University of Arizona		Homework#4	Dependency Parsing
CSC 585 Algorithms for NLP		Due date:	October 24, 2023
Fall 2023		**Name:**	
Instructor: Eduardo Blanco			

Instructions

- Bring a hard copy to class with your answers to Questions 1
- Submit your implementation for Questions 2 on GitHub Classroom. You will find the invite link for the repository in d2l. Follow these steps:

```
# Accept the invite link for the assignment available at d2l.
# It will create a repo named 04-dep-parsing-GITHUB_USER
#   Please use a github user name that is your name (or something close to it).
# Then set up the environment
> git clone YOUR_REPO
> cd 04-dep-parsing-GITHUB_USER
> python3 -m venv .env
> source .env/bin/activate
> pip install -r requirements.txt
```

You can run your code with the following command: `> python parser.py`.

Question 1 [20pt]
Go to `https://explosion.ai/demos/displacy`, uncheck *Merge Phrases* and run spaCy with the following sentences (do not change anything, copy and paste the sentences below (one at a time) without changing anything. Do not change punctuation, upper and lower case, etc.):

1. I ordered a pizza with my phone.
2. I ordered a pizza using my phone.
3. I ordered a pizza with anchovies.
4. Kevin McCarthy Ousted as House Speaker in Historic Vote.

First, copy the exact tokenization, part-of-speech tags, and dependencies you get from spaCy for the sentences above. Second, comment on the attachment errors you see (do not worry about dependency labels). Discuss why do you think the model is making mistakes.

Note: You may run spaCy from Python if you prefer. Spell out the version of spaCy and the specific model you use.

Question 2 [80pt]
Implement the arc-standard parsing algorithm. You are given:

- Code to download the training and test splits.
- An implementation of a dummy parser. This parser selects the first verb as root and then attaches all other tokens to this verb.
- A bare bones implementation of a less dummy parser.

- A bare bones implementation of the arc-standard parsing algorithm.

You can ignore the dependency types and only focus on attaching tokens to the right heads. Your job is to:

- Implement the `LessDummyParser`. It is supposed to be simple, just a slightly better heuristic than the `DummyParser`. Hard code some rules after looking at a few examples from the training split.
- Implement the `ArcStandardParser` parser. The `_train(self)` method should learn an oracle from the sentences in `self.train`. The high-level steps are to:
  - create the configurations and gold transitions (shift, left-arc, or right-arc), and
  - build an oracle based on the transitions.

  To be clear, this oracle will not be an actual oracle—it will make mistakes. You can implement the "most likely transition per configuration" as your oracle, but you are responsible to decide what to use from the configuration (which may be large). The `parse_sentence(self, sentence)` will apply the transitions according to the oracle and create the dependency tree accordingly. You are not expected to use machine learning to learn the oracle.

You will be graded as follows:

- `LessDummyParser`: 20 points if you get an unlabeled attachment score over 0.20 (the `DummyParser` gets 0.164).
- `ArcStandardParser`: 40 points if you implement the arc-standard parsing algorithm and create a sensible oracle. A parser that gets an unlabeled attachment score under 0.40 is not sensible.
- `ArcStandardParser`: up to 20 points depending on the unlabeled attachment score you get:
  - Students within 5% of the best performing parser will get 20 points.
  - Students within 10% of the best performing parser will get 15 points.
  - Students within 15% of the best performing parser will get 10 points.
  - Students within 20% of the best performing parser will get 5 points.

Work smart: train the parser with a handful of sentences until you know it works.