**Java 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.

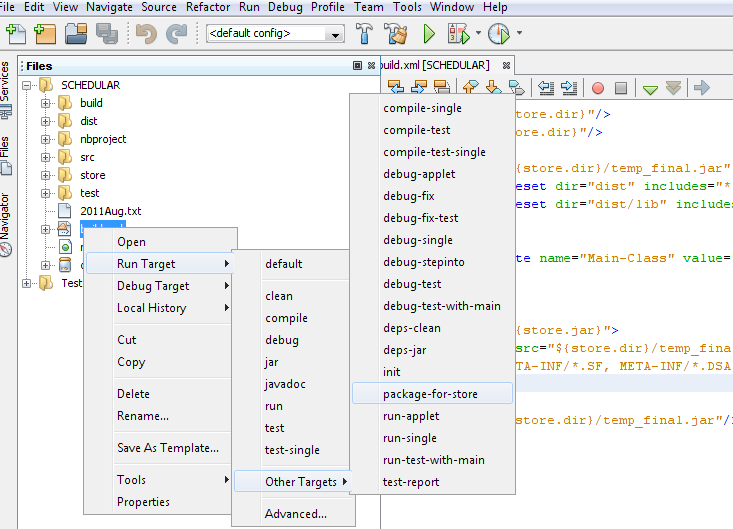
**2. Installation**

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 and quest-vanilla 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 and quest-vanilla libraries can also be modified and recompiled if necessary.

**3.1. libsvm**

The libsvm 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-modified” folder.

**3.2. quest-vanilla**

Like the libsvm library, quest-vanilla 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-vanilla is also distributed as a NetBeans project, and can be found in the “quest-vanilla” folder.

**4. Running**

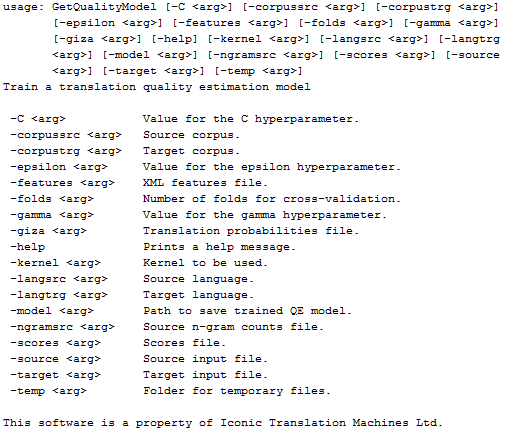
The Iconic Quality Estimator has two main functionalities: quality model training, and quality estimation. To each functionality, an individual callable class have been created. While the “GetQualityModel” class is responsible for training SVM models for quality estimation, the “EstimateQuality” class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

**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.



**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 Chapter xxx.

**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.

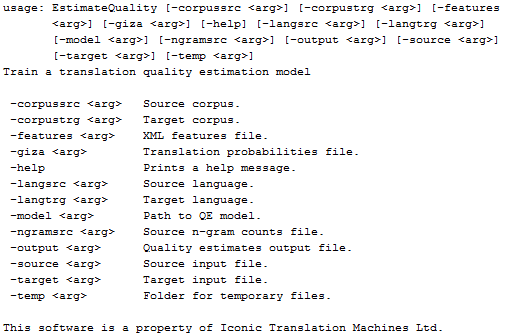
To train a model without cross-validation, the user must assign 0 to the “-folds” parameter, and provide valid floating-point values for the “-C”, “-gamma” and “-epsilon” parameters.

**4.2. 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.

****

**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.

**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.

**6. Iconic Quality Estimates**

For this project, we have