SMTWithApacheJoshua

Statistical Machine Translation with Apache Joshua (Incubating)

Introduction

The page provides detail on how to use Apache Joshua (Incubating) to undertake statistical machine translation (STM) via the Tika.translate API. This work is the result of development that has taken place both through TIKA-1343 Create a Tika Translator implementation that uses JoshuaDecoder and via close work within the Joshua community.

The benefits of using this approach for achieving language translation through Tika are as follows;

- It's free! As opposed to several other translation services currently available via Tika, STM via Joshua is free. You build the language models, you set up and manage the infrastructure and you have 100% control over the resulting translation
- You are not restricted under some usage ceiling. As there is no paid service, you can use this method completely unrestricted.
- The language model generation and quality are completely transparent. Nowadays a large issue with the use of statistical models (or more generally any models utilized within learning processes) is typically not shared and hence it is difficult to fully quantify or justify the results you get. For example, if we were to use Google Translate, we have absolutely no insight into how the translations are undertaken, what accuracy they achieve, etc. The method and work which is proposed here address this concern entirely. Everything is 100% transparent.

The downsides of using this approach are as follows;

- Joshua, the underlying STM toolkit is quite a complex piece of software. This should by no means be a surprise... after all STM is an extremely difficult and active research area. Some of the world's largest companies e.g. Google, Yahoo!, Bing, IBM, etc are investing large sums of money and significant resources trying to address the issues. The fact that we have STM available via Tika is a huge step towards building the STM open source community.
- Depending upon your translation requirements, you may be required to build your own language models. This however depends on which models are available via the Joshua community. If you do need to build your own models/language packs, this is not exactly a trivial process however you can find loads of help on this topic over on the Joshua mailing lists.
- Depending on the availability of good hardware, you may encounter performance issues. The loading of large language models, STM tasks generally, and building new language packs tend to benefit from powerful machines with lots of RAM. If this is not available then you may encounter issues.

With the above in mind, let us continue with configuring and provisioning Tika for STM with Joshua.

Step 1: Retrieve the Joshua Language Pack

In this example, we will be using a Spanish-to-English n-gram language model pack which was generated on October 6th 2016 and built using BerkeleyLM. For more detail on the language pack itself and how it was produced, see the Language Pack Details.

Grab the language pack and set it up as follows

```
$ cd /usr/local
$ wget "http://cs.jhu.edu/~post/files/apache-joshua-es-en-2016-10-06.tgz"
$ tar -zxvf apache-joshua-es-en-2016-10-06.tgz
$ cd apache-joshua-es-en-2016-10-06
```

To run the language pack, invoke the command

```
./joshua [OPTIONS ...]
```

The Joshua decoder will start running, accepting input from STDIN and writing to STDOUT. Joshua expects its input in the form of a single sentence per line. Each sentence should first be piped through `prepare.sh`, which normalizes and tokenizes the input for the language pack's source language. Here we have a passage.txt for you to try. If you run out of memory then please increase it within the `joshua` script.

```
cat passage.txt | prepare.sh | joshua > output.txt
```

It takes some time (sometimes as much as a minute) to load all of the models into memory, which means there is high latency from startup until the first translation. To reduce this time, Joshua can also be run in server mode, implementing either a direct TCP-IP interface, or implementing a Google-translate style RESTful API. To run Joshua as a TCP-IP server, add the option

```
joshua -server-port 5674
```

You can then connect directly to the socket using nc or telnet:

```
cat passage.txt | prepare.sh | nc localhost 5674 > output.txt
```

Take a look at `output.txt` and you will see your translated passage... pretty cool eh?

You can set the RESTful interface by also passing '-server-type http':

```
joshua -server-port 5674 -server-type http
```

The RESTful interface is used when running the browser demo (see `apache-joshua-es-en-2016-10-06/web/index.html`) or when using the Josh ua Translation Engine.

Step 2: Using the Joshua Translation Engine

In this step we establish a translation engine server written in Python that provides translations of documents http requests as responses to http requests. It provides a very convenient way for establishing a remotely accessible translation service which can be used in a RESTful manner which is exactly what Tika does when configured to use the JoshuaNetworkTranslator implementation.

Lets check out the Joshua Translation Engine source and start the service

```
$ cd /usr/local
$ git clone https://github.com/joshua-decoder/joshua_translation_engine.git
$ cd joshua_translation_engine
$ python app.py -b /usr/local/joshua_resources/apache-joshua-es-en-2016-10-06 -s es -t en -p 5674
...
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

The Python application can also print us basic help instructions to explain the above parameters

```
usage: app.py [options]
Specify at least one bundle and source and target languages for each bundle. The order of -s and -t correspond
to the order of -b options

Start a translation engine server.

optional arguments:
  -h, --help            show this help message and exit
  -b BUNDLE_DIR [BUNDLE_DIR ...], --bundle-dir BUNDLE_DIR [BUNDLE_DIR ...]
                        path to directory generated by Joshua's run_bundler.py
                        script
  -s SOURCE_LANG [SOURCE_LANG ...], --source-lang SOURCE_LANG [SOURCE_LANG ...]
                        the two-character language code of the input text.
  -t TARGET_LANG [TARGET_LANG ...], --target-lang TARGET_LANG [TARGET_LANG ...]
                        the two-character language code of the output text
  -p [PORT [PORT ...]], --port [PORT [PORT ...]]
                        the TCP port(s). Either specify just one port, and the
                        rest of the bundles will start on automatically
                        incremented port numbers, or specify one port number
                        per bundle. Omitting this option defaults to setting
                        the first port number to 56748.
```

Once the above is stable you can then progress with configuring Tika!

Step 3: Configure and Provision Apache Tika

So now let us grab the Tika source, configure, compile and deploy it such that we can utilize Joshua's STM functionality.

```
$ cd /usr/local
$ git clone https://github.com/apache/tika.git
$ cd tika
$ mvn clean install -DskipTests
$ java -jar tika-server/target/tika-server-1.15-SNAPSHOT.jar
...

Oct 25, 2016 10:16:25 AM org.apache.tika.server.TikaServerCli main
INFO: Starting Apache Tika 1.15-SNAPSHOT server
Oct 25, 2016 10:16:26 AM org.apache.cxf.endpoint.ServerImpl initDestination
INFO: Setting the server's publish address to be http://localhost:9998/
Oct 25, 2016 10:16:26 AM org.slf4j.impl.JCLLoggerAdapter info
INFO: jetty-8.y.z-SNAPSHOT
Oct 25, 2016 10:16:26 AM org.slf4j.impl.JCLLoggerAdapter info
INFO: Started SelectChannelConnector@localhost:9998
Oct 25, 2016 10:16:26 AM org.apache.tika.server.TikaServerCli main
INFO: Started
```

We can now utilize the Tika Translate REST API to undertake language translation for us.

```
curl http://localhost:9998/translate/all/org.apache.tika.language.translate.JoshuaNetworkTranslator/spanish
/english -X PUT -T passage.txt
```

The response you receive should be the translated text for the passage you posted. Pretty neat eh?

Contact/Assistance

If you have issues with the Tika components of this document, you can get help from the Tika community mailing lists. If you have issues with the Joshua components of this document, you can get help from the Joshua community mailing lists.

Language Pack Details

Details

The language pack being used in this example was trained on the following bitexts, with a 4-gram language model built over the target side of the bitext. Many of the parallel documents were sources from OPUS, A Collection of Multilingual Parallel Corpora with Tools and Interfaces (http://opus.lingfil.uu.se/).

- Europarl version 7. Proceedings of the European Parliament.
  http://www.statmt.org/europarl/
- DGT: A collection of translation memories provided by the JRC.
  https://ec.europa.eu/jrc/en/language-technologies/dgt-translation-memory
- EMEA. PDF documents from the European Medicines Agency.
  http://www.emea.europa.eu/
- Global Voices. Parallel news stories.
  https://globalvoices.org/
- JRC-Aquis: legislative text of the European Union from 1950 on.
  https://ec.europa.eu/jrc/en/language-technologies/jrc-acquis
- News Commentary v10. News commentaries provided by WMT.
  http://www.casmacat.eu/corpus/news-commentary
- Tatoeba.
  http://tatoeba.org
- Fisher & Callhome Spanish. Translations of conversational Spanish.
  https://catalog.ldc.upenn.edu/ldc2014t23

Benchmark

Test set	BLEU score	Meteor score	Description
Newstest2013	27.46	33.39	News
Fisher dev2	37.13	36.75	Conversational speech
Callhome evltest	32.44	33.39	Conversational speech
Global Voices	36.33	36.40	News

Configuration

```
# MERT optimized configuration
# decoder /export/projects/mpost/language-packs/es-en/5/tune/model/run-joshua.sh
# BLEU 0.2655 on dev /export/projects/mpost/language-packs/es-en/5/data/tune/corpus.es
# We were before running iteration 4
# finished Tue Sep 27 12:03:42 EDT 2016
# This file is a template for the Joshua pipeline; variables enclosed
# in <angle-brackets> are substituted by the pipeline script as
# appropriate.  This file also serves to document Joshua's many
# parameters.

# These are the grammar file specifications.  Joshua supports an
# arbitrary number of grammar files, each specified on its own line
# using the following format:
#
#   tm = TYPE OWNER LIMIT FILE
#
# TYPE is "packed", "thrax", or "samt".  The latter denotes the format
# used in Zollmann and Venugopal's SAMT decoder
# (http://www.cs.cmu.edu/~zollmann/samt/).
#
# OWNER is the "owner" of the rules in the grammar; this is used to
# determine which set of phrasal features apply to the grammar's
# rules.  Having different owners allows different features to be
# applied to different grammars, and for grammars to share features
# across files.
#
# LIMIT is the maximum input span permitted for the application of
# grammar rules found in the grammar file.  A value of -1 implies no limit.
#
# FILE is the grammar file (or directory when using packed grammars).
# The file can be compressed with gzip, which is determined by the
# presence or absence of a ".gz" file extension.
#
# By a convention defined by Chiang (2007), the grammars are split
# into two files: the main translation grammar containing all the
# learned translation rules, and a glue grammar which supports
# monotonic concatenation of hierarchical phrases. The glue grammar's
# main distinction from the regular grammar is that the span limit
# does not apply to it.

tm = phrase -owner pt -maxspan 0 -path model/grammar.gz.packed

# This symbol is used over unknown words in the source language

default-non-terminal = X

# This is the goal nonterminal, used to determine when a complete
# parse is found.  It should correspond to the root-level rules in the
# glue grammar.

goal-symbol = GOAL

# Language model config.
#
# Multiple language models are supported.  For each language model,
# create one of the following lines:
#
# feature-function = LanguageModel -lm_type TYPE -lm_order ORDER -lm_file FILE
# feature-function = StateMinimizingLanguageModel -lm_order ORDER -lm_file FILE
#
# - TYPE is one of "kenlm" or "berkeleylm"
# - ORDER is the order of the language model (default 5)
# - FILE is the path to the LM file. This can be binarized if appropriate to the type
#   (e.g., KenLM has a compiled format)
#
# A state-minimizing LM collapses left-state. Currently only KenLM supports this.
#
# For each LM, add a weight lm_INDEX below, where indexing starts from 0.
# The suffix _OOV is appended to unknown source-language words if this
# is set to true.

mark-oovs = false

# The search algorithm: "cky" for hierarchical / phrase-based decoding,
# "stack" for phrase-based decoding
search = stack

# The pop-limit for decoding.  This determines how many hypotheses are
# considered over each span of the input.

pop-limit = 100

# How many hypotheses to output

top-n = 1

# Whether those hypotheses should be distinct strings

use-unique-nbest = true

# This is the default format of the ouput printed to STDOUT.  The variables that can be
# substituted are:
#
# %i: the sentence number (0-indexed)
# %s: the translated sentence
# %t: the derivation tree
# %f: the feature string
# %c: the model cost

output-format = %S

# When printing the trees (%t in 'output-format'), this controls whether the alignments
# are also printed.

include-align-index = false

# And these are the feature functions to activate.
feature-function = OOVPenalty
feature-function = WordPenalty

## Model weights #####################################################

# For each langage model line listed above, create a weight in the
# following format: the keyword "lm", a 0-based index, and the weight.
# lm_INDEX WEIGHT


# The phrasal weights correspond to weights stored with each of the
# grammar rules.  The format is
#
#   tm_OWNER_COLUMN WEIGHT
#
# where COLUMN denotes the 0-based order of the parameter in the
# grammar file and WEIGHT is the corresponding weight.  In the future,
# we plan to add a sparse feature representation which will simplify
# this.

# The wordpenalty feature counts the number of words in each hypothesis.


# This feature counts the number of unknown words in the hypothesis.


# This feature weights paths through an input lattice.  It is only activated
# when decoding lattices.
feature-function = LanguageModel -lm_order 4 -lm_file model/lm.gz -lm_type berkeleylm
```

```
feature-function = Distortion
feature-function = PhrasePenalty



lowercase = -project-case

lm_0 0.242132004556722
WordPenalty -0.111308832033767
OOVPenalty 0.0101534888932218
tm_pt_2 0.0241130425384253
PhrasePenalty -0.0240605834315291
tm_pt_0 0.0262269656358665
tm_pt_1 0.0535319307753204
Distortion 0.121100027216756
tm_pt_3 0.191275104853519
tm_pt_4 0.119489193983075
tm_pt_5 0.076608826081799
```