PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 102 OCTOBER 2014 5–16

Qualitative: Open source Python tool for Quality Estimation over multiple Machine Translation outputs

Eleftherios Avramidis, Lukas Poustka, Sven Schmeier

German Research Center for Artificial Intelligence (DFKI Berlin)

Abstract

"Qualitative" is a python toolkit for ranking and selection of sentence-level output by different MT systems using Quality Estimation. The toolkit implements a basic pipeline for annotating the given sentences with black-box features. Consequently, it applies a machine learning mechanism in order to rank data based on models pre-trained on human preferences. The preprocessing pipeline includes support for language models, PCFG parsing, language checking tools and various other pre-processors and feature generators. The code follows the principles of object-oriented programming to allow modularity and extensibility. The tool can operate by processing both batch-files and single sentences. An XML-RPC interface is provided for hooking up with web-services and a graphical animated web-based interface demonstrates its potential on-line use.

1. Introduction

Having been a topic of research for many years, Machine Translation (MT) has recently reached a wide range of applications for big audiences. Methods and analyses produced in academic labs have been adopted by the industry and transferred to products and services with numerous use-cases. This has increased the interest for the field and generously empowered a broad spectrum of research activities and further development.

In this long history of conveying MT research into everyday use, the software developed by the relevant research community has been of high value. Through the open source implementation of majorly important MT and natural language processing tools (Koehn et al., 2007; Federico et al., 2008), researchers had the opportunity

 Corresponding author: eleftherios.avramidis@dfki.de
Cite as: Eleftherios Avramidis, Lukas Poustka, Sven Schmeier. Qualitative: Open source Python tool for Quality Estimation over multiple Machine Translation outputs. The Prague Bulletin of Mathematical Linguistics No. 102, 2014, pp. 5–16. doi: 10.2478/pralin-2014-0009.

to test and expand proposed methods and easily introduce new ideas. Additionally, the industry found pretty strong software engineering prototypes that were not too far from user-oriented programs and many of them were directly plugged into user interfaces.

Adopting this line of MT-oriented software development, we provide a Free Software implementation in the direction of Quality Estimation (QE). This can be directly used for further research/applications on MT output ranking, system selection, hybrid machine translation etc. Moreover, it provides the basics for further community development on QE, aiming to gather a wide range of contributions.

The following sections contain a comparison of our software with already existing open source tools (section 2), a description of the basic functionality (section 3), an explanation of the methods used in the background (section 4), an introductory guide for further development (section 5) and the plans for further work (section 6).

2. Previous work

The first collaborative work for development on this field was done in the frame of the WS'03 Summer Workshop at the John Hopkins University on *Confidence Estimation of MT* (Blatz et al., 2004), but to our knowledge no application or code has been publically available, as a result of this effort. Additionally, relevant contributions introduced several open-source tools offering MT evaluation (e.g. METEOR (Banerjee and Lavie, 2005)), Hjerson and Addicter (Berka et al., 2012)), whereas a good amount of such metrics and scripts were gathered in Asiya (Giménez and Marquez, 2010). All this work is based on comparing the produced translations with reference translations, which generally falls out of the scope of QE.

A major contribution directly to the field of QE has been done by the QuEst tool (Specia et al., 2013), which included an implementation of various features by numerous contributors. Whereas there is serious overlapping of QuEst with our submission, there are notable differences. In contrast to the regression-based orientation of QuEst, our software is aimed to sentence-level ranking and selection of MT-output, by implementing comparative Quality Estimation. Additionally, not the same quality features have been implemented in both toolkits, whereas Python as a dynamic language allows provides more flexible data structures and architecture. Nevertheless, our software includes a QuEst wrapper for conformity with WMT baselines.

3. Basic use

The out-of-the box functionality of the software is based on a use-case, where one source sentence is translated by several MT systems. The software therefore analyzes properties of all translations and suggests the proper *ranking* (ordering), trying to predict human preference. This can be used for ranking system outputs or combining several systems on the sentence level.

3.1. Installation

The toolkit is implemented in Python 2.7 and has been developed and tested for operation in a Linux operating system. The code has to be downloaded[1] and the Python path needs to be set to the `/src` directory, where all python scripts, modules and packages reside. Much of the functionality is provided by several publically available Python extensions, which need to be installed prior to any execution. All extensions required for the out-of-the-box functionality are provided by the Python *pip* package management system, so it is enough to run the respective `pip install` commands for the packages detailed in `INSTALL.txt`, These installations can easily take place on the user's home folder, without requiring root access (e.g. in experiment server environments).

The toolkit interacts with several java applications whose 'jar' and class files have to be placed in the directory `/lib`. An installation script that automatically downloads all required java dependencies is provided. Additionally, one needs to execute externally the LM server by Nitin Madnani.[2]

3.2. Resources and Configuration

The quality features for the given sentences are generated within a pipeline of NLP analysis tools. Many of these tools require specific resources to be acquired prior to the execution of the program. In particular, for the out-of-the-box installation and pre-trained models, one needs a language model, a PCFG grammar and a truecaser model for the source and target languages, which are also provided by an installation script through our servers.

All file locations and several other parameters (e.g. the translation language direction) can be specified in one or more complementary configuration files. Sample configuration files are provided in `/cfg/autoranking` and can be modified accordingly to fit the user's specific installation. The configuration files may also include references to many language-specific resources; the program will only use the ones which are relevant to the chosen languages.

The reason for allowing many configuration files is that one may want to split the configuration parameters depending on the environment, i.e. some settings may be generic and applicable to all computational servers, whereas some others may change from machine to machine.

---

[1] `http://www.dfki.de/~elav01/software/qualitative`

[2] `http://desilinguist.org` At the time that this paper is submitted, the Language Model scores are provided by using LM server, which wraps around SRILM (Stolcke, 2002) and needs to be compiled and executed separately. It is in our plans to remove this dependency and include KenLM (Heafield, 2011)

3.3. Execution

There are several ways the program can be executed. All relevant scripts can be found in the directory `/src/app/autoranking`. In particular, the basic functionality is provided as following:

- **Command-line interaction** (`application.py`): This script allows the user to interact with the sentence selection on the commandline. A configuration file and a pre-trained selection model need to be passed as parameters. Then the user can sequentially type the source sentence and the respective translations one by one. The purpose of this script is mainly to verify that all installation and settings are correct.
- **Batch decoding** (`decode_batch.py`): This script serves for cases where multiple sentences with their respective translations need to be analyzed in a row (e.g. for translating full documents). Apart from the configuration file and the trained classifier, this script accepts the decoding data in an XML file.
- **XML-RPC interface** (`xmlrpcserver.py`): This instantiates and XML-RPC awaiting translation requests for one sentence at a time. The server responds to the command `rank`, having as parameters the source and any number of respective translations. It is useful for binding to web-applications.

In order to assess the installation process we provide pre-trained model for German-English.

3.4. Demonstration server

As a demonstration for the use of the toolkit in a web service, an additional piece of software is provided. It is a web-interface[3] implemented in PHP which allows the user to enter source text to be translated. Consequently, it communicates with external translation services (e.g. Moses server, Google Translate API, Lucy RBMT), and fetches the translations. Finally, the produced translations are given to our ranking/selection mechanism and the result is visualised graphically.

The demonstration server is provided as an example. It is not included in the architecture of the rest of the toolkit and therefore is distributed as a separate package. Users have to modify the script, in order to parametrise the necessary URL addresses and configuration.

4. Under the hood

The core of the program is based on comparative Quality Estimation. The given sentence is first processed within a pipeline of modules. These modules perform text pre-processing and various NLP analyses to generate features that indicate the quality

[3]The demo web interface can be accessed at `http://www.qt21.eu/demonstrator`

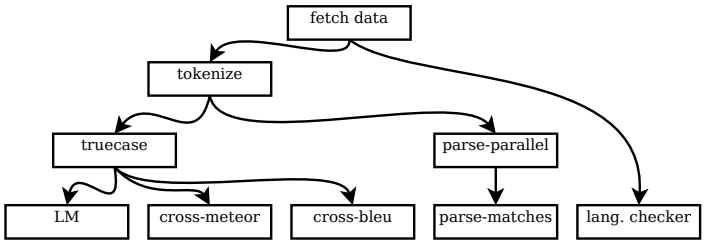

Figure 1. Sample pipeline of feature generators for one of the most successful feature sets.

of the translation. The generated features are consequently fed to a machine learning algorithm, which employs a statistical model for putting the translations in an order of preference. This statistical model has been previously trained on human-annotated data.

In principle, there can be various combinations of features and machine learning algorithms, depending on what performs best given various properties of the quality estimation task. The optimal combination, which performs similarly to human decisions, is subject of constant research in the field. The provided vanilla implementation and pre-trained model follow one of the most successful experiments on German-English, which includes the feature set #24 with Logistic Regression as described in Avramidis (2013a).

4.1. Generation of features

As explained above, the generation of features is a crucial step for acquiring quality hints regarding the processed sentence. For this purpose, the toolkit provides a set of modules, thereof called *feature generators*. A sample pipeline can be seen in Figure 1.

- **Language model** (LM): It sends a query to the language model in order to acquire LM probabilities for unigrams, bigrams, trigrams etc., and also detects and counts words unknown to the language model. It also saves the sentence position (index) of the unknown words and the n-grams with the lowest and highest probability. Features also include the average and the standard deviation of the n-grams probabilities and the sentence positions. The queries to the LM can be sent via XML-RPC to an external LM server, or to call the respective functions from an imported LM library (e.g. KenLM).
- **PCFG parsing**: It loads a language-specific PCFG grammar and handles the parsing of source and target sentences by PCFG parsing. It extracts the overall log-likelihood, the best parse confidence and the count (k) of k-best parse trees generated. The best parse is included as string meta-data, so that following fea-

ture generators can re-use it for producing their features. One of them counts how many times each tree node label appears in the parse and calculate their respective sentences position statistics. Another performs naïve source-to-target tree-label matching, i.e. calculates the ratio of VPs on the source with the respective VPs on the target. The current implementation supports the Berkeley Parser (Heafield, 2011) via either as included library or as an external XML-RPC server. There are pre-trained grammars for English, German, Spanish (Taulé et al., 2008), French, Russian, Chinese and Arabic freely available.

- **Cross-target BLEU and METEOR**: It encapsulates the calculation of well-known reference-aware n-gram-based metrics. For the scenario of multiple alternative translations by different systems, each particular system receives as a feature its own metric score, as if the other competitive systems were used as reference translations. For certain scenarios, this may indicate cases when a system output is very different than the majority of the other competitive system outputs. Here, smoothed sentence level BLEU (Papineni et al., 2002) and all METEOR components (precision, recall, fragmentation penalty and overall weighed score) (Lavie and Agarwal, 2007) are supported.
- **Language correction**: This investigates the usability of language-correction software. Such software is based on hand-written rules which detect grammatical, syntactic and other expresional inconsistencies on monolingual text usually while being written in word processors; the errors are automatically flagged and suggestions are given to authors. We use the count of each error type in every sentence as a separate feature, a tactic that unfortunately produces rather sparse features. A feature generator wraps around the open source Language Tool library (Naber, 2003; Miłkowski, 2012), whereas remote querying towards the optional proprietary software Acrolinx IQ (Siegel, 2011) is also supported via the SUDS protocol.
- **IBM1 probabilities**: This feature generator supports reading an IBM-1 model and includes the produced sentence probability as an additional feature.
- **Length features** include simple counting of the number of tokens, characters and the average number of characters per word for each sentence.
- **Ratios and differences**: A last step calculates differences and ratios between every source and its respective translation feature, if available. Defining explicit features for known relations between features may be useful for some ML algorithms.

The code includes the default normalisation, tokenisation and truecasing scripts of Moses (Koehn et al., 2007). Additional tokenisation and other pre-processing actions, when needed, are done via NLTK (Loper and Bird, 2002). Also, some functions from automatic error extraction of Hjerson (Popović, 2011) are included.

4.2. Machine Learning

The ranking algorithm is based on binary classifier decisions, which are recombined up into a full ranking (Usunier et al., 2009; Avramidis, 2012). The decision is every time taken on a sentence level, given the numeric features generated for this particular sentence and translation. The final algorithm has no access or use for the given/translated text. Although the pre-trained models use logistic regression (Hosmer, 1989) as a pairwise classifier, the interface allows many other learning methods to be employed and stored, such as SVM (Joachims, 2006), Naive Bayes, k-Nearest neighbours (Coomans and Massart, 1982) and Decision Trees (Quinlan, 1986).

4.3. Training

Training of new models takes places in two stages, *annotation* and *learning*. First, the training data need to be processed by the feature generators pipeline using the batch annotation script[4] and a given configuration script. Implementing the interface provided by the Ruffus library (Goodstadt, 2010) allows for parallelisation of intermediate steps into a number of CPU cores, whereas it keeps track of finished and unfinished steps so that they can be resumed if something crashes. Heavy and slow tasks, such as parsing, are also parallelised after being split into multiple parts.

On the second phase, the learning script[5] trains and evaluates the machine learning algorithms given the features generated previously at the annotation phase. Both learning algorithms and feature sets can be defined in a configuration file. The learning pipeline is organised through a modified version of the Python Experiment Suite (Rückstieß and Schmidhuber, 2011), so intermediate steps are saved on the disk for resuming or further use, as for example the pickled classifier models.

Machine learning algorithms are primarily provided by Orange (Demšar et al., 2004) and optionally by Scikit-learn (Pedregosa et al., 2011). The entire set of RankEval scripts (Avramidis, 2013b) are included as part of the software. Many of the evaluation functions and calculations are based on NumPy and SciPy (Oliphant, 2007).

5. Development

The architecture of the program has been designed to allow for further development. The code is open and collaborative efforts are centralised within a Git repository.[6]. The development has been divided in several python packages. Each package serves a different function, so that the code can be modural and re-usable.

[4] `/src/app/autoranking/annotate_batch.py`

[5] `/src/app/autoranking/suite.py`

[6] Please check `http://www.dfki.de/~elav01/software/qualitative` for details on how to clone the repository and commit

The code has been documented based on inline comments following the EpyDoc standard, and therefore an automatically generated API documentation is provided for the vast majority of functions and classes. The more important classes and functions are detailed in the following sections.

5.1. Understanding the data structures

The data structures used in all parts of the software are contained in the `sentence` package. The basic structures are:

- The `SimpleSentence` is the basic data structure. It is a class which wraps the original sentence text as a string. Additionally, it contains a dictionary of "attributes". These can be features and meta-data provided by the original data; they are further augmented by the annotation process (see feature generators) and they are the ones who are important for the machine learning algorithms, which also put their results as attributes.
- The `ParallelSentence` is a class that represents the basic unit of a parallel corpus. It contains one source sentence, a list of target sentences and optionally a reference, All encapsulated sentences are an instance of `SimpleSentence`. The `ParallelSentence` also includes its own attribute dictionary, with the sentence id, the source and target language asclasses the most common attributes.
- The `DataSet` is an iterable class which stands one level above, as it encapsulates a list of parallel sentences and several convenience functions.

Any development effort should use the data structures, which allow for optimal flow of data between the various modules. In this python package, one can find more classes which extend the basic data structures. These extensions have been developed to support breaking the parallel sentences into a set of pairwise sentence comparisons.

5.2. Reading and writing files

In order to facilitate processing data for sentence-level ranking, external data processing is based on a XML format called "JCML" for *Judged Corpus Markup Language*. In contrast to line-separated simple-text formats used for regression-based QE, this format allows for a variable number of target sentences per source[7], whereas all features are aggregated in the same file. In the package `support.jcml` we provide several utility scripts (e.g. for joining and splitting) and also a utility for converting from multiple plain text files to JCML.

Reading and writing external data is taken care through the classes in the package `dataprocessor`. The modules contained in the package allow for reading and writing using several alternative python libraries, e.g. *minidom* (batch all-in-memory),

[7] The ranking-based human evaluation tasks of WMT provides 2-12 system outputs per source sentence. For this reason JCML was used for the QE ranking task at WMT13

SAX and *CElementTree* (incremental disk writing/reading). The most commonly-used reader is from `ce.cejcml`, whereas the most common writer is `IncrementalJcml` from `sax.saxps2jcml`.

5.3. Adding new features

The classes responsible for generating features reside in the `featuregenerator` package. One can add features by developing a new feature generator. The required steps are:

- create a new module with a new class that extends `FeatureGenerator` (from `featuregenerator`). The initialisation function should load necessary resources as named arguments. If more modules are required, place them in a new package.
- override the unimplemented function `get_features_tgt`, perform the required analysis of each target sentence and return a dictionary containing the resulting feature names and values. This function will be automatically repeated for all target sentences and can also process the source sentence.
- optionally override the unimplemented function `get_features_src` if you want to provide features that refer only to the source sentence. Similarly prefixed functions can be overridden for attaching features as attributes to other parts of the parallel sentence.
- optionally override the functions `add_features_tgt` and similarly prefixed function if you also want to modify the string of the processed sentences (e.g. for pre-processing, tokenisation etc.).
- add the new feature generator in the annotation process of the relevant application. An initialised instance of the class should be added in a list with all other feature generators that are executed. The order of the feature generators in the list matters, particularly when some generators require pre-processing or meta-data from previously executed generators. If possible, parameters should be loaded from a configuration file.

It is also possible to encapsulate tools and libraries written in Java through the Py4J library, following the example of the `BerkeleyParserSocket` class.

5.4. New QE applications and machine learning algorithms

The provided code includes implementation for the ranking mechanism by using pairwise classifiers. Since there is infrastructure for feature generation and machine learning, other experiments and applications can be easily developed. Additional **QE applications** can be added as separate packages in `apps` by following the example of the `autoranking` package. These are mainly executable scripts for annotation and training (see section 4.3), or other functions (test module, commandline app or XML-RPC server).

As mentioned, this package already provides support for several functions of Orange and Scikit-learn. Further **machine learning algorithms** can be included in `ml.lib` by implementing the abstract functions of the classes in the package `ml`.

6. Further work

Whereas the provided code contains a fully functional implementation and a modular architecture, several parts are subject of further improvement. We are currently working on improving the architecture, e.g. to provide abstract classes for machine learning methods or better templates for new "apps". Additionally, a constant goal are more feature implementations, to cover at least the so-called *baseline* features. Readers of this paper are advised to check the latest version of documentation and architecture in the official web-page.

Acknowledgements

This work has received support by the EC's FP7 (FP7/2007-2013) under grant agreement number 610516: "QTLeap: Quality Translation by Deep Language Engineering Approaches". Early stages have been developed with the support of the projects TaraXŰ and QT-Launchpad. Many thanks to: Slav Petrov for modifying the Berkeley Parser in order to allow modification of parsing parameters; Hieu Hoang, Philipp Koehn, Maja Popović, Josh Schroeder and David Vilar as parts of their open source code have been included in some of our scripts; Aljoscha Burchardt and Prof. Hans Uszkoreit for the support.

Bibliography

Avramidis, Eleftherios. Comparative Quality Estimation: Automatic Sentence-Level Ranking of Multiple Machine Translation Outputs. In *Proceedings of 24th International Conference on Computational Linguistics*, pages 115–132, Mumbai, India, Dec. 2012. The COLING 2012 Organizing Committee.

Avramidis, Eleftherios. Sentence-level ranking with quality estimation. *Machine Translation (MT)*, 28(Special issue on Quality Estimation):1–20, 2013a.

Avramidis, Eleftherios. RankEval: Open Tool for Evaluation of Machine-Learned Ranking. *The Prague Bulletin of Mathematical Linguistics (PBML)*, 100:63–72, 2013b. doi: 10.2478/pralin-2013-0012.

Banerjee, Somnath and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*. Association for Computational Linguistics, 2005.

Berka, Jan, Ondřej Bojar, Mark Fishel, Maja Popović, and Daniel Zeman. Automatic MT Error Analysis: Hjerson Helping Addicter. In *8th International Conference on Language Resources and Evaluation*, pages 2158–2163, 2012. ISBN 978-2-9517408-7-7.

Blatz, John, Erin Fitzgerald, George Foster, Simona Gandrabur, Cyril Goutte, Alex Kulesza, Alberto Sanchis, and Nicola Ueffing. Confidence Estimation for Machine Translation. In Rollins, M., editor, *Mental Imagery*. Yale University Press, 2004.

Coomans, D. and D.L. Massart. Alternative k-nearest neighbour rules in supervised pattern recognition. *Analytica Chimica Acta*, (138):15–27, Jan. 1982. ISSN 00032670.

Demšar, Janez, Blaž Zupan, Gregor Leban, and Tomaz Curk. Orange: From Experimental Machine Learning to Interactive Data Mining. In *Principles of Data Mining and Knowledge Discovery*, pages 537–539, 2004.

Federico, Marcello, Nicola Bertoldi, and Mauro Cettolo. IRSTLM: an open source toolkit for handling large scale language models. In *Interspeech*, pages 1618–1621. ISCA, 2008.

Giménez, Jesús and Lluís Marquez. Asiya: An Open Toolkit for Automatic Machine Translation (Meta-)Evaluation. *The Prague Bulletin of Mathematical Linguistics*, 94:77–86, 2010. doi: 10.2478/v10108-010-0022-6.

Goodstadt, Leo. Ruffus: a lightweight Python library for computational pipelines. *Bioinformatics*, 26(21):2778–2779, Nov. 2010. ISSN 1367-4803.

Heafield, Kenneth. KenLM: Faster and Smaller Language Model Queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland, July 2011. Association for Computational Linguistics.

Hosmer, David. *Applied logistic regression*. Wiley, New York [u.a.], 8th edition, 1989. ISBN 9780471615538.

Joachims, Thorsten. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226. ACM, 2006. ISBN 1595933395.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open Source Toolkit for Statistical Machine Translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 177–180, Prague, Czech Republic, June 2007.

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, June 2007. Association for Computational Linguistics.

Loper, Edward and Steven Bird. NLTK: The Natural Language Toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ETMTNLP '02, pages 63–70, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

Miłkowski, Marcin. *Translation Quality Checking in LanguageTool*, pages 213–223. Corpus Data across Languages and Disciplines. Peter Lang, Frankfurt am Main, Berlin, Bern, Bruxelles, New York, Oxford, Wien, 2012.

Naber, Daniel. A rule-based style and grammar checker. Technical report, Bielefeld University, Bielefeld, Germany, 2003.

Oliphant, Travis E. SciPy: Open source scientific tools for Python, 2007.

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, July 2002. Association for Computational Linguistics.

Pedregosa, F, G Varoquaux, A Gramfort, V Michel, B Thirion, O Grisel, M Blondel, P Prettenhofer, R Weiss, V Dubourg, J Vanderplas, A Passos, D Cournapeau, M Brucher, M Perrot, and E Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Popović, Maja. Hjerson: An Open Source Tool for Automatic Error Classification of Machine Translation Output. *The Prague Bulletin of Mathematical Linguistics*, 96(1):59–68, 2011. doi: 10.2478/v10108-011-0011-4.

Quinlan, J. R. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, Mar. 1986. ISSN 0885-6125.

Rückstieß, Thomas and Jürgen Schmidhuber. Python Experiment Suite Implementation. *The Python Papers Source Codes*, 2:4, 2011.

Siegel, Melanie. Autorenunterstützung für die Maschinelle Übersetzung. In *Multilingual Resources and Multilingual Applications: Proceedings of the Conference of the German Society for Computational Linguistics and Language Technology (GSCL)*, Hamburg, 2011.

Specia, Lucia, Kashif Shah, José Guilherme Camargo de Souza, and Trevor Cohn. QuEst - A translation quality estimation framework. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 79–84, Sofia, Bulgaria, Aug. 2013. Association for Computational Linguistics.

Stolcke, Andreas. SRILM – An Extensible Language Modeling Toolkit. In *Proceedings of the Seventh International Conference on Spoken Language Processing*, pages 901–904. ISCA, Sept. 2002.

Taulé, Mariona, Antònia Martí, and Marta Recasens. AnCora: Multilevel Annotated Corpora for Catalan and Spanish. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, May 2008. European Language Resources Association (ELRA). ISBN 2-9517408-4-0.

Usunier, Nicolas, David Buffoni, and Patrick Gallinari. Ranking with ordered weighted pairwise classification. In *Proceedings of the 26th Annual International Conference on Machine Learning ICML 2009 Montreal Quebec Canada June 1418 2009*, pages 1057—-1064. ACM, 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