Jesse Waite

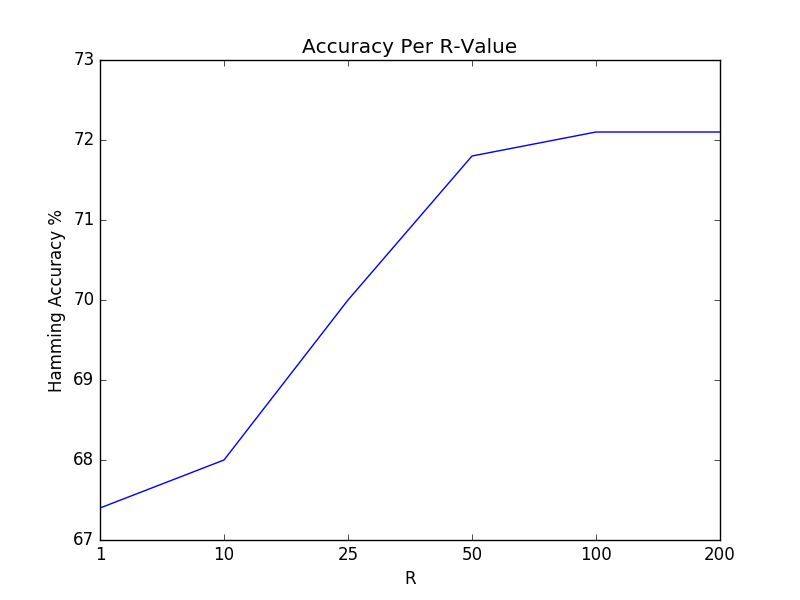
10752500

Structured Prediction, homework 1

1. The LaSO framework uses the concept of priority queue-based tree search to learn a parameter vector **w** capable of guiding the search toward y-good terminal nodes. The problem of cluster-based coreference resolution corresponds with finding a suitable assignment of mentions (“John”, “he”, “his”) to clusters C1, C2, thru Ci. So for the sentence In the context of cluster-based coreference resolution, the Karps algorithm???

2. b. The plots of accuracy for R=20, eta=0.01 are shown for the net-talk and ocr datasets, for maxIterations of 1, 10, 25, 50, and 100. For both datasets, the accuracy did not improve much after the first few iterations, as shown:

2. d. Next, we tested the effect of varied the R parameter to observe its impact on test performance. The hamming accuracy is shown below for R = 1, 10, 25, 50, 100, and 200, using only first-order features. Increasing R improved accuracy, however not considerably, and the improvement tapered after around R = 50.



2.e. One observation of randomized greedy search is that the uniform randomness of the start point for each search creates instability in the learning algorithm. Generating random label sequences will often yield sequences with labels that closely map to the input x-sequence, and thereby the sequence is already biased to climb toward knowledge it has already learned. In short, randomly-generated sequences are not the worst-case starting point for beginning a search procedure, since even randomness lends some bias by generating label positions that closely match their corresponding positions in the input x-sequence.

One way to get around this is to generate worst-case strings at every iteration. That way, the search procedure is forced to start learning from the label sequence furthest from the target sequence. This is a method for maximizing the amount of search performed by the search procedure, such that we maximize the number of mistakes made by the learner, and minimize the likelihood of its learning inferences being partially and accidentally correct.

To test this, instead of generating random sequences within *InferRGS()*, I simply return a list of the same random character repeated *n* times, where *n* is the length of the input sequence. For instance, “aaaaaa” or “bbbbbbb”. The effect is that the search procedure is forced to hill-climb according to only its current weights, not from positions with some small chance of being partially correct.

As shown, the search method works surprisingly well, for 15 iterations, eta=0.01, and first-order features. The search method appeared to work better than using completely random sequences, but not significantly. At most, it can be claimed the method didn’t harm performance.

[R10=70.3, 71.7, 68.5, 69.9 ,69.3]