11-711: Algorithms for NLP

Homework Assignment 4a: Deductive Weighted Agenda Parsing

Out: October 29, 2012
Due: November 7, 2012

Introduction

This assignment involves hands-on coding. To help you with time management, we have divided it into 2 parts (4a and 4b). For the first part, you have only one and a half weeks.

Get started now.

Background

Background reading: "Semiring Parsing." *Computational Linguistics* 25(4):573–605, December 1999, http://aclweb.org/anthology-new/J/J99/J99-4004.pdf. The relevant sections are 1–2 (pp. 573–589).

In this assignment, you will be exploring a very simple but general parser.

This parser solves the following dynamic program:

```
constit(X, I-1, I)  ⊕= unary(X, W) ⊗ word(W, I-1, I).
  constit(X, I, K)  ⊕= binary(X, Y1, Y2) ⊗ constit(Y1, I, J) ⊗ constit(Y2, J, K).
             goal   ⊕= constit(``S'', 0, N) if length(N).
```

Note that the `word` axioms are indexed with two integers with a difference of 1. Think of the integers as indexing inter-word positions, with 0 being the position before the beginning of the sentence, 1 being the position after the first word, and so on. So the two indices for a word are its "start" and "end" positions.

The agenda processing algorithm works as following:

1. Add each of the rules to a table of axiom production rules[1]
2. Add each of the words to the agenda as axioms

[1]The more general agenda algorithm presented in class would first add these to the agenda and then add them to a more complex chart, which remembers words, constituents, *and* rules. We'll keep things simple for this assignment.

3. Sort the agenda using your agenda ordering
4. While the agenda is non-empty:
   (a) Pop the highest-priority item off the agenda
   (b) Aggregate its value into the "chart" (using your semiring $\oplus$ to calculate values)
   (c) If it caused the chart to change and this agenda item along with other previously completed `constit` items in the chart can act as antecedents to form any consequent `constit` items, then:
      i. Append the value of the new consequent `constit` items to the agenda (using your semiring to calculate the consequent values by applying $\otimes$ to the current chart values of the antecedents)
   (d) Sort the agenda using your agenda ordering
5. Go to 4
6. Return the `goal` item

Provided Grammar

We provide you with a small dataset, which you will be using throughout this assignment. The grammar (`rules.txt`), was estimated on a small section of the Penn Treebank by collapsing unary productions in the original trees, extracting rule instances, counting them and then performing relative frequency estimation of $P(\alpha \mid A)$ for the rule $A \rightarrow \alpha$. Gaussian noise was added to break ties between rules (this means that the scores are no longer guaranteed to be probabilities). The file `sents.txt` contains plaintext sentences, which have been tokenized and escaped to match the expectations of the grammar; we guarantee that these sentences are in the language of the grammar.

Provided Boilerplate Code

We provide you with some boilerplate code in `starter.py`, which you should copy and separately modify for each question on this homework. It imports and uses `parser.py`, which implements the general agenda parsing algorithm described above. You may not modify `parser.py`. However, you may find it helpful to look through the code if you are uncertain how a particular part of the algorithm works.

You can test the boilerplate code by running: `cat sents.txt | ./starter.py rules.cnf`(which you will generate from `rules.txt`). Try replacing `cat` with `head` if you want to try just a few sentences.

1 Grammar Pre-Processing [5 points]

In real world NLP systems, it is often desirable to convert grammars to normal forms to improve parsing efficiency. You will begin by converting the given grammar to Chomsky normal form. This conversion was presented in class for unweighted CFGs. To extend this to the weighted case, all rules added in the conversion process have weight 1, and the first rule in each new right-branching chain has the weight of the original rule that the chain replaces.

For example, the rule:

```
[NP-SBJ-1] -> [DT] [NN] [NN]  / 1.09655914232
```

is rewritten

```
[NP-SBJ-1] -> [DT] [X4] / 1.09655914232
[X4] -> [NN] [NN] / 1.0
```

- **[2 points]** Run the provided script `cnf_convert.py` on `rules.txt` to convert the grammar to Chomsky normal form. How many rules are in (1) the original grammar and (2) the CNF grammar?
- **[3 points]** What is the largest number of **right-hand side** symbols to appear in a rule in the original grammar? How many rules in the CNF grammar correspond to this single rule in the original grammar?

2 Agenda Orderings [20 points]

For *each of the four orderings* in this section, you will answer the following questions:

- **[1 point]** Submit your working code (required to get points for the following; note that this code is already given to you for the first ordering).
- **[1 point]** How many agenda-add operations did it take to complete the parse of the first example sentence?
- **[2 points]** Explain why this ordering takes more or fewer operations than the other orderings. Mention how the ordering effects updates to memoized items (item in the "chart") and compare the ordering to the standard CKY algorithm. (You should answer this question for each ordering, but obviously after experimenting with all the orderings.)
- **[1 point]** Is this ordering asymptotically optimal for the Viterbi semiring?

(I.e., there are **20** points, 5 per ordering.)

You should start with `starter.py` (which implements the Viterbi semiring) and modify the `agendaComparator()` function to produce the desired ordering. Name each of your python implementation files `orderingN.py` where N is the ordering from below (e.g. `ordering1.py` for the first ordering). You will be using the Viterbi semiring without any pruning in this section.

Ordering 1: Order the agenda by increasing span sizes (narrow to wide), then left to right (by start position), then by largest score first.

Ordering 2: Order the agenda first by increasing span sizes (narrow to wide), then right to left (by start position), and then by largest score first.

Ordering 3: Order the agenda first by decreasing span sizes (wide to narrow), then left to right (by start position), and then by largest score first.

Ordering 4: Order the agenda by increasing span sizes (narrow to wide), then left to right (by start position), then by smallest score first.

3 Basic Semirings [15 points]

A semiring is defined by a set of values, 2 special values in that set, and 2 binary operators. At a high level:

- $\mathcal{K}$: The set of values.
- $\mathbb{0}$: An annhilator such that $\mathbb{0} \otimes x = \mathbb{0}$ (used for initializing blank chart entries).
- $\mathbb{1}$: An identity element $\mathbb{1} \otimes x = x$ (used for seeding agenda with axioms such as input words).
- $\otimes$: A multiplication-like operator (technically, a monoid), used for aggregating neighboring antecedents in a production rule as well as the weight of the production rule.
- $\oplus$: A commutative addition-like operator (technically, a monoid) used for aggregating the results of $\otimes$ with other previous results within a chart cell.

In addition, we define two more functions:

- `R(rule)` (Goodman, Section 2.3), which transforms a rule into a value usable by $\otimes$.
- `A(word, sOne)`, which is called whenever a new axiom is seeded onto the agenda. By default, this function returns semiring one, $\mathbb{1}$.

Note that $\otimes$ is *not* commutative in general. In this assignment, you can assume that binary productions will be evaluated in the following order (this may become important in later sections):

`constit(X, I, J)` $\oplus=$ `( R( X -> Y Z )` $\otimes$ `constit(Y,I,K) )` $\otimes$ `constit(Z,K,J)`

In this section and the rest of this homework, you'll be using the left-to-right agenda ordering with increasing span sizes with smallest values first, which is the default agenda comparator for `parse()`.

For each semiring in this section, you will implement the semiring in Python by starting with `starter.py`, which implements the Vierbi semiring, and modifying `semiZero, semiOne, semiPlus(), semiTimes()` and `R`. **Hint:** The semirings in this section require only trivial modifications to the provided code.

For each semiring, answer the following questions (5 points per semiring):

- **[2 points]** Submit your working code (required to get points for the following).
- **[1 point]** How many agenda-add operations did it take to execute this semiring for the first example sentence?
- **[2 points]** Why is this semiring more/less efficient in terms of agenda-add operations?

Your implementation of each semiring should be named `semiringN.py` where N is the number of the semiring. (e.g. `semiring1.py`).

Semiring 1: Recognition.
This semiring should return True if the string is in the language of the grammar and False otherwise. Your final output should be in the following format and should contain these lines (once for each sentence):

```
AGENDA ADDS: 35
GOAL SCORE: 1
```

You will receive some points for "working code" based on grepping your solution's output for "AGENDA ADDS:" and "GOAL SCORE".

Semiring 2: Derivation Counting
This semiring should count the number of complete derivations of a given sentence under the grammar.
Your final output should be in the following format and should contain these lines (once for each sentence):

```
AGENDA ADDS: 35
GOAL SCORE: 2
```

You will receive some points for "working code" based on grepping your solution's output for "AGENDA ADDS:" and "GOAL SCORE:".

Semiring 3: Inside (String) Score
This semiring returns the sum of the scores of all complete derivations of a given sentence under the grammar.
Your final output should be in the following format and should contain these lines (once for each sentence):

```
AGENDA ADDS: 35
GOAL SCORE: 2.76059169625e-08
```

You will receive some points for "working code" based on grepping your solution's output for "AGENDA ADDS:" and "GOAL SCORE:".

Submitting Your Code

In addition to submitting two hard copies of your assignment, all code for this assignment should be submitted at http://demo.ark.cs.cmu.edu:8265/turnin/. You should submit a single `.tar.gz` file containing exactly the following files:

- `ordering*.py`
- `semiring*.py`

Your tar file should be called `submission.tar.gz` and should *not* contain any internal directory structure. You can create such a tarball using:

```
tar -cvzf submission.tar.gz ordering*.py semiring*.py
```