Part-of-Speech Tagging Using the Brill Method

Maria Larsson and Måns Norelius
Lund Institute of Technology
Lund, Sweden
d00ml@efd.lth.se, d00mno@efd.lth.se

Abstract

Part-of-speech tagging is the process of associating each word in a text with it's part-of-speech category and possibly a set of morphosyntactic features. This information is represented by part-of-speech tags. This paper describes an implementation of a part-of-speech tagger for Swedish based on the Brill method. The basic idea is to apply a set of rules to an initial annotation achieved using a simple algorithm. The rules are found using transformation-based learning applied to a manually tagged training corpus. The paper also addresses the problem of tagging unknown words, i.e. words that don't appear in the training corpus.

1 Introduction

1.1 Part-of-Speech Tagging

The first step in implementing a part-of-speech tagger is to build a lexicon, where the part-of-speech of a word can be found. Unfortunately many words are ambiguous, and each word can therefore have several classifications. As an example, the word "note" can be either a noun or a verb. It is the object of the part-of-speech tagger to resolve these ambiguities, using the context of the word. Another problem is the handling of words that have no entries in the lexicon.

There are basically two approaches to part-of-speech tagging: rule-based tagging and stochastic tagging. This paper describes an implementation using the rule-based approach, where the rules are generated using transformation-based learning.

1.2 Transformation-Based Learning

Transformation-based error-driven learning is a machine learning method typically used for classification problems, where the goal is to assign classifications to a set of samples. An initial classification is produced using a simple algorithm. In each iteration the current classification is compared to the correct classification and transformations are generated to correct the errors. The output of the algorithm is a list of transformations that can be used for automatic classification, together with the initial classifier.

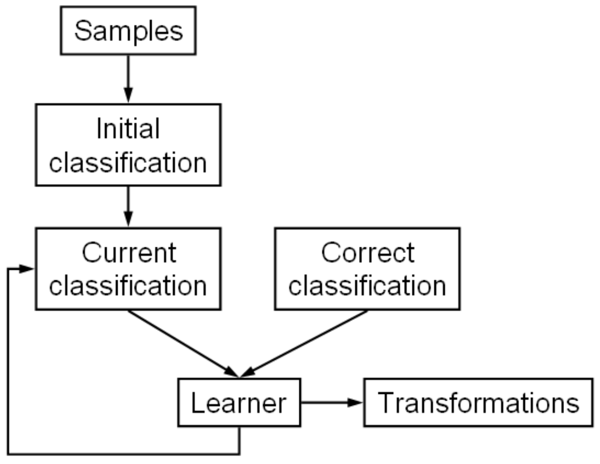

Figure 1: Transformation-based learning

There are two components of a transformation: a rewrite rule and a triggering environment. The rewrite rule says what should be done (e.g. change the class from A to B) and the triggering environment says when it should be done (e.g. if the preceding sample is of class C).

Transformation-based learning is used in many different areas, and has proven to be a very successful method in the field of natural language processing. The algorithm was introduced for POS tagging by Eric Brill in 1995.

2 Brill's Learning Algorithm

The algorithm first assigns every word it's most likely part-of-speech, i.e. the most common tag for that word. This initial annotation is compared to a hand-annotated corpus, and a list of errors is produced. For each error, rules to correct the error are instantiated from a set of rule templates. Each instantiated rule is evaluated by computing it's impact on the whole corpus. The rules are compared by assigning each rule a score, which is the difference between the number of good transformations and the number of bad transformations the rule produces. The rule with the highest score is applied to the text and added to the result list. The transformed corpus is then used to generate a new rule in the next iteration. The algorithm stops when a certain criteria has been fulfilled, e.g. the error rate is below a specified threshold. The algorithm is outlined in figure 2.

```
while (nbr of errrors > threshold)
  for (each error)
    for (each rule r correcting the error)
      good(r) = nbr of good transformations
      bad(r) = nbr of bad transformations
      score(r) = good(r) - bad(r)
  Apply the rule with highest score and
  append it to the rule list
```

Figure 2: Pseudo code for Brill's learning algorithm

2.1 Rule Templates

The learning algorithm instantiates rules given a set of templates. The rules change the tagging of a word based on the tagging of the neighbouring words, and are therefore called contextual rules. The rule templates proposed by Brill are presented below.

Change tag $\boldsymbol{a}$ to tag $\boldsymbol{b}$ when:

1. The preceding (following) word is tagged $\boldsymbol{z}$
2. The word two before (after) is tagged $\boldsymbol{z}$
3. One of the two preceding (following) words is tagged $\boldsymbol{z}$
4. One of the three preceding (following) words is tagged $\boldsymbol{z}$
5. The preceding word is tagged $\boldsymbol{z}$ and the following word is tagged $\boldsymbol{w}$
6. The preceding (following) word is tagged $\boldsymbol{z}$ and the word two before (after) is tagged $\boldsymbol{w}$

where $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{z}$ and $\boldsymbol{w}$ are tag variables.

Instantiating a rule using one of these templates means that the variables are assigned tags corresponding to the specific error to be corrected.

3 Tagging of Unknown Words

A simple way of dealing with unknown words is to assign them the most common part-of-speech and then rely on the contextual rules to correct the errors. There are however more sophisticated methods that can be used to achieve higher accuracy. Brill suggests a special set of rules to apply only to the unknown words, generated in basically the same way as the contextual rules. These rules are applied after the initial tagging and before applying the contexual rules.

3.1 Rule Templates

The rule templates used only for unknown words changes the tag based on properties of the actual word, and not based on the tagging of neighbouring words. The following list describes the templates designed by Brill.

Change the tag of an unknown word from $\boldsymbol{a}$ to $\boldsymbol{b}$ when:

1. Deleting the prefix (suffix) $\boldsymbol{x}$ results in a word
2. The prefix (suffix) of the word is $\boldsymbol{x}$
3. Adding the prefix (suffix) $\boldsymbol{x}$ results in a word
4. The preceding (following) word is $\boldsymbol{w}$
5. The character $\boldsymbol{c}$ appears in the word

where $\boldsymbol{x}$ is a string of length 1 to 4.

4 Implementation

4.1 Corpus

The corpus used is the SUC 1.0 corpus, which was developed as part of a joint research project between the Departments of Linguistics at Stockholm University and Umeå University respectively. The POS tags used in the implementation are identical to the tags used in the SUC project.

4.2 Previous Work

As a project for last year's course a Java implementation of the Brill tagger was written by François Marier and Bengt Sjödin. It contained the basic Brill algorithm and the contextual rule templates, but was too slow for practical use. This program has worked as a foundation for our implementation. Shorter running time and handling of unknown words are the main improvements in this year's implementation.

4.3 Optimization

Because of unexpected low accuracy of the implemented tagger, it was suggested that the program of last year might contain a bug. A set of tests was performed, which resulted in the conclusion that the program worked correctly. The reason for the bad results was instead that the training algorithm was too slow. It was only able to work on small texts, which could not be expected to generate very good results. Our first goal was therefore to optimize the program to be able to run it on a large training corpus.

4.3.1 Problems Identified and Solved

After analyzing the code and extracting profile information, we were able to identify the main reasons for the slow running time of the learner. First we found that a large percentage of the running time was spent on string comparisons. This was because the POS tags were represented by strings, and the algorithm contains many comparisons between tags. The problem was solved by representing the tags by integers, which are faster to compare, and introducing a new class to handle the translation.

Secondly, we discovered that the rule evaluation was done in an inefficient way. Rules were evaluated by actually applying them to the text and then counting the number of errors in the resulting tagging. This meant that the whole corpus had to be copied for every rule evaluation. In our implementation we count the number of good and bad transformations without applying it. Only the best rule is applied and hence there is no need for copying.

Finally, a number of rules were instantiated in linear time with respect to the number of words in the training corpus. This had a great effect on the total time complexity, and increased the running time when using larger texts. In the final implementation all rules are instantiated in constant time.

To further improve the time complexity, all rules are generated in the beginning of the program, and not in each iteration. This means we only consider rules that corrected errors in the original tagging. Consequently, there is a risk that selected rules introduce new errors that no rule is able to correct. However, the output of the algorithm was not affected by this change, which indicates that these missed rules should have been disgarded anyway.

4.3.2 Test Results

In order to demonstrate the process of the optimization, a comparison has been made between the running times of the learning algorithm of the original program and three optimized versions. The versions have an increasing level of optimization and the main changes added to each version are presented below.

1. The tags are represented by integers instead of strings.
2. All rules are instantiated in constant time. All rules to consider are generated once.
3. The rules are evaluated by counting the number of good and bad transformations. Only the best rule is applied, which removes the need for copying.

The programs were tested on a training corpus of 23 000 words extracted from 10 files of the SUC corpus. There are no unknown words, since the dictionary was generated using the whole SUC corpus. The algorithm was stopped after 100 learned rules and the result was the same for all programs.

All tests were performed on a Pentium M 1.4 GHz with 512 MB of RAM and Windows XP Professional operating system.

Program version	Running time
Original	585 min
Optimized 1	121 min
Optimized 2	14 min
Optimized 3	3 min

As seen in the table above, the final optimized version is almost 200 times faster than the original program. Since the time complexity has improved as well, this ratio increases as the training corpus grows. Figure 3 and 4 illustrate the difference in time complexity between the two programs. Running time is plotted as a function of training corpus size.

4.4 Handling of Unknown Words

The second part of the project was to develope a system for handling unknown words. In the original program all words not present in the dictionary were marked as unknown and ignored. A first approach was to found out which tag was the most common and tag every unknown word with this tag. Disgarding

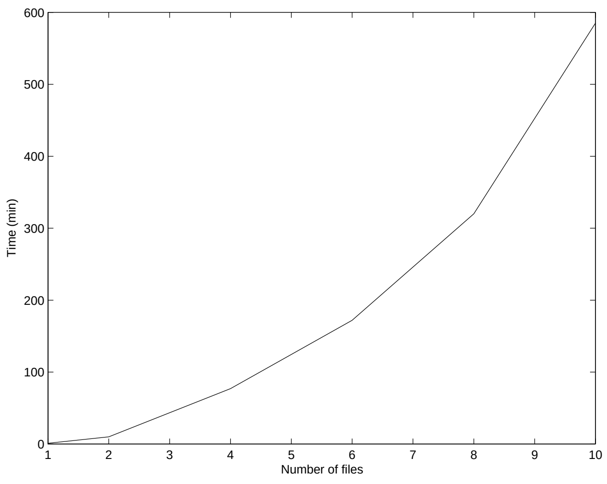

Figure 3: Running time of the original program

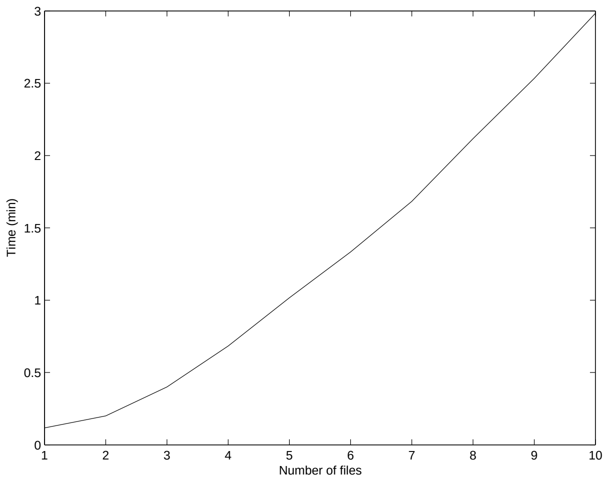

Figure 4: Running time of the optimized program

closed word classes, the most frequent tag was concluded to be “NN UTR SIN IND NOM”, which represents a type of noun (see appendix for details). This simple step correctly classified about 15 % of the unknown words.

4.4.1 Initial Tagging

Although a few steps have been added to the first approach, the initial tagging is still very simple. The unknown words are divided into three groups: numbers, proper nouns and common nouns. If the word contains digits it is tagged as a number. If the it is capitalized and not in the beginning of a sentence it is tagged as a proper noun. All other words are tagged with the most common tag, i.e. common noun.

Also more sophisticated methods were tested to improve the initial tagging. For instance, we tried to conclude some features based on word endings. Although this improved accuracy, we later decided to keep the initial tagger simple. During the developement work of the learner of unknown word rules, it became clear that these types of problems could be better solved by generated rules than by our manually written rules.

However, one extra feature is part of the final implementation. It was found that many of the unknown words are compound words. As a result, we try to divide the unknown word in such a way that the last part forms a known word. If it succeeds, the unknown word is tagged with the default tag for that word. As an example, the unknown word “solstol” is tagged in the same way as the known word “stol”.

4.4.2 Learning Rules for Unknown Words

The handling of unknown words was done in the way suggested by Brill. A program was written for generating rules instantiated from the rule templates described in section 3.1. The learning algorithm is very similar to the algorithm described in figure 2, with the difference that only errors concerning unknown words are considered. Another small difference is the way of counting good and bad transformations of a rule. The good and bad counts should only increase once for each unique unknown word. This prevents a rule that only corrects the tagging of one unknown word from getting a good count just because there are many occurrences of that word. Also, the training corpus must of course contain unknown words and should therefore not be part of the corpus used to create the dictionary.

The rules for unknown words process words instead of tags, and therefore the algorithm for unknown word rules is slower than the algorithm for contextual rules. In addition, tests showed that it is necessary to generate the rules in each iteration in this case.

4.4.3 Tagging

The tagging is done in three steps.

1. Initial tagging
2. Application of unknown word rules
3. Application of contextual rules

4.5 System Overview

The class diagram below presents an overview of the system.

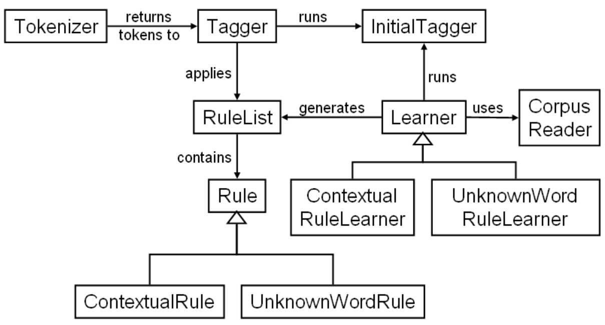

Figure 5: Class diagram

The system can be divided into two programs. First the learning algorithm must be run (one time for each type of rules). After that the rules are generated and can be used in the program performing part-of-speech tagging. **Learner** is the main class of the learning program and **Tagger** is the main class of the tagging program. As can be seen in the class diagram, many classes are used by both programs.

For clarity reasons some classes are omitted in the diagram. These include the classes for specific rule templates along with the WordDictionary and TagDictionary classes used by most other classes. The rule templates for contextual rules are represented by 13 subclasses to the class ContextualRule. In the same way the unknown word rule templates are represented by 9 subclasses to the class UnknownWordRule.

4.6 Class Descriptions

Learner is the main class of the learning program and contains the general learning algorithm. It is an abstract class that requires the subclasses to implement some parts of the algorithm.

ContextualRuleLearner is a subclass of Learner and is responsible for the contextual rule learning.

UnknownWordRuleLearner is also a subclass of Learner and is responsible for learning the unknown word rules.

Rule is the superclass of all rules. It contains the abstract methods instantiate, predicate, evaluate and apply.

ContextualRule is the superclass of all contextual rules.

UnknownWordRule is the superclass of all unknown word rules.

RuleList is a class for maintaining a list of rules.

CorpusReader is responsible for extracting words and tags from the manually annotated corpus.

InitialTagger is responsible for the initial tagging of each word with it's most common tag. Unknown words are tagged according to a few simple rules.

Tagger is the main class of the tagging program. It takes an untagged text as input and produces a tagged text as output.

Tokenizer is used by the Tagger to divide the input text into tokens.

WordDictionary contains all words of the training corpus. It is used for finding the most likely tag for a word, investigating if a word exists and searching for prefixes or suffixes of words.

TagDictionary is responsible for the translation between the string and integer representation of the part-of-speech tags.

4.7 User Instructions

Before running the learner och tagger programs, the dictionaries must be created. This is done by running the two Dictionary classes with the directory containing the training corpus passed as an argument. This creates the files word_dict.dat and tag_dict.dat which are used by the other programs. If the corpus is large, it might be necessary to increase the heap size. An example is shown below. Note that the package name

```
se/lth/cs/BrillTagger
```

has been omitted in the examples to save space.

```
java -Xmx256M WordDictionary dir
java -Xmx256M TagDictionary dir
```

Now the two learning programs can be run. The directory of the training corpus must be passed as an argument. It is important that the training corpus for UnknownWordRuleLearner has not been used when creating the dictionary. Files containing the generated rules are created.

```
java -Xmx256M ContextualRuleLearner dir
java -Xmx256M UnknownWordRuleLearner dir
```

Finally the part-of-speech tagger is ready for use. The argument is a file containing the untagged text. The output is printed to standard out.

```
java Tagger file.txt
```

In order to test the accuracy of the tagger, a small testing program has been developed. It works like the Tagger but takes a manually annotated corpus as input. That way it can compute the accuracy of the tagging. The test program is started with the following command, where the argument can be either a file or a directory.

```
java Test dir
```

5 Test Results

The implemented tagger has been evaluated by computing the accuracy of the tagging on a test corpus of 120 000 words, both with an open and closed vocabulary. The time to learn the rules have also been recorded. The tests were performed on a Pentium M 1.4 GHz with 512 MB of RAM and Windows XP Professional operating system.

5.1 Rule Learning

The contextual rules were learned in 9 hours using a training corpus of 470 000 words. The learner was stopped when 200 rules had been generated. The dictionary was built using the whole SUC corpus.

The learner of unknown word rules used a training corpus of 230 000 words and was finished after 11 hours and 30 minutes. The whole SUC corpus except the texts in the training corpus was used to generate the dictionary. The algorithm was stopped after 100 generated rules.

Figure 6 and 7 show the first ten rules learned by each learner.

5.2 Closed Vocabulary Test

A closed vocabulary means that there are no unknown words in the text to tag. Therefore the whole SUC corpus could be used to build the dictionary.

The results are presented as the percentage of correctly tagged words after initial tagging, which is called the baseline, and after applying the rules. Only the contextual rules are applicable here, since there are no unknown words. As a comparsion it can be mentioned that an accuracy of 97.0 % was reported by Brill, making the closed vocabulary assumption.

From tag	To tag	Condition
IE	SN	Tag 1, 2 or 3 after is VB PRS AKT
PN NEU SIN DEF SUB/OBJ	DT NEU SIN DEF	Tag 1, 2 or 3 after is NN NEU SIN DEF NOM
IE	SN	Tag 1, 2 or 3 after is VB PRT AKT
JJ POS UTR/NEU PLU IND/DEF NOM	JJ POS UTR/NEU SIN DEF NOM	Tag 1, 2 or 3 before is DT UTR SIN DEF
JJ POS UTR/NEU SIN DEF NOM	JJ POS UTR/NEU PLU IND/DEF NOM	Next tag is NN UTR PLU IND NOM
NN NEU PLU IND NOM	NN NEU SIN IND NOM	Tag 1 or 2 before is DT NEU SIN IND
HP - - -	KN	Tag 1 or 2 after is NN UTR SIN IND NOM
DT UTR SIN DEF	PN UTR SIN DEF SUB/OBJ	Next tag is VB PRS AKT
DT UTR/NEU PLU DEF	PN UTR/NEU PLU DEF SUB	Next tag is VB PRS AKT
PN NEU SIN DEF SUB/OBJ	DT NEU SIN DEF	Tag 1 or 2 after is NN NEU SIN IND NOM

Figure 6: The first ten contextual rules

From tag	To tag	Condition
NN UTR SIN IND NOM	NN UTR PLU DEF NOM	Suffix is "rna"
DT UTR SIN IND	NN UTR SIN DEF NOM	Suffix is "en"
PM NOM	PM GEN	Suffix is "s"
UO	NN NEU SIN DEF NOM	Suffix is "et"
NN UTR SIN IND NOM	PC PRS UTR/NEU SIN/PLU IND/DEF NOM	Suffix is "ande"
UO	NN UTR PLU DEF GEN	Suffix is "rnas"
AB	NN UTR SIN DEF GEN	Suffix is "ens"
DT UTR/NEU PLU DEF	VB PRT AKT	Suffix is "de"
JJ POS UTR SIN IND NOM	NN UTR SIN DEF NOM	Suffix is "n"
PM NOM	NN UTR PLU DEF NOM	Suffix is "rna"

Figure 7: The first ten rules for unknown words

Baseline:	91.92 %
Final accuracy:	95.18 %

5.3 Open Vocabulary Test

To test the performance of the tagger with an open vocabulary, the test corpus could not be part of the corpus used to build the dictionary. The ratio of unknown words in the test corpus were 7.44 %. The figures below show the percentage of correctly tagged known and unknown words after the three main steps of tagging.

Accuracy after initial tagging (baseline)

Known words:	90.78 %
Unknown words:	59.67 %
All words:	88.46 %

Accuracy after applying unknown word rules

Known words:	90.78 %
Unknown words:	73.37 %
All words:	89.48 %

Accuracy after applying contextual rules

Known words:	94.41 %
Unknown words:	74.70 %
All words:	92.94 %

This can be compared to the accuracy claimed by Brill. With a baseline of 92.4 %, he reported accuracies of 96.3 % for all words and 82.0 % for unknown words.

6 Conclusions

6.1 Optimization

The optimization of the original tagger was very successful, resulting in a running time almost 200 times faster when using a training corpus of 23 000 words.

6.2 Handling of Unknown Words

Unfortunately, most of the rules for unknown words turned out to give very poor results. A possible reason is that the rule templates were developed with the English language in mind. The rule template that gave by far the best results was the one that changes the tag depending on the suffix of the word. It resulted in very reasonable rules like "Change from nominative to genitive if the word ends with *s*".

6.3 The Results

The final result of our work can be summarized in the figures showing the accuracy of the tagger. The resulting accuracy was computed to 95.18 % and 92.94 %, with a closed and open vocabulary respectively. These figures are significantly lower than the ones reported by Brill. The main difference between our result and that of Brill is the baseline, which is much lower in our implementation (88 % compared to 92 %). The difference between the baseline and the resulting accuracy is about 4 percentage points for both implementations. However, it may be difficult to make comparisons between the two implementations since our is for the Swedish language and Brill's is for English.

7 Further Work

For future extenders of this work, the baseline will presumably be the focal point of attention. It may be interesting to examine why the baseline in this and Brill's implementation differs and see if improvements can be made.

A way to further increase the accuracy of the tagger, would be to introduce the lexicalized rules also suggested by Brill. It is a set of contextual rule templates that make reference to words instead of tags. However, according to Brill, these rules only improve accuracy slightly (0.2 percentage points).

Another interesting task would be to investigate in what ways the rule templates for unknown words could be adjusted to make them more suitable for the Swedish language.

8 Acknowledgements

We would like to thank Pierre Nugues for valuable help and suggestions during our work. We also wish to acknowledge the work of François Marier and Bengt Sjödin, which resulted in a clear and useful Brill tagger implementation.

References

Eva Ejerhed and Gunnel Källgren and Ola Wennstedt and Magnus Åström. 1992. *The Linguistic Annotation System of the Stockholm-Umeå Project.*

Pierre Nugues. 2004. *An Intruduction to Language Processing with Perl and Prolog.*

Eric Brill. 1995. *Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part-of-Speech Tagging.*

Grace Ngai and Radu Florian. 2001. *Transformation-Based Learning in the Fast Lane.*

Johan Carlberger and Viggo Kann. 1999. *Implementing an efficient part-of-speech tagger*

François Marier and Bengt Sjödin. 2003. *A part-of-speech tagger for Swedish using the Brill transformation-based learning*

Appendix - The SUC 1.0 Corpus

Text Categories

The corpus consists of 500 text files, with approximately 2000 words each. Each file has a unique name, containing information of which category the text falls under. There are ten main text catagories and each of them has a number of sub-categories. The distibution of files over the main categories are presented below.

Category	Number of files
A. Press, Reportage	44
B. Press, Editorials	17
C. Press, Reviews	27
E. Skills, trades and hobbies	58
F. Popular lore	48
G. Biographies, essays	26
H. Miscellaneous	70
J. Learned and scientific writing	83
K. Imaginative prose	127

Format

The SUC 1.0 corpus is available in two different formats called SUC1A and SUC1B. The format used in the project and described here is the SUC1A format.

The corpus is divided into text elements generally called tokens. Tokens are normally words, but also include punctuations, numbers etc. Each token is tagged with it's part-of-speech category along with a number of morphosyntactic features. The base form of the word is also part of the tag. Below is an example of a tokenized and tagged sentence, with a reference number for each token. Note that the swedish letters å, ä and ö are encoded }, { and |.

```
("<Det>" <1142>
    (PN NEU SIN DEF SUB/OBJ "det"))
("<{r>" <1143>
    (VB PRS AKT "vara"))
("<viktigt>" <1144>
    (JJ POS NEU SIN IND NOM "viktig"))
("<att>" <1145>
    (IE "att"))
("<inte>" <1146>
    (AB "inte"))
("<st|ra>" <1147>
    (VB INF AKT "st|ra"))
("<f}glarna>" <1148>
    (NN UTR PLU DEF NOM "f}gel"))
("<under>" <1149>
    (PP "under"))
("<h{ckningstiden>" <1150>
    (NN UTR SIN DEF NOM "h{ckningstid"))
("<.>" <1151>
    (DL MAD "."))
```

Part-of-Speech Categories

All tags begins with one of the two letter codes representing the part-of-speech.

Code	Swedish category	Example	English translation
AB	Adverb	inte	Adverb
DT	Determinerare	denna	Determiner
HA	Relativt adverb	när	Relative Adverb
HD	Relativ determinerare	vilken	Relative Determiner
HP	Relativt pronomen	som	Relative Pronoun
HS	Relativt possessivt pronomen	vars	Relative Possessive
IE	Infinitivmrke	att	Infinitive Marker
IN	Interjektion	ja	Interjection
JJ	Adjektiv	glad	Adjective
KN	Konjunktion	och	Conjunction
NN	Substantiv	pudding	Noun
PC	Particip	utsänd	Participle
PL	Partikel	ut	Particle
PM	Egennamn	Mats	Proper Noun
PN	Pronomen	hon	Pronoun
PP	Preposition	av	Preposition
PS	Possessivt pronomen	hennes	Possessive
RG	Grundtal	tre	Cardinal number
RO	Ordningstal	tredje	Ordinal number
SN	Subjunktion	att	Subjunction
UO	Utländskt ord	the	Foreign Word
VB	Verb	kasta	Verb

Morphosyntactic Features

Parentheses show that a feature only applies to some members of the part-of-speech or that not all the values of a feature are applicable.

Feature	Value	Legend	POS where feature applies
Gender	UTR	Uter (common)	DT, HD, HP, JJ, NN, PC, PN, PS, (RG, RO)
	NEU	Neuter	
	MAS	Masculine	
Number	SIN	Singular	DT, HD, HP, JJ, NN, PC, PN, PS, (RG, RO)
	PLU	Plural	
Definiteness	IND	Indefinite	DT, (HD, HP, HS), JJ, NN, PC, PN, (PS, RG, RO)
	DEF	Definite	
Case	NOM	Nominative	JJ, NN, PC, PM, (RG, RO)
	GEN	Genitive	
Tense	PRS	Present	VB
	PRT	Preterite	
	SUP	Supinum	
	INF	Infinite	
Voice	AKT	Active	
	SFO	S-form[1]	
Mood	KON	Subjunctive[2]	
Participle form	PRS	Present	PC
	PRF	Perfect	
Degree	POS	Positive	(AB), JJ
	KOM	Comparative	
	SUV	Superlative	
Pronoun form	SUB	Subject form	PN
	OBJ	Object form	
	SMS	Compound [3]	All parts-of-speech