Abstract

This work presents a hybrid part of speech tagger which successfully combines a statistic model with a rule based system. The rules behave as constraints used to reduce the ambiguity of the tokens. The novelty is in the tool used for building such rules. Graphical Grammar Studio (GGS) is an open source software designed for matching/finding sequences of tokens in a similar manner as the regex language is designed to match sequences of characters. Differences are that GGS can also annotate the matched sequences and that a GGS rule / grammar is a Recursive Transitional Network which can be edited using a user friendly visual tool. To ease the manual building of such rules, a custom made tool was used, which classifies tagging errors and shows the precision yielded by the rules. Finally, the paper presents the results of the part of speech (POS) tagging model for a Romanian.

1. Introduction

When a simple statistic part of speech (POS) tagger generates a type of error systematically, the only solution to fix it is to tweak various parameters. This usually leads to a trial and error approach which is not guaranteed to fix the problem. Modifying some of the parameters might also require retraining of the entire statistic model, which can take a lot of time.

By detecting the linguistic conditions for which the classifier generates a type of error, the hybrid model described in this paper can be configured to fix such errors with little effort.

Most common POS taggers use a POS dictionary for constraining each word to a small set of possible output tags. Then, a statistic model is used to disambiguate among these. The Hybrid POS tagger presented uses a method of reducing the ambiguity even further, by using rules which can take into account any features of the words within the input sentence.

POS taggers sometimes fail to correctly classify cases for which linguists can easily decide the correct part of speech. These types of errors are generated due to noise in the training data but also because a machine learning method cannot yet detect all the linguistic phenomena in the training set. The main goal of this work is to overcome this limitation.

Another goal is to introduce Graphical Grammar Studio, a tool for creating rules in a visual environment, displayed as networks of tokens, which are used for matching and annotating sequences. With this tool, a system for constructing constraint rules has been
implemented. The rules are very easy to edit, and understand due to the visual representation.

2. Graphical Grammar Studio as a constraint rules system

Before applying the rules, each word of the input sentence is associated with a list of possible tags, based on a POS dictionary. If a word is not found in the POS dictionary, it is associated with a predefined list of tags, considered the guesser tagset.

Applying a rule on a word, results in the reduction of the number of the possible POS tags associated to some tokens (usually, for the token for which it is applied, but other tokens can also be affected). All rules are applied on all words, from left to right.

The general architecture of the hybrid part of speech tagger is not new (Simionescu 2011). The novelty lies in the solution for the partial disambiguation rules system. In the previous approach (Simionescu 2011), a constraint rules language was designed, particularly for this task. This approach was similar to Constraint Grammars (Karlsson, et al. 1995) and JAPE (Cunningham, Maynard and Tablan 2000). After testing intensively with these, while building more and more complex rules, limitations of such “text based” languages were reached. It becomes very difficult to manage and understand the behavior of complex rules, written in programming languages, because they end up having a very elaborate complicated and unorganized look.

2.1. Graphical Grammar Studio

Graphical Grammar Studio (GGS) is a tool developed by the author and published as an open source project on SourceForge\(^1\). GGS offers the tools for creating and applying an extended variation of Recursive Transitional Networks (RTN) on tokenized text. It has been successfully used for creating a rule based deep Noun Phrase Chunker for Romanian.

At its core, GGS is a tool very similar with Nooj (Silberztein, Nooj: an Object-Oriented Approach 2004), but the latter is meant to become a comprehensive development environment for NLP tasks. Nooj is an improved version of the INTEX system (Silberztein, INTEX: a corpus processing system 1994). It has support for dictionaries, morphological grammars, paradigmatic representations etc. and it can read an impressive number of input formats. It also provides means to create chains of grammars.

GGS on the other hand is oriented only towards sequence matching and annotating. It is a tool meant for easy integration in various processing chains. It doesn’t have support for dictionaries like Nooj and it doesn’t require the presence of certain attributes for its input tokens. All token attributes are treated as key-value pairs which a GGS grammar can refer to, using regular expressions for both keys and values. GGS is the more out-of-the-box type of tool, because it doesn’t require any prior configuration/installation whatsoever.

\(^1\) [http://sourceforge.net/projects/ggs/](http://sourceforge.net/projects/ggs/)
Being a tool specialized on matching and annotating sequences of tokens, GGS contains several features which Nooj lacks in this aspect, like recursive depth and loop limits, or look ahead and look behind assertions. But before anything else, GGS is meant to be an open source project which can be used by anyone. Also, the Java platform might offer some technical advantages for some users, over the .NET platform.

GGS’s main component is the GGS Editor. A secondary component is a java library used for integrating the GGS engine in java code.

GGS grammars are composed mainly of token matching/consuming nodes and empty nodes (which are used for visually organizing the aspect of a network of nodes, and for other features of GGS; empty nodes do not consume tokens from the input). The nodes are structured in sub networks (graphs). There are also jump nodes which can transition from one graph to another. Recursive jumps can thus be described, making this manner of defining matching grammars a very convenient one for many NLP rule based tasks. Each GGS grammar has a main graph. The starting and ending nodes of this grammar are the actual start and final states of the machine that runs behind the scenes.

A GGS grammar can be applied on xml input. The name of the tags which represent text units (usually sentences) must be provided. GGS expects to find sequences of token tags as the children of these xml nodes, which will be considered the input stack of tokens. A token tag can have an unlimited number of XML attributes. A token matching node accepts/consume the first unconsumed input token based on a condition which is described as a sequence of key-value pairs which must or must not be present in these attributes map. Moreover, both the keys and the values can be specified in such a condition using regular expressions, providing a great amount of flexibility.

The main graph in Figure 1 matches all pair of words in which the first one is “the”. The “<>” node matches any token because it doesn’t impose any conditions. For the first token, the code is interpreted as: the next input token will be consumed if it has an attribute „WORD” equal to „the”.

The main graph in Figure 2 matches any sequence of words, where the first and the last word must be “the” and the second word must be “is”. The “<>” node matches any token because it doesn’t impose any conditions.

Figure 1: Simple GGS graph example

Figure 2: Secondary graph
Figure 2 and 3 show a secondary graph and how it is transitioned to, by nodes from the main graph. The mySequence contains a loop and will match any sequence of adjectives separated by optional conjunctions or commas (provided that input tokens which are adjectives have the attribute ana="#Afp"). The code <+ana=/#Nc.*> is interpreted as: the next input token will be consumed if it has the attribute +ana maped to a value which matches the regex “#Nc.*” (which will match tags which stand for common nouns, in this particular example). Regular expressions can also be used on the attributes names. The code <+/.*/=\p{Lu}./> will match the input token if it has any attribute (regex .*) having a value which matches the regex \p{Lu}.* (which stands for string starting with upper case letters).

The syntax of this language can be used to specify multiple constraints related to the attributes of an input token. Such constraints can also be used for consuming an input token only if it doesn’t have a particular attribute.

### 2.2. Constraint rules system based on GGS

For the task of partial POS disambiguation, constraint rules must be able to define a conditional part, which can relate to the target token’s (the token for which the rule is applied / the current token) neighbors and their attributes. Rules must also contain specifications for the actions to be taken in case their conditions are satisfied. These consist in manipulations of the lists of possible POS tags associated to tokens from the input sentence.

GGS is ideal for the conditions part. The system presented uses GGS for this; constraint rules are actual grammars which are used for matching sequences of tokens. This way, one can create such rules using GGS visual editor. GGS can also annotate matched sequences. This mechanism can be used for specifying what actions to be taken for particular tokens (a rule usually affects only its target token, but it can also affect any other token from a sentence).

An xml format for feeding the input to GGS is required. The matching engine can take one sentence at a time. Each token tag must model using XML attributes, among other details, its list of possible POS tags (based on a morphologic dictionary). Below is a sample of the chosen feed format. This is important in the creation process of rules.

```xml
<s>
  <W in_dict="true" lemma0="un" msd0="Timsr">Un</W>
  <W in_dict="true" lemma0="vrea" lemma1="om" lemma2="om" msd0="Vaip1p" msd1="Ncmsrn" msd2="Ncmson">om</W>
</s>
```
There is an unknown word in the example. This has 97 different possible POS tags because that is the size of the guesser tagset.

In the workflow of the entire process, a piece of code looks up words in the morphologic dictionary and creates the input to feed GGS, for each sentence. Then, the GGS matching process comes into action and, for each token (from left to right) and for each rule, it creates an output composed from the same input text with some of the tokens annotated with action specifications. Below is an example of such output.

The annotated output is interpreted by a piece of code. For the previous example, all the POS tags which match “Np.*” will be kept in the list of possible POS tags of the token identified by the child tag and the rest will be removed. In the same manner, an action for REMOVE can be specified.

By default, GGS was designed for finding matches anywhere in the given input text unit. It does this by actually applying the matching process repeatedly, offsetting the input each time by 1 position to the right. In the case of these constraint rules, this is not required. With the GGS java library, it was possible to explicitly request only one matching attempt starting from a particular offset from the input stream. This is how GGS was tweaked to create one output for each token-rule pair. GGS’s flexibility made it very easy to be wrapped in a standalone NLP component.

### 2.3. Example rules
These are actual rules from the Romanian pos tagger implemented and described in the next section.

![Figure 4: A simple pos reduction rule](image)

Figure 4: A simple pos reduction rule shows a very simple rule, which if a token word matches „fără” (the i at the end indicates that the match is case insensitive) and the next word is “să” or is compound and has “să” at the end (e.g. “ca să”, “în loc să”, “are să” are words tokenized as one), then the first token can only be tagged with Cs (which stands for conjunction). This way, „fără” will not be confused by the statistic model for a preposition, in these cases.

![Figure 5: A more complex rule](image)

Figure 5: A more complex rule

The rule from Figure 5 deals with imperative verbs. If a token can be an imperative verb (it has an attribute starting with „msd” and equal to a value starting with „Vmm”) and it is followed by a word which starts with a hyphen and can be a Personal Pronoun (in the first or second person), then the verb must certainly be an imperative one. This rule deals with cases as in „[culcă][-te]” (translation: “go to sleep”) or “[ascultă][-mă]” (translation: “listen to me”) or “[ridicaţi][-le]” (translation: “lift them”) (brackets represent token boundaries), in which the verbs morphological features are usually confused by the prediction system.

For the system to behave as desired, it is necessary to make sure that the target token of a rule has a fixed position in the sequence which can be matched by its grammar. This is a problem, only for the rules which are looking to/consuming a variable number of tokens to the left of their target. Given the fact that rules are applied from left to right on each offset, there might be cases where such rules don’t behave as desired – some tokens might get skipped by such rules; also, such rules might be applied multiple times, resulting in redundant operations. To overcome this, rules which look at a variable number of tokens to the left of their target should use look behind assertion conditions on the node which is supposed to consume the target token. This way the target token has a fixed position in the matched sequence of the rule, and the tokens to its left are checked by a look behind condition (which is actually a secondary grammar being applied). Not obeying this design rule results in slight changes in the constraint rules’ behavior. In addition, assertions conditions in general can be used to define rules which cannot be defined otherwise. For instance: a word can be a conjunctive verb, only if there is a words “să” at a maximum distance of 5 tokens to its left, with the condition that there is no other potential conjunctive verb in between.
The rule from Figure 6 deals with a gender confusion for adjectives which have the same form in both masculine and feminine declination (e.g. „mare”, ”tare”). The desambiguation is done based on the presence of a masculine article or numeral to the left, with the possibility of two adverbs in between („cea mai tare” - feminine vs „cel mai tare” - masculine).

Figure 7 shows a rule which uses a negative look behind assertion. If a word can be a conjunctive verb but there is no word „să” to its left at a maximum distance of five tokens then restrict the possibility of the word to be tagged as conjunctive verb.

Assertions are actually secondary grammars which are applied on input text. Look behind assertions particularly are consuming the tokens from right to left, and that is also the same order in which their nodes are parsed. The positive or negative attribute of assertions state whether the grammar must or must not match respectively, on the input tokens.

3. A rule editing tool

Writing rules to enhance the precision of the POS tagger can be difficult without any information about the errors generated at evaluation. For this reason a custom made application was developed which, while evaluating the model on a test corpus, shows various statistics regarding the fail cases detected. The software also provides the functionality to evaluate only a certain rule (or set of rules) and see the increase or decrease of precision yielded by it. This tool is completely language independent.

The rules are separated into final and test rules. The evaluation of the model when using only the final rules is a reference point for determining the differences yielded by the test rules. Evaluating the test rules results in actually evaluating the final rules plus the test rules together. Only the fail cases which are tagged differently in the full evaluation are considered test rules errors. For both final rules errors and test rules errors, the interface provides classifications by the following criteria:

- Most failed output tags
- Most failed expected tags
- Most frequent confusions(most frequent output-expected tags pair)
- Most problematic words
For each class of errors the user can visualize the sequences of words from the test corpus for which the POS tagger has failed (Figure 8).

By analyzing the final rules’ fail cases, a linguist can detect contextual features for solving frequent errors, and write a new rule. This can then be easily refined because the user can see the exact fail cases and statistics for the new rule (by making it the single test rule in the system).

The application doesn’t require multiple initializations of the POS dictionary and the statistic model, and that is why the work flow is smooth. Evaluating usually takes less than a few seconds. Nevertheless, the interface offers the possibility to use only a fraction of the testing corpus, to speed things up, if necessary.

4. Romanian Hybrid POS tagger

A Romanian POS tagger has been developed by the author using the system described in the previous sections. The resources (morphologic dictionary, training and test corpora, tagset) and statistical model used were the same from the previous work (Simionescu 2011).

The morphologic dictionary was built with the help of the DexOnline.ro\(^2\) database and Wikipedia’s proper nouns collection, and it contains 1.25 million words associated to 230 000 distinct lemmas. The tagset used (406 tags) is based on the MSD classification (Erjavec 2004) used in MULTEXT-East\(^4\), and it is a reduced version of the one used by

\(^2\) http://www.dexonline.ro is the digitalization of some prestigious Romanian dictionaries. Part of the database used by DexOnline is available under the GNU license

\(^3\) http://ro.wikipedia.org

\(^4\) http://nl.ijs.si/ME
the Research Institute for Artificial Intelligence, Romanian Academy⁵ (Tu ş 2000) (about 600 tags).

The training corpus used is composed of NAACL 2003⁶ plus 28,000 sentences extracted from JRC-ACQUIS⁷ and tagged with the RACAI POS-tagger – 67000 sentences in total. The test corpus is the Multext-East ”1984” corpus having around 6000 sentences.

The statistical model and its configuration are also the ones used in the previous reference research (Simionescu 2011). The statistic model used is the maximum entropy model. In his work (1998) Adwait Ratnaparkhi describes the use of maximum entropy for POS tagging. The maximum entropy model is used by the state of the art POS tagger for English –Stanford Tagger– having precision of 97.32% (Toutanova, et al. 2003). Only recently has this score been overtaken (97.50%) (Søgaard 2011).

An online version of this POS tagger is available, both as a web application and a web service, at http://nlptools.infoiasi.ro/WebPosTagger/.

Table 1 shows the precisions obtained with and without applying the rules for the implemented Romanian Hybrid POS tagger.

<table>
<thead>
<tr>
<th>Precision</th>
<th>Without rules</th>
<th>With rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>For unknown words</td>
<td>88.88%</td>
<td>93.31%</td>
</tr>
<tr>
<td>For all words</td>
<td>95.12%</td>
<td>97.03%</td>
</tr>
</tbody>
</table>

The slight increase of precision from the previous version (96.66) is due to the fact that there were a few more rules added.

5. Conclusions

GGS is a tool which can be wrapped very nicely into a contraint grammar tool. With the implemented mechanism, one can easily create very complex rules which can be organized so that they become simple in aspect and structure. The new method of creating rules, using a visual tool opens up possibilities for less experienced users to experiment with what is one of the first bricks in computational linguistics – POS tagging.

The author intends to release the hybrid POS tagging model presented, as an open source project in the near future.

---

⁵ http://www.racai.ro referred in the NLP community as “RACAI”
⁶ A parallel corpus for Romanian-English created at the HLT/NAACL 2003 workshop, titled ”Building and Using Parallel Texts: Data Driven Machine Translation and Beyond”
⁷ JRC-ACQUIS is the largest parallel corpus. It is composed of lows for the EU Member States, since 1958 till present, translated and aligned for 23 languages
Bibliography


