CIS526 Homework 3: Evaluation

Joe Trovato

March 6, 2015

Motivation

After having built a aligner and decoder, I realized that these modules are useless without the ability to evaluate the proposed translations. Evaluation is integral to statistical machine translation because it determines how the translator learns, and what it believes to be a good translation. There are many metrics by which a translation can be evaluated but all of these metrics strive to encompass human understandability. While this quality is rather hard to quantify and codify, numerous evaluation techniques have been designed to accomplish this. In this report, I will describe my implementation of both the METEOR method as well as classify sentence according to ROSE.

Algorithm

As I learned more and more about evaluation metrics, my algorithm evolved from a simple unigram matching comparator to a sophisticated SVM with complex data features. I started by implementing the METEOR metric, which provided better results than the simple unigram matching metric provided. After a few hacks I had code that was beating the baseline. After I achieved baseline results, I sought for a more accurate method of evaluation, so I implemented ROSE. While I could not any mistakes, ROSE preformed worse than METEOR, contrary to the papers I read.

METEOR

when deciding how to expand the default code, I was between BLUE and METEOR. I choose to use METEOR because, it is shown to perform better on sentence-level evaluation, while BLUE does better with document level evaluation. Furthermore, METEOR was attractive because it is based on precision and recall of words in the sentences. See the equation below. My initial attempt did not include a chunking penalty. I did tune the alpha parameter to slightly favor recall in this calculation. ($\alpha = 0.75$) I found empirically that this slightly improved scores.
METEOR without chunking penalty:

$$score = \frac{P(h,e)R(h,e)}{(1-\alpha)R(h,e) + \alpha P(h,e)}$$

where

$$P(h,e) = precision \qquad R(h,e) = recall$$

After seeing decent results I introduced a chunking penalty. This algorithm basically looked for fewest number of chunks of reference sentence words the hypothesis sentence could be divided into. So if the sentences were identical, then there would be one chunk; if the sentence both consisted of the same tow clauses, but out of order, then there would be two chunks. Finally, if words in the hypothesis are not found

in the reference, it is automatically a chunk. It follows that two sentences sharing no words would have a number of chunks equal to the number of words in the hypothesis.
METEOR with chunking penalty:

$$score = (1 - \gamma(\frac{c}{m}^{\beta}))\frac{P(h,e)R(h,e)}{(1-\alpha)R(h,e) + \alpha P(h,e)}$$

where c is the number of chunks and m is the number of unigram matches between hypothesis and reference.

Some Hacks

To get the best results from our grader, a few hacks were employed. First, the evaluations were not very accurate to begin with, so the evaluator would misclassify about half of the data. While this is better than random, it is no better by much. After looking the confusion matrix, I noticed that the number of misclassifcations of hypothesis 2 outweighs the correct classifications when the hypothesis are equally likely. This means that I could take advantage of the class imbalance between a specific hypothesis being the best translation or both hypothesis being decent. Instead of classifying a set of hypothesis as 0, if it was not a definitive 1 or -1 I classified it as -1. This boosted the prediction of hypothesis 2 and killed any prediction of the hypotheses being equal but because of the imbalance of data in the data set, this hack allowed my score to go up.

Another useful hack was to check if the unmatched words in a hypothesis were English words. I edited the code to first match words from the hypothesis to reference sentence, then for all of the remaining words, I imparted a penalty if they were not in an English dictionary. I read in my computers dictionary as a list from /usr/share/dict/words then converted to a dict in Python for fast look-up times. I found that translations that could not translate a word at all were usually bad translations, so if the program encountered a word in a hypothesis that is not found in the English dictionary, it reduced the score by half, essentially throwing away that hypothesis. This yielded marginal improvements in accuracy, but was enough to put me over the baseline.

ROSE

After seeing decent results with METEOR, I attempted to implement the more sophisticated ROSE method of evaluation. This method involves transforming a hypothesis sentence into a feature vector that encodes, n-gram matching between hypothesis and reference, word counts, precision, and recall. There is also an extension that includes part of speech data but, I chose to stick with the basics. The exact features I used are shown in figure 1 labelled 1-17.

Once the hypothesis sentence is encoded as a feature vector a classifier can be trained given labelled data. Our data was label with the answer of which hypothesis sentence is better, but ROSE calls for labels that rate the quality of the translation. I was able to adapt the data given by converting the label according to Table 1. This way the better translation gets a confident score and if it is a tie they get medium scores.

After labelling the data and generating the features, I trained a classifying SVM or SVC from the skikit learn package for Python. The results were underwhelming. I experimented with only having GOOD and BAD classes but results still did not improve.

Results

Table 1 shows my results as my evaluator evolved. Despite claiming to have better accuracy, ROSE performed worse that METEOR. This could have been an issue with my implementation or with the data set. The Data specified by Song and Cohn in their paper describing ROSE, was labelled with three labels of translation confidence: GOOD, EDIT, and BAD. The data that we had was slightly different and my adaptation of the given data to this three class system may have skewed the results.

Table 1: Label Conversion Chart

Sentence	Original Label	New Label
hyp1	-1	0
hyp2	-1	2
hyp1	1	2
hyp2	1	0
hyp1	0	1
hyp2	0	1

ID	Description
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
14	Words count
15	Function words count
16	Punctuation count
17	Content words count
18-21	n-gram POS precision, n=1...4
22-25	n-gram POS recall, n=1...4
26-29	n-gram POS f-measure, n=1...4
30-33	n-gram POS string mixed precision, n=1...4

Figure 1: The ROSE features used for training and classification

Table 2: Accuracy of Evaluation

Evaluation Method	Approximate Accuracy Score
Word Counts (default code)	0.44
METEOR without chunking	0.46
METEOR without chunking, tuned parameters	0.48
METEOR with chunking, tuned parameters	0.50
METEOR with chunking and dictionary lookup	0.51
METEOR with chunking, dictionary lookup, and no ties	0.52
ROSE without POS features	0.41
ROSE without POS features, without ties	0.21