FGT - A Framework for Generating Rule-based Taggers

Zoltán Alexin[1], Tamás Váradi[2], Csaba Oravecz[2], Gábor Prószéky[3], János Csirik[4], and Tibor Gyimóthy[4]

[1] Dept. of Applied Informatics, József Attila University, Árpád tér 2, H-6720 Szeged Hungary, alexin@inf.u-szeged.hu
[2] Research Institute for Linguistics, Hungarian Academy of Sciences, varadi, oravecz@inf.u-szeged.hu
[3] Morphologic Ltd. Hungary, proszeky@morphologic.hu
[4] Research Group on Artificial Intelligence, Hungarian Academy of Sciences, Aradi vértanuk tere 1, H-6720 Szeged, csirik, gyimi@inf.u-szeged.hu

Abstract. In this paper we give an overview of a tool called FGT (Framework for Tagger generation) which is able to assist in the development of rule-based POS taggers. In the tagging generation process we need programs to determine ambiguity tag classes, produce lexicons, generate training and test data for different learners, and transform the learnt disambiguation rules into working taggers. C and Prolog programs have been written for such tasks and we are going to integrate them into the FGT. The modules of the FGT system have already been tested for several applications. Our findings show that the FGT system can effectively be used for producing POS taggers from different annotated texts and for different learning methods.

1 Introduction

Part-of-speech (POS) tagging is an important intermediate step between the morphologic analysis and syntactical parsing of natural languages. There is much research and development efforts for producing POS taggers by using annotated corpora [3, 7]. The applied methods can be divided into two main types: statistical approaches and rule-based methods. Most of the statistical methods are based on Hidden Markov Models (HMM) [9], while Inductive Logic Programming (ILP) algorithms are widely used to produce rule-based POS taggers for different languages [3, 4, 7]. Several papers were published comparing the accuracies of these taggers generated by different methods [4, 6, 7].

In this paper we give an overview of a tool called FGT (Framework for Tagger Generation) which is able to assist in the devolopment of rule-based POS taggers. The generation process of these taggers is based on large-size corpora so auxiliary programs are required to handle annotated data. More precisely, we need programs to determine ambiguity classes, produce lexicons, generate training and test data for different learners, and transform the learnt disambiguation rules into working taggers. We have developed C and Prolog programs for these tasks and we are going to integrate them into the FGT. The modules of the FGT system have already been tested in several applications. By using the Hungarian part of the MULTEXT corpus [5] and four rule-based learner methods, POS taggers were generated with the FGT system. In this paper we give a short overview of our experiences using FGT for producing Slovene and Hungarian taggers for a large (MSD) tag set. Our results show that the FGT system can be effectively used for producing POS taggers from different annotated texts, and for different learning methods. Currently only a loosely integrated version of the different modules has been implemented in the FGT system. (The modules of the FGT system and the experimental data are both available on the web site: `http://amadea.inf.u-szeged.hu/lll`). An integrated interactive environment for the FGT system is under development.

The next section provides an overview of the modules of the FGT system. In Section 3 some experiments are described, and in the final section we give a summary of our future work.

2 The Main Modules of FGT

In this section the most important modules of FGT are presented. As mentioned earlier the main aim of the development of FGT is to provide a toolbox that can simplify the work of creating rule-based taggers from annotated corpora when using learning systems. Since the work of making disambiguation modules for Hungarian began, such things were generally performed by hand. But now most of the manual work is carried out by the modules of FGT.

The presented system contains utilities for the particular steps of tagger generation. It consists of several loosely connected modules. Each of them does one specific task in the processing of annotated corpora. The modules read in the corpus files, generate learning tasks for the learning systems, and finally generate rules for a working tagger framework.

Conversion programs have been written in ANSI C, while all other modules have been written in SICStus Prolog 3.7.1. Choosing such a development platform gives a good solution for maintaining maximal portability of the sources. The main development machine was a Pentium-II 266MHz dual CPU machine with Linux Redhat 5.2 operating system.

The majority of the modules of the FGT system can already be downloaded (see [1]), while the remaining modules will be finished and put into the Web in the near future.

2.1 Conversion of the Data to Prolog

The annotated corpora for the tagger generation are given in several different text file formats. In order to remove all unnecessary information the content (*sentences*) is converted into a standard prolog form. Each sentence is mapped to a prolog fact that contains only the *good* tagging (*annotation*) and the list of possible tags, as in the following example:

```
s('Ohu.1.2.1.1', [(asn, [asn, vmis3s]), wpunct, asn, asn, nsn,
   vmis3s, wpunct, (t, [psn, t]), npn, rg, ms, vmis3p, spunct]).
```

The training and testing material for the learning tools are all generated from such types of lists of Prolog facts. There are two widely-used tag sets for Hungarian: the CTAG and MSD codes. Both tag sets are supported by the modules of the FGT system. MSD codes are the most accurate as for in Hungarian there are c.a. 10000 different MSD codes. On the other hand many of them are rarely used, e.g. in the Hungarian Orwell text we counted only 400 different MSD tags, and 566 different MSD codes in the Slovene. The Hungarian CTAG set contains 125 different codes and 100 out of them occurred in the text. More details can be found in [2, 7].

Corpus	File format
Multext-East corpus (Orwell, Hungarian)	SGML file, CES[8]
Revised (in 1999) Orwell, Hungariani without morphological analysis	DIS file, Tab separated values CTAG annotations
Revised (in 1999) Orwell, Hungarian with full morphological analysis	DIS file, Tab separated values both CTAG and MSD annotations
Multext-East corpus (Orwell, Slovene) without morphological analysis	Space separated pairs of words and MSDs

Table 1. The supported file formats in FGT

The supported corpus formats are listed in Table 1. The conversion programs are written in pure ANSI C and expected to be portable to every possible platform.

2.2 Generating Learning Tasks for Learning Systems

Currently four learning tools are supported: C 4.5, AGLEARN, Progol and PHM [7]. The learning task generator modules produce all the necessary files (subdirectories, training data and test data files for the chosen ambiguity classes, batch files). When the generation is completed the learning phase can be started by simply typing in the name of a batch file e.g. `doall`. It then takes the learning tasks one by one and runs them with the corresponding command-line parameters and input files. The learned rules, decision trees, or tables are converted into Prolog rules in the next step (see subsection 2.3).

Our idea behind the tagger construction is to decompose the problem into ambiguity classes. It means that first the most frequent ambiguity classes are selected, then a separate learning task is performed for each of these ambiguity classes. Learning tools result in rules that are able to determine the good tagging of a word belonging to a specific ambiguity class. We convert these rules to Prolog predicates like `choose_b__a_b_c/2`, where $[a, b, c]$ is an ambiguity class and

$b \in [a, b, c]$. The two arguments of the predicates are the left and right contexts. Both contexts are represented by lists of MSD (or CTAG) token codes. We argue that the accuracy of the learnt rules should be better when the learning focuses on one specific ambiguity class. With this technique the size of the training data set can also be reduced (sometimes too much in extent) so the running time of learning tools can be decreased.

C 4.5 and AGLEARN learning

The learning task generator makes a separate subdirectory for each ambiguity class. The generator module can set up a learning task with different window sizes i.e. the left and the right contexts are cut to fixed length prefixes of the complete lists. We work with window sizes: $1 \ldots 7$, so each subdirectory will contain 7 learning tasks for each different window size. According to our studies the window size $= 2$ gives the best accuracy. A batch file is also generated that goes into each subdirectory and runs C 4.5 or AGLEARN on the files available there.

The AGLEARN method can be considered as a preprocessor for C 4.5, which allows the use of linguistic background knowledge [2].

Progol learning

The Progol learning task generator module makes separate learning tasks for each c_i choice within a given ambiguity class $[c_1 \ldots c_n]$ $(1 \leq i \leq n)$. Positive examples will be those configurations where the annotation of the focus word is c_i, while all other configurations when the annotation of the focus word is different from c_i will be the negative examples of the learning task. The training and testing data files are generated by the learning task generator module.

PHM learning

The PHM algorithm is written in ANSI C [7]. It requires two text files to prepare, namely: a training and a test data file. Both of them are compressed versions of the prolog corpora, the token codes being replaced by integers in the files. The PHM learner then decomposes the task into the most frequent ambiguity classes, learns the rules, and finally evaluates the learned rules.

2.3 Generating Working Taggers

For the tagger generation there are three modules available: the tagger standard part (a framework), the Prolog predicate generator, and a module for generating combined taggers.

The tagger standard part dynamically senses the *choose* predicates present. This technique is a flexible solution when we do not have rules for all possible ambiguity classes. The tagger tries to resolve the next leftmost ambiguity in a sentence, and when it fails (e.g. there is no rule for this ambiguity, or the rule fails) it moves further right. It is repeated while there is an applicable disambiguation rule to the words of the sentence.

The Prolog predicate generator module is able to read in the output files of a particular learning tool and generate the corresponding *choose* predicates that are saved as SICStus prolog source files for each ambiguity class. It walks through all of the generated subdirectories.

The tool for generating combined taggers can apply a meta-rule (*strategy*) for combination. If we consider 3 taggers and want to implement a majority voting combination of them, then the following meta-rule should be applied:

```
majority(Choose) :- (c45:Choose), (progol:Choose).
majority(Choose) :- (c45:Choose), (phm:Choose).
majority(Choose) :- (progol:Choose), (phm:Choose).
```

Where `c45`, `progol`, and `phm` are prolog modules that contain the learned rule sets by C 4.5, Progol and PHM respectively.

2.4 Evaluation of the Learnt Tagger

The tagger framework can write its output to disk files. The tagger evaluation module is able to compare two files, the reference and the disambiguated (the tagger output) files. The result is a set of statistics that describes the behaviour of the tagger in detail. The report contains the number of *good*, *bad* and *unsolved* cases for each ambiguity class.

3 Expriments of using FGT

In [7] we presented taggers based on the CTAG annotations. The presented taggers were generated by the modules of FGT. To test the capabilities of the FGT framework on an MSD annotated corpus we attempted to generate two working taggers. In both cases the C 4.5 attribute value learner was used. Each decision tree inferred by the learner is converted to Prolog clauses. The test part of the corpus was used to evaluate the obtained taggers.

MSD level tagger for Hungarian using C 4.5

We selected the most frequent ambiguity classes (whose occurrances in the training data set was larger than 50). There were 53 such classes. From the 18784 ambiguities in the training 15458 (82.29%) belong to these classes; from the 6742 ambiguities in the test 5546 (82.26%) belong to these classes. The resulting tagger obtained a 86.06% accuracy on the ambiguous words.

MSD level tagger for Slovene using C 4.5

Similarily, we selected the most frequent ambiguity classes. There were 66 such classes. 19337 (71.47%) from the 27054 ambiguities in the training belong to these classes; 8754 (71.37%) from the 12265 ambiguities in the test belong to these classes. The resulted tagger obtained 75.81% accuracy on the ambiguous words.

4 Further Work

The FGT system gives a solution for automatic tagger generation from annotated corpora using them as training and test data in the learning algorithms. The system offers much assistance in the creation of experimental taggers. Its components proved their usefulness both separately and as a whole. The majority of the modules of the FGT can be downloaded from [1].

The FGT system can be arbitrarily augmented with new modules when required. An interactive graphical user interface will be developed that will intergate all existing modules of FGT.

Another possible direction for development is a module for generating combined taggers. Combined taggers reportedly achieve better performances than any of their individual components [6]. By supplying a meta combination rule (*strategy*) the module is then able to generate a combined tagger from the rulesets learned by different learning tasks. The development of such a tool is planned in the medium term.

We would like to integrate the Hungarian taggers generated by FGT with a morphological analyser. After that we will be able to test our taggers against unknown texts. Hungarian Morphologic Ltd. has already developed a morphological parser (called HuMor), but it uses its own annotation mechanism. The conversion of HuMor codes to MSD codes can be done using a new module for FGT.

References

[1] Z. Alexin and T. Gyimóthy. Learning and Natural Language Processing at RGAI Szeged, 1999. `http://amadea.inf.u-szeged.hu/lll/`.

[2] Z. Alexin, S. Zvada, and T. Gyimóthy. Application of AGLEARN on Hungarian Part-of-speech Tagging. In D. Parigot and M. Mernik, editors, *Second Workshop on Attribute Grammars and their Applications, WAGA'99*, pages 133–152, Amsterdam, The Netherlands, 1999. INRIA Rocquencourt.

[3] J. Cussens. Part-of-speech tagging using Progol. In N. Lavrač and S. Džeroski, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of *LNAI*, pages 93–108. Springer, Sept. 17–20 1997.

[4] S. Džeroski, T. Erjavec, and J. Zavrel. Morphosyntactic tagging of slovene: Evaluating pos taggers and tagsets. Ijs techreport, Jozef Stefan Institute, Ljubljana, Slovenia, 1999.

[5] T. Erjavec, A. Lawson, and L. R. (eds.). East meets West: A compendium of multilingual resources, 1998. CD-ROM, produced and distributed by TELRI Association e.V., ISBN:3-922641-46-6.

[6] H. v. Halteren, J. Zavrel, and W. Daelemans. Improving data driven wordclass tagging by system combination. In *Proc. of COLING-ACL'98, Montreal, Canada*, pages 491–497, 1998.

[7] T. Horváth, Z. Alexin, T. Gyimóthy, and S. Wrobel. Application of Different Learning Methods to Hungarian Part-of-speech Tagging. In D. Page, editor, *Proc. 9th Int. Conference on Inductive Logic Programming (ILP99)*, Berlin, New York, 1999. Springer Verlag. to appear.

[8] C. Oravecz. Morfoszintaktikai annotáció a magyar nemzeti szövegtárban. Technical report, Research Institute for Linguistics, Hungarian Academy of Sciences, 1998. (in Hungarian).

[9] R. Steetskamp. An Implementation of a Probabilistic Tagger. Technical report, TOSCA Research Group, University of Nijmegen, 1995.