Homework 3 - Report

Jim Chen

March 6, 2015

First Attempt: Basic Precision/Recall Weighting :

Motivation and Description of Model

The first implementation I experimented with was a simple weighted precision recall as the scoring method. Precision and recall were calculated as the size of set intersections over set sizes (of the target and reference sentences). This in theory should be better than the pure match counts method.

Results

This naive implementation of the decoder runs slightly better than the default ranking and gives a 0.469 correlation to human ranking. Adjusting the parameters I was able to get to 0.523, which actually brings us close to the baseline

Second Attempt: Using classification :

Motivation and Description of Model

I next attempted to use classification algorithms to enhance the predicted results by training on the given dev answers. Training was done with the following features:

- N-gram (N=1,2,3,4) Precision
- N-gram (N=1,2,3,4) Recall
- Stemmed N-gram (N=1,2,3,4) Precision
- Stemmed N-gram (N=1,2,3,4) Recall
- Sentence length ratio (test/ref)

The training uses a maximum entropy classifier (Megam) and also uses stemming provided by nltk's porter stemmer. We divide sentences into three classes, comparatively positive (+1), comparatively negative (-1) and neutral (0). The scores for each classified sentence is the probability of being positive and neutral minus the probability of being negative. We compare the scores of the two sentences and assign the one with higher score as better.

Results

The results were less than optimal. On the 20000 sentence pairs and answers trained through a portion of local data, we were able to get high prediction accuracies 0.603 for testing on the rest of the dev data, however, when we train with the entire dev data (locally achieves 0.61) and submit, we get a very low score of 0.477. This may be the fact that the classifier is overfitting on our training data and causing the unlearned examples to not accurately reflect the optimal arrangements.

Further testing

Some further testing was done with the adjusting algorithms for precision and recall, adding chunked METEOR as another feature and varying the weights. None proved helpful enough to get us past baseline. We also experimented with logic to determine which sentence is better by adjusting weights to the probabilities or poposing new score functions. However they also proved to be of limited help.

Third Attempt: METEOR with chunking and N-gram METEOR :

Motivation and Description of Model

For the next part I implemented the basic chunking penalty for METEOR (with a constrained search to minimize chunk count). Another feature borrowed from our second classification attempt was to incorporate an N-gram METEOR score (treat each N-gram instead as a word). The METEOR scores would be added together by a weight $(0.8, 0.2)$ to create the final score we use to compare.

Results

The results are somewhat interesting but the increase in accuracy was limited. With simply combined bigram METEOR we were able to hit the baseline at 0.538 and with chunking plus a few other tweaks to the parameters, we got around to 0.540. It seems that bigrams help the most, as introducing 3-grams or higher either doesn't affect scores in cases, or even slightly brings the score lower due to the higher n-gram levels adding too many constraints. Further tweaks of the parameters and cuttoffs for $-1, 0, 1$ were tested, and many, while giving local improvements, did not perform better on the submitted full test set.

Adding WordNet as Matcher

I also experimented with using WordNet as part of the matching process when calculating values of m. WordNet path distance was used and a cutoff of 0.6 was selected to increase accuracy. These combined with stemming gave 0.0003 improvement and the best is still at 0.540.

Normalizing

Normalizing was used to improve data matches. For unigrams we incorproate both stemming match and WordNet path distances. For bigrams we use stemming matches. Also text was made lower case. However the most improvement was achieved by filtering each word to only allow alphanumeric characters. This final improvement brings us to 0.547972.

Other enhancements were not attempted due to lack of time to explore choices.