CS 388 Natural Language Processing
Homework 3: Active Learning for Statistical Parsing

Due: April 8, 2015

Note: A FAQ for this assignment is available.

When learning PCFGs from data, we often want to get good performance while using as few annotated examples as possible. Constructing large training corpora like the Penn Treebank requires significant expert human effort, which can be quite costly. In this homework we'll roughly replicate an experiment in (Hwa, 2000) on active learning for PCFG parsing, which attempts to reduce annotation cost.

Active learning puts more of the burden of data collection on the learning system. In particular, *sample selection*, requires that the learner itself select the examples for annotatation from a large sample of initially unannotated data. The goal is to pick training examples wisely in order to minimize the amount of data that needs to be annotated to achieve a desired level of performance. For statistical parsing, a training instance is a sentence and the annotation is a parse tree supplied by a linguistic expert. First, the system is initially trained on a small randomly-selected sample of annotated instances to get started. Next, using the current learned model, the system selects a small batch of the most useful examples for annotation and asks the expert to annotate them. It then retrains on all the annotated data. Based on what it has learned, the system repeatedly selects small batches of examples for the user to annotate until the desired level of performance is reached or some resource limit is exhausted.

In experiments on sample selection, a corpus of completely annotated data is used to simulate active learning. First, a disjoint portion of data is set aside for testing. The remaining data is left for training, but is initially assumed to be unannotated. When the system requests annotation for a particular instance, the annotation for that instance is retrieved from the dataset. To measure performance, the accuracy of the current learned model is tested on the test set after every batch of labeled data is selected and the model is retrained. By comparing to the learning curve of a system that selects training examples randomly, the advantage of active learning can be ascertained.

However, using number of sentences is not a fair measure of training set complexity since longer sentences are clearly more difficult for a human to annotate. Active learners have an inherent bias to select more-complex, longer sentences, so just counting the number of training sentences would give them an unfair advantage. Therefore, using the number of words or the number of phrases (i.e. the number of internal nodes in the gold-standard parse tree, called the number of "brackets" in (Hwa, 2000)), is a fairer measure of training set size. Hwa (2000) plots the number of brackets in the training set on the Y axis and parsing accuracy on the X axis. Learning curves that instead plot training set size (e.g. number of words or brackets) on the X axis and test accuracy on the Y axis are more normal, so you should present results as learning curves.

The simplest approach to selecting training examples is *uncertainty sampling*. The learner first tries to annotate all of the remaining unannotated training examples itself, using its probabilistic model to assign a certainty to its labeling of each example. It then selects for annotation those examples in which it is most uncertain. By obtaining feedback on the cases in which it is most uncertain, it hopes to learn more than obtaining labels on random sentences in which its existing model is perhaps already quite confident in, and from which it would therefore not learn much.

For statistical parsing, there are several ways of measuring the uncertainty in the automatic annotation of a sentence.

1. As a naive method, just use the length of the sentence (number of words) since long sentences are usually more syntactically complex and so their parses are inevitably less certain. However, they will also be harder for the expert to annotate.
2. Use the probability assigned to the selected parse tree, the lower the probability, the higher the uncertainty that it is correct. However, since longer sentences require more production applications to generate, they will inevitably have lower probability. Assuming a binary parse tree, a sentence of length n has $n-1$ internal nodes or productions. Therefore, the probability of the parse is a product of $n - 1$ numbers, therefore taking the ($n - 1$)th root could help normalize for sentence length.
3. *Tree entropy* as defined by Hwa (2000), uses the probability distribution across all parses of a sentence as a measure of uncertainty. If a method can compute only the top K parses, then the entropy of this set of top parses only can be used to approximate tree entropy (i.e. let the set of parses V in the equation defining $TE(s,G)$ be just the set of top-K parses rather than all possible parses).

Using the Stanford parser as the underlying statistically trained parser, compare uncertainty sampling for active selection using each of these different ways of measuring uncertainty to random sample selection. Follow the steps outlined below and then turn in any code you write electronically and a hard-copy write-up describing your methods, your experimental results, and your interpretation and explanation of these results. Specific information regarding how to submit the electronic part is here.

1. Get a copy of the Stanford parser here. The FAQ page is a good starting place for info, but you may want to look through the

script `makeSerialized.csh` to get an idea for how to interact with the parser. The Stanford parser combines a powerful *unlexicalized* parser with a lexicalized dependency parser. If you're interested, you can read more about the unlexicalized parser here and the full factored model here.

2. The department machines have default Java version "1.7". In order to run the Stanford parser, you may want to download the newest version of Java SE Development Kit.

3. To familiarize yourself with the parser, run the PCFG parser on the annotated WSJ corpus located in `/projects/nlp/penn-treebank3/parsed/mrg/wsj/` and collect the labeled output:

```
java -cp "stanford-parser.jar:" -server -mx1500m edu.stanford.nlp.parser.lexparser.LexicalizedParser
     -evals "tsv" -goodPCFG \
     -train /projects/nlp/penn-treebank3/parsed/mrg/wsj/ 200-270 \
     -testTreebank /projects/nlp/penn-treebank3/parsed/mrg/wsj/ 2000-2100 \
  >  labeled_output.txt
```

The file `labeled_output.txt` will contain the parses of the WSJ sentences occurring in files 2000 to 2100 (roughly section 20) and will be in the Penn Treebank format. Training on 200-270 yields roughly 1000 sentences of input. The final output indicating performance will look something like this:

```
Testing on treebank done [108.0 sec].
pcfg LP/LR summary evalb: LP: 70.55 LR: 72.81 F1: 71.66 Exact: 12.56 N: 2069
dep DA summary evalb: LP: 0.0 LR: 0.0 F1: 0.0 Exact: 0.0 N: 0
factor LP/LR summary evalb: LP: 70.55 LR: 72.81 F1: 71.66 Exact: 12.56 N: 2069
factor Tag summary evalb: LP: 89.71 LR: 89.71 F1: 89.7 Exact: 14.35 N: 2069
factF1  factDA  factEx  pcfgF1  depDA   factTA  num
71.67           12.57   71.67           89.71   2069
```

This gives the bracketed scoring results (described in your book) including precision (LP), recall (LR) and F1. Pay attention to the line starting with `pcfg LP/LR` as it gives the scores for the unlexicalized pcfg parse. If you would like to try the factored parser, replace the argument `-goodPCFG` with `-goodFactored` and limit the sentence length (to limit memory use) by adding `-maxLength 40`. Note that the factored parser is about five times slower than the PCFG parser alone.

4. Create an initial training set, an "unlabeled" training pool for active learning, and a test set. To create the initial training set, extract the first 50 sentences from section 00. For the unlabeled training set, concatenate sections 01-03 of WSJ. This will give you roughly 4500 additional potential training sentences (approximately 100,000 words). For testing, use WSJ section 20.

5. Using the `ParserDemo.java` class as a example, develop a simple command line interface to the `LexicalizedParser` that includes support for active learning. Your package should train a parser on a given training set and evaluate it on a given test set, as with the bundled `LexicalizedParser`. Additionally, choose a random set of sentences from the "unlabeled" training pool whose word count totals approximately 1500 (this represents approximately 60 additional sentences of average length). Output the original training set plus the annotated versions of the randomly selected sentences as your next training set. Output the remaining "unlabeled" training instances as your next "unlabeled" training pool. Lastly, collect your results for this iteration, including at a minimum the following:
   - Iteration number
   - Number of training words
   - PCFG F1 score

6. Execute 20 iterations of your parser for the random selection function, selecting approx 1500 words of additional training data each iteration. You may wish to write a simple test harness script that automates this for you. The random selection function represents a baseline that your more sophisticated sample selection functions should outperform.

7. Implement the three selection functions describe above (sentence length, normalized probability of the top parse, and tree entropy using the top 10 PCFG parses), use each of them to replace random selection in the previous run, and collect results for each. Make sure to collect enough data to plot a learning curve (F1 score versus number of training words) for each sample selection function.

8. If you have additional time, you may wish to experiment by
   - changing the batch size for word count totals
   - changing the initial training set size
   - changing the unlabeled training set
   - changing the Top K parses in Tree entropy
   - changing the maximum iteration for each sampling function

Additionally, there are a set of tips for working with the Stanford Parser for this assignment.

Report

Your hard-copy report (of approximately 4-6 pages) should contain a concise but detailed discussion of the experiments you ran, including nicely formatted learning curves presenting the results. In your discussion, be sure to address at least the following questions:

1. Do the active learning methods perform better than random selection of training examples? Why?
2. Does active learning help across the complete learning curve, or are there parts of the learning curve where it performs best? Why?
3. How do the different methods for measuring uncertainty perform compared to each other? Which ones seem to work best? Try to explain any observed differences between methods.
4. How do your results compare to those presented by Hwa (2000)?

Code

Be sure to turn in any code you wrote, including any third party libraries you use, such as stanford parser jar files. That is, your turnin should be self contained, and is ready to run under the department machines. However, do not include the datasets in your submission. Further, be sure to include a README file including the specific commands you used for this assignment, and an example script if possible. Lastly, follow the steps here to turnin the code in Canvas.

References

(Hwa, 2000), "Sample selection for statistical grammar induction." In the Proceedings of the 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora.