Unisa system submission at NLP4TM 2016

Friedel Wolff
Academy of African Languages and Science
College of Graduate Studies
University of South Africa

Abstract

Our entry for the NLP4TM shared task is a machine learning system that learns to classify erroneous translation pairs. It is trained on features relating to segment length, translation checks, spelling and grammar errors, and additionally use external data for detecting problems with fluency and lexical choice. The system was the best in 4 of the 9 subtasks, and shares the first place in an additional subtask.
Keywords: translation memory, cleaning, quality

1. Introduction

This paper describes our entry to the NLP4TM 2016 shared task.[1] The problem is approached as a classification task solved by machine learning similar to the approach taken by (Barbu, 2015).

Finding errors in human produced text is a long-time activity in natural language processing. Spell checkers and grammar checkers are arguably the most well known due to integration in word processors. Identifying erroneous translation pairs can amount to a large undertaking, depending on the type of errors expected. Barbu mentions a task of identifying random text, chat, incorrect language and partial translations in a translation memory. This is a much different matter from identifying incorrect style, lack of fluency, less than ideal lexical choice or small typographic errors, such as spacing and punctuation.

We select features (section 2) based on previous work and existing tools, and in anticipation of certain errors. Missing content in the source or target might be due to alignment errors or translator mistake. A number of orthographic and non-linguistic parts of text, such as punctuation, numbers, spacing, URLs, e-mail addresses and XML tags should most often be consistent between source and target texts. Where they are not, it might be an indication of an error. Errors in terms of spelling and grammar are of a more linguistic kind. Fluency and lexical choice are often more subjective in nature, but we also anticipate some errors of this kind.

The discussion above of spell checkers and grammar checkers are not only of an introductory nature. Our system combines several quality checkers, including a spell checker and grammar checker. We additionally employ rule-based translation checkers as additional features in this category.

Lastly we include features using external data, both bilingual and monolingual to detect issues with fluency and lexical choice.

We combine information from these tools as features in a machine learning approach (section 3), which was competitive in the shared task (see the results in section 4).

2. Features

We include some features based on statistics and measuring (such as length related features) as well as features based on simple heuristics. Some of the error categories reported by (Barbu, 2015) are handled differently in our feature set as described in the following subsections.

2.1. Length features

We include the Gale-Church ratio from the provided baseline as a feature. It follows a formulation given by (Tiedemann, 2011). In contrast with the provided baseline, however, we compute it based on all characters (including whitespace) as recommended by (Gale and Church, 1993). In the notation below, $|s|$ is the length of the string s.

$$CG = \frac{|s| - |t|}{\sqrt{3.4(|s| + |t|)}}$$

Other features derived from the lengths of the source and target segment are also included. We include the source and target lengths as is, as well as a feature calculated on the difference between the two. Since the difference between source and target lengths are much larger for longer strings than for shorter strings, we perform a normalisation in an attempt to produce a more consistent feature. We calculate the standard score (or z-score) in the hope that the feature values of different categories would be more easily separable in a linear way. For a source segment s and target segment t, we calculate this difference this way:

$$\Delta_{length} = \frac{|s| - \mu_s}{\sigma_s} - \frac{|t| - \mu_t}{\sigma_t}$$

where μ_s and σ_s is the mean and standard deviation respectively of the lengths of the source texts in the training data, and similarly for μ_t and σ_t. This standardisation is best done on normally distributed data. Since the distribution of lengths is not normally distributed, a more suitable normalisation should be considered, or a transformation on the data.

2.2. pofilter

The Translate Toolkit[2] is a toolkit and application library for software localisation and localisation engineering. It is widely used in Free and Open Source Software (FOSS) localisation, and includes functionality and customisation for several popular FOSS file formats and projects. It is used in several end-user applications for software localisation such as Pootle, Virtaal, Weblate, Damned Lies, Pontoon and

[1] http://rgcl.wlv.ac.uk/nlp4tm2016/shared-task/

[2] http://docs.translatehouse.org/projects/translate-toolkit/

the Wordforge translation editor. One area of functionality is for quality checking of translations. These are based on rules and heuristics to detect issues in translations that might warrant further review. These are accessible through an API or a command line tool, `pofilter`. [3]
Most of the tests simply compare some aspect of the source and target text. If an issue is identified by a check, it might be serious, such as errors with XML tags or printf variables, or of a less serious nature, for example punctuation consistency. The pofilter documentation lists the categories of checks as critical, functional, cosmetic and extraction.[4] For our purposes only the first three of these four categories are relevant. Some of the individual tests are for project specific issues (such as the `gconf` test for GNOME localisation), or require configuration, such as `notranslatewords` (a test for words that should remain unchanged in the target. We therefore only use a subset of these tests, and view each of these as a feature in our machine learning approach.
These tests are all binary in nature: a segment pair is either classified as failing or not. For several of the features it might be the only classification that makes sense. For example, spacing at the end of the segments is either similar or not, and it seems doubtful that measuring the degree of similarity will be of much use. However, other features could be foreseen to count the number of possible issues with the translation, such as the number of problematic XML tags or the number of problematic quotation marks. However, in our work we just use these as binary features as the most straightforward way of using this library. Since some of these quality checks were similar in spirit to some of the other features of our machine learning setup, we removed the ones were we had a non-binary outcome for similar quality issues. Examples include the pofilter tests `short`, `long` and `spelling`.
These tests cover several types of possible quality issues, not all of which are present in the training data for the shared task. Many of them are specifically geared towards software localisation, and in some cases, specifically for certain FOSS projects. Many of the checks didn't find anything in the training data, and therefore were zero features, but might be useful in another context.

2.3. Spelling and grammar

Spell checking is a common assistive technology used during writing and translation. Spelling errors could be used as an indicator of bad translation or careless writing in general. We use the Free and Open Source Hunspell[5] spell checkers for the relevant languages through the enchant framework. Instead of merely counting the unrecognised words in the target text, we additionally keep track of unrecognised words in the source text. These unrecognised source words are then allowed verbatim in the target text under the assumption that they could be named entities that would be unrecognised in most languages. A similar approach is used in the spellcheck test of pofilter. The difference is that we count the number of possible errors, while pofilter only makes a binary judgement per segment pair. We normalise the count by the number of tokens in the target text.
LanguageTool[6] is an Open Source proofreading program. It has support for more than 20 languages, although the level of support for each language varies considerably. It has support for all four languages in the data of the shared task. The level of support as mentioned on their web page is summarised in table 1 for the relevant languages of the shared task. Apart from merely analysing the target text, we enable the "bitext mode" where knowledge of false friends with another language is taken into account.

Language	XML rules	Java rules	False friends	Confusion pairs
en	1318	16	356	485
de	2076	24	126	26
es	92	1	57	0
it	135	2	37	0

Table 1: Rules in LanguageTool

We calculate two features based on the grammar checker:

- the number of issues raised in the target text
- the difference between the number of issues reported for the source and target texts

In addition to performing basic style and grammar checking, LanguageTool can also perform spell checking. Since we perform spell checking in a separate feature, this was disabled for the features calculated using LanguageTool.

2.4. Statistical features using external data

2.4.1. Language probability

Fluency is often mentioned in the evaluation of translation. A language model is employed in statistical machine translation to ensure fluent output. As another feature for our classifier, we use a language model to estimate the probability of a segment of text. There could be many reasons for a low probability estimation of a given text: it could simply be about a less frequently discussed topics, for example. We therefore estimate the probability of both the source and the target text, and use the difference as the learning feature. If the topic under discussion is rare, the probability of both the source and the target should be low, and the difference between them should be comparable to the difference between the probabilities for a more common topic.
The probability of a given text depends greatly on the length of the segment. A longer segment has a lower probability. This causes a skewed distribution of probabilities, and causes a greater difference between source and target probabilities in the longer segments — also for well translated segments. We therefore try to normalise the probability to lessen the effect of segment length somewhat. This way the difference between the two probabilities are hopefully a more consistent feature between segments of different lengths.

[3] http://docs.translatehouse.org/projects/translate-toolkit/en/stable-1.14.0/commands/pofilter.html
[4] http://docs.translatehouse.org/projects/translate-toolkit/en/stable-1.14.0/commands/pofilter_tests.html
[5] https://hunspell.github.io/
[6] https://languagetool.org/

For a source segment s of length $|s|$, we obtain a probability $p(s)$ estimated according to the language model, and then adjust it for the segment length by multiplying with $10^{|s|}$. Since we handle the probabilities as log probabilities, it simply means that we obtain a feature value for s as

$$log_{10}p(s) + |s|$$

and similarly for the target segment. These values are standardised by calculating z-scores for both source and target, and the difference between these standardised scores is used as the feature, similar to the length difference mentioned above.

This feature should be able to detect some agreement errors, incorrect word order, and possibly even typos in common words, as all of these should affect the probability of one side of the source-target pair in a way that should be reflected in the difference.

We trained language models on Europarl (Koehn, 2005) for the four languages of the shared task. Although this is not a balanced corpus, we argue that it doesn't matter as much as that it is unbalanced in the same way for every language. Since we are only using the difference in probability between the two languages' language models, we hypothesise that the bias of the language model would be similar in both the source and target language.

We implemented a 5-gram language model using Kenlm[7] (Heafield et al., 2013) which uses modified Kneser-Ney smoothing.

2.4.2. Lexical translation choice

We also investigated the use of external bilingual data. From bilingual data we can train word alignment models. We limited our approach to only doing alignment with IBM model 2 (Brown et al., 1993). We used the bilingual aligned Europarl corpus (Koehn, 2005) for each language pair.

We calculate a simple feature based on lexical translations. (Barbu, 2015) suggested using a bilingual dictionary instead of full-blown machine translation. We use the lexical translation probabilities and extract only high-probability translations in both directions. We then test for the presence of the expected translation, and count the number of "missing" dictionary entries in both directions. This way, no information about token order or word alignment is used. This is much faster than calculating the translation probability over the whole segment according to IBM model 2 or similar, since no word alignment step is required. The usefulness of this feature is expected to be heavily influenced by the training data that the bilingual dictionary is extracted from. Using hand-crafted terminologies of a translation project might give even better results, as long as ambiguity can be avoided or handled adequately. Here we simply used the texts in a bag-of-word fashion with no disambiguation.

2.5. Normalisation

The value of several of the features mentioned above depends heavily on segment length. Some of our features performed better when the raw values (such as the number of spelling errors) were normalised to the segment length. This makes some sense when considering that the instructions for the classification specifically mentions that annotators should consider the number of errors relative to the length of the translation.[8] A few different normalisation methods were attempted as described above, but a more methodological approach to these should still be attempted at a later stage.

3. Classifier

We use a support vector classifier with a linear kernel. The system is implemented in Python using scikit-learn. (Pedregosa et al., 2011) Apart from specifying the use of a linear kernel, the default settings were used for the support vector classifier.

In initial testing comparable results were obtained with a logistic regression classifier, as well as a random forest approach. While the support vector classifier seemed to perform slightly better, the other two classifiers trained faster, and seem like reasonable substitutes if runtime performance was a greater concern.

4. Results

The results for our system is summarised in table 2. The column containing the F_1 score indicates the average F_1 score for the binary tasks, and the weighted F_1 score for the fine-grained task.

Subtask	F_1 score	Correctly classified
Binary I (en-de)	**0.71**	**83.9%**
Binary I (en-es)	0.755	81.4%
Binary I (en-it)	**0.745**	**78.0%**
Binary II (en-de)	**0.68**	**88.3%**
Binary II (en-es)	0.76	86.0%
Binary II (en-it)	0.765	88.0%
Fine-grained (en-de)	**0.80**	**83.4%**
Fine-grained (en-es)	0.77	79.5%
Fine-grained (en-it)	**0.73**	**75.7%**

Table 2: Official results for the Unisa system. Winning scores are indicated in bold.

The Unisa system was indicated as top-scoring system for all three tasks in the language pair English-German, and for the fine-grained task in English-Italian. For the task Binary I in English-Italian, Unisa shared the first place. The fine-grained task is the hardest task, and here our results are consistently good amongst the competing systems.

There is some variance between the scores for the different languages. Our system seems to perform best for the English-German pair. Especially for the fine-grained classification task, we perform much better in this pair than the other two. The most obvious language specific difference lies in LanguageTool—it has very different levels of support for different languages (see table 1). The monolingual and bilingual corpora and the spelling checkers are language specific, but can be expected to provide similar training opportunities to the classifier.

[7] http://kheafield.com/code/kenlm/

[8] e.g. http://rgcl.wlv.ac.uk/wp-content/uploads/2015/12/InstructionsAnnotators_Spanish.pdf

5. Bibliographical References

Barbu, E. (2015). Spotting false translation segments in translation memories. In *Proceedings of the Workshop Natural Language Processing for Translation Memories*, pages 9–16, Hissar, Bulgaria, September. Association for Computational Linguistics.

Brown, P. F., Pietra, V. J. D., Pietra, S. A. D., and Mercer, R. L. (1993). The mathematics of statistical machine translation: parameter estimation. *Comput. Linguist.*, 19(2):263–311.

Gale, W. A. and Church, K. W. (1993). A program for aligning sentences in bilingual corpora. *Comput. Linguist.*, 19(1):75–102, March.

Heafield, K., Pouzyrevsky, I., Clark, J. H., and Koehn, P. (2013). Scalable modified Kneser-Ney language model estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 690–696, Sofia, Bulgaria, August.

Koehn, P. (2005). Europarl: A Parallel Corpus for Statistical Machine Translation. In *Conference Proceedings: the tenth Machine Translation Summit*, pages 79–86, Phuket, Thailand. AAMT, AAMT.

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

Tiedemann, J. (2011). *Bitext Alignment*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.