CMPT 413/825 Project
Using ROSE metric for Machine Translation Evaluation

Luiz Peres de Oliveira lperesde@sfu.ca
Turash Mosharraf tmosharr@sfu.ca
Jin Jung sjjung@sfu.ca
Justin Lew jylew@sfu.ca

1 Motivation

The motivation of using regression and ranking based optimisation in the machine translation evaluation problem is to improve the accuracy of the evaluation beyond the baseline implementation. The baseline implementation used the METEOR metric for finding precision and recall (Lavie and Agarwal, 2007). The METEOR algorithm did not allow for customization of features. However, the ROSE implementation allowed to attempt new features along with existing ones. Although we noticed that running the algorithm with some of the features present in the original ROSE implementation had a high time complexity and did not improve the results, we could our own set of features which would increase the accuracy of the evaluation.

2 Approach

The approach taken to improve from the baseline is partially based on the implementation of the ROSE metric (Song and Cohn, 2011). The algorithm uses a 27 length feature vector. We did not use the 33 elements that is specified in the original ROSE metric. We realize that many of the features such as the scoring post taggers would not improve the results, instead we used a customized feature vector consisting of the parameters:

- 1-4: n-gram precision, n=1...4
- 5-8: n-gram recall, n=1...4
- 9-12: n-gram f-measure, n=1...4
- 13: average n-gram precision for sentence
- 14: n-gram score at the document level
- 15-18: n-gram precision for sentence excluding stopwords, n=1...4
- 19-22: n-gram recall for sentence excluding stopwords, n=1...4
- 23-26: n-gram f-measure for sentence excluding stopwords, n=1...4
- 27: average n-gram precision for sentence excluding stopwords, n=1...4

We used the WordNet synonym sets and stemmer in order to improve the matching between hypothesis and reference sentences.
The first four elements of the feature vector are the n-gram precision for n=1..4, given by the formula

$$P_{h,r} = \frac{|h \cap r|}{|r|}$$

The next four elements are the n-gram recall for n=1...4, given by the formula

$$R_{h,r} = \frac{|h \cap r|}{|h|}$$

where h is the n-grams for the hypothesis sentence, and r is the n-grams for the reference sentence. The n-gram f-measure for n=1...4 for elements with ID 9-12 is defined as

$$\frac{2 \times P_{h,r} \times R_{h,r}}{P_{h,r} + R_{h,r}}.$$

Element 13 which is the average n-gram precision for hypothesis sentence is described in the paper by

(Song and Cohn, 2011). Elements 14 is customized such that we score the n-gram at document level. The n-gram model for element 14 is produced by a similar algorithm as the chunker in assignment 2 where we score the sentence according to the document.
Elements 15-27 of the feature vector are the same as the elements 1-13, except that we take every single hypothesis h and take its representation without stopwords h'. Stopwords is a set of high frequency words in corpora and may consist of words such as "a", "I", "the", and etc. Suppose a hypothesis sentence h is given by "it is a good day" and the reference sentence r is given by "what a good day." $h' \subseteq h$ and $r' \subseteq r$ where $h' = h - stopwords$ and $r' = r - stopwords$. Both h' and r', are going to be equal to "good day" in this case, therefore we associate a higher weight to elements 15-27 of the feature vector.

Our algorithm initializes the weight vector with random weights and improve until convergence using a neural network analogous to the approach used in assignment 2.

3 Data

The data file to train the evaluation model is from hyp1-hyp2-ref. The file consists of a triple (h1, h2, and r) where h1 and h2 are two translations to which is evaluated by the algorithm, along with a reference sentence of the hypothesis curated by a human translator.

The data file dev.answers contains the preference between the two hypothesis translations by a human translator. The numbers correspond to outputs of the function,

$$f(h_1, h_2, e) = \begin{cases} 1, & \text{if } h_1 \text{ is preferred to } h_2 \\ 0, & \text{if } h_1 \text{ is as good as } h_2 \\ \text{-1}, & \text{if } h_2 \text{ is preferred to } h_1 \end{cases}$$

where h_1 and h_2 are the two hypothesis translation and e is the reference translation.

4 Code

Data: (h_1, h_2, ref) in $\mathcal{D}$
Result: output $\alpha(h_1, h_2, e)$
Load n-gram model $ngram\_dict$ document level;
Initialize weight vectors wt randomly;
for $(h1, h2, e)$ *in* $\mathcal{D}$ **do**
  Preprocess triple (h_1, h_2, e);
  Create feature vectors vc_1, vc_2;
  Score feature vectors vc_1, vc_2;
  $\alpha = \sum_{i=1}^{27} wt_i(vc_{1_i} - vc_{2_i})$;
  output α;
end

5 Experimental Setup

Our experiment compares the accuracy between ROSE and the METEOR metric as shown in the baseline. Method 1 was the baseline implementation of the METRO metric. Methods 2 through 8 are modifications to the ROSE implementation (Song and Cohn, 2011) and the modifications to the sentence structure of the data set. Table 1 shows the methods implemented to improve the accuracy of the evaluator.

5.1 Results

Method	Time Execution [1]	Dev Score	Test Score
1	1min 33sec	0.510169	0.529
2	11sec	0.512868	0.529
3	50sec	0.517365	0.533
4	48sec	0.519008	0.539
5	55sec	0.520103	0.541
6	4min 51sec	0.523115	0.546
7	5min 43sec	0.526166	0.547
8	6min 17sec	0.530742	0.548
9	6min 17sec	0.537038	0.555
10	14min 30 sec	0.549867	0.564

[1]Time of execution is based on the performance of a 2.2GHz quad-core Intel Core i7 processor MacBook Pro

Figure 1: Dev Score and Test Score for each Methods

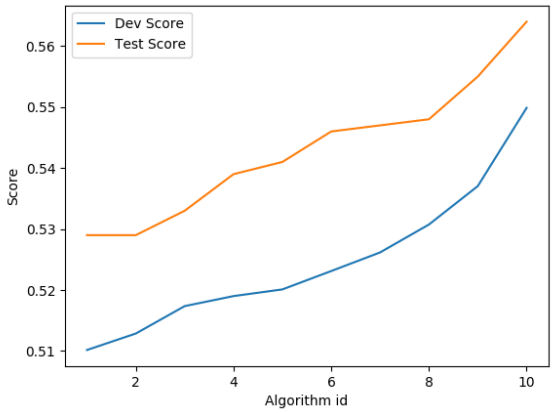
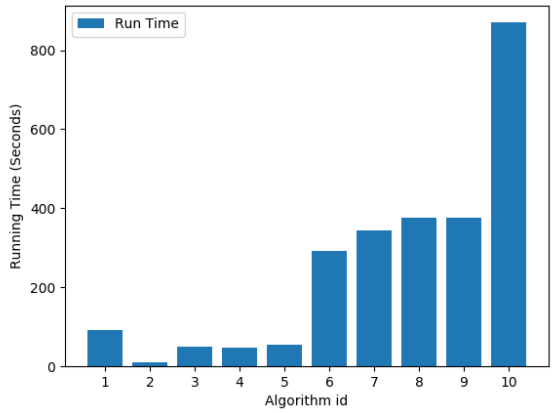


Figure 2: Execution Time for each Methods



Method	Description
1	METEOR
2	ROSE, only one feature vector with 13 elements
3	ROSE, added second feature vector with sentences without stopwords
4	Removed all characters with punctuation
5	Included scores for n-grams at sentence level. First feature vector contains 14 elements.
6	Removed all unicode characters and used WordNet to lemmatize sentence input.
7	Used WordNet to check similarities among words
8	Used levenshtein distance
9	Customised ROSE, uses 27 features, 13 from original ROSE algorithm + 14 custom features.
10	Machine Learning with Neural Network for 30 epochs.

5.2 Analysis of the Results

Our baseline is METEOR metric. The METEOR considers both precision and recall and adjusts the relative weight by parameter tuning.

First attempt to improve the baseline was to implement modified ROSE metric. One feature vector with 13 elements (precisions for 1-gram 2-gram, 3-gram, and 4-gram; recall for 1-gram, 2-gram, 3-gram, and 4-gram; f-measure for 1-gram, 2-gram, 3-gram, and 4-gram; average precision) was implemented initially which resulted in improving the time execution by 1 minute 22 seconds. Although the dev score increased from 0.510169 to 0.512868, the test score remained relatively same. Assuming that one feature vector was not enough, secondary feature vector was added to see if there were any improvement.

Additional to the implementation of secondary feature vector, stopwords (less important words) were also removed. By removing the stopwords, subset of strings were created from hypothesis and reference to be compared using the secondary feature vectors of 13 elements. The second feature vector is same as the first feature vector except the fact

that it is extracted from the modified hypothesis reference pair (the one without stopwords). This resulted in increasing the time of execution on by 49 seconds from Method 2; however, both the dev score and test score improved to 0.517365 and 0.533, respectively.

The next attempt was to remove all characters with punctuations. As there were less input to be compared, running time was faster but only by little. Although the dev score and the test score increased to 0.519008 and 0.539, respectively, the precision score actually went down. Also, it is possible that removing the characters with punctuations might be a drawback in the future.

In Method 5, implementation of 14th feature in the first feature vector was made. This 14th feature contained the sum of n-gram scores of reference at the document level. Basically, n-gram language models were applied to the reference to fetch scores for each and every word of the reference. This could have been expanded to the dictionary to get better precision score, but we decided to use the set of words that were originally provided. The result of this method ended up having a little longer execution time as additional n-gram language models had to run, which ended up with 55 seconds of running time. The dev score and the test score increased to 0.520103 and 0.541, respectively.

To improve the score further, all unicode characters were removed and Part-Of-Speech(POS) tagger was implemented in order to use WordNet stemmer, which could lemmatize sentence input. Due to the lemmatization of the input, the execution time increased tremendously because the WordNet stemmer had to go through every words of the input. Both the dev score and the test score increased to 0.523115 and 0.546, respectively. While the lemmatization increased the scores, it is possible that the result could have lost some precisions.

Both Method 7 and Method 8 were minor updates from Method 6. In Method 7, WordNet synsets was used to check similarities among input words. As more words were checked for the synonyms, the execution time increased. The scores improved a little bit, which ended up where the dev score resulting of 0.526166 and the test score resulting of 0.547. In Method 8, levenshtein distance was implemented, which resulted another minor improvement in the scores where the dev score resulted of 0.530742 and the test score resulted of 0.548.

Method 9 was resulted from combining the two feature vectors together to make one large feature vector of length 27. The POS tagger was also used. The weights were tuned by many trials. The accuracy of both dev and test corpus increased by a large margin. The dev score resulted in 0.537 and the test score resulted in 0.555. On the other hand, running time was relatively similar.

In Method 10, the neural network was implemented to learn the optimum weights of features. Using the first 25568 lines of the input file and dev.answer file as ground truth, the learning algorithm ran for 50 epochs. The total running time went up to 14 minutes 30 seconds due to the learning process; however, the accuracy of 0.549 on dev score and 0.564 on test score were achieved.

6 Future Work

Future work that extends the partial implementation of the ROSE metric would be to use a special kernel for categorizing text documents (Lodhi et al., 2002). In order to improve upon ROSE and BLEU, methods for combining scores from partial syntactic dependency matches along with n-gram matches using a statistical parser as presented in the paper (Kahn et al., 2009).

References

Jeremy G Kahn, Matthew Snover, and Mari Ostendorf. 2009. Expected dependency pair match: predicting translation quality with expected syntactic structure. *Machine Translation*, 23(2):169–179.

Alon Lavie and Abhaya Agarwal. 2007. 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. Association for Computational Linguistics.

Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. 2002. Text classification using string kernels. *Journal of Machine Learning Research*, 2(Feb):419–444.

Xingyi Song and Trevor Cohn. 2011. Regression and ranking based optimisation for sentence level machine translation evaluation. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 123–129. Association for Computational Linguistics.