
slow	14,752	2,288
normal	92,290	9,787

Table 4: Correlation between annotation speed and correctness of tokens. Tokens are defined as "slow" if their annotation took longer than one standard deviation above the mean time.

6 Conclusions and future work

In this paper, we have studied the difficulty of syntax annotation in a dependency-based framework, in the context of the Finnish language and the Stanford Dependency (SD) scheme. We have studied the different kinds of errors by the annotators and compared these errors with those of a baseline parser. In addition, we have trained an automatic system that orders single-annotated sentences so that sentences that are most likely to contain errors are offered for inspection first.

We find that there are several different kinds of mistakes that humans make in syntax annotation. In this data, different kinds of clausal complements and modifiers were often erroneously marked, as were comparatives, appositions and structures with parataxis. Nearly one third of the erroneous dependencies marked by annotators were such that only the type of the dependency was wrong. Morphological and semantic closeness of two phenomena seemed to mislead annotators, as for instance adverbial modifiers were often confused with nominal modifiers, and nominal modifiers with direct objects. Annotators also made some mistakes that were not due to any linguistic resemblance, but rather an artifact of annotation user interface shortcut keys that were adjacent or capital and non-capital versions of the same letter. The last type of errors suggests how

[10] Variations of this definition were tested and had no effect on the overall conclusion.

this particular annotation user interface in question could be improved, or how the usability of possible future software could be increased.

We also find that our automatic sentence ranker notably outperforms a random baseline. This means that using this classifier to order single annotated sentences for inspection, it is possible to significantly reduce the amount of double annotation or other careful inspection needed in a compromise setting where full double annotation is not possible or desired. For instance, if one wanted to correct 50% of errors in a treebank, using the proposed method, they could inspect only 25% of all sentences instead of the 50% expected by random selection — a remarkable decrease in effort.

In the future, the knowledge gained in this work could be used for developing new methods helpful for inspecting manual annotations, and for the benefit of large annotation efforts in general. Also studies in for instance the field of active learning, where the goal is to keep the amount of data annotated for machine learning purposes to a minimum, could be conducted.

Acknowledgments

We would like to thank Lingsoft Ltd. for their kind permission to use FinTWOL and FinCG analyses in TDT. We are also grateful to the corpus text authors for the use of their text. This work has been supported by the Academy of Finland.

References

A. B. Clegg and A. Shepherd. 2007. Benchmarking natural-language parsers for biological applications using dependency graphs. *BMC Bioinformatics*, 8(1):24.

D. Cohn, Z. Ghahramani, and M. Jordan. 1996. Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4:129–145.

M. Dickinson. 2010. Detecting errors in automatically-parsed dependency relations. In *Proceedings of ACL'10*, pages 729–738.

D. Dligach, R. Nielsen, and M. Palmer. 2010. To annotate more accurately or to annotate more. In *Proceedings of the Fourth Linguistic Annotation Workshop*, pages 64–72.

J. Hajič. 1998. Building a Syntactically Annotated Corpus: The Prague Dependency Treebank. In *Issues of Valency and Meaning. Studies in Honour of Jarmila Panevová*, pages 106–132. Karolinum, Charles University Press, Prague, Czech Republic.

A. Hakulinen, M. Vilkuna, R. Korhonen, V. Koivisto, T-R. Heinonen, and I. Alho. 2004. *Iso suomen kielioppi / Grammar of Finnish*. Suomalaisen kirjallisuuden seura.

K. Haverinen, F. Ginter, V. Laippala, T. Viljanen, and T. Salakoski. 2009. Dependency annotation of Wikipedia: First steps towards a Finnish treebank. In *Proceedings of TLT8*, pages 95–105.

K. Haverinen, T. Viljanen, V. Laippala, S. Kohonen, F. Ginter, and T. Salakoski. 2010. Treebanking finnish. In *Proceedings of TLT9*, pages 79–90.

T. Joachims and C-N. John Yu. 2009. Sparse kernel SVMs via cutting-plane training. *Machine Learning, Special Issue from ECML PKDD 2009*, 76(2–3):179–193.

F. Karlsson. 1990. Constraint Grammar as a framework for parsing unrestricted text. In *Proceedings of COLING'90*, pages 168–173.

K. Koskenniemi. 1983. Two-level model for morphological analysis. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 683–685.

D. Lin. 1998. A dependency-based method for evaluating broad-coverage parsers. *Natural Language Engineering*, 4(2):97–114.

M. Marcus, M. A. Marcinkiwicz, and B. Santorini. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2):313–330.

M-C. de Marneffe and C. Manning. 2008a. Stanford typed dependencies manual. Technical report, Stanford University, September.

M-C. de Marneffe and C. Manning. 2008b. Stanford typed dependencies representation. In *Proceedings of COLING'08, Workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 1–8.

J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryiğit, S. Kübler, S. Marinov, and E. Marsi. 2007. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

K. Tomanek and U. Hahn. 2010. Annotation time stamps — temporal metadata from the linguistic annotation process. In *Proceedings of LREC'10*, pages 2516–2521.

K. Tomanek, U. Hahn, S. Lohmann, and J. Ziegler. 2010. A cognitive cost model of annotations based on eye-tracking data. In *Proceedings of ACL'10*, pages 1158–1167.

Š. Zikánová, L. Mladová, J. Mírovský, and P. Jínová. 2010. Typical cases of annotators disagreement in discourse annotations in prague dependency treebank. In *Proceedings of LREC'10*, pages 2002–2006.