# QTLeap Manager

The purpose of `qtlm` (QTLeap Manager) is to make life easier for QTLeap developers working on TectoMT-based translation systems.

Comments and suggestions for improvement are welcome (luis.gomes@di.fc.ul.pt).

*Note:* `qtlm` as well as this document are *work in progress*.

## Installation

Before starting, make sure that you have a working Treex installation. You can find instructions at the Treex web page.

The remainder of this installation guide assumes that you have checked-out the TectoMT repository into `$HOME/code/tectomt` as follows (adjust if necessary):

```
mkdir -p $HOME/code
url=https://svn.ms.mff.cuni.cz/svn/tectomt_devel/trunk
svn --username public co $url $HOME/code/tectomt
```

Pre-requisites/dependencies of `qtlm`:

- TectoMT (>= rev14386)
- Perl (>= v5.14.2)
- Python (>= 3.2.3)
- Bash (>= 4.2)
- Gawk (>= 3.1.8)
- GIZA++ (>= 1.0.7)

Note: the BLEU scores reported for Pilot 1 by the EN-PT system are based on `tectomt` revision 14386.

Extract the *QTLeap Manager* archive `qtlm_rev274.tgz` into `$HOME/code/qtlm`:

```
mkdir -p $HOME/code
tar xzf qtlm_rev273.tgz -C $HOME/code
```

Add the following ling to your `$HOME/.bashrc`:

```
source $HOME/code/qtlm/conf/env/default.sh
```

And, run the previous command on your active terminal before proceeding.

## Usage

If you type `qtlm help` on your terminal you shoud get the following usage summary:

```
Usage: qtlm <command> <args>...

List of commands:

  train
        Trains the transfer models for the current configuration.

  serve
        Starts two MTMonkey workers, one for each translation direction.

  evaluate <src> <trg> [testset]...
        Evaluates current pipeline using given testset or all configured
        testsets if a testset is not specified.

  list scores
        Lists BLEU scores from all evaluations in current directory.

  clean <src> <trg> [testset]...
        Cleans cache files from last evaluation. Use this if you changed
        the <src> languages analysis.

  save <testset> <description>
        Saves a snapshot of the current evaluation of <testset>.
        <description> should be a brief description of what changed since
        last save. Note that <testset> must have been evaluated in both
        translation directions before saving a snapshot.
        Snapshots are uploaded to the share server.

  list snapshots
        Lists all snapshots in reverse chronological order.

  compare [snapshot_id]
        Compare current evaluations with specified snapshot (or with
        latest snapshot if snapshot_id is not given).

  translate <src> <trg>
        Translates text from STDIN (expects one sentence per line) and
        writes translations to STDOUT.
        If environment variable $save_trees is set, then trees are saved
        into the directory specified by the variable.

  help
        Shows this help.
```

```
version
      Shows qtleap script version.
```

*Note:* `save`, `list snapshots`, `compare` commands are under development.

Most of these commands require an environment variable `$QTLM_CONF` to be set. This variable should contain a string with three components separated by a forward slash (`/`):

1. the language pair (in the form of `L1-L2`);
2. the training dataset name;
3. the date when the transfer models were trained (formatted as `YYYY-MM-DD`)

Example: `QTLM_CONF=en-pt/ep/2015-02-12`

The two languages must be lexicographically ordered (`en-pt` is OK, `pt-en` is not). The same configuration identifier is used for both translation directions. According to the `$QTLM_CONF` variable defined above, the file `$QTLM_ROOT/conf/datasets/en-pt/ep.sh` must exist (see Dataset Configuration section below for further details). The date suffix (in this case `2015-02-12`) indicates when the transfer models were trained.

### Training

Please see the relevant configuration for training Training transfer models (both translation directions are trained in parallel):

```
qtlm train
```

The training process will create a directory named

```
train_${DATASET}_${LANG1}-${LANG2}_${TRAIN_DATE}
```

which would be `train_ep_en-pt_2015-02-12` for the previous example. The training process will create several files and sub-directories within that directory. For example, when training models for English-Portuguese, we get the following files and directories:

```
.
‘-- train_ep_en-pt_2015-02-12
    |-- [*] qtlm.info  # contains versioning information about qtlm
    |-- [*] qtlm.stat  # output of "hg stat" on $QTLM_ROOT repository
    |-- [*] qtlm.diff  # unified diff of the $QTLM_ROOT repository
```

```
|-- [*] tectomt.info # contains versioning information about tectomt
|-- [*] tectomt.stat # output of "svn stat" on the $TMT_ROOT repository
|-- [*] tectomt.diff # unified diff of the $TMT_ROOT repository
|-- dataset_files/   # downloaded from central share server
|-- corpus/          # plain text split into chunks of 200 sentences
|-- lemmas.gz        # GIZA input files
|-- giza/            # GIZA itermediate files
|-- alignments.gz    # GIZA final alignments
|-- models/
|   |-- en-pt/       # models for EN to PT transfer
|   |   |-- formeme/
|   |   |   |-- [*] maxent.model.gz
|   |   |   `-- [*] static.model.gz
|   |   |-- lemma/
|   |   |   |-- [*] maxent.model.gz
|   |   |   `-- [*] static.model.gz
|   |   `-- v/       # input vectors for machine learning
|   |-- pt-en/       # models for PT to EN transfer
|   |   |-- formeme/
|   |   |   |-- [*] maxent.model.gz
|   |   |   `-- [*] static.model.gz
|   |   |-- lemma/
|   |   |   |-- [*] maxent.model.gz
|   |   |   `-- [*] static.model.gz
|   |   `-- v/       # input vectors for machine learning
|-- logs/            # logs for all training stages/tools
|-- atrees/          # analytical-level trees
`-- ttrees/          # tectogrammatical-level trees
```

When training is finished, the files prefixed with `[*]` in the above tree are automatically uploaded to the share server into the directory `$upload_ssh_path/$QTLM_CONF`. See Sharing Configuration section for details about `$upload_ssh_path` and related variables.

### Translation

Translating from English to Portuguese (reads one sentence per line from `STDIN` and writes one sentence per line on `STDOUT`):

```
qtlm translate en pt
```

If you want to save the trees of each translated sentence (for debugging purposes for example), then set the target directory in the environment variable `$save_trees`:

```
save_trees=somedir qtlm translate en pt
```

This will read from `STDIN` and write to `STDOUT` as previously, but it will also create a file named `somedir/######.treex.gz` for each input line (`######` is replaced by the number of the line, starting with `000001`).

### MTMonkey XML-RPC workers

To start MTMonkey workers for the current configuration, just run:

```
qtlm serve
```

This will launch a pair of `treex-socket-servers` (one for each translation direction) and a pair of `treex-mtmworkers` which provide a XML-RPC interface to the (plain-text) socket servers. The ports of these 4 servers should be configured in the dataset configuration file. See Dataset Configuration section.

### Evaluation

Evaluating the current pipeline on a specific evaluation set (in this example `qtleap_2a`):

```
qtlm evaluate en pt qtleap_2a
```

For this command to succeed the file `$QTLM_ROOT/conf/testsets/en-pt/qtleap_2a.sh` must exist and define a variable named `testset_files` as described below in Testset Configuration section.

A new directory `eval_qtleap_2a` will be created in the current directory with the following structure:

```
.
‘-- eval_qtleap_2a
    |-- about.txt               # contains versioning information
    |-- qtleap_2a.en2pt.bleu    # output of ‘mteval-v13a.pl‘
    |-- qtleap_2a.en2pt.cache.treex.gz # trees before synthesis stage
    |-- qtleap_2a.en2pt.final.treex.gz # final trees
    |-- qtleap_2a.en2pt.html    # original, reference and MT side by side
    |-- qtleap_2a.en2pt.ngrams  #
    |-- qtleap_2a.en2pt.resume  # output of Print::TranslationResume
    |-- qtleap_2a.en.txt        # original English text
    |-- qtleap_2a.pt_mt.txt     # machine translated (English to Portuguese)
    ‘-- qtleap_2a.pt.txt        # original Portuguese text
```

If you then evaluate on the other direction (Portuguese to English):

```
qtlm evaluate pt en qtleap_2a
```

The following files will be added to the directory:

```
.
‘-- eval_qtleap_2a
    ...
    |-- qtleap_2a.en_mt.txt     # machine translated (Portuguese to English)
    |-- qtleap_2a.pt2en.bleu    # output of ‘mteval-v13a.pl‘
    |-- qtleap_2a.pt2en.cache.treex.gz # trees before synthesis stage
    |-- qtleap_2a.pt2en.final.treex.gz # final trees
    |-- qtleap_2a.pt2en.html    # original, reference and MT side by side
    |-- qtleap_2a.pt2en.ngrams  #
    ‘-- qtleap_2a.pt2en.resume  # output of Print::TranslationResume
```

To evaluate the current pipeline on all evaluation sets listed in `$QTLM_ROOT/conf/testsets/en-pt` just omit the evalset name:

```
qtlm evaluate en pt
```

To list BLEU and NIST scores for all testsets evaluated under the current directory:

```
qtlm list scores
```

Which will output something like:

```
TESTSET          NIST    BLEU    SYSTEM
qtleap_2a.en2pt  5.4622  0.1942  qtleap:en-pt/ep/2015-01-19
qtleap_1a.en2pt  5.7766  0.2290  qtleap:en-pt/ep/2015-01-19
qtleap_2q.en2pt  4.8243  0.1419  qtleap:en-pt/ep/2015-01-19
qtleap_1q.en2pt  4.5370  0.1224  qtleap:en-pt/ep/2015-01-19
```

**Cleaning cached intermediate trees** If you are developing the synthesis and you want to re-evaluate the pipeline you just repeat the above commands to re-synthesize the translations.

The re-runs will be much faster than the first evaluation because `qtlm evaluate` will reuse the previously created `*.cache.treex.gz` files (which contain the trees after analysis and transfer), and only the synthesis step is done.

*However, if you have changed the analysis or transfer steps*, then you should remove the cached trees by running:

```
qtlm clean
```

This will clean the cached trees for all configured testsets that have been already evaluated in the current directory.

### Snapshots

*Note:* snapshots are under development.

A snapshot is a bundle of current evaluations together with all information needed to recover the exact state of the current pipeline.

**Creating a snapshot** To create a snapshot first you must ensure that all configured testsets have been evaluated using the current `$QTLM_CONF` for both translation directions. Then you may run:

```
qtlm save "brief description of what changed since last snapshot"
```

This command will create a new directory `snapshots/YYYY-MM-DDL` (year, month, day, and a letter) within the current directory and it will copy all current evaluations into it.

The value of the `$QTLM_CONF` variable is saved into `about.txt` within the snapshot directory, as well as the current mercurial and SVN revision numbers of `$QTLM_ROOT` and `$TMT_ROOT` respectively, and the current revision of the remote lxsuite service.

Furthermore, uncommited changes to the `$QTLM_ROOT` and `$TMT_ROOT` repositories are also saved in the form of a unified diff (`qtlm.diff` and `tectomt.diff`), allowing us to recover the current source code in full extent.

*WARNING*: only files already tracked by mercurial and SVN will be included in the unified diff of every snapshot, ie, all files appearing with a question mark when you issue the commands `hg status` or `svn status` *WILL NOT* be included in the diff.

The snapshot is also uploaded to the configured share server, making it readily available for comparison and analysis to other users. The URL of a snapshot is `$download_http_base_url/snapshots/LANGPAIR/DATASET/YYYY-MM-DDL`, where `$download_http_base_url` is a configuration variable described in Sharing Configuration, and `LANGPAIR` and `DATASET` are the first two components of `$QTLM_CONF`.

**Listing snapshots** Listing all saved snapshots, from the most recent to the oldest:

```
qtlm list snapshots
```

This will fetch an updated list of snapshots from the share server for the current `$QTLM_CONF`. The list is presented as follows:

```
------------------------------------------------------------------------
 Snapshot      | en2pt | pt2en | Description
---------------|-------|-------|---------------------------------------
 * 2015-02-09a | 12.81 |  6.27 | added some exceptions to the rules
   2015-02-02a |  9.56 |  4.69 | some reordering rules for noun phrases
------------------------------------------------------------------------
```

Columns `en2pt` and `pt2en` show the average BLEU scores over all configured evalsets for both translation directions. Snapshots marked with an asterisk (`*`) exist both locally and on the server. Unmarked snapshots exist only on the server.

**Comparing snapshots** To compare current translations/evaluations with the ones from last snapshot:

```
qtlm compare
```

To compare current translations/evaluations with a specific snapshot (in this case 2015-01-20):

```
qtlm compare 2015-01-20
```

Note: if the specified snapshot does not exist locally (ie, it does not appear marked with an asterisk in the list of snapshots), then the comparison will take longer because the snapshot will be automatically downloaded from the server.

## Configuration

All configuration files are kept in directory `$QTLM_ROOT/conf`.

### Environment Configuration

The shell environment is configured by adding the following line to your `$HOME/.bashrc`:

```
source $HOME/code/qtlm/conf/env/default.sh
```

This file defines and exports the following variables: `QTLM_ROOT`, `TMT_ROOT`, `TREEX_CONFIG`, `PATH`, and `PERL5LIB`. If you installed the qtlm and tectomt repositories into the recommended place (`$HOME/code/qtlm` and `$HOME/code/tectomt`), then you don't have to change this file. Else, you should create a file with your username and source it from your `$HOME/.bashrc` like this:

```
source $QTLM_ROOT/conf/env/$USER.sh
```

### Host Configuration

The file `$QTLM_ROOT/conf/hosts/$(hostname).sh` will be used if it exists, else the file `$QTLM_ROOT/conf/hosts/default.sh` is used instead. Either of these files must define the following variables:

**$num_procs** The maximum number of concurrent processes that should be executed. Specify a number lower than the number of available processors in your machine.
(default: `2`)

**$sort_mem** How much memory can we use for sorting?
(default: 50%)

**$big_machine** Set this to `true` only if your machine has enough memory to run several concurrent analysis pipelines (for example a machine with 32 cores and 256 GB RAM).
(default: `false`)

**$giza_dir** Where GIZA++ has been installed.
(default: `"$TMT_ROOT/share/installed_tools/giza"`)

### Sharing Configuration

Corpora and transfer models are downloaded/uploaded automatically, without user intervention. All data is stored in a central server, which is configured in `$QTLM_ROOT/conf/sharing.sh`:

**$upload_ssh_*** These variables configure SSH access for automatic uploading of transfer models after training. Example:

```
upload_ssh_user="lgomes"
upload_ssh_host="nlx-server.di.fc.ul.pt"
upload_ssh_port=22
upload_ssh_path="public_html/qtleap/share"
```

**$download_http_*** These variables configure HTTP access for automatic downloading of datasets, testsets, and transfer models as needed. Example:

```
download_http_base_url="http://nlx-server.di.fc.ul.pt/~lgomes/qtleap/share"
download_http_user="qtleap"
download_http_password="paeltqtleap"
```

### Dataset Configuration

A dataset is a combination of parallel corpora that is used to train the transfer models. For each `DATASET` we must create a respective file `$QTLM_ROOT/conf/datasets/L1-L2/DATASET.sh` and it must define the following variables:

**$dataset_files** A space-separated list of files (may be gzipped), each containing tab-separated pairs of human translated sentences. The file paths specfied here must be relative to `$download_base_url` configured in `$QTLM_ROOT/conf/sharing.sh`.

Example: `dataset_files="corpora/europarl/ep.enpt.gz"`

**$train_hostname** The hostname of the machine where the transfer models are to be trained. This must be the exact string returned by the hostname command. It is used as a safety guard to prevent training on an under-resourced machine. You may use an `*` to allow training of this dataset on any machine.

**$*_train_opts** Four variables set the options affecting the behaviour of the machine learning algorithms for training each transfer model:

- $lemma_static_train_opts
- $lemma_maxent_train_opts
- $formeme_static_train_opts
- $formeme_maxent_train_opts

Refer to `$TMT_ROOT/treex/training/mt/transl_models/train.pl` for further details. Example:

```
static_train_opts="--instances 10000 \
    --min_instances 2 \
    --min_per_class 1 \
    --class_coverage 1"

maxent_train_opts="--instances 10000 \
    --min_instances 10 \
    --min_per_class 2 \
    --class_coverage 1 \
    --feature_column 2 \
    --feature_cut 2 \
    --learner_params 'smooth_sigma 0.99'"
```

```
lemma_static_train_opts="$static_train_opts"
formeme_static_train_opts="$static_train_opts"

lemma_maxent_train_opts="$maxent_train_opts"
formeme_maxent_train_opts="$maxent_train_opts"
```

**$rm_giza_files** If `true` then GIZA models are removed after the aligment is produced.

**$treex_socket_server_ports** This variable defines the two ports of Treex socket servers (one for each translation direction).

Example:

```
treex_socket_server_ports="7001 7002"
```

**$treex_mtmworker_ports** This variable defines the two ports of Treex MT-Monkey XML-RPC servers (one for each translation direction).

Example:

```
treex_mtmworker_ports="8001 8002"
```

### Testset Configuration

A testset is a combination of parallel corpora that is used to test the whole pipeline. For each `TESTSET` we must create a respective file `$QTLM_ROOT/conf/testsets/L1-L2/TESTSET.sh` and it must define the following variables:

**$testset_files** A space-separated list of files (may be gzipped), each containing tab-separated pairs of human translated sentences. The file paths specified here must be relative to `$download_base_url` configured in `$QTLM_ROOT/conf/sharing.sh`.

Example: `testset_files="corpora/qtleap/qtleap_1a.gz"`

### Treex Configuration

Treex configuration for each user is kept in `$QTLM_ROOT/conf/treex/$USER/config.yaml`. If you wonder why we don't simply use `$QTLM_ROOT/conf/treex/$USER.yaml`, it is because Treex expects its configuration file to be named exactly `config.yaml`.

Here's a Treex configuration (`$QTLM_ROOT/conf/treex/luis/config.yaml`) for guidance:

```
---
resource_path:
  - /home/luis/code/tectomt/share
share_dir: /home/luis/code/tectomt/share
share_url: http://ufallab.ms.mff.cuni.cz/tectomt/share
tmp_dir: /tmp
pml_schema_dir: /home/luis/code/tectomt/treex/lib/Treex/Core/share/tred_extension/treex/resources
tred_dir: /home/luis/tred
tred_extension_dir: /home/luis/code/tectomt/treex/lib/Treex/Core/share/tred_extension
```