Assignment: Dependency Parsing

Adapted from Prof Joachim's Nivre'scourse page

1 Report and Submission

In this assignment, you are going to implement some key components of a transition-based dependency parser for Hindi and analyze its behavior. Please use the python programming language to complete this assignment using the dataset provided above. Please submit the following for evaluation:

1. A PDF report answering the questions mentioned below.
2. Link to a github repository containing your code (embedded in the report).

The report should be written in your own words and the code should be written by you.

2 Starter Code and Data

Please make use of the following materials to solve this assignment:

1. **Starter code**: https://www2.lingfil.uu.se/cl/sara/courses/parsing24/assignment_data/dep_starter_code.tar.gz
2. **Data**: Link to the HUTB corpus of dependency trees: https://ltrc.iiit.ac.in/treebank_H2014/HDTB_pre_release_version-0.05.zip
3. **Reference**: Chapter 3 of Kubler's textbook https://talkbank.org/aphasia/publications/2009/K%C3%BCbler09.pdf

3 Transition System

A transition system for dependency parsing consists of a set of configurations (or parser states) and a set of transitions (or parse actions) for moving between configurations. You are going to implement the arc-eager transition system (see Section 3.4.1 in Kübler et al. textbook mentioned above), where configurations consist of a stack, a buffer, and an arc set, all represented as lists. Tokens are represented by integers corresponding to sentence positions (with 0 for the special root node), and dependency arcs are represented as triples (h, d, ℓ), where h is the head token, d the dependent token, and ℓ the dependency label.Given the sentence *"the cat sits on the mat today"*, a possible configuration is:

```
stack  = [0, 2]
buffer = [3, 4, 5, 6, 7]
arcs   = [(2, 1, "det")]
```

This configuration has the root and "cat" on the stack, "sits", "on", "the", "mat", and "today" in the buffer. The arc set contains a single dependency arc connecting the head word "cat" to the dependent word "the" with relation `det`. Make sure you understand how configurations are represented. Let us call the first word in the stack **top** and the first word in the buffer **next**. The transitions of the arc-eager system are:

- **SH** — move `next` from the buffer to the stack
- **RE** — remove `top` from the stack
- **(RA, label)** — add (`top`, `next`, `label`) to the arc set; move `next` from the buffer to the stack
- **(LA, label)** — add (`next`, `top`, `label`) to the arc set; remove `top` from the stack

Starter code: The file `transition.py` contains a dummy implementation of a transition-based parser that parses the sentence above by initializing the parser configuration to:

```
stack  = [0]
buffer = [1, 2, 3, 4, 5, 6, 7]
arcs   = []
```

and performing the following sequence of transitions:

```
SH
(LA, "det")
SH
(LA, "nsubj")
SH
SH
SH
(LA, "det")
(LA, "case")
(RA, "nmod")
RE
(RA, "nmod")
```

The method `transition(trans, stack, buffer, arcs)` is only a stub, which currently only implements `SH`. Your task is to implement the correct behavior for all four transitions. You can test your implementation by verifying that the parser produces the correct dependency tree for the sentence.

4 Oracle (20 marks)

Training a transition-based dependency parser means training a model for predicting the next transition given the current configuration. To do this, we first need to construct an oracle, a function that correctly predicts the next transition when given access to the correct dependency tree. You can find a definition of a static oracle for the arc-eager system in Goldberg and Nivre (CoLing 2012, Algorithm 1): `https://aclanthology.org/C12-1059.pdf`

Starter code: The file `oracle.py` implements an oracle parser that reads a set of syntactically annotated sentences from a file and tries to derive the correct trees by following the predictions of an `oracle()` method. This method is a stub that always predicts `SH`, and your task is to implement a better oracle. In addition, you have to replace the `transition()` method by your own implementation from above. You can test your implementation on the sentence "she gave the cat some food" by running:

```
python3 oracle.py < example.tab
```

or

```
python3 oracle.py tab < example.tab
```

if you want to see the result in the same tab-based format as the input file `example.tab`.

5 Testing and Error Analysis on a Real Treebank (10 marks)

Once your oracle works, test it on some real treebank data. For this purpose, use the development set of the Hindi treebank (simplified for clarity):

```
1 bhajpa NNP n 27
2 ne PSP psp 1
3 kendr NNPC _ 4
4 aur CC avy 6
5 keral NNPC _ 4
6 sarkar NNP n 27
7 par PSP psp 6
```

The columns contain tokens, part-of-speech tags, syntactic heads, and dependency labels. To run your oracle parser on the whole development set and print the derived trees in the same format, run:

```
python3 oracle.py tab < en-ud-dev.tab > en-ud-dev.out
```

Ideally, the trees output by your oracle parser should be identical to the original trees from the treebank; if not, analyze the discrepancies and discuss potential solutions.

6 Extra Credit (20 marks)

You must do one of the following (refer to Chapter 3 of the Kubler textbook):

- Implement a different transition system (e.g., arc-standard).
- Implement a dynamic oracle for the arc-eager transition system.
- Implement a projectivization algorithm for dependency trees.