Nerf – readme

26 czerwca 2012

Jakub Waszczuk

Institute of Computer Science Polish Academy of Sciences

About

NERF is a Named Entity Recognition (NER) tool based on the bigram Conditional Random Fields (CRF) modelling method. The tool can be trained to recognize tree-like structures of Named Entities (i.e., with recursivelly embeded NEs). The tool has been developed at the Institute of Computer Science, Polish Academy od Sciences, Warsaw, as a part of of the National Corpus of Polish project. It is released under the GNU General Public License v3.

NERF homepage:

- official: `http://zil.ipipan.waw.pl/Nerf`
- next version developed at: `https://github.com/kawu/nerf`

Authors:

- Jakub Waszczuk [jakub.waszczuk@ipipan.waw.pl]
- Michał Lenart [Michal.Lenart@ipipan.waw.pl]

License: GPL v.3

1 Prerequisites

- Python (version 2.6.* or 2.7.* http://www.python.org/)
- Cython (version 0.14 or newer, http://cython.org/)
- GNU make, gcc
  - Under Windows, gcc can be acquired by downloading and installing MinGW, `http://www.mingw.org/wiki/Getting_Started`; the make tool can be downloaded from `http://gnuwin32.sourceforge.net/packages/make.htm`.
- pycrf library, distributed together with the NERF tool. You can find description of the pycrf's installation process below.

2 Input and output format

To train a NERF model you need a corpus in TEI NKJP xml format with all annotation levels up to morphosyntactic (ann_morph.xml files) and named entity annotation level (ann_named.xml files).

In named-entity-recognition mode NERF can read plain text and TEI NKJP format. In the first case, output consists of a list of recognized named entities in separate lines (see 4.1 for more details). In the second case, output is stored in ann_named.xml files compatible with TEI NKJP format.

3 Installation

Currently, the only option is to install pycrf library and NERF tool from source.

3.1 pycrf

Linux Open a terminal, cd to the ./crf/libcrf directory in the pycrf source distribution and run:

```
make all
```

Then return to the main pycrf source directory and run setup.py script (with administrator privileges) to install the crf module:

```
python setup.py install
```

Windows Insallation procedure under Windows is similar, but:

- Make sure that paths to newly installed software (make and gcc) are listed in `PATH` environment variable.
- Run setup.py script in the following manner:
  `python setup.py build --compiler=mingw32`
  `python setup.py install`
- If *unrecognized option -mno-cygwin* error occures during compilation, see `http://stackoverflow.com/questions/6034390/compiling-with-cython-and-mingw-produces-gcc-error-unrecognized-command-line-o`.

3.2 NERF

NERF doesn't need any additional installation steps and is ready to use immediatelly after unpacking of the distribution package.

4 Basic usage

NERF is a command line tool. It doesn't provide any graphical user interface.

If you want to use it only in named-entity-recognition mode (i.e., you want to use the NERF model provided with the distribution package) read only the 4.1 section. Otherwise, you should also learn about NERF configuration and model training described in the 4.2 section. If you plan to evaluate NERF on the TEI NKJP corpus, see the 4.3 section.

4.1 NER

TEI NKJP format Use annotate.py script to recognize named entities in TEI NKJP corpus.

```
$ python scripts/annotate.py PATH-TO-CORPUS NERF-MODEL
```

Invoke with –help option to see the detailed description.

Plain text When the NER model has been successfully produced, simple ner_service.py program can be used to test the model by hand. The ner_service.py file is also a good example of how the NER tool can be used as a NER service.

NOTE: We assume, that the model uses only information from the textual level.

To use the ner_service.py with the NER model saved in model.tgz file, run:

```
$ python scripts/ner_service.py model.tgz
Loading model...
Done
```

Enter each sentence in a separate line. Sentences will be divided into segments (with break on whitespace and punctuation characters) and a list of recognized NEs will be printed below.

```
> Szpital im. Św. Jadwigi.
Organization : Szpital im. Św. Jadwigi (0, 23)
...
```

To quit the program, enter CTRL+c.

Another, even simpler ner_io.py script can be used to embed Nerf tool in a more complex data processing system. The script reads successive sentences from standard input and writes results to standard output. Input and output format is the same as with the ner_service.py script. You can invoke the ner_io.py script with the following command (note, that it needs some time to load the model before it can start processing input):

```
$ python scripts/ner_io.py model.tgz
```

4.2 Training NER model

Use train.py script from the scripts directory to train NER model on TEI NKJP corpus. TEI corpus should be unpacked from the archive file in advance.

Observation schema Training process can be customized using an observation schema. Schema defines types of observations, which will be used (i.e., values of respective observation types) during the named entity recognition. Examples of schemes may be found in the data directory bundled with this software package.

Following chart shows structure (grammar) of the schema file:

```
schema: schema_line NEWLINE [schema]
schema_line: feature_type ',' name ',' range
feature_type: "SINGLE" | "PAIR"
range: '[' list_of_ints ']'
list_of_ints: INT [',' list_of_ints]
```

Here is a simple schema example, which satisfies the specification above:

```
SINGLE, WORD(), [-1, 0]
SINGLE, SUFFIX(3), [-1, 0]
SINGLE, SHAPE(), [-1, 0]
PAIR, ID(), [0]
```

Each line of the schema corresponds to one observation type. Type of feature (SINGLE or PAIR) determines, whether observation value will be used in normal feature function (of the form f(observation value, label value)), or in transition feature function (of the form f(observation value, current label, previous label)). Name of observation (e.g., WORD()) relates to one of observation classes defined in ner/features.py source file. The last component of observation type is range. Range indicates, for which segments (with respect to the current position) observation value will be acquired. In the schema above, values of WORD, SUFFIX and SHAPE observations for i-th and (i-1)-th segments will be considered as observations related to the i-th position in the sentence. On the other hand, ID observation (which is equall to 1 for every segment in the sentence) will be acquired only for the i-th segment, when collecting observations for the i-th position in the sentence. Generally, `PAIR, ID(), [0]` observation type should be present in every schema, unless you want to obtain other than HMM-like CRF structure and use PAIR with other observation types.

Train.py script Here is an example of the train.py script usage:

```
$ python scripts/train.py --tei-corpus=X --schema=Y --model-out=Z
```

At least path to the source corpus (–tei-corpus) and path to the schema file (–schema) should be supplied. Model output file can be set with –model-out option (otherwise, resultant model will not be saved).

There are also other useful options:

- `--iter-num`: number of training iteration on the training corpus,
- `--threads`: number of threads to use,
- `--scale0`: SGD method parameter; fixed between 0.01 and 0.1 gave best results for NER model training task. When scale0 value is too low, SGD might not get close to the optimum point. On the other hand, when the scale0 parameter is too high, training might not converge. Using –pre-train option can help in some situations. Use –evaluate-every=1 option to detect problems with convergence early in the training process.

Run `python scripts/train.py --help` to see complete list of training options.

4.3 NER model evaluation

Cross validation Usage of cross_validate.py script is similar to the usage of train.py script. Additional option -k N can be provided (default: 10-cross validation). Training-specific options will be used to train each of N models during the validation process.

```
$ python scripts/cross_validate.py --tei-corpus=X --schema=Y -k 10 ...
```

Invoke with –help option to see the detailed description.

Error analisys Script error_analisys.py can be used to perform error analisys given NER model and evaluation corpus. The script should be invoked with the same corpus-related arguments (–tei-corpus, –select and –relative-eval-size) as the train.py script used to train the NER model, for example:

```
$ python scripts/train.py   --tei-corpus=X --select=Y --model-out=Z ...
$ python scripts/error\_analisys.py --tei-corpus=X --select=Y --model-in=Z ...
```

Corpus division will be exactly the same and also evaluation parts will be the same in both cases.

Invoke with –help option to see the detailed description.