How To: Script calc_amfm.py

Written by
Luis Fernando D'Haro and Rafael E. Banchs
Institute for Infocomm Research – Singapore
July 2016

Purpose of the document:

This document describes briefly how to use the script for calculating the Adequacy Metric (AM) and Fluency Metric (FM) [1], together with a combined score using both.

Installation and Requirements:

Before using the script make sure to install the following programs and packages:

a. Python 2.7
b. Numpy (version 1.11.0 or higher)
c. Sklearn (version 0.17.1 or higher)
d. Scipy (version 0.17.1 or higher)

These programs can be installed on Linux using the following commands:

a. sudo apt-get install python2.7
b. sudo apt-get install python-pip
c. sudo pip install numpy --upgrade
d. sudo pip install sklearn –upgrade
e. sudo pip install scipy –upgrade

Improvements:

This new version is based on a client-server architecture which can speed up the processing time by two main factors:

1. The AM and FM models are loaded just one time in the server and keep in memory while the server is running
2. The server implements some dynamic cache memory (using hashes) to avoid some calculations when the same sentence is used again during the time the server is running. This could highly

speed the time if several submissions sharing similar source or target sentences are tested at the same time and the server is keep running without interruption.

Few other improvements to the algorithm have been done to reduce the processing times. Finally, it is important to mention that we have implemented a multi-threaded server which should be able to handle different clients accessing the service without problems.

Running the programs:

There are two main scripts: a server and client, which are called with the following arguments:

python calc_amfm_server.py –h
Usage: calc_amfm_server.py [-h] [-port PORT] dataset lang

Positional arguments [Compulsory]:
dataset an identifier of the data used to train the models, e.g. ASPEC_JP_EN, BPPT_IN_EN
lang target language available according to the given dataset [e.g. en|jp|ko|in|hi]

Optional arguments:
-h, --help show this help message and exit
-port PORT, --port PORT Port used to listen to the clients

For the client the usage is as follows:
Usage: calc_amfm_client.py [-h] [-port PORT] [-list] [-fm] [-am] ref out lang

Positional arguments [Compulsory]:
ref file with the gold standard sentences
out file with the submission sentences
lang target language [must be the same as the one used to load the models in the server, e.g. en|jp|ko|in|hi]

Optional arguments:
-h, --help show this help message and exit
-port PORT, --port PORT Port used to connect to the server [By default 52000]
-list, --list ref and out files contains lists of parallel submission files to process. This speeds up the process since models are loaded only one time. Submissions must be for the same language
-fm, --fm Do not calculate FM score
-am, --am Do not calculate AM score

In this stage, the program is delivered with predefined AM and FM models for the following datasets and languages:

ASPEC: Japanese and English (jp-en), Japanese and Chinese (jp-zh).
JPO: Japanese and English (jp-en), Japanese and Chinese (jp-zh), Japanese and Korean (jp-ko).
BPPT: Indonesian and English (in-en).
HINDEN: Hindi and English (hi-en).

The server includes specific configuration information when using each of the models. To avoid asking the server with a pair of sentences (reference and submission) with a wrong language, the server requests the client to send information about the language of the pair of sentences and compares that the requested language is the same as the model previously loaded. In case there is a mismatch, the interface shows an error and the client is automatically disconnected.

NOTE: In order to minimize the requirement for language-dependent tokenizers, the server pre-process the submissions and references using a generic tokenizer based on spaces for the following languages: English, Korean, Indonesian and Hindi; for Japanese and Chinese the tokenizer is based on characters. Future systems should include more specific tokenizers as well as additional pre-processing for handling numbers, names, and several punctuation marks.

Usage case 1: Process a single submission for English

Type the following commands in the shell:

Python calc_amfm_server.py –port 1234 ASPEC_JP_EN en

python calc_amfm_client.py –port 1234 ref.txt submission.txt en

Where ref.txt and submission.txt are parallel UTF-8 files containing the English reference and output sentences. Each line in the file corresponds to a single sentence.

Ref.txt	Submission.txt
This is the first line	This is the first line
This is a reference	This is the translated output
This is the reference	But here the system translates another unrelated thing

The output of the command will be this:

On the server:
Listening clients on localhost:1234

```
Starting loading models for language en ...
Loading FM model...
Loading AM model
Finished loading models for language en...
Waiting for connections...

On the client:
Connected to server on localhost:port 1234
********* START PROCESSING SUBMISSION ************
Processing submissions ref=ref_en.txt and output=submission_en.txt
N_SENT,         FM,             AM,             AM_FM
0,              1.00000,        1.00000,        1.00000
1,              0.16549,        0.52744,        0.34647
2,              0.00896,        0.24786,        0.12841

GLOBAL AVERAGE FM:                  0.07990
GLOBAL AVERAGE AM:                  0.55523
GLOBAL AVERAGE AM_FM (0.50):        0.31756
********* END PROCESSING SUBMISSION ************
```

The output shows for each evaluated sentence the FM, AM and combined score, as well as the average score for all the sentences. We can see that for lines containing the same translation (1st line) the AM/FM and interpolated scores (AM_FM) are 1.0, for sentences that are not completely similar (2nd line) the score is lower, while for completely unrelated sentences (3r line) the score is even lower and it could be even 0.0.

Usage case 2: Multiple submissions using a list

In this case, let's assume that the user wants to evaluate multiple systems at the same time all of them for the Japanese language, the recommendation is to use the following command:

python calc_amfm_server.py –port 4321 ASPEC_JP_EN jp [**NOTE**: we run this server on a different port for the Japanese language to keep alive and independent the English server]

python calc_amfm_client.py –port 4321 –list ref_list.txt sub_list.txt jp

In this case, ref_list.txt and sub_list.txt contain the absolute path to the plain text reference and submission txt files to process. Notice that the reference could be the same for all the submissions, but nevertheless the application requires both files to contain the same number of lines and the target language must be the same for all of them. Finally, the number of lines for each reference and submission text files must be the same since they contain parallel sentences. Let's consider the following example:

Ref_list_ja.txt	Sub_list_ja.txt
ABSOLUTE_PATH_TO_JAPANESE_REFERENCE	ABSOLUTE_PATH_TO_JAPANESE_OUTPUT_1
ABSOLUTE_PATH_TO_JAPANESE_REFERENCE	ABSOLUTE_PATH_TO_JAPANESE_OUTPUT_1

In this case, the system will go line by line and process each submission against its reference as in example case 1. The main advantage of using lists is that the models are loaded just one time in memory.

Contact

In case of doubts, comments or improvements regarding this document or the scripts please contact to the authors at: luisdhe@i2r.a-star.edu.sg or rembanchs@i2r.a-star.edu.sg

Bibliography

[1] Banchs, Rafael E., Luis F. D'Haro, and Haizhou Li. "Adequacy–fluency metrics: Evaluating MT in the continuous space model framework." Audio, Speech, and Language Processing, IEEE/ACM Transactions on 23.3 (2015): 472-482.