Tools for Machine Translation Quality Inspection

Jan Berka, Ondřej Bojar, Mark Fishel
Maja Popović, Daniel Zeman

Introduction

This report describes Addicter, tool for automatic error detection and evaluation, providing its user also with graphical interface useful for browsing through the dataset.

Section 1 introduces Addicter, Section 2 describes the installation process, Section 3 shows how to prepare an experiment for evaluation using Addicter, Section 4 gives information about the evaluation process and its results. The graphical user interface is then described in Section 5. Related work is adressed in Section 6.

1 About Addicter

Addicter stands for Automatic Detection and DIsplay of Common Translation ERrors. It will be a set of tools (mostly scripts written in Perl) that help with error analysis for machine translation.

The work on Addicter has started at the MT Marathon 2010 in Dublin, within a broader 5-day project called Failfinder (Dan Zeman, Ondřej Bojar, Martin Popel, David Mareček, Jon Clark, Ken Heafield, Qin Gao, Loïc Barrault). The code that resulted from the project can be freely downloaded from `https://failfinder.googlecode.com/svn/trunk/`. The nucleus that existed just after the MT Marathon (4 Feb 2010) is Addicter version 0.1, to reflect that this was by no means deemed a final product.

In 2011, the viewer was accompanied by an automatic error recognizer and classifier, thanks to Mark Fishel. The development has been moved to ÚFAL StatMT SVN repository (i.e. failfinder.googlecode.com is currently not maintained). Currently, Addicter can do the following:

- Find erroneous tokens and classify the errors in a way similar to Vilar's taxonomy.
- Browse the test data, sentence by sentence, and show aligned source sentence, reference translation and system hypothesis.
- Browse aligned training corpus, look for example words in context.
- Show lines of the phrase table that contain a given word.
- Summarize alignments of a given word. This feature can also serve as a primitive corpus-based dictionary.

- Search and group words sharing the same lemma. That way morphological errors can be highlighted.

The viewing and browsing is performed using a web server that generates web pages dynamically (to avoid pre-generating millions of static HTML documents). Words in sentences are clickable so that the user can quickly navigate to examples and summaries of other than the current word. If you have access to a webserver you may use Addicter with it; otherwise you can use Addicter's own lightweight server. A small subset can be also generated as static HTML files and viewed without a web server: the test data browser.

2 Installation

- Addicter is written in Perl and you need a Perl interpreter to run Addicter. This is usually no problem on Unix-like systems but you may need to install Perl version ≥ 5.8 if you are working on Windows. Options include Active Perl and Strawberry Perl.

- Addicter uses some general-purpose Perl libraries that are maintained in a separate repository. Download these first, using username public and password public. Then make sure that Perl finds these libraries. In Linux/bash, the following commands will do that:

  ```
  svn --username public checkout
  https://svn.ms.mff.cuni.cz/svn/dzlib ~/lib
  export PERL5LIB=~/lib:$PERL5LIB
  ```

  In Windows, you can use TortoiseSVN to access the repository.

- Check out the current version of Addicter from the StatMT SVN repository, again using username public and password public:

  ```
  svn --username public checkout
  https://svn.ms.mff.cuni.cz/svn/statmt/trunk/addicter addicter
  ```

- There are three subfolders, testchamber, prepare and cgi. For every experiment whose data shall be explored by addicter, create a subfolder in cgi, e.g. cgi/fr-en-01

3 Experiment Preparation

This section describes chronologically the set of actions for preparing the machine translation experiment for evaluation using Addicter.

3.1 Data Acquisition

The first obvious step is to get the data. In order to complete the automatic error detection, classification and summarization task, Addicter needs the source text, reference translation and system translation (for which we will use the term *hypothesis*). All text must be sentence-aligned, i.e. source, reference and hypothesis must have the same number of lines and corresponding sentences on equal lines.

To be able to use the Word Explorer (see Section 5.3) a training corpus with source and its reference translation with the alignment between them together with alignment between hypothesis and source text of the experiment are needed. This is covered in more detail in Section 3.4.

For the example experiment in this paper 200 sentences from the English-Czech translation task of the fourth Workshop on Statistical Machine Translation[1] were used as a source and reference translation (English was the source and Czech the target language). The hypothesis was obtained by the TectoMT system [9].

The target side of the corpus (reference and hypothesis translations) can have additional information, for example lemmas and part of speech tags. Addicter needs the tokens to be in this format:

surface form|info1|info2

This is example of one sentence of reference translation of the data introduced above, where the info1 is information, whether the token is punctuation, content or auxiliary word, and the info2 part is token's lemma:

„|punct|" potřebujeme|content|potřebovat víc|content|hodně citu|content|cit a|aux|a víc|content|hodně driblingu|content|dribling ,|punct|, "|punct|" důrazně|content|důrazný řekl|content|říci jol|content|jola .|punct|.

3.2 Experiment Folder

For each experiment, you must make a folder in the /cgi directory. In this case, we create a folder named TectoMT_WMT09. We then move our dataset into it, naming the source corpus test.src, the machine translator output test.system.tgt and the reference translation test.tgt.

File tree of the cgi directory after creating the new experiment looks as follows:

```
cgi
|-- AddicterHTML.pm
|-- ReadFindErrs.pm
|-- TectoMT_WMT09
|-- browsetest.pl
|-- browsetest_static.pl
|-- cu-bojar
|-- de-en-jane
|-- example.pl
\-- index.pl
```

Addicter comes with two sample experiments (the directories cu-bojar and de-en-jane), so everyone can try it even without his own data. The file tree of the experiment TectoMT_WMT09 is very simple:

```
TectoMT_WMT09/
|-- test.src
|-- test.system.tgt
\-- test.tgt
```

As we will continue with this example, the file tree will grow bigger.

[1] http://www.statmt.org/wmt09/

source sentence	once somniloquy appears , try to keep calm .
reference translation	když se objeví somnilokvie , pokuste se zachovat klid .
machine translation	jednou somniloquy se zdá , že zkusí držet klid .
align-hmm.pl output	2-1 4-4 8-8 9-9

Table 1: HMM Alignment Example

3.3 Reference to Hypothesis Alignments

To be able to automatically find and flag errors in translation, Addicter first needs to have reference and hypothesis translations aligned.

There can be multiple alignments for a given experiment. Each alignment has to be in its own subfolder in the experiment folder. The name of the subfolder is then interpreted as the alignment name, when using the graphical user interface (see Section 5). Addicter automatically applies all its alignments to the experiment and creates subfolders for them.

Addicter implements a couple of alignment algorithms. The scripts are stored in the `testchamber` directory. They are:

- `align-hmm.pl`: An alignment based on hidden Markov model
- `align-greedy.pl`: A greedy injective alignment
- `align-lcs.pl`: The alignment is done via the Longest Common Subsequence (LCS) algorithm
- `align-viasource.pl`: reference to hypothesis alignment done via combining the source to reference and source to hypothesis alignments

Commands to run any of the Addicter's internal alignment algorithms all have the same structure.

```
align-xxx.pl [parameters] reference_file hypothesis_file
```

The resulting reference to hypothesis alignment is passed to standard output.

Table 1 shows an example of align-hmm output on single sentence. You can use any alignment tool you wish, as long as its output format is the same.

The user can add his own alignments from other sources (human annotation, Giza++, Meteor and others) manually just by creating a subfolder with the alignment name and placing the alignment there under the name test.refhyp.ali.

Automatic alignment algorithms of course do mistakes, as the task is in principle difficult and even humans cannot often agree on one correct alignment, so for experiments with smaller corpus a tool for manualy editing alignments can be handy. Addicter solves this problem by the script `alitextview.pl`, which transforms the format of an alignment into beautiful tables in plain text, which are easily human-readable and editable.

Addicter uses these alignment markers:

- * – sure
- o – possible
- O – phrasal
- @ – phrasal, sure

- ? – phrasal, possible

The script `dealitextview.pl` serves for switching the format back from text tables. Once you are done with the alignment, put it into a subfolder of your experiment under the name `test.refhyp.ali`. You can have various amount of alignments in one experiment, as long as each has its own subfolder.

3.4 Index Preparation

The word explorer (see Section 5.3) needs index files that will tell it where to look for examples of a particular word. These are created by the indexing script `addictindex.pl`. The indexer needs the following input files:

- train.src – source side of training corpus
- train.tgt – target side of training corpus
- train.ali – alignment of training corpus
- test.src – source side of test data
- test.tgt – reference translation of test data
- test.ali – alignment of the source and reference translation of test data
- test.system.tgt – system output for test data
- test.system.ali – alignment of the source and the system output for test data

Once all the input files are ready, the indexer is invoked as follows (the -o argument is the working directory, where the resulting index files will be stored):

```
addictindex.pl \
    -trs train.en -trt train.hi -tra train.ali \
    -s test.en -r test.hi -h test.system.hi -ra test.ali \
    -ha test.system.ali \
    -o working_folder
```

The indexer splits the output index into multiple files in order to reduce size of any individual file. All index files must be stored in the experiment folder so that the CGI scripts can find them.

However, the indexer simultaneously reads the input files and copies the corpus to the working directory, so it is safer to set the -o option to somewhere else and copy the resulting indexes to the experiment folder afterwards.

4 Experiment Evaluation

After obtaining data, preparing the experiment folder and aligning reference to hypothesis translation, we can run Addicter's automatic error detection and classification algorithm. Indexing the corpus is not necessary for this step.

Error taxonomy is taken from the work [1], which is based on the taxonomy proposed by [8]. The detection is described in [10].

The error detection and classification is done via the script detecter.pl in the `prepare` folder. The usage of this script is:

```
detecter.pl -s srcfile -r reffile -h hypfile \
    [-a alignment] [-w workdir]
```

The outputs of this script are two files called tcali.txt and tcerr.txt. The first one is just copy of the alignment, the second one is a XML file describing found errors and their classes. The outputs should be stored in the folder with used alignment, so Addicter can link errors to the alignment used for their detection, when showing the data in the GUI. When omitting the alignment option, Addicter uses its align-greedy.pl script for reference to hypothesis alignment.

The script rundetection.pl runs automatically all Addicter's alignment algorithms and applies error detection based on them, so the user does not have to make the alignments and copy them to the experiment file structure manually. Note that this script runs all alignments just with implicit parameters, so the user may want to run them manually even so. The script is invoked as follows:

```
rundetection.pl --src=src_file --ref=reference_file \
    --hyp=hypothesis_file [--work=workdir]
```

Addicter can also run the error detection and classification based on Hjerson from [6] and transform its output to the XML format readable by Addicter, so the errors found by Hjerson can be viewed in Addicter's GUI right next to errors found by Addicter itself. The script transforming Hjerson's output to Addicter's XML is hjersoner.pl located in the prepare folder, while the script runhjerson.pl in the same folder runs automatically Hjerson, gets from it the alignment and errors and transforms it all to Addicter format. The usage is following:

```
hjersoner.pl --cat=error.cats --ali=refhyp.ali \
    [--src=source.txt]

runhjerson.pl --ref=reference_file \
    --baseref=base_reference_file --hyp=hypothesis_file \
    --basehyp=base_hypothesis_file \
    [--src=source_file --work=workdir]
```

where error.cats is the output from Hjerson.

5 Graphical User Interface

All the experiment evaluation scripts work in command line interface, but one of the powerful features of Addicter is its graphical user interface allowing the user to easily gain more detailed view on the MT system behaviour.

The GUI uses a web server, that generates web pages dynamically (to avoid pre-generating millions of static HTML documents). If you have access to a webserver you may use Addicter with it; otherwise you can use Addicter's own lightweight server. It is invoked via the script `server.pl`. It will say something like

```
Please contact me at:  <URL:http://localhost:2588/cgi/index.pl>
```

which is the URL you should point your browser to. The server uses a randomly picked port number unless you specify it as a commandline parameter: `server.pl 8080`.

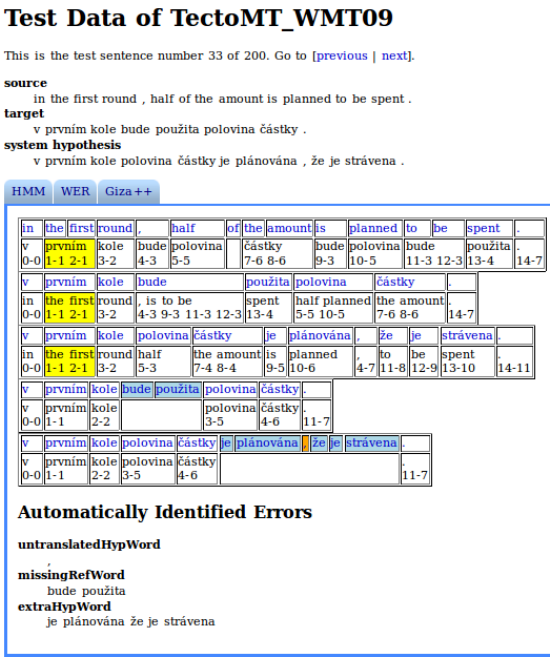


Figure 1: Example of a Sentence in Test Data Browser

The first page you will see is the list of experiments (stored in file structure described in Section 3.2). All experiment names in the list are links to their respective experiment main pages.

5.1 Experiment Main Page

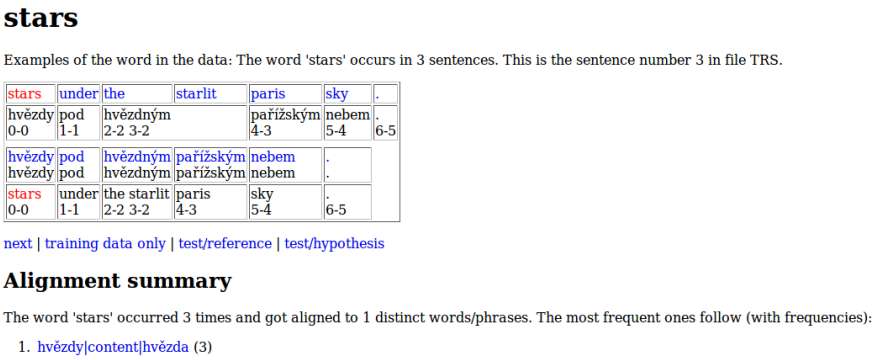
The experiment main page contains name of the experiment, first few sentences, interface for searching for words in the corpus starting with given letters or via Perl-style regular expressions, and links to the test data browser, word explorer and error summary pages.

5.2 Test Data Browser

Test data browser is a tool for easy visualisation of individual sentences in the test dataset. It displays the source text together with the reference and hypothesis translations and their alignments. The detected errors are highlighted (each class with a different color) and also summarized below.

The user can choose the underlying alignment which serves as the basis for error detection. All words in the sentence are clickable and link directly to the Word Explorer (5.3). By moving the mouse over a particular word, all words matched by the alignment are highlighted.

Figure 1 shows a sample sentence in the Test Data Browser. With the use of HMM (Hidden Markov Model) alignment errors of three categories are detected. Clicking on the "WER" tab would display the same sentence with different alignment (probably leading to detection of other errors).



stars

Examples of the word in the data: The word 'stars' occurs in 3 sentences. This is the sentence number 3 in file TRS.

stars	under	the	starlit	paris	sky	.
hvězdy 0-0	pod 1-1	hvězdným 2-2 3-2		pařížským 4-3	nebem 5-4	. 6-5
hvězdy hvězdy	pod pod	hvězdným hvězdným	pařížským pařížským	nebem nebem	. .	
stars 0-0	under 1-1	the starlit 2-2 3-2	paris 4-3	sky 5-4	. 6-5	

next | training data only | test/reference | test/hypothesis

Alignment summary

The word 'stars' occurred 3 times and got aligned to 1 distinct words/phrases. The most frequent ones follow (with frequencies):

1. hvězdy|content|hvězda (3)

Figure 2: The Word “stars” in Word Explorer

The test data browser can be also generated as static HTML files and viewed without a web server.

5.3 Word Explorer

The word explorer enables to browse training and test data and search for sentences containing a given word. It also displays simple statistics of co-occurrence in the data and observed translations.

An example of World Explorer usage is shown in Figure 2. The word “stars” occurred in three sentences in the corpus and got translated to the same Czech word “hvězdy” every time. Word Explorer automatically displays the first sentence with the word “stars” (source text with translation and alignment), enabling quick navigation through the rest.

The user can navigate into the Word Explorer from the Test Data Browser by clicking on any word from the dataset, or directly from the experiment main page (see Figure 3). All words are indexed by their first letter. Thus by clicking on a letter or words starting with it are listed (on the Figure 3 all words in the source data starting with the letter “p” are listed) and their Word Explorer page is then accessible with next click.

The word explorer can be used only if the index was properly built by the `addictindex.pl` script as described in Section 3.4. However, all other functionalities of Addicter are independent of this step and thus Addicter can be reasonably used without the need of training data and bilingual alignments.

5.4 Error Summary

Error summary page contains global, summarized information about the detected errors with respect to given alignment. At the top, there is a table showing absolute and relative counts of detected errors of different classes.

Under the table, there are numbers of sentences containing given error, serving also as direct links to these sentences in the test data browser.

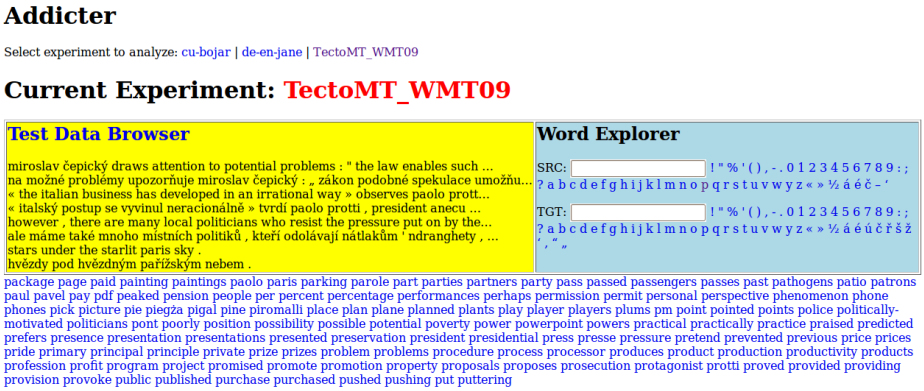


Figure 3: Experiment Main Page

6 Related Work

Meteor [2] is an automatic machine translation evaluation system. It scores hypotheses by aligning them to one or more reference translations. Alignments are based on exact, stem, synonym, and paraphrase matches between words and phrases. Segment and system level metric scores are calculated based on the alignments between hypothesis-reference pairs. Meteor X-ray uses XeTeX and Gnuplot to create visualizations of alignment matrices and score distributions from the output of Meteor. These visualizations allow easy comparison of MT systems or system configurations and facilitate in-depth performance analysis by examination of underlying Meteor alignments. Final output is in PDF form with intermediate TeX and Gnuplot files preserved for inclusion in reports or presentations.[2]

Meteor is written in Java and is released under LGPL (includes some files subject to the WordNet license) and is available for download at `http://www.cs.cmu.edu/~alavie/METEOR/`. Once downloaded and unpacked, you can run it without any arguments (like this: `java -jar meteor-*.jar`, * denoting the version) to get a help message on its usage.

It also has a tool for visualizations, called the Meteor x-Ray (written in Python). The x-Ray displays alignment tables as well as histograms of score distribution either for one or for comparison of two systems

A tool for scoring machine translation with interactive and visual approach is iBLEU [5]. Like Addicter viewer, it runs in web browser (Firefox v4 or higher; v5 recommended). You can use it to explore output of a translation system or compare outputs of two different systems, or compare output to Google Translate or Bing Translator. It is available under the MIT license on `http://code.google.com/p/ibleu/`, where there is also a video showing the use of this tool.

If you download and unpack iBLEU, you can start computing BLEU and viewing different segments just by opening the `bleu.html` file in the browser. It will look like something on Figure 4. After choosing files with source texts

[2]Taken over from `http://www.cs.cmu.edu/\~alavie/METEOR/`

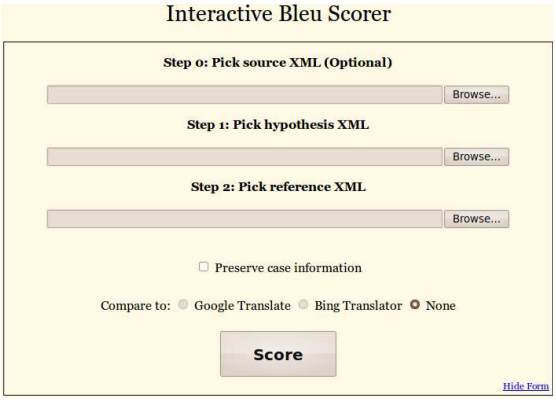


Figure 4: iBLEU start page

(optional), hypothesis and reference translation – all in the NIST XML[3] – and clicking on the "Score" button the overall score will be displayed together with a list of graphs of scores on individual sentences in individual documents. The screenshot is in Figure 5.

You can further explore the individual segments in a document by clicking on the columns in the graph. Information about the segment will appear together with source text, reference translation and hypothesis and their differences highlighted by red color. Figure 6 shows a screenshot from exploring one segment from the demo dataset.

The iBLEU also allows to compare two systems and to easily (by one click) compare your system to Google Translator or Bing Translator systems. However, unlike Addicter it does not classify errors in any way.

Asiya is an open toolkit for automatic machine translation evaluation. It is available at `http://nlp.lsi.upc.edu/asiya/` under LGPL license. It is "a common interface to a compiled collection of evaluation and meta-evaluation methods" (from [3]). Asiya operates on source text and a set of candidate translation and set of reference translations. The tool is written in Perl and operated by giving a main script `Asiya.pl` a path to a text configuration file. The usage is documented on its webpage and in [3] (together with one use case).

Asiya has also its online version (available at `http://falkor.lsi.upc.edu/asiya/asiya_online.php`), which is currently under development, but is usable "to obtain automatic evaluation scores according to a selected set of metric representatives, together with ULC combined score (i.e., arithmetic mean) over a heuristically defined set of metrics"[4]. A screenshot of the online version with user interface is on Figure 7, while Figure 8 shows a screenshot of results over a demo corpus from iBLEU.

Unlike iBLEU or Asiya, Hjerson [6] is a tool which detects and classifies errors into five classes: morphological, reodering, missing words, extra words and lexical. It works with one or more reference translations and one candidate translation together with their corresponding base forms. The output is count

[3]Described in `ftp://jaguar.ncsl.nist.gov/mt/resources/mteval-xml-v1.5.dtd`
[4]from `http://nlp.lsi.upc.edu/asiya/`

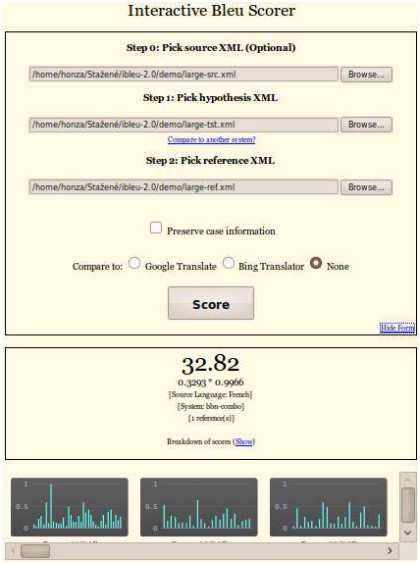


Figure 5: iBLEU overall score

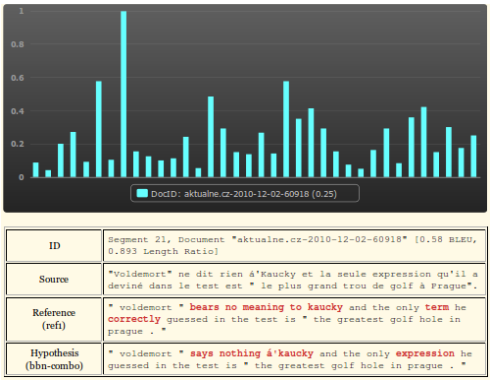


Figure 6: iBLEU sentence score

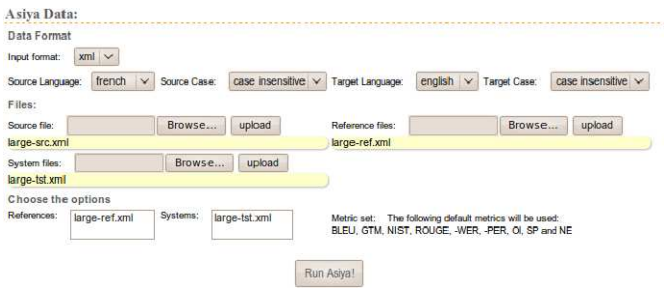


Figure 7: Asiya online tool interface

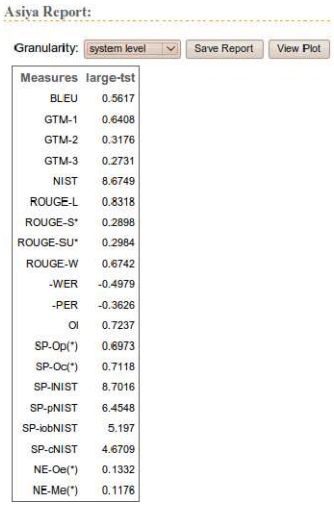


Figure 8: Asiya online tool results

and rate for each error class at the document and sentence level, as well as original translations labelled with corresponding error class in text and HTML format.

Hjerson is publicly available at `http://www.dfki.de/~mapo02/hjerson/` under the GPL license. It is operated by a Python script `hjerson.py`. The usage is very simple, the script needs just paths to file with reference translation, its base form, candidate translation (hypothesis) and its base form, so it looks something like this:

```
./hjerson.py -R reference -H hypothesis -B base form reference -b
base form hypothesis
```

The command `./hjerson --help` will print out the description of all options. Moreover, Hjerson usage is described together with how it works in [6].

Woodpecker[5] is a tool for diagnostic evaluation of machine translation via a set of linguistic tests called checkpoints [11], such as an ambiguous word, noun phrase and others. Woodpecker can extract the checkpoints automatically from source and target sentences. Sadly it is available for Windows only and it is currently aimed primarily at errors in English to Chinese and Chinese to English translation because it relies on a pre-defined set of linguistic phenomena.

The Experimental Management System [4] is a tool allowing smart execution of training and testing of machine translation experiment with one configuration file, automatically detecting re-usable steps for multiple runs with changed settings. It also provides analysis of the experiment run, such as n-gram precision and recall and color-coded output in web browser.

Blast [7] is a tool for qualitative error analysis. It provides a graphical user interface for manual annotation of errors, the user can work with different error classification taxonomies and even define his own taxonomy. Blast can also automatically preprocess the data and find similarities between the reference

[5] `http://research.microsoft.com/en-us/downloads/ad240799-a9a7-4a14-a556-d6a7c7919b4a/default.aspx`

and hypothesis translations, summarize existing annotation in similar way to Addicter error table and save and load annotations. Its main aim is manual annotation.

Acknowledgements

This research has been supported by the grant of the Czech Ministry of Education no. MSM0021620838 (2010), by the grants of the Czech Science Foundation no. P406/11/1499 and P406/10/P259 and the Estonian Science Foundation target financed theme SF0180078s08 (2011).

References

[1] Ondřej Bojar. Analyzing Error Types in English-Czech Machine Translation. *The Prague Bulletin of Mathematical Linguistics*, 95, 2011.

[2] Michael Denkowski and Alon Lavie. Meteor 1.3: Automatic Metric for Reliable Optimization and Evaluation of Machine Translation Systems. In *Proceedings of the EMNLP 2011 Workshop on Statistical Machine Translation*, 2011.

[3] Jesús Giménez and Lluís Màrquez. Asiya: An Open Toolkit for Automatic Machine Translation (Meta-)Evaluation. *The Prague Bulletin of Mathematical Linguistics*, (94):77–86, 2010.

[4] Phillip Koehn. An Experimental Management System. *The Prague Bulletin of Mathematical Linguistics*, 94, 2010.

[5] Nitin Madnani. iBLEU: Interactively Debugging & Scoring Statistical Machine Translation Systems. In *Proceedings of the Fifth IEEE International Conference on Semantic Computing*, 2011.

[6] Maja Popović. Hjerson: An Open Source Tool for Automatic Error Classification of Machine Translation Output. *The Prague Bulletin of Mathematical Linguistics*, 96, 2011.

[7] Sara Stymne. Blast: A tool for error analysis of machine translation output. In *Proceedings of ACL-HLT 2011 System Demonstrations*, pages 56–61, Portland, Oregon, USA, 2011.

[8] David Vilar, Jia Xu, Luis Fernando D'Haro, and Hermann Ney. Error analysis of machine translation output. In *Proc. of LREC'06*, pages 697–702, Genoa, Italy, 2006.

[9] Zdeněk Žabokrtský, Jan Ptáček, and Petr Pajas. TectoMT: Highly modular MT system with tectogrammatics used as transfer layer. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pages 167–170, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[10] Daniel Zeman, Mark Fishel, Ondřej Bojar, and Jan Berka. Addicter: What Is Wrong with My Translations? *The Prague Bulletin of Mathematical Linguistics*, 96, 2011.

[11] Ming Zhou, Bo Wang, Shujie Liu, Mu Li, Dongdong Zhang, and Tiejun Zhao. Diagnostic evaluation of machine translation systems using automatically constructed linguistic check-points. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 1121–1128, Manchester, UK, 2008.