Iconic Quality Estimator Manual

Gustavo Henrique Paetzold

1. Introduction

In this document, we introduce the Java Quality Estimation tool developed for Iconic Translation Machines Ltda. The tool is capable of not only training models over annotated data, but also to use such models to predict the quality of unseen translations, and also to evaluate the accuracy of the quality estimates produced.

2. Installation

Since the tool is composed of a compiled JAR file, there is no need for installation. To use it, one needs only to place the IconicQualityEstimator.jar file in a certain folder, and use a terminal or command prompt to explore the tool's functionalities.

3. Source Code

The tool's code is distributed as a NetBeans project, and can be found in the "IconicQualityEstimator" folder. It is compiled over Java's JDK 1.8, and uses two libraries: libsvm and quest-vanilla. To re-compile the project, the user can open the project in NetBeans and use the "Clean and Build" option. A more practical alternative, however, is to compile the project as a "packed" (or "fat") JAR, which means that, when compiled, the JAR file will not require for the libsvm_iconic and quest_iconic libraries to be distributed along with it, since they will be "packed" inside the main JAR itself. To re-compile the project as a packed JAR, the user must follow the steps below:

1. Open the projects in NetBeans.
2. Use the "Clean and Build" function (Shift+F11).
3. Change from the "Projects" to "Files" view.
4. Right-click the "build.xml" file and select the "Run Target -> Other Targets -> package for store" option. Figure 1 illustrates this process.

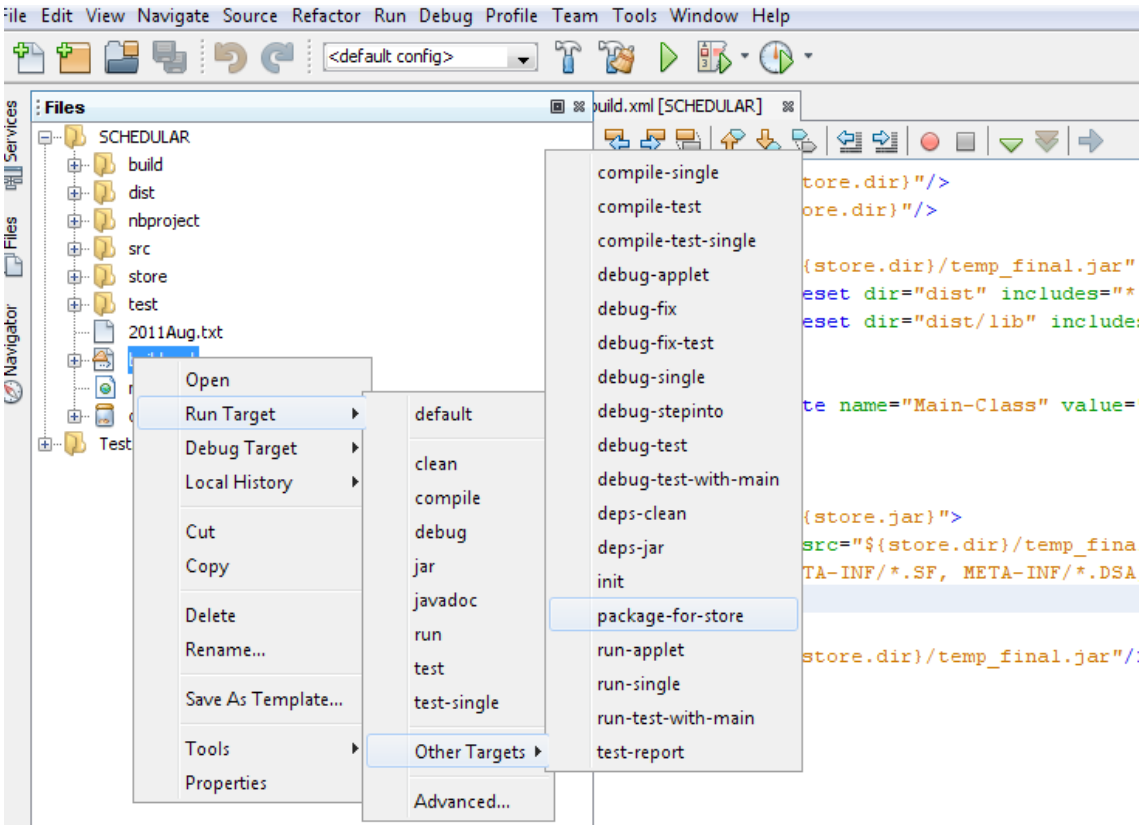

Figure 1 – Packed JAR compilation process

If the steps are performed correctly, the user will find the packed IconicQualityEstimator.jar file in the "store" folder, inside the NetBeans project's root folder. The libsvm_iconic and quest_iconic libraries can also be modified and recompiled if necessary.

3.1. libsvm_iconic

The libsvm_iconic library allows for one to train Support Vector Machine (SVM) models easily. The version used by the Iconic Quality Estimator is personalized, and contains classes and functions modified specifically to fit the requirements of this project. The modified source code of libsvm is also distributed as a NetBeans project, and can be found in the "libsvm_iconic" folder.

3.2. quest_iconic

Like the libsvm_iconic library, quest_iconic is a personalized version of QuEst++ (https://github.com/ghpaetzold/questplusplus), created specifically for this project. It is much lighter than the original version of QuEst++, and supports only sentence-level quality estimation. The source code of quest_iconic is also distributed as a NetBeans project, and can be found in the "quest_iconic" folder.

4. Running

The Iconic Quality Estimator has two main functionalities: quality model training, and quality estimation. To each functionality, an individual callable class have been created. While the "GetQualityModel" class is responsible for training SVM models for quality estimation, the "EstimateQuality" class uses the model produced to estimate the quality of new translations, and the "EvaluateQualityEstimates" class evaluates the accuracy of the quality estimates produced.

4.1. GetQualityModel

The GetQualityModel class takes as input several documents and resources, and produces as output an SVM model trained over the data provided. By running the command line below, one can learn more about the parameters required:

java -cp IconicQualityEstimator.jar main.GetQualityModel -help

If ran correctly, the instructions illustrated in Figure 2 will be presented.

```
usage: GetQualityModel [-C <arg>] [-corpussrc <arg>] [-corpustrg <arg>]
       [-epsilon <arg>] [-features <arg>] [-featurevalues <arg>] [-folds
       <arg>] [-gamma <arg>] [-giza <arg>] [-help] [-kernel <arg>]
       [-langsrc <arg>] [-langtrg <arg>] [-model <arg>] [-ngramsrc <arg>]
       [-scores <arg>] [-source <arg>] [-target <arg>] [-temp <arg>]
Trains a translation quality estimation model.

 -C <arg>               Value for the C hyperparameter.
 -corpussrc <arg>       Source corpus.
 -corpustrg <arg>       Target corpus.
 -epsilon <arg>         Value for the epsilon hyperparameter.
 -features <arg>        XML features file.
 -featurevalues <arg>   File containing pre-calculated feature values.
 -folds <arg>           Number of folds for cross-validation.
 -gamma <arg>           Value for the gamma hyperparameter.
 -giza <arg>            Translation probabilities file.
 -help                  Prints a help message.
 -kernel <arg>          Kernel to be used.
 -langsrc <arg>         Source language.
 -langtrg <arg>         Target language.
 -model <arg>           Path to save trained QE model.
 -ngramsrc <arg>        Source n-gram counts file.
 -scores <arg>          Scores file.
 -source <arg>          Source input file.
 -target <arg>          Target input file.
 -temp <arg>            Folder for temporary files.

This software is a property of Iconic Translation Machines Ltd.
```

Figure 2 – Input parameters for the GetQualityModel class

The resulting trained model will be placed at the value provided for the "-model" parameter.

4.1.1. Resources

The GetQualityModel class requires for certain resources, such as language models, ngram count files and raw text corpora. Such resources must be produced in the following way:

- **N-gram Count Files:** They must be in the format produced by the NGramSorter class included in QuEst++. To produce them, first obtain a raw n-gram counts file from SRILM through the following command line:

  ngram-count –text <corpus> -order <order> -write <raw_counts>

  In sequence, download and install QuEst++ from the website https://github.com/ghpaetzold/questplusplus, and run the following command line:

  java -cp QuEst++.jar shef.mt.util.NGramSorter <raw_counts> <number_of_slices> <ngram_file_order> <frequency_cutoff> <final_counts_file>

  The resulting n-gram counts file will be placed in <final_counts_file>.

- **Source and Target Corpora:** The corpora required must be in plain text format, and must contain one tokenized sentence per line.

- **Features File:** Must be an XML file in the same format as the ones found in the "/config/features" folder in https://github.com/ghpaetzold/questplusplus.

- **Translation Probabilities File:** Must be produced by GIZA++. For a tutorial on how to produce this file, please refer to the tutorial provided at: http://www.opentag.com/okapi/wiki/index.php?title=GIZA%2B%2B_Installation_and_Running_Tutorial.

- **Source and Target Languages:** The languages in which the sentences in the source and target files are written. Both languages must be written in their entirety, such as "english", "spanish", "german" and "chinese".

- **Source and Target Input Files:** Must contain the translations to be used during training. While the source input file must contain original sentences in source language, the target input file must contain one machine translation for each sentence in the source file. Both files must be in plain text, have the same number of lines, and contain one tokenized sentence per line.

- **Scores File:** Must have the same number of lines as both source and target files, and must contain one quality score per line. The quality scores can be inferred from TER scores between the sentences in the source and target files. To learn more on how to do so, please refer to Section 4.3.1.

4.1.2. Learning Options

Using GetQualityModel, the user can train a model in two distinct ways: with, or without cross-validation. For both configurations, the user must provide a valid value to the "-kernel" parameter. The valid kernels supported are:

- linear
- rbf
- poly
- sigmoid

To train a model with cross-validation, the user must provide a value greater than 0 to the "-folds" parameter. The user can then ignore the "-C", "-gamma" and "-epsilon" parameters.

To train a model without cross-validation, the user must assign 0 to the "-folds" parameter, and provide valid floating-point values for the "-C", "-gamma" and "-epsilon" parameters.

4.1.3. Feature Estimation Options

The GetQualitymodel class also allows for flexibility during feature estimation. The user can either require for the class to calculate features for the datasets provided, or it can provide their own feature values. If the user wishes for the class to estimate features itself, it must provide valid values for the following parameters:

- -corpussrc
- -corpustrg
- -features
- -giza
- -langsrc
- -langtrg
- -ngramsrc
- -source
- -target

If the user wishes to provide the feature values themselves, they may ignore all aforementioned parameters and simply provide a valid value for the "-featurevalues"

parameter. The value must be the path to a plain text file containing the feature values to be used. The file must contain the same number of lines as the file provided for the "-scores" parameter. Each line of the file must contain the same number of floating point values separated by a tabulation marker.

The following parameters are mandatory for both settings:

- -scores
- -model
- -temp
- -kernel
- -folds

4.2. EstimateQuality

The EstimateQuality class takes as input a quality estimation model along with several documents and resources, and produces as output quality estimates for the input translations provided. By running the command line below, one can learn more about the parameters required:

java -cp IconicQualityEstimator.jar main.EstimateQuality -help

If ran correctly, the instructions illustrated in Figure 3 will be presented.

```
usage: EstimateQuality [-corpussrc <arg>] [-corpustrg <arg>] [-features
       <arg>] [-featurevalues <arg>] [-giza <arg>] [-help] [-langsrc
       <arg>] [-langtrg <arg>] [-model <arg>] [-ngramsrc <arg>] [-output
       <arg>] [-source <arg>] [-target <arg>] [-temp <arg>]
Produces quality estimates for translations.

 -corpussrc <arg>       Source corpus.
 -corpustrg <arg>       Target corpus.
 -features <arg>        XML features file.
 -featurevalues <arg>   File containing pre-calculated feature values.
 -giza <arg>            Translation probabilities file.
 -help                  Prints a help message.
 -langsrc <arg>         Source language.
 -langtrg <arg>         Target language.
 -model <arg>           Path to QE model.
 -ngramsrc <arg>        Source n-gram counts file.
 -output <arg>          Quality estimates output file.
 -source <arg>          Source input file.
 -target <arg>          Target input file.
 -temp <arg>            Folder for temporary files.

This software is a property of Iconic Translation Machines Ltd.
```

Figure 3 – Input parameters for the EstimateQuality class

As output, the class will produce a file with quality estimates in the path provided for the “-output” parameter. The file will have the same amount of lines as both source and target input files, and will contain one quality estimate per line. The format of all files required by the parameters is the same described in Section 4.1.1.

4.2.1. Feature Estimation Options

Much like the GetQualityModel class, the EstimateQuality class also offers support for both pre-computed and automatically generated features. If the user wishes for the class to estimate features itself, it must provide valid values for the following parameters:

- -corpussrc
- -corpustrg
- -features
- -giza
- -langsrc
- -langtrg
- -ngramsrc
- -source
- -target
- -temp

If the user wishes to provide the feature values themselves, they may ignore all aforementioned parameters and simply provide a valid value for the “-featurevalues” parameter. The value must be the path to a plain text file containing the feature values to be used. Each line of the file must contain the same number of floating point values separated by a tabulation marker. Note that the features used must be the same ones used to train the model provided for the “-model” parameter.

The following parameters are mandatory for both settings:

- -model
- -output

4.3. EvaluateQualityEstimates

The EvaluateQualityEstimates class takes as input a set of predicted quality estimates along with a gold-standard, and produces as output several statistics about the accuracy of the estimates provided. By running the command line below, one can learn more about the parameters required:

java -cp IconicQualityEstimator.jar main.EvaluateQualityEstimates -help

If ran correctly, the instructions illustrated in Figure 4 will be presented.

```
usage: EvaluateQualityEstimates [-est <arg>] [-help] [-ref <arg>]
Evaluates the quality of predicted quality estimates.

 -est <arg>   File containing predicted quality estimates.
 -help        Prints a help message.
 -ref <arg>   File containing gold-standard quality estimates.

This software is a property of Iconic Translation Machines Ltd.
```

Figure 4 - Input parameters for the EvaluateQualityEstimates class

Both the gold-standard and predicted estimates file must have the same number of lines, and must contain one Iconic QE Score per line. The Iconic QE Scores required and the Iconic QE Accuracy metrics produced as output are a novelty, and have been developed specifically for this project. Figure 5 shows an example of the output produced.

```
Iconic QE Sum: 10932
Iconic QE Mean: 0.4822232024702222
Iconic QE Standard Deviation: 0.9609320239724449
Iconic QE Gravity Counts:
        0: 17836
        1: 274
        2: 3022
        3: 1538
```

Figure 5 – Output produced by the EvaluateQualityEstimates class

In the following Sections, we describe the Iconic QE Scores and Iconic QE Accuracy metrics in more detail.

4.3.1. Iconic QE Scores

The Iconic QE Score is a novel metric that describes the quality of a given machine translation. It is composed of an integer score in the [0, 5] range, where 0 describes a perfect translation, and 5 an entirely unusable translation. The scores can be inferred from TER scores between a machine translation and a perfect reference translation. The transformation function between the TER scores and the Iconic QE Score is described in Table 1.

TER Range	Iconic QE Score
0	0
1~20	1
21~40	2
41~50	3
51~70	4
71+	5

Table 1 – TER to Iconic QE Score function

4.3.2. Iconic QE Accuracy

The Iconic QE Accuracy metrics are a novel strategy for the evaluation of QE models. All metrics are calculated with respect to the Iconic QE Matrix, illustrated in Table 2. In the matrix, a column represents a gold-standard Iconic QE Score *A*, a line represents a predicted Iconic QE Score *B*, and the cell value represents the error gravity associated to predicting *B* instead of *A*.

	0	**1**	**2**	**3**	**4**	**5**
0	0	0	0	1	2	2
1	0	0	0	1	2	2
2	0	0	0	0	2	2
3	1	1	0	0	0	2
4	3	3	3	0	0	0
5	3	3	3	3	0	0

Table 2 – The Iconic QE Matrix

Given a set of gold-standard and a set of predicted scores, the following accuracy metrics can be calculated:

- **Sum:** The sum of the gravity errors obtained.
- **Mean:** The average between the gravity errors obtained.
- **Standard Deviation:** The standard deviation of the gravity errors obtained.
- **Gravity Counts:** The number of times errors of a certain gravity were made.

5. Documentation

The Iconic Quality Estimator also has a complete Javadoc documentation of every class included in the tool. The documentation is present in the “javadoc” folder of the tool’s NetBeans project.