SMT Corpus Tools Documentation

Release 1.0.0

Leo Jiang <leo.jiang.dev@gmail.com>

May 16, 2017

Contents

Welcome! This is the documentation for SMT Corpus Tools 1.0, last updated May 16, 2017.

SMT Corpus Tools aims to provide kinds of tools to process corpus files for machine translation. SMT Corpus Tools is a sub-project of Moses Suite released under BSD 2-Clause License.

Contents:

CHAPTER 1

Introduction

How to install

The steps to install corpus tools:

- Sync the moses suite code from github.
- Copy the folder of subproject corpus-tools to destination folder.
- Set an environment variable PYTHONPATH point to *destination*/corpustools.

Some tools, e.g. corpus clean tool, need to get support from external tools. So we need to install those external tools and write their infos into tools configuration file. Please refer the installation instructions for individual tool in *External corpus tools*.

List of SMT Corpus Tools

- *Moses corpus clean tool*
- *TMX2Text Converter*

CHAPTER 2

Moses corpus clean tool

Overview

A common task for SMT is cleaning up the corpus files. Clean up the long sentences and replace illegal characters before feed the corpus files for training or creating language model in Moses system. Furthermore, we are cleaning/replacing the strings through regular expression according to some rules to improve the trained translation model.

Please refer *clean_corpus Module* module for command line syntax and *Clean Config* for writing your clean steps.

Most of clean steps can be implemented as *Regular Expression Clean*, while others can be implemented as *Predicate Clean* which drop the corpus align if predicate is failed. The predicate clean way simplified the code writing for this kind of clean.

Moses corpus clean tool is designed to be very extensible by external clean modules. User can *Write own clean module* to implement the clean step.

Some external corpus tools is needed, e.g. kinds of tokenizer or segmenter for different languages. These external corpus tools should installed separately, and configured in *corpus tools config*.

Clean Config

A configuration file describe the user-defined clean steps in json format. As in json format, we can edit this configuration file easily, add/remove steps or modify attributes for one step even in a simple text editor.

You can find other attriubtes in the instance of CleanConfig to represent other facters in a cleaning process, e.g. files, directories, languages etc. Please refer *config.clean_config Module* module.

Reference: A sample configuration of clean steps.

Clean Steps

Regular Expression Clean

Most of cleanup are deleting strings or replacing strings in align sentences. User can specify regular expression in configuration. A typical regex clean step should include `description`, `action`, `pattern` at least. The value of action can be `delete_line`, `delete` or `replace`. If action is replace, `repl` is needed. pattern is a regular expression, but in json format every backslash should be escaped, e.g. write the regex `\d` as `\\d` in json cofiguration. The only character '' need to be escaped.

The additional option can be specified:

- `apply_to` indicate which sentence should be cleaned, default Both.
- `unicode` indicate regular expression is unicode awareness. default true.
- `case_sensitive` indicate whether or not search is case sensitive, default false(insensitive).

```
{
  "description": "integer",
  "action": "replace",
  "pattern" : "\\d+",
  "repl" : "num",
  "apply_to": "source",
  "unicode": true,
  "case_sensitive": true
}
```

Predicate Clean

Now we have the following predicate clean modules built in. In predicate clean module, the function `predicate` must be implemented. Please refer *predicate clean* for signatures of these functions.

- clean align beyond the length limit.
- clean wrong sentence ratio align
- clean length diff align

Write own clean module

You can write own clean module to extend the corpus clean tool to support new clean rules. The new clean module should be put into the sub-package `corpustools.clean`, and have an entry function *run(clean, tools, step)*. You can get whole clean information from clean configuration, and get external corpus tools from tools configuration. The parameter step indicate the configuration for current step. Please refer the source code for example.

For predicate clean, corpus corpus tool had implemented the common code, so you only need to provide a function to return True or False according to current sentence align. Also please refer the built-in modules for example.

Module API Documentation

`clean_corpus` Module

Corpus Clean Tool

Clean a bitext file according to clean steps. The corpus file should be like 'path/filename.en-zhcn.bitext'. The config file of clean steps is a json style file. A working directory as well as output directory can be specified in command line, otherwise all intermediate result files will be put in same folder as bitext file.

Users can implement their own cleanup modules with python language, and put modules into folder "corpustools.clean". Most of cleanup steps can be implemented as regular expression clean, some of them can be implemented as predicate clean. Sometimes, we need to run tokenization and lowercasing in cleanup steps. These steps are implemented by calling external tools.

Current support external tools:

- Tokenizer : Stanford Chinese Segmentor (Chinese)
- Tokenizer : Chasen (Japanese)
- Tokenizer : Moses tokenizer (multilingual European languages)
- Caser : Moses case tool

Command line Syntax:

```
Usage: clean-corpus.py [options] corpus_file clean_steps

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -c FILE, --config=FILE
                        specified corpus tools config
  -w DIR, --working-dir=DIR
                        working directory
  -o DIR, --output-dir=DIR
                        output directory

Args:
    corpus_file:    The path to corpus file.
    clean_steps:    Configuration file of clean steps.
```

corpustools.clean_corpus.**main**(*argv*)
entry function.

corpustools.clean_corpus.**argv2conf**(*argv*)
Parse command line arguments, and construct the corpus clean and external corpus tools configuration.

For external tools configuration, read the system-wide and user default configuration file first. If user give a configuration file of external tool in command line, it will be loaded also. A configuration file for cleanup steps must be provided as second argument in command line.

Parameters **argv** – command line arguments.

Returns Exit program if arguments wrong, or failed to construct configuration files. Else return a tuple, corpus tools configuration and cleanup configuration, (corpustools_config, corpus-clean_config).

corpustools.clean_corpus.**clean_corpus**(*corpustools_config*, *clean_config*)
Clean the bitext file.

Copy the corpus file into working directory, run the user-specified clean steps, keep the result for every steps, finally put the clean corpus file into output directory.

corpustools.clean_corpus.**predicate_clean**(*clean_config*, *step*, *predicate*)
Clean the corpus in a way called 'predicate clean'.

Predicate clean can be invoked for those clean rules which only accept or drop the TUs from corpus according result returned by a predicate function (a function return True or False). Drop the align if predicate is True.

config.clean_config Module

clean.regex Module

Regular expression clean module.

corpustools.clean.regex.**run**(*clean_config*, *corpustools_config*, *step*)
entry function.

class corpustools.clean.regex.**RegexClean**(*clean*, *step*)
Class RegexClean run regular expression clean on source and target corpus.

run()
clean the corpus.

compile_relist()
Compile the regular expressions to re objects before using them to improve performance. The compiled pattern is assigned back to clean step to replace the string form of pattern.

relist_clean(*line*)
Clean the line with a list of re steps.

re_clean(*sentence*)
Clean the sentence with clean step, return cleaned corpus sentence.

Parameters **sentence** – unicode string, corpus sentence.

Example of clean step.

```
{
  "description": "delete cdata",
  "action": "replace",
  "pattern" : "CDATA",
  "repl" : "",
  "apply_to": "source",
  "unicode": true,
  "case_sensitive": true,
  "log": "detail"
}
```

re_del(*sentence*, *pattern*)
Return empty string if pattern matched.

Parameters

- **sentence** – unicode string, corpus sentence.
- **pattern** – re object.

re_repl(*sentence*, *pattern*, *repl*)
Return substituted sentence.

Parameters

- **sentence** – unicode string, corpus sentences.
- **pattern** – re object.
- **repl** – unicode string.

predicate clean

corpustools.clean.length_diff.**predicate**(*source*, *target*, *constraint*)
Return True if the distance between source and target is beyond the limit.

corpustools.clean.length_limit.**predicate**(*source*, *target*, *constraint*)
Return True if the length of source and/or target is beyond the limit.

The length limit for GIZA++ in moses is 100 tokens.

corpustools.clean.sentence_ratio.**predicate**(*source*, *target*, *constraint*)
Return True if the sentences ratio is beyond the threshold.

sentences ratio = source length / target length or target length / source length

The threshold of ratio in moses system is 9.

clean.tokenize Module

Tokenize module in corpus clean tools

corpustools.clean.tokenize.**tokenize**(*clean*, *tools*, *step*, *lang*)
Tokenize the corpus files in corpus clean working directroy.

Actually, this function works as router to dispatch the request to tokenizers in token subpackage. The modules in token subpackage are adapters to external tokenizer tools.

Parameters

- **clean** – corpus clean configuration.
- **tools** – external tools configuration.
- **step** – clean step.
- **lang** – specify the language of which corpus be tokenize.

corpustools.clean.tokenize.**run**(*clean*, *tools*, *step*)
Clean module interface function, run tokenization for corpus files.

clean.lowercase Module

Lowercase module for corpus clean tool

corpustools.clean.lowercase.**run**(*clean*, *tools*, *step*)
Clean module interface function, lowercase corpus files.

corpustools.clean.lowercase.**lowercase_corpus**(*clean*, *lang*, *ext*)
Lowercase corpus files, dispatch the lowercase request to lowercase module in corpustools.case.

CHAPTER 3

TMX2Text Converter

Overview

Before feeding the file into moses system for training, we should convert them into plain text first. TMX2Text Converter(tmx2txt.py) is designed for converting TMX file(s) into plain text files in UTF-8 encoding.

Command line syntax:

```
Usage: tmx2txt.py [options] file|directory source_lang target_lang

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -o DIR, --output-dir=DIR
                        output directory
  -l FILE, --log=FILE   log file
  -D, --Debug           logging debug message
```

Module API Documentation

`tmx2txt` Module

`format.tmxparser` Module

TMX Parser Module

class `corpustools.format.tmxparser.`**`TMXParser`**

TMXParser read TMX file and extract the specified languages sentence align.

This tmx parser use xml.parsers.expat as xml parser engine.

`parse_file`(*filename*, *source_lang*, *target_lang*)

Expat parser callback function.

start_element_handler(*name*, *attributes*)
Expat parser callback function.

end_element_handler(*name*)
Expat parser callback function.

char_data_handler(*data*)
Expat parser callback function.

CHAPTER 4

External corpus tools

Corpus tools config

Corpus tools configuration is a ini style file in which we can write the options for tools. The most important option is the path of tool executable program. Other programs, e.g. corpus clean tool, can get the information of these external tools, then call them in apropriate way.

For accessing the value, class CorpusToolsConfig support a more intuitive way, i.e. use section.option as key, e.g. `tools["moses.scripts_path"]`.

You can find a sample of external tools configuration file corpustools.conf from repository in which the path of essential tools (moses scripts and two tokenizer) is configured. It's external tool's responsibility to write the correct info into this configuration file.

External Tools

Tokenizer

For tokenization, we call some external tokenizer for specified language(s):

- The tokenizer Perl script in moses
- Stanford Word Segmenter
- Chasen Japanese Segmenter

You can download the latest release from official website of Stanford Word Segmenter. Put them somewhere, and configure the path in corpus tools configuration. Currently corpus tools will call the script provided by Stanford word segmenter directly.

I re-package the Chasen to simplify the compilation and support corpus files with UTF-8 encoding directly. Please refer another sub-packages chasen-moses in project Moses Suite for detail. As this package need to be built into binary executable, you can follow the instruction in sub-package chasen-moses to build and install it. Or following the instruction to get a pre-compiled binary. After installing, don't forget to configure it in tools configuration.

Module API Documentation

config.corpustools_config Module

token.moses Module

Tokenizer Module for Moses built-in tokenizer

corpustools.token.moses.**tokenize**(*infile*, *outfile*, *lang*, *tools*, *step*)
Call moses built-in tokenizer for corpus.

Moses built-in tokenizer support European languages.

Parameters

- **infile** – input filename.
- **outfile** – output filename.
- **lang** – language of corpus.
- **tools** – external tools configuration.

token.stanford_segmenter Module

Tokenizer Module for Stanford Segmenter

corpustools.token.stanford_segmenter.**tokenize**(*infile*, *outfile*, *lang*, *tools*, *step*)
Call Stanford Segmenter for Chinese text.

Parameters

- **infile** – input filename.
- **outfile** – output filename.
- **lang** – corpus language.
- **tools** – external tools configuration.
- **step** – tokenizer configuration in step.

token.chasen Module

Tokenizer Module for Japanese Segmenter Chasen

corpustools.token.chasen.**tokenize**(*infile*, *outfile*, *lang*, *tools*, *step*)
Call chasen (Japanese Segmenter) for Japanese text.

Parameters

- **infile** – input filename.
- **outfile** – output filename.
- **lang** – language of corpus.
- **tools** – external tools configuration.

CHAPTER 5

Frequently Asked Questions

Looking forward to get feedback from you. Thank you!

CHAPTER 6

ChangeLog

SMT Corpus Tools 1.0

Corpus Clean Tool 1.0

- Add the log for predicate clean and regex clean.

SMT Corpus Tools 0.9

Corpus Clean Tool 0.9

- Support predicate clean
- Support regular expression clean
- Implement lowercase in python which is faster than perl script in moses
- Implement three built-in predicate clean: lenght diff, length limit, sentence ratio.
- Internal language code converter
- Support moses tokenizer, chasen and Stanford Word Segmenter.
- Check whether number of lines of corpus is identical after tokenization.

TMX2Text Converter 1.0

- Parsing TMX file and extract the specified languages sentence align.
- Parsing all .tmx files in a directory.
- Logging the results when parsing tmx files in a directory.
- Check the number of lines in generated corpus.

CHAPTER 7

Copyright

SMT Corpus Tools and this documentation is:

CHAPTER 8

License

SMT Corpus Tools is distributed under FreeBSD License (The BSD 2-Clause License).

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

> Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index