Integration of QuEst and Okapi

Final Activity Report

Participants:

Gustavo Paetzold, Lucia Specia and Kashif Shah (Sheffield University)

Yves Savourel (Okapi)

1. Introduction

The proposed project aimed at exploring the integration of the QuEst toolkit within the Okapi Framework, allowing for users to perform translation quality estimation along with Okapi’s many functionalities.

QuEst (http://www.quest.dcs.shef.ac.uk/) is an open source toolkit for machine translation quality estimation which has been developed at the University of Sheffield, and is capable of both crafting translation quality estimation models and applying them onto translations in order to estimate their quality. The Okapi framework (http://www.opentag.com/okapi/) is an open source and cross-platform set of tools and applications designed to help in creating or improving translation/localization processes.

The project’s description proposed three main objectives, each of which was accomplished within the deadline initially estimated:

- **[April-15-2014]** A new Okapi Step was added to the Okapi Pipeline Editor, which has the purpose of employing QuEst in calculating feature values and annotating translation candidates with a predicted quality value, for already existing prediction models.

  [June-30-2014] An Okapi tool was developed to allow users to build prediction models and the relevant resources for these models, through existing or novel functionalities added to QuEst. This tool covers the entire pipeline of resource generation, training (model building) and testing (generation of quality predictions).

- **[July-30-2014]** A pilot user study was performed on how quality predictions could be used within the Okapi workflow. The focus of the experiment was on the selection of translations with a certain quality band for error annotation within Okapi.

In what follows we describe all activities developed throughout the entirety of the project schedule, which cover in more detail the three main goals just described. We also provide experiments conducted with the resources developed along with a discussion on the results obtained.

2. Activities Developed

2.1. The QuEst Step

As previously mentioned, the main objective concluded by 15 of April for the Integration of QuEst and Okapi project is the development of an Okapi Step for translation quality estimation.

From the beginning of February until the beginning of March, the group focused on the task of inspecting the Okapi Framework in order to become more familiar with its functionalities and also with the software engineering techniques/programming tools used in its development. The design of the new Okapi Step for translation quality estimation began at the beginning of March. To allow for the group to work on the new Okapi Step with both security and versioning control, a Google Code project was created, which is currently hosted at https://code.google.com/p/okapi-quest/ and available for all group members to access and modify.

A "vanilla" version of QuEst, which is available for download at http://www.quest.dcs.shef.ac.uk/, was the one chosen to be integrated with the Okapi Framework. The objective was to compile the QuEst project into a Java library, which could be easily imported into the Okapi Framework library repositories. The code of the QuEst project was initially submitted to an adaptation, which allowed for the group to create a QuEst version more suited for the task. To the "vanilla" version were added some of the new translation features supported by the QuEst Online version, which allows for it to extract up to 50 features. Were also corrected minor coding issues such as hard-coded function parameters, improperly handled buffered streams and unnecessary method calls.

The new Okapi Step for quality estimation is represented inside the Okapi Framework by the following three Java classes, all of which make use of the adapted QuEst library:

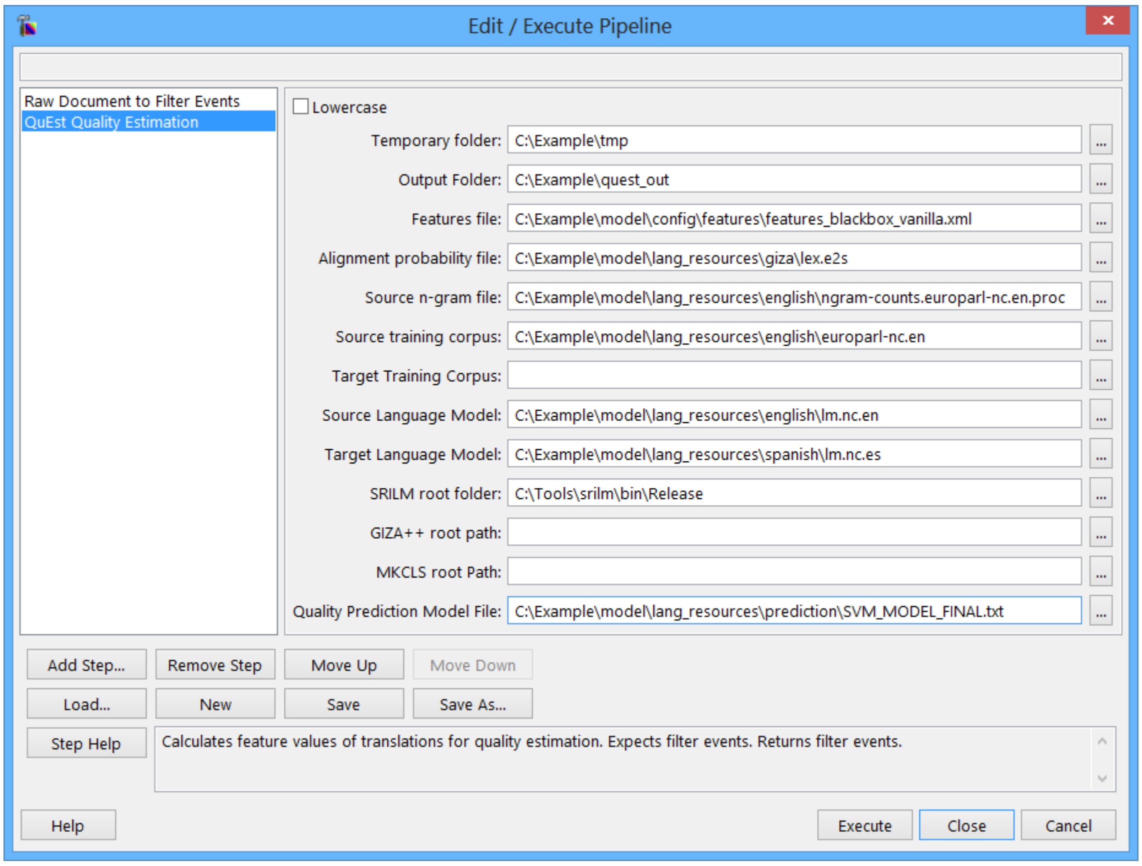
- **QuestStep.java**: Extends the BasePipelineStep class from the Okapi Framework, and implements the methods necessary for the step to be made available in the Okapi Pipeline Editor. It reads all translations from the input files provided by the user and invokes an instance of the QuestProcessor class to produce quality estimations for them.

- **QuestProcessor.java**: Invokes the methods and functions of the QuEst library in order to calculate feature values, and uses the Java SVM (Support Vector Machine) library to load the prediction model provided by the user and then generate translation quality scores. It has also a built-in resource generation helper: if the user does not know how to generate the language model or ngram count files necessary for the step to run, it automatically generates such resources.

- **Parameters.java**: Implements the IEditorDescriptionProvider interface, and is responsible for providing all data necessary for the Okapi Framework to build the interface for the user to setup the settings of the quality estimation step in the Okapi Pipeline Editor.

Along with the three main classes were also created unit tests, which automatically configures and runs the step onto a set of test parameters, allowing for the group to easily debug the step and find possible bugs and errors.

When the user adds the quality estimation step to a custom pipeline, the setup interface shown in Figure 1 appears in the right-hand side of the Okapi Pipeline Editor. It allows for the user to provide the necessary files for the QuEst methods to run.



Raw Document to Filter Events
QuEst Quality Estimation

☐ Lowercase

Temporary folder:	C:\Example\tmp
Output Folder:	C:\Example\quest_out
Features file:	C:\Example\model\config\features\features_blackbox_vanilla.xml
Alignment probability file:	C:\Example\model\lang_resources\giza\lex.e2s
Source n-gram file:	C:\Example\model\lang_resources\english\ngram-counts.europarl-nc.en.proc
Source training corpus:	C:\Example\model\lang_resources\english\europarl-nc.en
Target Training Corpus:	
Source Language Model:	C:\Example\model\lang_resources\english\lm.nc.en
Target Language Model:	C:\Example\model\lang_resources\spanish\lm.nc.es
SRILM root folder:	C:\Tools\srilm\bin\Release
GIZA++ root path:	
MKCLS root Path:	
Quality Prediction Model File:	C:\Example\model\lang_resources\prediction\SVM_MODEL_FINAL.txt

Calculates feature values of translations for quality estimation. Expects filter events. Returns filter events.

Figure 1 – QuEst Step parameters

The input parameters necessary for the quality estimation step to perform successfully are the following:

- **Lowercase**: Determines whether or not the input data should be lowercased.

- **Temporary Folder:** Folder where the intermediate processed input files will be stored. Some of the files which may be stored in the input folder are the ones containing the lowercased versions of the input translations and their perplexity measures.

- **Output Folder:** Folder where the files containing the quality estimation scores will be stored. In its current version, the quality estimation step produces output in three distinct formats: plain text, XML and TMX.

- **Features File:** An XML file describing which features QuEst should calculate for the given translations.

- **Alignment Probability File:** A file containing the alignment probability for the words of source and target languages of the translations provided. The file may be produced by any bilingual word aligner, but must be in the same format produced by GIZA++.

- **Source N-gram File:** File containing the N-gram counts for the translations' source language.

- **Source and Target Training Corpus:** Files containing sentences in the source and target languages.

- **Source and Target Language Models:** Two separate files containing a language model for the translations' both source and target languages. The files must be in ARPA format, and may be produced by any language modelling tool.

- **SRILM Binaries Folder:** Path of the folder containing the compiled binaries of the user's SRILM installation.

- **GIZA++ Root Path:** Path of the folder containing the compiled binaries of the user's GIZA++ installation.

- **MKCLS Root Path:** Path of the folder containing the compiled binaries of the user's MKCLS installation.

- **Quality Prediction Model File:** Path for the prediction model for the calculation of quality estimations. The file must be in the same format produced by the Java SVM library.

Once the parameters are set and the user runs the pipeline, the step begins to execute. Initially the step reads and stores all translations. Then, the sentences are passed onto a pre-processing phase, where they are submitted to tokenization, lowercasing (if selected by the user) and measure calculation by the SRILM Language Modelling tool. Once finished the pre-processing routine, the methods provided by the QuEst library calculate the feature values for each translation. The SVM prediction model is then loaded, and the feature values previously produced, along with the given model, are used in the estimation of quality scores. Both feature values and quality estimation scores are then finally stored in a TMX file, which is placed in the output folder specified by the user during setup.

2.2. The SVM Model Builder Step

The main goal to be concluded by the 30th of June, according to the project's schedule, was the development of a tool to facilitate the creation of quality prediction models in the Okapi Framework. For that purpose, we chose to create another Okapi Step.

The SVM Model Builder Step has the goal of allowing for the user to easily create a translation quality prediction model to be used by the QuEst Step. In order to produce such models, we have created an adapted version of the LibSVM (Fan et. al, 2005), an open-source Java library which allows for one to train Support Vector Machines onto quality estimation datasets and save the models produced in text format. The adaptation made to the original LibSVM consisted on modifying it in order for the library to be easily imported into the Okapi Framework.

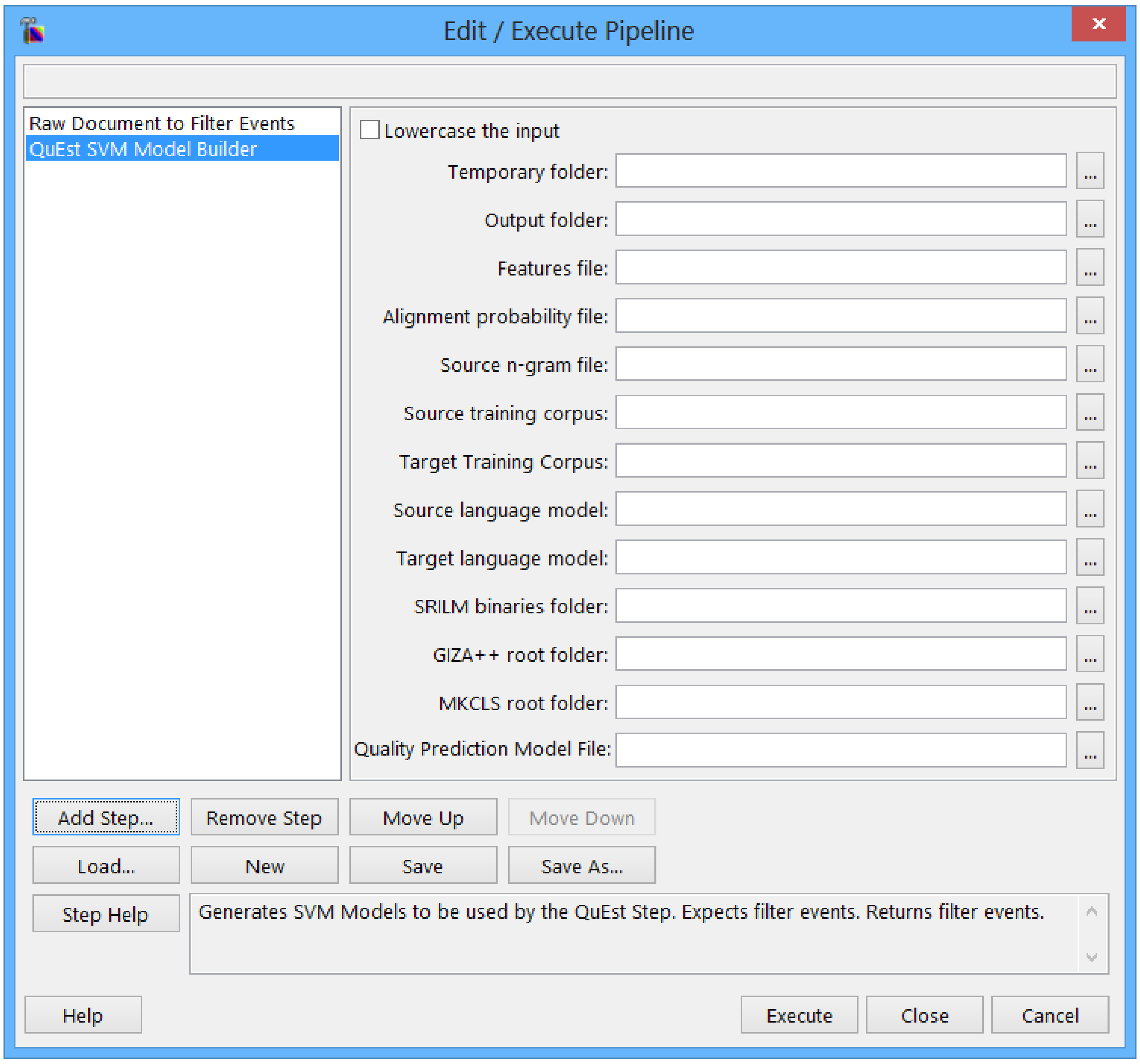
The SVM Model Builder Step is represented inside the Okapi Framework by the following three Java classes:

- **SVMModelBuilderStep.java**: Extends the BasePipelineStep class from the Okapi Framework, and implements the methods necessary for the step to be made available in the Okapi Pipeline Editor. It reads three input files, which must be provided by the user: two files containing *n* sentences each in source and target languages, as well as a file containing the score for each of the *n* translations. After reading the input files, it calculates the prediction feature values, as specified by the user through the step's interface, by employing the QuestProcessor.java class, described in Section 2.1. With feature values calculated, it invokes the SVModelBuilderProcessor.java in order for the model to be quality prediction model to be built and saved in a text file.

- **SVMModelBuilderProcessor.java**: It receives as input the feature values previously produced as well as the training quality prediction scores read by the SVMModelBuilderStep.java class. With such data loaded, it invokes the adapted LibSVM in order for a translation quality prediction model to be trained and saved onto a text file. It optimizes SVM parameters by automatically performing 5-fold cross validation.

- **Parameters.java**: Implements the IEditorDescriptionProvider interface, and is responsible for providing all data necessary for the Okapi Framework to build the interface for the user to setup the settings of the quality estimation step in the Okapi Pipeline Editor.

Like the QuEst Step, it also has a graphical interface, which allows for the user to easily load the files necessary for the step to run. Figure 2 illustrates the graphical interface of the SVM Model Builder Step.



Generates SVM Models to be used by the QuEst Step. Expects filter events. Returns filter events.

Figure 2 – SVM Model Builder Step parameters

The parameters in Figure 2 can be described as such:

- **Lowercase**: Determines whether or not the input data should be lowercased.

- **Temporary Folder:** Folder where the intermediate processed input files will be stored. Some of the files which may be stored in the input folder are the ones containing the lowercased versions of the input translations and their perplexity measures.

- **Output Folder:** Folder where the files containing the quality estimation scores will be stored. In its current version, the quality estimation step produces output in three distinct formats: plain text, XML and TMX.

- **Features File:** An XML file describing which features QuEst should calculate for the given translations.

- **Alignment Probability File:** A file containing the alignment probability for the words of source and target languages of the translations provided. The file may be produced by any bilingual word aligner, but must be in the same format produced by GIZA++.

- **Source N-gram File:** File containing the N-gram counts for the translations' source language.

- **Source and Target Training Corpus:** Files containing sentences in the source and target languages.

- **Source and Target Language Models:** Two separate files containing a language model for the translations' both source and target languages. The files must be in ARPA format, and may be produced by any language modelling tool.

- **SRILM Binaries Folder:** Path of the folder containing the compiled binaries of the user's SRILM installation.

- **GIZA++ Root Folder:** Path of the folder containing the compiled binaries of the user's GIZA++ installation.

- **MKCLS Root Folder:** Path of the folder containing the compiled binaries of the user's MKCLS installation.

- **Quality Prediction Model File:** Path of the file in which to save the translation quality prediction model produced by the step in plain text format.

Note that, since the SVM Model Builder Step uses the QuestProcessor.java class to produce feature values, most of its input parameters are identical to the ones of the QuEst Step, described in Section 2.1. As already mentioned, when finished running, the SVM Model Builder Step produces a text file containing a translation quality prediction model, which may then be passed onto the Okapi pipeline to the QuEst Step.

In order to make it simpler for unexperienced users to configure the SVM Model Builder Step, we have included in the step a series of routines which automatically produce some of the required resources previously listed. Those resources are:

- **Source N-gram File:** If not provided by the user, the step runs SRILM in order to create it. The resulting file is stored in the "Temporary Folder" in order for the user to use it again, if necessary.

- **Source and Target Language Models:** If either or both of such language models are not provided by the user through the interface, the step calls SRILM in order to create them automatically. The resulting files are stored in the "Temporary Folder" in order for the user to use it again, if necessary.

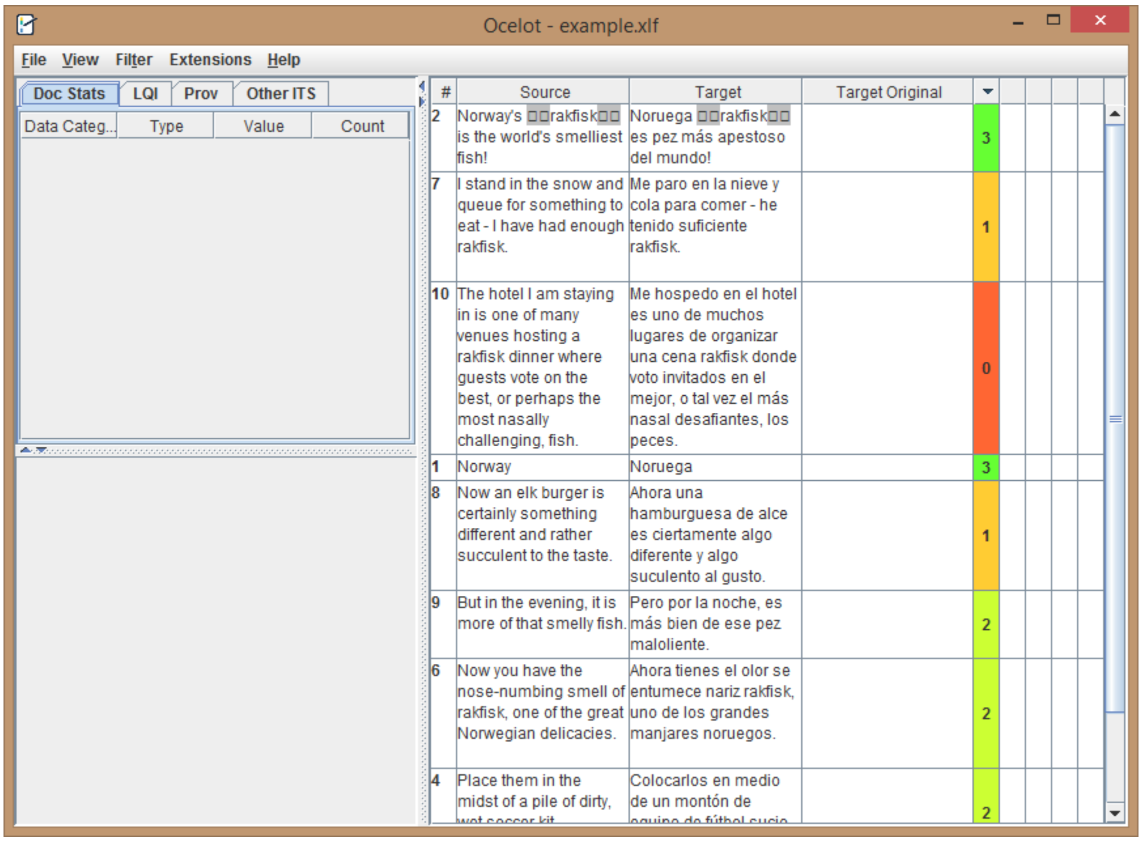
It is very important to mention that, in order for the step to be able to create the missing resources, the user must provide valid files for the "Source and Target Training Corpus" parameters, as well as a valid path for the "SRILM Binaries Folder" parameter.

2.3. The Properties Setting Step

In order to allow for one to use the data produced by the QuEst Step in annotation tools with graphical interfaces, we devised the Properties Setting Step. This step has the purpose of using the TMX files produced by the QuEst Step, which contain translation quality estimates, to annotate XLIFF files.

The XLIFF format (XLIFF, 2014) is based on the well-known XML format, and is very frequently used by localization, translation and annotation tools. In the context of translation quality estimation, XLIFF files can be used to store translations along with any type of property associated to them, such as quality estimates. Ocelot (Ocelot, 2014) is an example of open-source tool which uses XLIFF files containing translation data to allow for one to annotate such translations in many distinct ways. Figure 3 illustrates the Ocelot tool being used in the annotation task of the experiment described in Section 3.2.

Such tools are an interesting alternative when it comes to displaying translation quality estimates in a comprehensible fashion. Ocelot, for an example, is capable of not only presenting translations along with its scores, but also of assigning visual cues to different ranges of scores, as can be observed in Figure 3. This resource allows for one to more easily assess issues and limitations of quality estimation approaches.



#	Source	Target	Target Original	
2	Norway's □□rakfisk□□ is the world's smelliest fish!	Noruega □□rakfisk□□ es pez más apestoso del mundo!		3
7	I stand in the snow and queue for something to eat - I have had enough rakfisk.	Me paro en la nieve y cola para comer - he tenido suficiente rakfisk.		1
10	The hotel I am staying in is one of many venues hosting a rakfisk dinner where guests vote on the best, or perhaps the most nasally challenging, fish.	Me hospedo en el hotel es uno de muchos lugares de organizar una cena rakfisk donde voto invitados en el mejor, o tal vez el más nasal desafiantes, los peces.		0
1	Norway	Noruega		3
8	Now an elk burger is certainly something different and rather succulent to the taste.	Ahora una hamburguesa de alce es ciertamente algo diferente y algo suculento al gusto.		1
9	But in the evening, it is more of that smelly fish.	Pero por la noche, es más bien de ese pez maloliente.		2
6	Now you have the nose-numbing smell of rakfisk, one of the great Norwegian delicacies.	Ahora tienes el olor se entumece nariz rakfisk, uno de los grandes manjares noruegos.		2
4	Place them in the midst of a pile of dirty,	Colocarlos en medio de un montón de		2

Figure 3 – Ocelot opening displaying the content of an XLIFF file

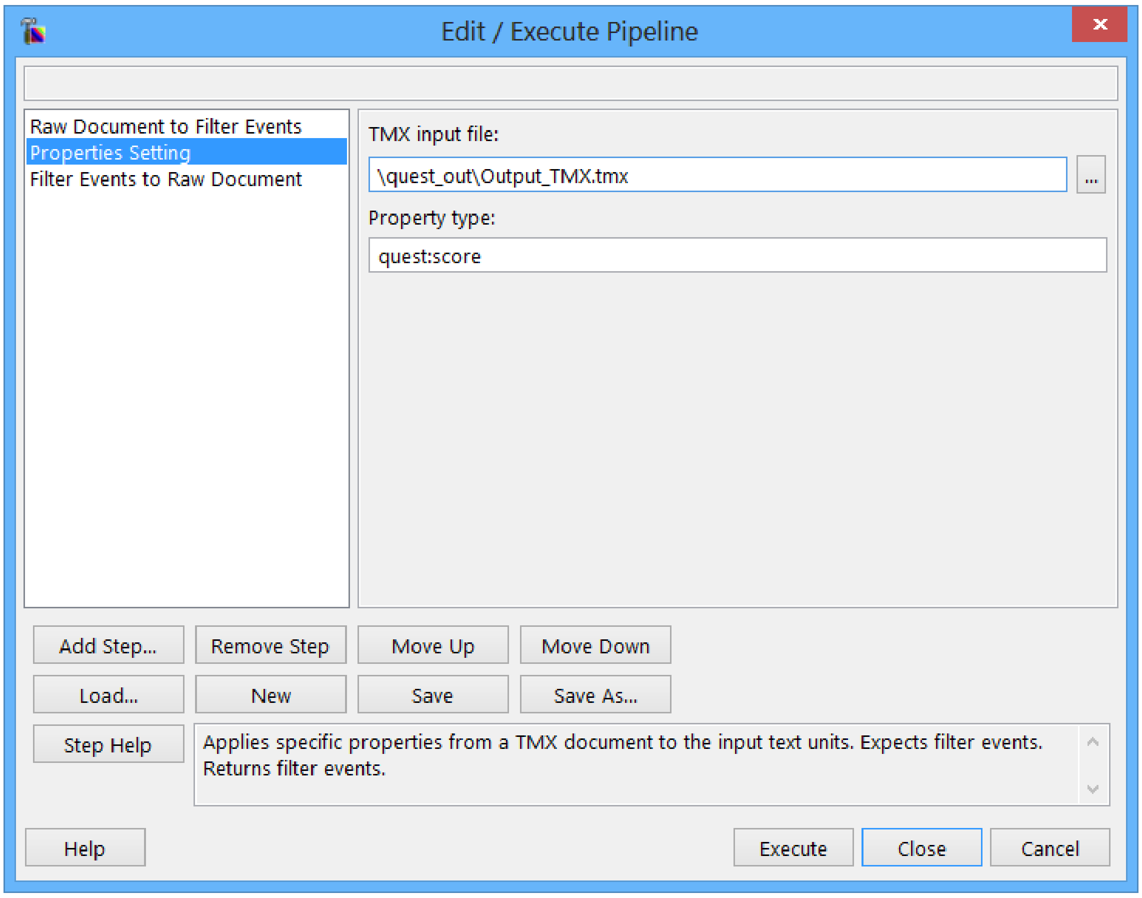
The Properties Setting Step is represented within the Okapi Framework by the three following Java classes:

- **PropertiesSettingStep.java**: Extends the BasePipelineStep class from the Okapi Framework, and implements the methods necessary for the step to be made available in the Okapi Pipeline Editor. It initially reads all translations from the input files provided by the user, along with a TMX file with translation quality estimates produced by the QuEst Step. It then invokes the PropertiesSetter.java class, in order for it to produce an XLIFF file of annotated translations.

- **PropertiesSetter.java**: Extracts the translation quality estimates from the input TMX file and use them to annotate an output XLIFF file containing the translations provided by the user.

- **Parameters.java**: Implements the IEditorDescriptionProvider interface, and is responsible for providing all data necessary for the Okapi Framework to build the interface for the user to setup the settings of the quality estimation step in the Okapi Pipeline Editor.

When added to the Okapi pipeline, the Properties Setting Step exhibits the interface illustrated in Figure 4.



TMX input file: \quest_out\Output_TMX.tmx

Property type: quest:score

Applies specific properties from a TMX document to the input text units. Expects filter events. Returns filter events.

Figure 4 – Properties Setting Step parameters

The parameters in Figure 4 can be described as such:

- **TMX Input File:** An XML file produced by the QuEst Step containing translations annotated with quality estimates.

- **Property Type:** A text field describing the label of the property in which the translation quality estimates are stored in the TMX file.

Once the XLIFF file is produced by the step, it may then be used for any type of annotation task. In Section 3.2 we describe an experiment in which we employ Ocelot, as well as all three steps produced in this project.

3. Experiments

3.1. Running Time

In order to evaluate the performance of the quality estimation step on producing translation quality estimates, a performance experiment was conducted. In this experiment, we ran the quality estimation step onto multiple sets containing distinct numbers of translations from English to Spanish. The four selected translation sets contained 62, 125, 250 and 500 test translations. The quality prediction model was trained with the same set of 2540 translations for all test sets. The configuration files necessary for the step to run are the ones referenced at http://www.quest.dcs.shef.ac.uk/ under the WMT13 QE Task 1.1 section. The SVM prediction model was also built based on the WMT13 QE Task 1.1 training set, by using the Java SVM Library.

The experiment was run in a desktop computer with an Intel® Core i7-4500U 1.8GHz and 8Gb of RAM running at 1600MHz. For each individual run of the quality estimation step was measured the time taken for it to perform the following 7 routines:

1. Quality Prediction Model Creation
2. Lowercasing
3. Language Model Loading
4. Calculation of Perplexity Measures (SRILM)
5. Feature Values Calculation
6. Feature Values Normalization (Java SVM Library)
7. Prediction Model Loading
8. Quality Scores Calculation

The routines selected represent the most processing intensive subtasks performed by the quality estimation step while producing quality scores. The results obtained are shown in Table 1.

Nº of Translations:	**62**	**125**	**250**	**500**
Routine 1	114969ms	114969ms	114969ms	114969ms
Routine 2	17ms	16ms	14ms	14ms
Routine 3	16688ms	15971ms	16304ms	16245ms
Routine 4	3027ms	2834ms	3014ms	3709ms
Routine 5	47ms	79ms	164ms	417ms
Routine 6	24ms	44ms	62ms	92ms
Routine 7	174ms	161ms	203ms	198ms
Routine 8	92ms	129ms	271ms	652ms
Total	**138.172s**	**138.330s**	**137.965s**	**138.001s**

Table 1 – Processing time measures of the performance experiment

One may notice that the three most time consuming routines are the quality prediction model estimation, the calculation of perplexity measures by SRILM and the loading of the language model. Although rather time consuming, the quality prediction model estimation only needs to be done once, which means that the user will not need to run the SVM Model Builder Step every time they want to produce quality estimates for a given dataset.

It is also possible to notice is that there are three routines which grow in processing time linearly as the number of the translations double, which are the Feature Values Calculation, Feature Values Normalization and the Quality Scores Estimation.

The total processing time measures also reveal that, because the two most time consuming tasks take several seconds to execute, doubling the number of input translations doesn't greatly affect the overall time it takes for the step to finish its processing.

3.2. Experiments with Quality Prediction for Sampling

These experiments are aimed at assessing a possible way to use the quality predictions generated using the Okapi version of QuEst: sampling "interesting" translations for further human inspection, for the purposes of quality assurance, for example, via error categorisation. The assumption is that interesting translations will be those with average to high quality; since very low quality translations are too difficult if not impossible to manually inspect, as they contain too many errors and these cannot be easily categorised.

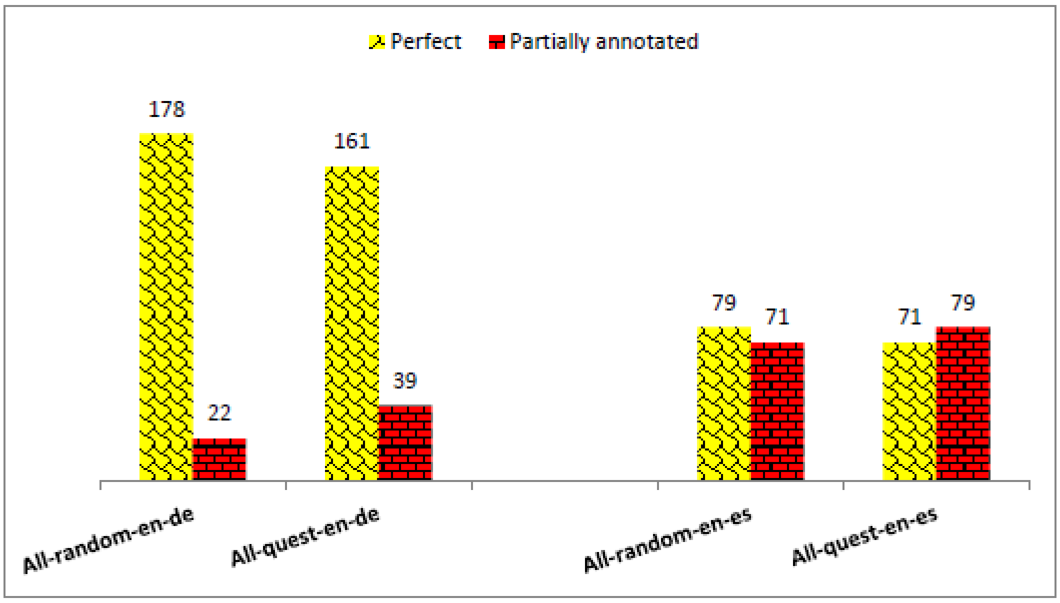
The experiments and results were submitted along with additional uses of QuEst for sampling, to the Language Resources and Evaluation Journal. We are still waiting for the reviews. In what follows we provide a brief summary of these experiments.

We compare the task of error categorisation on data selected at random against data selected according to quality predictions for two language pairs. The training contain news excerpts extracted from the WMT12 training set for only two language pairs: 2,209 English-Spanish (en-es) sentences and only 500 English-German (en-de) sentences. The test sets selected are the ones provided for WMT12, being composed of 3,003 pairs of translations for en-es and 2,737 translations for en-de. No true quality labels are available for these test sets. The goal is to take small samples for annotation from them based on their predicted quality. We took samples composed by the 50 highest scored sentences from both en-de and en-es test sets.

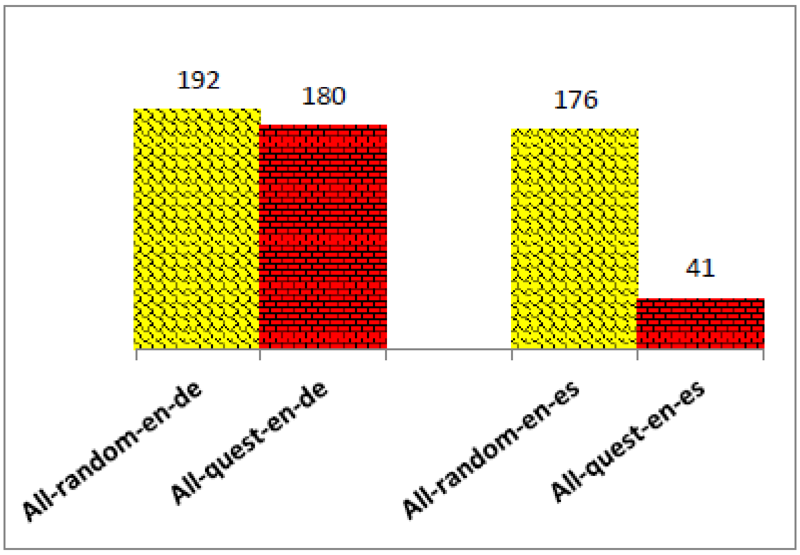
These 50 QuEst-based samples were mixed with a control set of 50 randomly selected samples of source segments and their respective translations. en-de segments were annotated by four translators and en-es were annotated by three translators. The annotation tool chosen for this set was Ocelot within Okapi, and all translators were properly instructed on using it before being presented to the actual sets of sentences. The error categories were a subset of MQM (Lommel et al., 2014): Terminology, Mistranslation, Omission, Addition, Untranslated, Register/Style, Spelling, Typography, Grammar, Uncategorized.

In what follows we highlight some interesting results.

- Perfect (no errors) segments versus segments annotated with errors (any type and number):



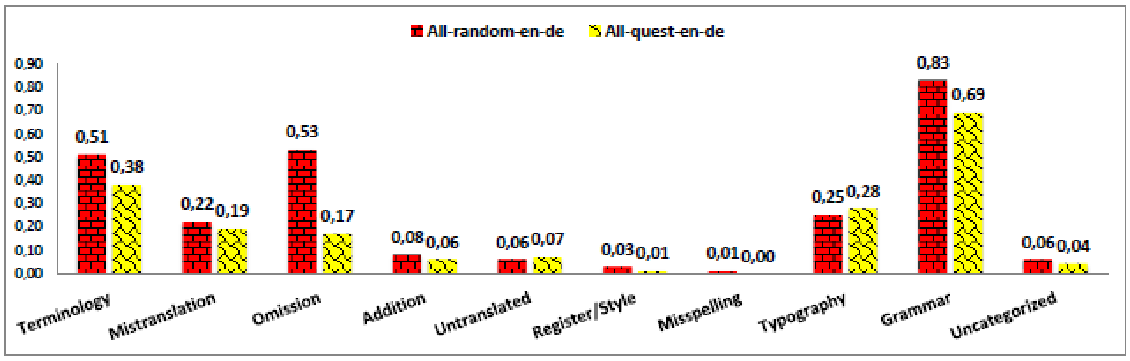
- Total errors annotated:



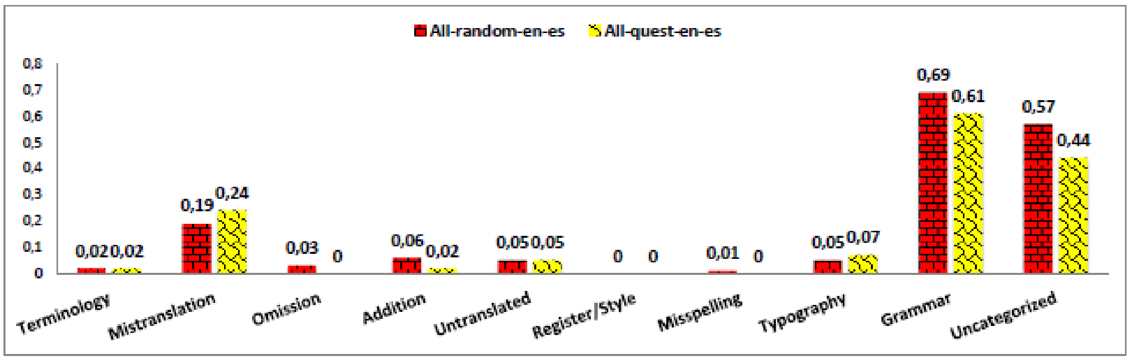
We noticed a very interesting phenomenon: even though there was a much higher proportion of segments judged perfect in the en-de than in the en-es set, the annotators of the en-de set seem to have found many more errors in QuEst scored segments than the annotators of the en-es set. Such observation leads us to believe that, in the case of en-de translations, there is not a lot of distinction in the nature of the sentences selected randomly and based on the QuEst score, since both selection methods resulted in a very similar amount of perfect segments, while the remaining annotated segments had nearly the same amount of errors. We believe that such a phenomenon provides further evidence that the main cause of this issue is the lack of adequacy of the features selected for the estimation of the quality prediction model used for the en-de dataset.

Such problem does not seem to have affected the en-es dataset in the same manner, since the segments selected randomly deemed not perfect had many more errors than the ones selected by using the QuEst score. In this case, the hypothesis that high quality translations selected by automatic quality estimation scores do not contain enough errors for them to be suitable for human quality assessment does not stand, because even though en-es segments selected by both criteria had almost the same proportion of perfect segments, the QuEst scored annotated segments still had a substantial amount of errors to be addressed.

- Distribution of errors:



(a) en-de



(b) en-es

This figure shows the distribution of the errors annotated. For both en-de and en-es sets, errors of grammatical nature are the most frequently observed, however, the proportion of uncategorized errors is much higher for en-es, which is an indication that the overall structural adequacy of many both randomly selected and QuEst scored Spanish translations have been more frequently compromised. The error type distribution is very similar for randomly selected and QuEst scored segments for both sets. Interesting differences between the errors found in the sets can be noticed: while en-es errors tend to be much more frequently related to grammaticality, annotations for en-de segments have a more even distribution between errors related to grammar, typography, terminology, mistranslation and omission. In counterpart, segments for both sets have seldom been annotated for errors related to addition of unnecessary lexical components, segment composition style, misspelling of words and untranslated segments.

As a general conclusion, the experiment has shown that the quality of prediction models can be greatly affected by the quality of the features selected. We also found that, contrary to our intuition, selecting the best scoring translations from a set can also be a reliable strategy for human assessment. Even though many segments were judged to be perfect by annotators, segments which were partially annotated still had a substantial, yet reduced, amount of errors to be addressed.

4. Budget

The budget planned for the project development and data collection has been spent as planned with the research intern and translators who participated in the annotation for Section 3.2. The remaining budget will be use used to support the participation of the research intern in the EAMT-15 conference to demonstrate the software.

5. Conclusion

Through the series of experiments conducted with the quality estimation step, it is possible to determine that it is a reliable and useful tool for determining the quality of translations. The performance results show that, even though some of the routines necessary for the quality estimates to be produced are quite time consuming, the step is overall agile, allowing for hundreds of translations to be evaluated in reasonable time.

6. References

R. E. Fan, P. H. Chen, and C. J. Lin. Working set selection using second order information for training SVM. Journal of Machine Learning Research 6, 1889-1918, 2005.

Ocelot. Available at: https://github.com/vistatec/ocelot. 2014.

XLIFF Format Documentation. Available at: http://docs.oasis-open.org/xliff/xliff-core/v2.0/xliff-core-v2.0.html. 2014.

A. Lommel, M. Popovic, A. Burchardt. Assessing inter-annotator agreement for translation error annotation. LREC Workshop Automatic and Manual Metrics for Operational Translation Evaluation, 2014.

Appendices

1. List of Featres in XML Format

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<features>

    <feature class="shef.mt.features.impl.bb.Feature1001"
description="number of tokens in the source sentence"
index="1001"/>

    <feature class="shef.mt.features.impl.bb.Feature1002"
description="number of tokens in the target sentence"
index="1002"/>

    <feature class="shef.mt.features.impl.bb.Feature1006"
description="average source token length" index="1006"/>

    <feature class="shef.mt.features.impl.bb.Feature1009"
description="LM probability of source sentence"
index="1009"/>

    <feature class="shef.mt.features.impl.bb.Feature1012"
description="LM probability of target sentence"
index="1012"/>

    <feature class="shef.mt.features.impl.bb.Feature1015"
description="number of occurrences of the target word
within the target hypothesis (averaged for all words in the
hypothesis - type/token ratio)" index="1015"/>

    <feature class="shef.mt.features.impl.bb.Feature1022"
description="average number of translations per source word
in the sentence (as given by IBM 1 table thresholded so
that prob(t|s) > 0.2)" index="1022"/>

    <feature class="shef.mt.features.impl.bb.Feature1036"
description="average number of translations per source word
in the sentence (as given by IBM 1 table thresholded so
that prob(t|s) > 0.01) weighted by the inverse frequency of
each word in the source corpus" index="1036"/>

    <feature class="shef.mt.features.impl.bb.Feature1046"
description="percentage of unigrams in quartile 1 of
```

```
frequency (lower frequency words) in a corpus of the source
language (SMT training corpus)" index="1046"/>

    <feature   class="shef.mt.features.impl.bb.Feature1049"
description="percentage  of  unigrams  in  quartile  4  of
frequency  (higher  frequency  words)  in  a  corpus  of  the
source sentence" index="1049"/>

    <feature   class="shef.mt.features.impl.bb.Feature1050"
description="percentage  of  bigrams  in  quartile  1  of
frequency  of  source  words  in  a  corpus  of  the  source
language" index="1050"/>

    <feature   class="shef.mt.features.impl.bb.Feature1053"
description="percentage  of  bigrams  in  quartile  4  of
frequency  of  source  words  in  a  corpus  of  the  source
language" index="1053"/>

    <feature   class="shef.mt.features.impl.bb.Feature1054"
description="percentage  of  trigrams  in  quartile  1  of
frequency  of  source  words  in  a  corpus  of  the  source
language" index="1054"/>

    <feature   class="shef.mt.features.impl.bb.Feature1057"
description="percentage  of  trigrams  in  quartile  4  of
frequency  of  source  words  in  a  corpus  of  the  source
language" index="1057"/>

    <feature   class="shef.mt.features.impl.bb.Feature1058"
description="percentage of unigrams in the source sentence
seen in a corpus (SMT training corpus)" index="1058"/>

    <feature   class="shef.mt.features.impl.bb.Feature1074"
description="number  of  punctuation  marks  in  the  source
sentence" index="1074"/>

    <feature   class="shef.mt.features.impl.bb.Feature1075"
description="number  of  punctuation  marks  in  the  target
sentence" index="1075"/>

</features>
```

2. Output Example in XML Format

```
<?xml version="1.0" encoding="UTF-8"?>

<root><translation><source>Norway's rakfisk: Is this the
world's smelliest fish?</source>

<target>De Noruega rakfisk: Es el pescado smelliest del
mundo?</target>

<features>-0.7407407407407407 -0.728813559322034  -
0.1111111111111116 0.83229387418794    0.7586961028899839
     -1.0 -0.8671070096706909 -0.8917140211166443 -1.0 -
0.25 -1.0 -0.6938775379008748 -1.0 -1.0 -0.25     -
0.6363636363636364  -0.6363636363636364</features>

<score>3.9053006637995322</score>

</translation>


<translation><source>Norway's five million people enjoy one
of the highest standards of living, not just in Europe, but
in the world.</source>

<target>Noruega de cinco millones de personas disfrutar de
uno de los más altos niveles de vida, no sólo en Europa,
pero en el mundo.</target>

<features>-0.2962962962962963 -0.22033898305084743     -
0.4370370370370369  0.5349145202982803  0.5645882250556289
     0.13360286564931934 -0.850641552367107  -
0.9702647121591003  -1.0 0.6000000000000001  -1.0
     0.35338334747583344 -1.0 -0.010100962020201165
     0.5294118000000001  -0.4545454545454546 -
0.4545454545454546</features>

<score>3.8672816179550944</score>

</translation>


<translation><source>Could the secret of the country's
success be connected to the local appetite for some
exceedingly smelly fish?</source>
```

```
<target>¿Podría el secreto del éxito del país estar
relacionado con el apetito locales para algunos
excesivamente malolientes pescado?</target>

<features>-0.37037037037037035     -0.423728813559322  -
0.2839506666666667  0.56056346858057    0.5124092613954379
     -0.46153844497041374     -0.7386034131850072 -
0.705089973303977   -1.0 0.5555555999999999  -1.0
     0.13445384903961521 -1.0 -0.20454543863636332     0.75
     -1.0 -0.6363636363636364</features>

<score>3.666694226241736</score>

</translation>

</root>
```

3. Output Example in TMX Format

```
<?xml version="1.0" encoding="UTF-8"?>

<tmx version="1.4"><header creationtool="unknown"
creationtoolversion="unknown" segtype="sentence" o-
tmf="unknown" adminlang="en" srclang="en"
datatype="plaintext"></header><body>

<tu>

<prop type="quest:id">/C:/Users/GustavoH/okapi-
quest/steps/quest/target/test-
classes/net/sf/okapi/steps/quest/source_test.eng_1</prop>

<prop type="quest:features">-0.7407407407407407  -
0.728813559322034   -0.11111111111111116
     0.83229387418794    0.7586961028899839  -1.0 -
0.8671070096706909  -0.8917140211166443 -1.0 -0.25     -1.0
     -0.6938775379008748 -1.0 -1.0 -0.25     -
0.6363636363636364  -0.6363636363636364</prop>

<prop type="quest:score">3.9053006637995322</prop>

<tuv xml:lang="en"><seg>Norway&apos;s rakfisk: Is this the
world&apos;s smelliest fish?</seg></tuv>

<tuv xml:lang="es"><seg>De Noruega rakfisk: Es el pescado
smelliest del mundo?</seg></tuv>

</tu>
```

```
<tu>

<prop type="quest:id">/C:/Users/GustavoH/okapi-
quest/steps/quest/target/test-
classes/net/sf/okapi/steps/quest/source_test.eng_2</prop>

<prop type="quest:features">-0.2962962962962963  -
0.22033898305084743 -0.4370370370370369 0.5349145202982803
     0.5645882250556289  0.13360286564931934 -
0.850641552367107   -0.9702647121591003 -1.0
     0.6000000000000001  -1.0 0.35338334747583344 -1.0 -
0.010100962020201165     0.5294118000000001  -
0.4545454545454546  -0.4545454545454546</prop>

<prop type="quest:score">3.8672816179550944</prop>

<tuv xml:lang="en"><seg>Norway&apos;s five million people
enjoy one of the highest standards of living, not just in
Europe, but in the world.</seg></tuv>

<tuv xml:lang="es"><seg>Noruega de cinco millones de
personas disfrutar de uno de los más altos niveles de vida,
no sólo en Europa, pero en el mundo.</seg></tuv>

</tu>

</body>

</tmx>
```