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

Due: April 9, 2018

When learning parsers from treebank 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 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 or shift-reduce decisions to generate, they will inevitably have lower probability. For shift-reduce parsing, each word in a sentence requires a shift operator to remove it from the buffer and a reduce action to give it a parent in the parse tree. Therefore a parse probability is the product of $2n$ probabilities, one for each shift-reduce operation. Therefore, taking the $(2n)$th root could help normalize for sentence length. The Stanford neural dependency parser does not use probability, instead the parsing is performed by picking the transition with the highest final layer output at each step (more details can be found here). To get a probability measure for a transition, at any step in the parse, we use the softmax function over all the possible actions that can be taken at that step. To assign the probability to a parse tree we can use these transition probabilities in several ways
    1. *Raw Parse Probability:* Take the product of the probability assigned to every transition taken to get that parse
    2. *Margin Parse Probability:* Take the product of the margin between probabilities assigned to two top transitions at every step

Using the Stanford neural dependency parser, compare uncertainty sampling for active selection using each of these two different ways of measuring uncertainty to random sample selection. Follow the steps outlined below and then turn in any code you write electronically and an electronic copy of a report 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. The modified the Stanford CoreNLP jar can be found in `/u/mooney/cs388-code/nlp/active-learning/stanford-corenlp-jar`, and other details about the Stanford dependency parser is given here and its Javadoc can be found here.
2. The neural dependency parser requires the data to be in CoNLL-X format. The corpora have already been converted to this format and can be accessed at `/projects/nlp/penn-dependencybank`. This directory also contains an embedding vectors file which will be needed to train the dependency parser. There is a pre-trained model `english_SD.gz` which is needed if you want to use the parser off the shelf.
3. (Optional) To familiarize yourself with the parser, run it on the annotated WSJ corpus located in `/projects/nlp/penn-dependencybank/wsj-conllx` and collect the labeled output:

```
aquila$ cat /projects/nlp/penn-dependencybank/wsj-conllx/00/* > wsj_00.conllx
aquila$ cat /projects/nlp/penn-dependencybank/wsj-conllx/01/* > wsj_01.conllx
aquila$ java -cp /u/mooney/cs388-code/nlp/active-learning/stanford-corenlp-jar/*:. edu.stanford.nlp
-trainFile wsj_00.conllx
-testFile wsj_01.conllx
-embedFile /projects/nlp/penn-dependencybank/en-cw.txt -embeddingSize 50
-model output_model
-maxIter 50
-outFile annotations.conllx
```

   The file command_line.log illustrates the kind of output you should expect. The test set predictions can be found in the file annotations.conllx. The final performance will look something like this:

```
OOV Words: 6354 / 43148 = 14.73%
UAS = 70.3623
LAS = 60.3968
DependencyParser parsed 43148 words in 1807 sentences in 3.8s at 11487.8 w/s, 481.1 sent/s.
```

   This gives the Unlabelled (UAS) and Labelled (LAS) attachment scores. Note that the exact values may vary since there is randomization in the algorithm.
4. `DependencyParserAPIUsage.java` (found in `/u/mooney/cs388-code/nlp/active-learning`) class illustrates how to use the dependency parser programatically. It has example usage of functions to get *Raw* and *Margin* probability of the parses.
5. 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.
6. `DependencyParserAPIUsage.java` (found in `/u/mooney/cs388-code/nlp/active-learning`) class illustrates how to use the dependency parser programatically.
7. Using this, develop a simple command line interface to the `DependencyParser` 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 `DependencyParser`. 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
   - LAS score
8. 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.
9. Implement the three selection functions describe above (sentence length, normalized Raw probability of the top parse, normalized Margin probability of the top parse), 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 (LAS score versus number of training words) for each sample selection function.

Report

Your report (of approximately 3-5 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? Try to explain any observed differences between methods.
4. How do your results compare to those presented by Hwa (2000)? Try to explain any differences.

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.