PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 109 OCTOBER 2017 51–60

QE::GUI – A Graphical User Interface for Quality Estimation

Eleftherios Avramidis

German Research Center for Artificial Intelligence (DFKI), Language Technology Lab, Berlin, Germany

Abstract

Despite its wide applicability, Quality Estimation (QE) of Machine Translation (MT) poses a difficult entry barrier since there are no open source tools with a graphical user interface (GUI). Here we present a tool in this direction by connecting the back-end of the QE decision-making mechanism with a web-based GUI. The interface allows the user to post requests to the QE engine and get a visual response with the results. Additionally we provide pre-trained QE models for easier launching of the app. The tool is written in Python so that it can leverage the rich natural language processing capabilities of the popular dynamic programming language, which is at the same time supported by top web-server environments.

1. Introduction

Understanding the performance of Machine Translation (MT) is a necessary task for the development of MT systems and assessing the usability of their output. A wide variety of methods has given the possibility to analyze the results of the MT output and provide useful insights about the quality of the results. Already from the first years of MT, automatic evaluation metrics have provided scores representing the quality of the translation by scoring the MT output against the references (Papineni et al., 2002; Lavie and Agarwal, 2007). Recently, the efforts on MT evaluation have been expanded to the field of Quality Estimation (QE) which is capable of providing numerical qualitative judgments on the MT output without access to reference translations (Blatz et al., 2004; Specia et al., 2009). Being independent of reference translations, QE can be useful in real-life applications. Additionally, its inherent diverse linguistic features, combined with machine learning, can provide deeper and more specific qualitative indicators.

 Corresponding author: eleftherios.avramidis@dfki.de
Cite as: Eleftherios Avramidis. QE::GUI – A Graphical User Interface for Quality Estimation. The Prague Bulletin of Mathematical Linguistics No. 109, 2017, pp. 51–60. doi: 10.1515/pralin-2017-0038.

Whereas most methods for the development and evaluation of MT, including QE, have been available through open-source tools, there has been little effort towards making them easily applicable and user-friendly. Being oriented to researchers, most software can only be executed in the commandline of Linux environments and the installation effort can be non-trivial. Therefore, the use of these tools is confined to a relatively small group of experienced developers, while the entry barrier for newcomers is considerable. Additionally, despite the many levels of qualitative analysis, most results are presented only on a high level, with limited options for visualization. With QE::GUI, the software described in this paper, we attempt to solve some of these issues. We focus on QE and we aim at providing a graphical user interface (GUI) on top of common QE software. The GUI is offered via a web-based framework that can be hosted in any web-server and can be operated with a common browser. It provides the possibility for users to upload their data, have them analyzed with QE models and have visualized statistics over the results.

It must be noted that instead of providing a full-flegded solution, here we mostly aim at setting the standards for further development by suggesting a particular well-structured framework. Therefore, the current functionality is limited to only a few cases of QE analyses, that should nevertheless be easily extended to cover more scenarios depending on the needs of the research and potential end-users. Despite its limited initial functionality, the software complies with several engineering principles that allow extensibility and easy addition of features and functionality. Additionally, the entire framework is built within a unified architecture, and the backbone is written in a single programming language (Python), including the web interface, the database layer and the computational back-end.[1]

This paper is accompanying the open-source code[2] in order to provide more details on the functionality of the software. After comparing our efforts with previous works (Chapter 2), we outline the basic usage and the capabilities of the software (Chapter 3). The architecture and the design is given in Chapter 4. Finally, ideas for further work and conclusions are outlined in Chapters 5 and 6 respectively.

2. Related Work

Several tools related to MT evaluation provide graphical interfaces in order to aid the understanding of the scored MT Quality. First, we shortly review some systems which provide graphical user interfaces related to reference-based automatic metrics.

The **Experiment Management System** (EMS; Koehn, 2010) is a tool for organizing the training and testing of MT models based on the popular statistical system

[1]The web interface contains non-Python elements such as HTML templates and CSS stylesheets, whereas embedded external tools also use other languages.

[2]Code: `https://github.com/lefterav/qegui`
Demo of basic interface: `http://blade-3.sb.dfki.de:8100/qe/` username: `admin` ; password: `pbml2017`

Moses (Koehn et al., 2007). For the models tested on particular test-sets, it offers a web-based interface that displays the BLEU score per test-set. The output for every system can be further examined by displaying additional sentence-level statistics. EMS is a pipeline that combines scripts in different programming languages and the information is communicated with intermediate files. The web interface is written in PHP (albeit with no database layer), whereas the back-end scripts are mostly written in Perl and Bash. **ComparEval** (Klejch et al., 2015) is a tool that allows uploading files with MT output and displays tables and graphs on the performance of the system. Multiple MT systems can be compared in parallel with bar charts, while the progress of consequent development versions of the same system can be visualized through a linear chart. The tool also offers pairwise comparison between system pairs and sentence-level analytics, including sentence-level scores, word-level error annotations and n-gram scoring.

The only tool that offers some GUI for QE is a particular version of QuEst named QuEst-Online, as presented by Shah et al. (2014). This version allows users to access the tool remotely from a web browser. They can upload their source and translated text, which are consequently analyzed to produce qualitative features. These features are used with a pre-built QE model in order to predict an estimated quality score for the translated sentence, including ranking of multiple translations. The entire system combines three modules in several programming languages: PHP for the web interface, Java for the feature generation and Python for the machine learning, as opposed to our tool, which organizes all steps with the same programming language. Finally, in contrast to QuEst-Online, our tool offers visualizations and graphs for the results, an administration interface with configuration options for the models, whereas the requests are stored in a database and organized in a hierarchical structure, allowing future retrieval and examination of previous evaluations.

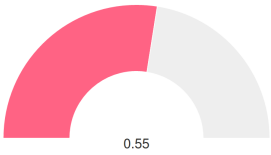
3. Usage

QE::GUI allows the users to submit documents translated by MT, including the source text and the MT output. These documents are analyzed and as a response, the user is given a set of statistics and graphs on the estimated quality of the translation.

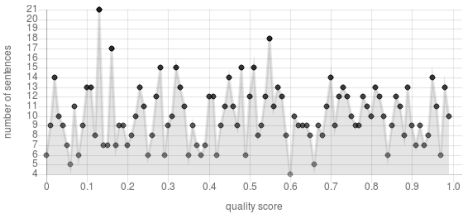
3.1. Organization of evaluation tasks

The interface is capable of storing the evaluation results in the long term, so that they can all be inspected in the future. Additionally, the user can apply hierarchical categorization to their uploads in order to maintain a clean structure of the contents.

The **document** is the basic organizational unit of the translations. Every document is a collection of source sentences with their respective MT outputs. The contents of a document can be uploaded from text files or inserted manually. All translated sentences in one document should be of the same language pair. The documents



(a) A gauge indicating the average score of all sentences in the document



(b) A linear area chart indicating mass distribution of the quality scores

Figure 1: Sample charts from the QE results

can be organized in evaluation tasks. Every **task** is effectively a folder which can contain many documents of different language pairs and sizes. The organization of documents into tasks is directly inflected on how the user can access the documents through the interface. After the list of the documents, the user is given the option to add an additional document.

3.2. Results and statistics

Once the document has been uploaded, a separate document page is created and the system starts the necessary operations for analyzing the sentences and performing the necessary calculations. When the calculations are finished, the document page gets populated with statistics about the estimated translation quality.

We show here two basic statistics accompanied with charts. For every evaluated document the page displays first the **average predicted score**, i.e. the mean score of all MT outputs. This is presented with a gauge which indicates the estimated score, as compared to the full range of the score. For instance, a predicted HTER score of 0.55 would be indicated by a gauge pointing a little more than half way in the range between 0 and 1 (Figure 1a). The indicator of the gauge can change color, depending on the level of quality.

Another possibly useful visual representation refers to the mass distribution of the sentence quality scores (Figure 1b). A linear chart indicates the amount of sentences that have been assigned to each possible quality score. This can be an indicator for how the quality of the translations ranges within the document. The conclusions can be complemented by observing a pie chart which indicated the distribution of the sentences scores among the 4 quartiles.

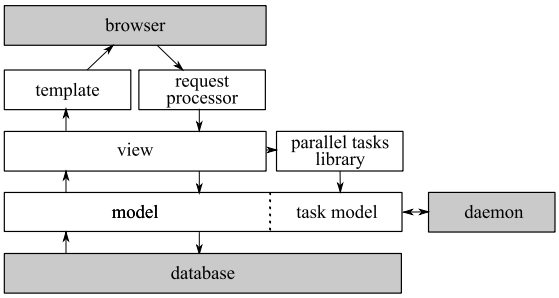


Figure 2: Generic internal architecture

3.3. Administration interface

The basic user interface has been deliberately designed so that it only contains the basic functionality for analyzing documents and observing the results of QE. Nevertheless, there is the possibility for advanced parametrization that falls out of the scope of simple usage. This is done through an advanced administration interface which contains functionality such as adding tasks, manually adding document sentences and translations, deleting tasks and documents and defining supported QE models and translation systems.

3.4. Deployment

The software can be deployed as a Python package and requires the creation of a Python virtual environment and the installation of some additional Python libraries. The QE framework may also require the installation of external natural language processing (NLP) required for the analysis of the text, prior to the application of machine learning. In its easiest form, the software can be run from a Django-compatible commandline bound on a SQLite database. For advanced scaling and better performance, it can be served as a proper website through common web servers (Apache, Nginx) with a more advanced database engine (MariaDB, PostgreSQL).[3]

4. Implementation

In this chapter we explain the main modules of the implementation. The generic architecture of the internal functionality is shown in Figure 2.

[3]Apache: `https://httpd.apache.org`, MariaDB: `https://mariadb.org`, Nginx: `https://nginx.org`, PostgreSQL: `https://www.postgresql.org`, SQLite: `https://www.sqlite.org`

4.1. Database structure

The implementation of the web-based interface and the underlying database structure is based on the latest version of the Django framework (ver. 1.11).[4] This has been chosen in order to take advantage of its powerful database modeling and its robustness, given the support of a wide developer community. Additionally, a wide range of libraries are easily available to extend the functionality. Last but not least, Django is fully Python-based, which makes it compatible and integrable with other Python applications. This is of a particular interest when it comes to QE, since several known open-source QE toolkits use Python (e.g. SciKit-Learn; Pedregosa et al., 2011) for their machine learning back-end.

As part of the Django framework, the web-based interface is built around a set of `models`, which are high-level representations of relational database tables. A simplified graphical representation of the models can be seen in Figure 3. To store the document structure presented above, we use the models `Task` and `Document`. A `Sentence` is the model which stores every source sentence and is associated with one or more MT outputs and reference translations through the models `MachineTranslation` and `ReferenceTranslation` (the latter is optional) respectively. Predicted quality scores for every MT output are stored in a `MachineTranslationEvaluation` model. The database also offers support for different evaluations of the same document (e.g. by different QE models),[5] which are organized by pointing from the separate `MachineTranslationEvaluation` instances to a `DocumentEvaluation` model.

4.2. Serving the data to the user

In order to allow access to the models, the framework employs a set of *v*iews (roughly a view for every page), which send queries to the models to retrieve the data and structure it as required for display. This data is then given to a set of templates, which take care of the setting text, objects, widgets and other elements in the final page shown to the user. To ease the representation of the page we use the responsive template framework Bootstrap, that reorders page elements depending on the screen size of the user's device. For creating the graphs, the data is passed to the open-source Javascript library JChart[6] which renders them in the HTML code of the page.

[4]Django :`http://djangoproject.com`

[5]The functionality of multiple evaluations is not available in the current preliminary version but is built into the database so that it can be extended in a next version.

[6]`https://django-jchart.matthisk.nl`

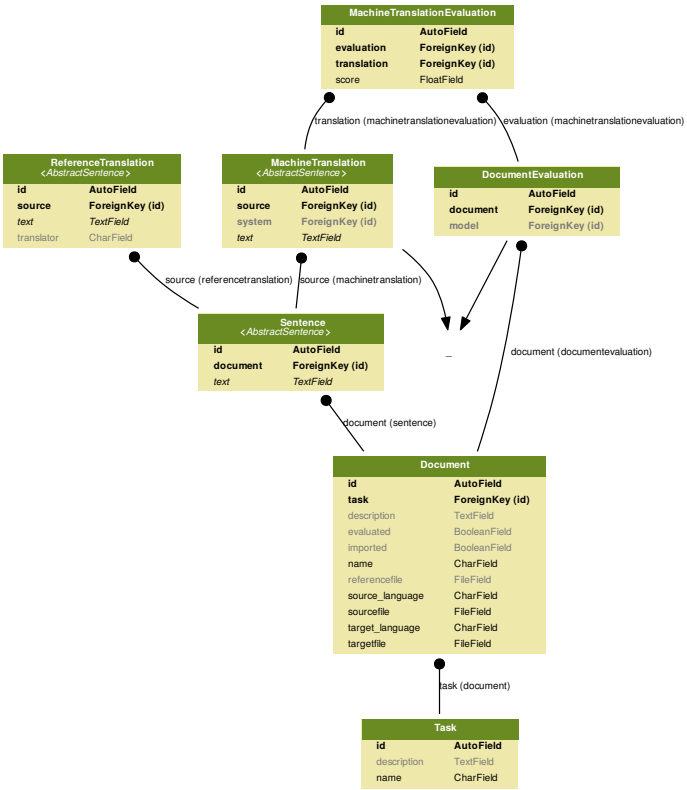


Figure 3: The structure of the Models that supports the web interface

4.3. Asynchronous data processing

A known issue for large NLP tasks is their scalability and low response times are a requirement for real-use applications. In our case the user is expected to upload a potentially large text file which contains source sentences and translations. The system is expected to store the sentences into the respective models of the database and then start processing them to estimate the quality scores. In order to avoid deadlocks and timeouts, we draw a line between the front-end (i.e. the interaction with the user) and the back-end (i.e. the processing of the data). The back-end tasks operate asynchronously, so that the loading of the front-end is not disturbed by time-consuming data-intensive processes. This way, the user can navigate away or close the document page but the processing they have requested will keep running in the background.

The implementation of the asynchronous processing uses the Django library `background tasks`. The steps followed are:

1. Every time the user uploads a new document, the library `background tasks` is instructed to create a new **asynchronous data processing task**.[7] The task is stored in a queue in the database, along with the parameters of the function and how it should be executed.
2. `Background tasks` provides a **background daemon**, which is then launched as a separated executable. The daemon regularly checks the queue and executes the pending tasks.
3. Having access to the Django database schema, the asynchronous tasks can deliver their results by populating the same models.
4. When every background task is finished, it notifies the front-end by enabling a boolean field in the relevant models.
5. The next time the front-end accesses the requested document evaluation, depending on the flag, will either proceed with visualizing the results, or display a message to the user to try again later.

It is also noteworthy that the daemon can run the queued tasks in parallel by launching several threads. The separate existence of the daemon also allows a distributed server set-up, where one server is responsible for the front-end and another one takes care of the back-end, but they both connect to the same database and use the same codebase.

4.4. Quality Estimation

In the last phase, a QE pipeline is triggered in order to process the data and deliver the estimation result. This pipeline consists of two major steps, that are accomplished by the existing QE toolkit: the feature generation and the application of a machine-learned model. During the feature generation, the quality estimation toolkit applies many NLP processes (e.g. language model scoring, parsing) in order to deliver numerical qualitative indicators (features). Then, these numbers are provided to a pre-trained QE model that delivers a *quality score*, i.e. a numerical judgment about the quality. The estimated quality score can then be stored in the model, associated with the respective MT output(s).

For the QE part, QE::GUI integrates existing code from other state-of-the-art Python-based tools. Qualitative (Avramidis, 2016) is used for the feature generation and prediction of sentence-level ranking, whereas models produced by the popular state-of-the-art QuEST (Shah et al., 2013) can be used for continuous score prediction. The database structure allows for loading and storing other Python-based QE models, provided that their usage is clearly documented.

[7] not to be confused with the *task* used for grouping documents in the part of the user interface

5. Further work

The presented software can cover functionality related to some basic use of QE and should still be considered as a preliminary version. Nevertheless, our aim is to suggest a unified framework that makes good use of state-of-the-art tools and is easily extensible to cover future needs. Further development can be directed towards covering more use-cases, including various types of QE (e.g. sentence binary filtering, word/phrase-level scoring). The graphical representation can be extended to compare the performance of different MT systems. Reference-based metrics can be integrated so that their scores can be displayed alongside the ones by QE, when reference translation are available. With the consideration of golden quality scores, this interface could also be used for automatically evaluating different QE approaches with correlation metrics. The principles of modularity and extensibility, along with the Git repository allow for wider collaboration and expansion of the development in a community scale.

6. Conclusion

We have introduced a new graphical user interface for Quality Estimation (QE) that exceeds any previous tool in terms of functionality and extensibility. It relies on a web-application built with Python Django. The user has the possibility to upload new documents to be analyzed by QE. The documents can be browsed through a user-friendly dashboard, categorized in tasks. For every document, the user can get the estimations by the QE along with several informative graphs and charts

The web interface is supported by a database layer which can store the data and its evaluation. Several state-of-the-art web libraries are seamlessly integrated to allow responsive appearance and drawing of charts. All data-intensive tasks, including QE, are executed asynchronously in a background queue by a separate daemon. Existing QE tools are integrated in order to perform feature generation and prediction of quality scores.

Acknowledgment This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement N° 645452 (QT21).

Bibliography

Avramidis, Eleftherios. Qualitative: Python Tool for MT Quality Estimation Supporting Server Mode and Hybrid MT. *The Prague Bulletin of Mathematical Linguistics (PBML)*, 106:147–158, 2016.

Blatz, John, Erin Fitzgerald, George Foster, Simona Gandrabur, Cyril Goutte, Alex Kulesza, Alberto Sanchis, and Nicola Ueffing. Confidence estimation for machine translation. In *Proceedings of the 20th international conference on Computational Linguistics (COLING 04)*, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

Klejch, Ondřej, Eleftherios Avramidis, Aljoscha Burchardt, and Martin Popel. MT-ComparEval: Graphical evaluation interface for Machine Translation development. *The Prague Bulletin of Mathematical Linguistics (PBML)*, 104:63–74, 2015. ISSN 0032-6585. doi: 10.1515/pralin-2015-0014. URL https://ufal.mff.cuni.cz/pbml/104/art-klejch-et-al.pdf.

Koehn, Philipp. An Experimental Management System. *The Prague Bulletin of Mathematical Linguistics*, 94:87–96, 2010. ISSN 1804-0462.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Chris Zens Richard a nd Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL '07, pages 177–180, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.

Lavie, Alon and Abhaya Agarwal. METEOR: An Automatic Metric for MT Evaluation with High Levels of Correlation with Human Judgments. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 228–231, Prague, Czech Republic, jun 2007. Association for Computational Linguistics.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, jul 2002. Association for Computational Linguistics.

Pedregosa, Fabian, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Shah, Kashif, Eleftherios Avramidis, Ergun Biçici, and Lucia Specia. QuEst: Design, Implementation and Extensions of a Framework for Machine Translation Quality Estimation. *The Prague Bulletin of Mathematical Linguistics*, 100:19–30, 2013.

Shah, Kashif, Marco Turchi, and Lucia Specia. An efficient and user-friendly tool for machine translation quality estimation. *LREC 2014. Proceedings of the Ninth International Conference on Language Resources and Evaluation*, pages 3560–3564, August 2014.

Specia, Lucia, M. Turchi, N. Cancedda, M. Dymetman, and N. Cristianini. Estimating the Sentence-Level Quality of Machine Translation Systems. In *13th Annual Meeting of the European Association for Machine Translation*, pages 28–35, Barcelona, Spain., 2009.

Address for correspondence:
Eleftherios Avramidis
eleftherios.avramidis@dfki.de
German Research Center for Artificial Intelligence (DFKI GmbH)
Language Technology Lab
Alt Moabit 91c
10559, Berlin, Germany