CompLing I: Problem Set 5

Due via Canvas at 11:59 PM April 25
(Accepted until 11:59 PM May 13, Absolutely No Extensions!)

In this assignment, we will build models that can process the latent structure beneath observed sentences. These models are known as *syntactic parsers*. In this assignment, we will focus on bottom-up CKY parsing as described in lecture.

Required Files:

- 3 python scripts (`netid_mle.py` (1); `netid_cky.py` (2); `netid_eval.py` (3))
- I strongly recommend that you use numpy to build the required charts.
- You are only allowed to use the following external libraries: sys, re, numpy, and the included tree.py

1. [15 pts.] Write a script that uses MLE to estimate a probabilistic context-free grammar from syntactic annotations.
    - Your script should read in an annotated training file and a float from sys.argv:
      python SCRIPT.py trees.super 0.8
    - The input file should be in this format: (S (NP (Det The) (NN apple)) ...)
    - The input float determines how much of the input file to use for training (vs validation) (0.8 will use the first floor(80%) of the sentences for training; the rest will be used for validation to determine <unk> probability, etc).
    - Your script should distinguish preterminal categories from other non-terminals by adding ^X (e.g., NN apple → NN^X apple)
    - Your script should collapse unary rules by combining the node categories.
      E.g., (NP (NN^X boy)) would become (NP+NN^X boy)
    - Your script should save the resulting grammar to model.pcfg
      Example lines:
      G S : NP VP -1.6153903109613956
      X JJR^X : higher -1.2039728043259361
2. (Total 35 pts.) Write a script to generate the best CKY parses for sequences of word tokens.
    - Your script should read in a PCFG file and an unannotated test file from sys.argv:
      python SCRIPT.py model.pcfg test.sents
    - The PCFG format should be as shown above
    - The input text should be as space-delimited word tokens, one input sequence per line: e.g.,
      I have matches .
      Fruit flies like a banana .
      Apples I like are red

- Your script should use the *CKY algorithm* along with Viterbi decoding to compute the likelihood of the best sequence.
- For each input sequence, your script should output to a file called output.parses...
  - [20 pts.] $\log_e$-likelihood of the sequence (or nan)
  - [15 pts.] the best parse (or FAIL if not a valid sequence)

  Example lines:
  LL0: -18.930603676602015
  (ROOT+S (NP+PRPˆX I) (S|<VP-.> (VP (VBˆX have) (NP+NNSˆX matches)) (.ˆX .)))
  LL1: -25.359349666103643
  (ROOT+S (NP+NNSˆX <UNK-T>) (S|<VP-.> (VP (VBZˆX flies) (PP (INˆX like) (NP (DTˆX a) (NNˆX <UNK-T>)))) (.ˆX .)))
  LL2: nan
  FAIL

3. (Total 35 pts.) Write a parser evaluation script to evaluate the quality of output parses to gold standard parses.
   - Your script should read in a parser output file and a file with gold parses from sys.argv: python SCRIPT.py output.parses gold.parses
   - The output.parses file should be formatted as shown above
   - The gold.parses file should have one gold parse per line
     (ROOT (S (PRP I) (VP (VB have) (NNS matches))) (. .))
     (ROOT (S (NP (NN Fruit) (NNS flies)) (VP (VBZ like) (NP (DT a) (NN banana)))) (. .))
     (ROOT (S (NP (NNS Apples) (RC (PNP I) (VBZ like))) (VP (VBZ are) (JJ red))))
   - For each gold parse, your script should output to output.eval...
     - [15 pts.] unlabeled precision
     - [15 pts.] unlabeled recall
     - [5 pts.] unlabeled F_1 score

     P 0.8571428571428571;R 0.8571428571428571;F 0.8571428571428571
     P 0.8181818181818182;R 0.8181818181818182;F 0.8181818181818182
     P 0.0;R 0.0;F 0.0