Machine Translation Evaluator

Haitang Hu
hthu@cs.jhu.edu

March 26, 2015

1 Evaluator Methods

1.1 BLEU and Smoothing

BLEU score[2] can be simply modified to fit on the sentence level, here we just consider the score within a line of samples, which contains 2 candidate and 1 reference sentence. We employed the *Modified Count* from the original paper, which ensure that the max n-gram count would not exceed the maximum n-gram counts in the reference sentence. But pure **BLEU** score performs badly, since for those sentences that has no n-gram, their score will simply be 0. Naturally, we could try to smooth our **BLEU** score, lots of paper have explore the possible options on smoothing technology, here are some of them:

1. **BLEU+1**[3]
   The idea is simple. Suppose we have our original modified counts as c_m, and candidate n-gram length to be l_m, our original BLEU score will be

   $$score = \frac{c_m}{l_m}$$

   while the smoothing version is simply

   $$score = \frac{c_m + s}{l_m + s}$$

   If we set $s = 1$, then we are using standard **BLEU+1**. Actually, the s here could be set empirically to be the "count" that give us best score on our dev data set.

2. **Exponential Invert Smooth**[4]
   This technique is used in NIST official **BLEU** toolkit. The intuition is giving less weight for a sentence if it has more 0 match.

3. **Average n-gram**[4]
   This technique tries to average **BLEU** with adjacent n-gram matching.

Except the last technique, all smoothing techniques are implemented in this homework, but none of them gives the best score. Performance will be discussed in *Optimization and Evaluation* section.

1.2 String Kernel

String kernel[6] ignores our domains knowledge on the word field, instead, it treat sentences to be simply a sequence of strings. So, we could measure our sentence level similarity by looking at their common string subsequences. Notice that this kernel could be very slow, the author proposed a recursive algorithm to solve this problem linearly in the length of two sentence. Even a lot of works have been done on this recursive algorithm implementation, it is still too slow(Possibly takes 12 hours or more to train).

1.3 SVM Classifier

A classifier[1] based on SVM gives the best score among all the implementations.
The idea of this classifier is to take sentence evaluation related features, and train them on supervised data set with SVM classifier to get the optimal weights. The features construction follows the paper, with a little modification. Here we don't use the puncation counts, also, for convenience, we ignored the **POS** part. Since we are using SVM, the kernel method is worth discussing. Here, we just simply pick **RBF Kernel** because it gives the best efficiency on training. A **RBF Kernel** is a method that measures the similarity on a high dimension space via computing *squared Euclidean distance*. The last thing worth mentioning is class numbers. In this case, we are dealing with 3 classes ($\{-1, 0, 1\}$) of possible outputs, so we can't simply train our data by applying a binary classifier. A "One-vs-ALL" classifier has been employed to deal with the multiple classes. To conclude, the following contains the complete process of training a classifier:

1. Construct training candidate/reference sentence pair, and their corresponding labels.
2. Read through sentence pairs, compress all candidate/reference features into a long vector for each instance.
3. Concat the vectors to be a large feature matrix.
4. Train data using SVM classifier.
5. Contruct features for unlabelled data, and predict using trained weights.

Note: The SVM training part used scikit-learn[7].

2 Optimization

2.1 Pre-processing

1. **Stop Words**
   Stop words should be filtered out before we start work on our language tokens. A typical list of English stop-words have been used in this homework.
2. **Stemmer**
   Word stemmer is a basic technique on word pre-processing, which gives us a suffix-stripped word, that could be used to do n-gram matching. Though this offers higher probabilities on matching, but it loses imformation on morphology, and verb tense. A standard **porter stemmer**[5] has been implemented, but the experiment shows pure stemming gives worse performance.

2.2 Variant Weight BLEU

Traditional **BLEU** uses equal weight for all n-grams matching. But we could encode our belief on this weight to optimize it. For example, we could assign more weight on 4-gram, since it is rare and if one matches that then we have more reason to believe it having higher score. In this homework, the experiments shows that a reverse weights has better performance(give unigram more weights and etc.).

2.3 SVM Tuning

Linear SVC is actually the first choice of this task, but it failed to train. So the more efficient **RBF** is used. Also, since **RBF** is also measuring similarity, so it is more likely to have positive effect.

3 Experiments

3.1 Results

The experiments results are shown in table below:

Method	Score
Simple Meteor	0.500900
Pure BLEU	0.340312
BLEU+0.1	0.506844
BLEU+1	0.507431
BLEU+1 2-gram	0.519321
BLEU+1 2-gram + sw	0.516857
BLEU+1 4-gram	0.511499
BLEU+1 2-gram + stm + stp	0.496284
BLEU+1 4-gram + stm + stp	0.488853
SVM with RBF	**0.560075**

Table 1: Scores for different methods
Note: *stm*: stemmer, *stp*: remove stop words and punctuation
sw : same weight BLEU

3.2 Evaluation

Note that we actually get worse performance when we drop stop words, punctuation and use stemmer, that could simply because when we dropped these terms we also dropped useful information such as morphology, verb tense, etc.
Also, when comparing the results, we also found that for SVM classifier, we got very less 0 predictions, which could be caused by less training data. A simple way to deal with this is to manully duplicate the 0 training to data, so that we get more "space" for this class in our "plane", but we didn't try it out for time limitation.

References

[1] Xingyi Song and Trevor Cohn. *Regression and Ranking based Optimisation for Sentence Level Machine Translation Evaluation*, 2011.

[2] Kishore Papineni, Salim Roukos, Todd Ward and Wei-Jing Zhu. *BLEU: a Method for Automatic Evaluation of Machine Translation*, 2002.

[3] Chin-Yew Lin and Franz Josef Och. *Automatic Evaluation of Machine Translation Quality Using Longest Common Subsequence and Skip-Bigram Statistics*, 2004.

[4] Boxing Chen and Colin Cherry. *A Systematic Comparison of Smoothing Techniques for Sentence-Level BLEU*, 2014.

[5] M.F.Porter. *An algorithm for suffix stripping*, 1980.

[6] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini and Chris Watkins. *Text Classification using String Kernels*, 2002.

[7] *Scikit-Learn*, 2015. `http://scikit-learn.org`