CS124 - Homework 4

Lucas Freitas

March 1, 2013

Problem 1

Design decision: for the implementation of this algorithm, I used padding to ensure that every possible boundary goes through the same number of comparisons with other n-grams.

The segmented file that was generated by the TANGO algorithm can be found in the submission folder under the name `alice_segmented.txt`.

Problem 2

1. `$ python freitas_hw3.py -f alice_tokenized.txt -p alice_segmented.txt -t 0.95 -n 10`

   ```
   precision 0.64873
   recall 0.80159

   1601 prediction lines used for evaluation
   ```

2. `$ python freitas_hw3.py -f alice_tokenized.txt -p alice_segmented.txt -t 0.95 -n 20`

   ```
   precision 0.65986
   recall 0.80326

   1601 prediction lines used for evaluation
   ```

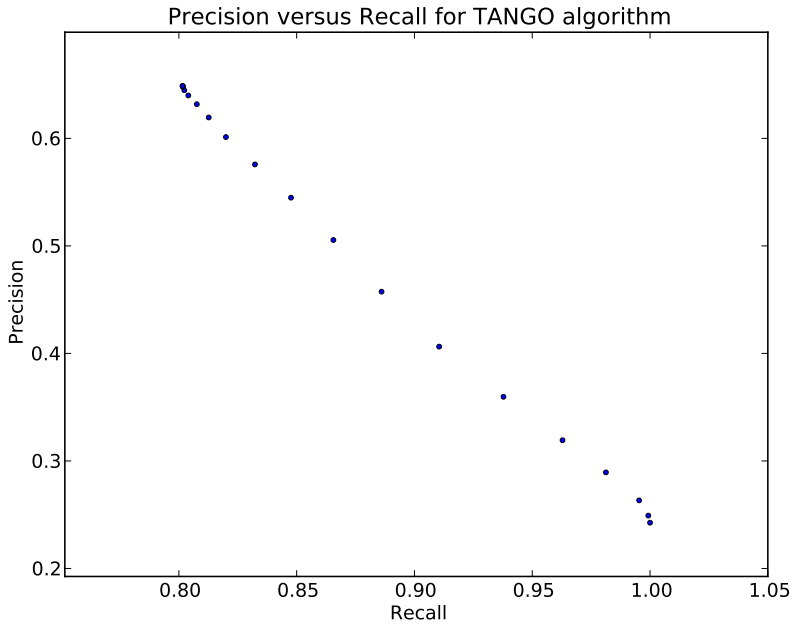
3. As we can see from the data above, the performance goes up when changing the maximum n-size from 10 to 20, but the increase is not that significant ($\approx 1.7\%$ increase in precision and $\approx 0.2\%$ increase in recall). That shows that having more possible n-grams, the algorithm considers the relation of position of word types more carefully, and the segmentation becomes more efficient.

   That is not that significative, however, for our performance. If we changed from an n-gram size of 2 to 10, we would see a very significant increase in performance (from 0.42759 to 0.64873 in precision and from 0.39285 to 0.80159 in recall), since we went from basically ignoring n-grams to actually understanding their relation to word segmentation, and considering that in our algorithm. That way, increasing our n-gram size will not help much at this point. For $n \geq 10$, we will get values that are extremely close to the maximum performance for our TANGO algorithm, and we should look for other methods, other than changing the maximum n-gram size, to increase performance.

   As a note, using F-measure, the performance goes from ≈ 0.717104 to ≈ 0.724533, leading to a $\approx 1.04\%$ increase in performance.

Problem 3

The graph showing the relation between precision and recall can be seen below (although not explicitly shown in the graph, an increase in the threshold leads to higher precision and lower recall).



The graph empirically shows two relations that we expected to happen:

1. Precision and recall have an inverse relationship. That is simple to see, however, since:

$$Precision = \frac{t_p}{t_p + f_p}$$

$$Recall = \frac{t_p}{t_p + f_n}$$

   If we decide to be reckless and guess spaces should be put everywhere, the number of true and false positives will be high. The number of false negatives, however, will probably be very low, and recall will be very close to one, while precision will be low. As we become more careful, the number of false positives would decrease, but the number of false negatives will increase, leading to higher precision and lower recall.

2. As the threshold increases, precision increases and recall decreases. That is not surprising either: a higher threshold just means that we are being more careful, which leads to higher precision and lower recall according to what we discussed in (a).

Problem 4

I would expect the performance of such algorithm to be considerably lower than the TANGO algorithm we used, since although the algorithm would know an approximation of the number of spaces to insert in the text corpus, it wouldn't know the distribution of lengths of words, or the relation between n-grams and segmentation. The algorithm would probably generate word tokens in an intermediate word size compared to possible lengths in the English dictionary word spectrum.

TANGO is usually biased into generating shorter word types, which leads to fairly good accuracy because the English language tends to have shorter words (it would be interesting to analyze how well TANGO performs for texts in German), while the algorithm described would be biased towards intermediately-sized words, which are not as common in the English language, and would lead to lower performance.