DELiC4MT
Tutorial

Antonio Toral, Sudip Kumar Naskar, Federico Gaspari

September 3, 2012

1 Introduction

This document contains a step-by-step tutorial for DELiC4MT [1].[1] It shows the way this tool works by applying it to a case study over a specific language pair, test set and linguistic checkpoint.

2 Set-up and Installation of the Required Software

First thing, we will create a folder for all the software, data and experiment files:

```
mkdir delic4mt && cd delic4mt
```

Next we will install all the software required, i.e. DELiC4MT, a word aligner, a PoS tagger, the Kybot engine and preprocessing tools.

DELiC4MT. You might want to check whether there is a newer version at `http://www.computing.dcu.ie/~atoral/delic4mt/`.

```
wget http://www.computing.dcu.ie/~atoral/delic4mt/delic4mt_110818.zip
unzip delic4mt_110818.zip
```

Word aligner, we use GIZA++. Download it:

```
wget http://giza-pp.googlecode.com/files/giza-pp-v1.0.5.tar.gz
tar -xvzf giza-pp-v1.0.5.tar.gz
```

If your version of g++ compiler is 4.4.x (check with `g++ --version`), you need to apply a patch to GIZA++:

```
wget http://www.openmatrex.org/giza-pp.patch
patch giza-pp/GIZA++-v2/file_spec.h <giza-pp.patch
```

Compile GIZA++:

```
cd giza-pp/
make
cd ..
```

[1] `http://www.computing.dcu.ie/~atoral/delic4mt/`

We use TreeTagger for PoS tagging:

```
mkdir treetagger && cd treetagger
wget ftp://ftp.ims.uni-stuttgart.de/pub/corpora/tree-tagger-linux-3.2.tar.gz
wget ftp://ftp.ims.uni-stuttgart.de/pub/corpora/tagger-scripts.tar.gz
wget ftp://ftp.ims.uni-stuttgart.de/pub/corpora/install-tagger.sh
wget ftp://ftp.ims.uni-stuttgart.de/pub/corpora/english-par-linux-3.2.bin.gz
wget ftp://ftp.ims.uni-stuttgart.de/pub/corpora/italian-par-linux-3.2-utf8.bin.gz
sh install-tagger.sh
cd ..
```

TreeTagger not only PoS tags the text but also tokenizes it. We do not want that since our text is already tokenised and retokenisation would break the correspondence of tokens with the word alignment.

```
cd treetagger/cmd
sed s/^\$TOK/#\$TOK/ tree-tagger-italian-utf8 > tree-tagger-notok-italian-utf8
sed s/^\$TOK/#\$TOK/ tree-tagger-english > tree-tagger-notok-english
cd ../..
```

The Kybot engine:

```
svn co https://kyoto.let.vu.nl/svn/kyoto/trunk/modules/mining_module/
```

It depends on dbxml (see the kybot's README for details). It has been tested with dbxml 2.4.16, download and install:

```
wget http://download.oracle.com/berkeley-db/dbxml-2.5.16.tar.gz
tar -xvzf dbxml-*.tar.gz
cd dbxml-*
sudo ./buildall.sh --prefix=/usr/local --enable-perl
cd ..
```

and finally preprocessing tools from Europarl:

```
wget http://www.statmt.org/europarl/v6/tools.tgz
tar xzvf tools.tgz
```

3 Preparing the test data to be evaluated

We will use the test data for Italian–English built in the CoSyne project [2].

```
wget http://www.computing.dcu.ie/~atoral/cosyne/D51_1.1.zip
unzip D51_1.1.zip
```

The two files that we will use for illustration purposes are `en-it.it.test` and `en-it.en.test` (in folder `D51_1.1/data/en_it/`).

The test set is PoS tagged using TreeTagger.

The following pipeline PoS tags the test set:

```
cat D51_1.1/data/en_it/en-it.it.test | perl tools/tokenizer.perl | \
perl delic4mt_*/scripts/treetagger_preserving_tokens_and_lines.pl italian \
2> treetagger_it.log | perl delic4mt_*/scripts/treetagger2kaf.pl -ri > en-it.it.test.kaf

cat D51_1.1/data/en_it/en-it.en.test | perl tools/tokenizer.perl | perl \
delic4mt_*/scripts/treetagger_preserving_tokens_and_lines.pl english \
2> treetagger_en.log | perl delic4mt_*/scripts/treetagger2kaf.pl -ri > en-it.en.test.kaf
```

It follows a sample of the KAF files produced for the Italian–English sentence pair 62 ("[...] la carne americana [...]", "[...] American meat [...]")[2]:

[2] "carne" is the Italian for "meat" and "americana" is the adjective for "American", in-

```
<text>[...]
 <wf wid="w62_4" sent="62" para="1">la</wf>
 <wf wid="w62_5" sent="62" para="1">carne</wf>
 <wf wid="w62_6" sent="62" para="1">americana</wf>
[...]</text>
<terms>[...]
 <term tid="t62_5" type="open" lemma="carne" pos="NOM">
  <span><target id="w62_5"/></span>
 </term>
 <term tid="t62_6" type="open" lemma="americano" pos="ADJ">
  <span><target id="w62_6"/></span>
 </term>
[...]</terms>
```

```
<text>[...]
 <wf wid="w62_3" sent="62" para="1">American</wf>
 <wf wid="w62_4" sent="62" para="1">meat</wf>
[...]</text>
<terms>[...]
 <term tid="t62_3" type="open" lemma="American" pos="JJ">
  <span><target id="w62_3"/></span>
 </term>
 <term tid="t62_4" type="open" lemma="meat" pos="NN">
  <span><target id="w62_4"/></span>
 </term>
[...]</terms>
```

The test set needs to be aligned at word level so that target equivalents of the source-language checkpoints can be identified. This is done by appending the test set to a bigger parallel corpus, i.e. Europarl,[3] because the quality of the alignment depends on the amount of text that is to be aligned. The text is preprocessed with the Europarl tokeniser; then GIZA++ is applied, and finally we get the word alignments of the sentences that make up the test set.

Download the Europarl corpus for Italian–English:

```
wget http://www.statmt.org/europarl/v6/it-en.tgz
tar xzvf it-en.tgz
```

Prepare the word alignment input data by joining Europarl and the test set:

```
cat europarl-v6.it-en.en D51_1.1/data/en_it/en-it.en.test > giza_input.en
cat europarl-v6.it-en.it D51_1.1/data/en_it/en-it.it.test > giza_input.it
```

Perform word alignment (the script `gizapp.sh` provided with DELiC4MT wraps the different phases of GIZA++)

```
delic4mt_*/scripts/gizapp.sh -sl it -tl en -sc giza_input.it -tc giza_input.en \
> alignment.it-en 2> alignment.it-en.log
```

Get the last n sentences of the alignment, those that correspond to the test set (1000 in our case):

```
tail -n 1000 alignment.it-en > alignment.clean.it-en
```

Next the word alignments for the sentence pair presented earlier are shown:[4]

flected for feminine singular (to agree grammatically with "carne") - note that in Italian the attributive adjective normally (though not necessarily) follows the noun to which it refers, whereas in English the opposite sequence is normally used.

[3] `http://www.statmt.org/europarl/`

[4] Note that identifiers in the alignment start from 0 while in the KAF files do from 1.

```
... 5-2           4-3    ... [word alignments]
carne-meat americana-American [tokens]
```

4 Creating a linguistic checkpoint

Kybots are used to extract the linguistic phenomena that are to be evaluated. A Kybot profile specifies which information to extract from the KAF documents. For example the profile that follows extracts under the element "event" the term identifiers of those nouns that are immediately followed by an adjective in the Italian side of the test set. Equivalent tokens in the target corpus of those found by the Kybots in the source are obtained using the word alignments.

```
<Kybot id="kybot_n_a_it">
  <variables>
    <var name="X" type="term" pos="NOM*" />
    <var name="Y" type="term" pos="ADJ*" />
  </variables>
  <relations>
    <root span="X" />
    <rel span="Y" pivot="X" direction="following" immediate="true" />
  </relations>
  <events>
    <event eid="" target="$X/@tid" lemma="$X/@lemma" pos="$X/@pos"/>
    <role rid="" event="" target="$Y/@tid" lemma="$Y/@lemma"
      pos="$Y/@pos" rtype="follows"/>
  </events>
</Kybot>
```

The following commands load the Italian test file in KAF and the Kybot profile:

```
perl ./mining_module/doc_load.pl --container-name docs_it en-it.it.test.kaf
perl ./mining_module/kybot_load.pl --container-name kybots_it kybot_n_a_it.xml
```

Then the Kybot profile can be applied on the KAF document, and the matching terms are output:

```
perl ./mining_module/kybot_run.pl --dry-run --profile-from-db --container-name docs_it \
--kybot-container-name kybots_it kybot_n_a_it.xml > out_n_a_it.xml
```

The following sample of the output shows the term "carne americana", terms 5 and 6 extracted from sentence 62, as it is a noun adjective sequence:

```
<kybotOut>
  <doc shortname="en-it.it.test.kaf">
   [...]
   <event eid="e66" target="t62_5" lemma="carne" pos="NOM" synset="" rank=""
    profile_id="kybot_n_a_it"/>
   <role rid="r66" event="e66" target="t62_6" lemma="americano" pos="ADJ"
    rtype="follows" synset="" rank="" profile_id="kybot_n_a_it"/>
   [...]
  </doc>
</kybotOut>
```

It is optional but recommended to filter out checkpoints for which the source and target tokens do not share the equivalent PoS tags. This reduces the amount of instances removing noisy ones.

```
perl delic4mt_*/scripts/filter_checkpoints.pl -kybot_out out_n_a_it.xml \
-alignment alignment.clean.it-en -kaf_tl en-it.en.test.kaf \
-constraints "NOM=NN;ADJ=JJ" > out_n_a_it.filtered.xml
```

This reads the checkpoint instances from the kybot_out file, and for each of them uses the alignment to get the corresponding terms in the kaf_tl file, and checks whether the PoS tags correspond checking the constraints. Constraints are separated by ;. In the example it will check that any noun in Italian (NOM*) corresponds to a noun in English (NN*) and the same for adjectives (ADJ* = JJ*).

5 Evaluating MT output for a checkpoint

The last step is to evaluate an MT system on the translation of noun adjective sequences from Italian into English. The tool that performs the evaluation is `delic4mt.jar`. It receives as input the word alignments, the source and target KAF files, the output of the Kybot and the output of the MT system. We will evaluate two MT systems:

```
java -jar delic4mt_*/evaluate/delic4mt.jar -alg alignment.clean.it-en -sl_kaf \
en-it.it.test.kaf -tl_kaf en-it.en.test.kaf -lc kybots/out_n_a_it.xml \
-run delic4mt_*/test/enit/mt1.out > mt1_n_a_it

java -jar delic4mt_*/evaluate/delic4mt.jar -alg alignment.clean.it-en -sl_kaf \
en-it.it.test.kaf -tl_kaf en-it.en.test.kaf -lc kybots/out_n_a_it.xml \
-run delic4mt_*/test/enit/mt2.out > mt2_n_a_it
```

The output files provide detailed information about the evaluation of each instance of the checkpoint. The last line provides the final score given to the MT system:

```
tail -n 1 mt*n_a_it
```

Finally, we can calculate whether the difference between the scores is statistically significant:

```
perl delic4mt_*/scripts/lingcheckp_sig.pl mt1* mt2* > stat_mt1_mt2_n_a_it 2>&1
```

6 DELiC4MT in one call

An alternative to running each of tools shown in Sections 4 and 5, is to use `delic4mt.sh`, a script that wraps the call to all of them. This can be very useful to automate the evaluation of different systems over various checkpoints. The user needs to provide a file with several variables (see `/test/delic4mt_vars*` for examples), namely: the checkpoints and systems to evaluate, test files and directories for each of the tools. Once the variables are in place the user can evaluate several systems executing:

```
perl delic4mt_*/scripts/delic4mt.sh delic4mt_*/test/delic4mt_vars.sh
```

References

[1] Sudip Kumar Naskar, Antonio Toral, Federico Gaspari, and Andy Way. A framework for diagnostic evaluation of mt based on linguistic checkpoints. In *Proceedings of the 13th Machine Translation Summit*, pages 529–536, Xiamen, China, September 2011.

[2] Antonio Toral, Federico Gaspari, Sudip Kumar Naskar, and Andy Way. A comparative evaluation of research vs. online mt systems. In Mikel L. Forcada, Heidi Depraetere, and Vincent Vandeghinste, editors, *Proceedings of the 15th Annual Conference of the European Association for Machine Translation*, pages 13–20, Leuven, Belgium, 2011. European Association for Machine Translation.