Automatic Processing of Data

Jan Štěpánek

29th November 2006

`run_all` from PDT

Located in `tools/machine-annotation/run_all`.

- Tokenization of the input plain text and segmentation into sentences.
- Morphological analysis and tagging (morphological disambiguation).
- Dependency parsing.
- Analytical (dependency) function assignment for all nodes of the parsed tree.

Limitations and requirements:

- Written in **C/C++**, **perl** and **tcsh**.
- Compiled for Linux on an i386 architecture.

`run_all` from PDT

Tokenization and segmentation

- Problems with full-stop (“.”) in Czech.
- Tested on `amw` data:
  - Segmentation:
    precision 98.0 %, recall 91.4 %, F-measure 94.6 %.
  - Tokenization:
    precision 100.0 %, recall 99.2 %, F-measure 99.6 %.

run_all from PDT

Morphological analysis

- All possible lemmas and tags.
- Dictionary of 350,000 entries, 12 million Czech word forms.
- Error rate: 2.5 % (foreign names and typos).

`run_all` from PDT

Morphological tagger

- Maximum entropy approach with greedy incorporation of selectors.
- Tagging – 93.08 % accuracy on evaluation test data.

run_all from PDT

Parsing

- Czech adaptation of the parser of Michael Collins — dependency based.
- Only shorter sentences (up to 60 words).
- Evaluation test data: 81.6 % parents assigned correctly (both training and test data tagged machinely).

run_all from PDT

Analytical function assignment

- Decision tree approach (Quinlan's **C5** classifier translated to **perl**)
- Uses **btred**.
- Precision around 92 %.

`run_all` from PDT

Conversion to PML

- All the previous steps use deprecated **CSTS** format.
- Conversion script uses **btred**.

`morph_chain` from CAC

Located in `tools/morph_chain`.

- Hidden Markov models — trained by Viterbi algorithm + averaged perceptron for evaluating transitions between HMM states (Collins)
- Trained on **PDT**: 91.8 % (93.1)

Dealing with the Structural Annotation

btred

Features (same as those of **TrEd**):

- Independent on operating system (MS Windows, Linux, OS X...)
- Open source program, available for free
- Written in **perl** (macros, predefined functions)

Requirements and installation:

- **perl** (5.8.3 or newer) with **Tk** library
- `http://ufal.mff.cuni.cz/~pajas/tred/`
  - For MS Windows: `tred_wininst_en.zip` → `setup.bat`
  - For Linux: `tred-dep-unix.tar.gz` → `install tred-current.tar.gz`

Dealing with the Structural Annotation

Why **btred**?

- Object-oriented tree representation — a rich repertory of basic functions for tree traversing and for many other basic operations on trees + several highly non-trivial functions suitable for linguistically motivated traversing of trees (e.g. solving the coordination relations).
- Reasonable stability because of long-time experience (development of **PDT**).
- Network (parallel) version (not for MS Windows).
- Powerful and fast search-engine (pipes).

Dealing with the Structural Annotation

Simplest examples

Basic syntax:

`btred -e <code> file(s)` OR `btred -I macro_file file(s)`

```
$ btred -e 'writeln("Hello world!");' sample0.a.gz
BTRED: Trying /export/common/lib/tred
Config file: /home/stepanek/.tredrc
BTRED: Resource path: /home/stepanek/tred/resources/
BTRED: Reading macros from /usr/tred/tred.mac...
BTRED: done.
BTRED: <script>
package TredMacro;
sub _btred_eval_ {
  writeln("Hello world!");
}
;
;
</script>
BTRED: Processing: sample0.a.gz (1/1)
Hello world!
BTRED: Done.
```

Dealing with the Structural Annotation

Simplest examples (2)

Traversing trees:

```
$ btred -QTe 'writeln($a++);' sample0.a.gz
 ...
52
53
```

Traversing trees *and nodes*:

```
$ btred -QNTe 'writeln($a++);' sample0.a.gz
 ...
864
865
```

More files:

```
$ btred -QNTe 'writeln($a++);' sample*.a.gz
 ...
7813
7814
```

Dealing with the Structural Annotation

Simple examples

Simple attributes:

```
$ btred -QNTe 'writeln($this->{afun})' sample0.a.gz
 ...
Atr
AuxK
```

Structured and list attributes

```
$ btred -QNTe 'writeln($this->attr("m/form"))' sample0.a.gz
 ...
založení
OSN
.
$ btred -QNTe 'my @ids = ListV($this->attr("coref_text.rf"));
  if (@ids){
    writeln(PML_T::GetNodeByID($ids[0])->{t_lemma});
  }' sample*.t.gz
 ...
#PersPron
Chodura
```

Dealing with the Structural Annotation

Examples

Methods — find the tree with the highest number of nodes (root `descendants`):

```
$ btred -QTe 'writeln(scalar($root->descendants))'
   sample*.t.gz | sort -n | tail -n1
 42
```

Similarly: `children`, `parent`, `lbrother`...

Dealing with the Structural Annotation

Examples (2)

perl functions `grep` and `map` — print verbs and their objects:

```
$ btred -QNTe 'if($this->attr("m/tag") =~ /^V/ ) {
  writeln join " ",
    $this->attr("m/form"),
    map {$_->attr("m/form")}
      grep {$_->{afun} eq "Obj"} $this->children;
}' sample1.a.gz
 ...
```

Nehodlá vyjadřovat
vyjadřovat
dokončil šetření
předal spis zastupitelství

Similarly: `first`

Dealing with the Structural Annotation

Complex examples

Effective children and parents — what semantical part of speech are the parents of **actors** and how often:

```
$ btred -QNTe '
  my $par;
  $par = join(" ",
    map( {
      $_->attr("gram/sempos")
    } PML_T::GetEParents())
  ),writeln($par) if $this->{functor} eq "ACT"
' sample*.t.gz | sort | uniq -c | sort -n
 ...
4 v v v
5 adj.denot
25
30 v v
108 n.denot
117 n.denot.neg
667 v
```

Dealing with the Structural Annotation

Complex examples (2)

Crossing layer boundaries — count all **actors** expressed by a noun in nominative (1^{st} case):

```
$ btred -QNTe 'writeln() if $this->{functor} eq "ACT"
  and ! $this->{is_generated}
  and first {
    my $t = $_->attr("m/tag");
    $t =~ /^N...1/
  } PML_T::GetANodes($this)
' sample*.t.gz | wc -l
 422
```

Dealing with the Structural Annotation

Searching and viewing results

TrEd function `FPosition()`:

```
$ btred -QNTe 'FPosition()
  if $this->{t_lemma} =~ /_.*_/' sample*.t.gz
 sample9.t.gz##14.22
```

```
$ btred -I macro-that-uses-FPosition *.t.gz |
    tred -l-
```

Dealing with the Structural Annotation

Speeding up

Crawling through all the tectogrammatical nodes by **btred** takes about *10 minutes*. Most time is spent by *opening and parsing* the data.

Possible solution: read all the data just once and keep them in the memory.

Problem: not enough memory.

Solution: distribute the data among several computers.

ntred (network-tred): **btred** servers + *hub*

Dealing with the Structural Annotation
Speeding up (2)

ntred requirements:

- Cannot run on MS Windows (problems with net sockets).
- All the computers running **btred**-servers must share a filesystem.
- Password-free access to all the computers is needed.
- Some macros have to be adjusted (e.g. overall statistics).