Homework 7: evaluate dependency parser

October 14, 2019

In this assignment you wil retrain a dependency parser from the spaCy NLP toolkit on a language of your choice and analyze its errors.

Pick a Dependency Treebank

Browse the Universal Dependencies website to see what dependency treebanks are available. Pick a language that you know, or are interested in. Then get the treebank over to your account on the GACRC teaching cluster using git as shown below. The prompt is cx-y to emphasize that you are are working at one of the numbered compute nodes rather than the login node.

```
cx-y$ git clone https://github.com/UniversalDependencies/UD_English-EWT
```

Encode for spaCy & retrain

Now convert the treebank you just transferred into the JSON format that spaCy uses. Follow the directions at https://spacy.io/usage/training#basics making sure to train only the tagger and the dependency parser. The example below abbreviates "English Web Treebank" by `ewt`; use an appropriate abbreviation for the language that you have chosen.

```
cx-y$ mkdir ewt-json
cx-y$ python -m spacy convert UD_English-EWT/en_ewt-ud-train.conllu ewt-json
cx-y$ python -m spacy convert UD_English-EWT/en_ewt-ud-dev.conllu ewt-json
cx-y$ python -m spacy convert UD_English-EWT/en_ewt-ud-test.conllu ewt-json
cx-y$ mkdir models
cx-y$ python -m spacy train en models ewt-json/en_ewt-ud-train.json ewt-json/en_ewt-ud-dev.json -p "tagger,parser"
```

You may wish to submit this last command as batch job; mine took 1h 45min and 827M RAM. Details about how to submit batch jobs are available on eLC in the September 5$^{\text{th}}$ module.

Evaluate how well it parses

After training has finished, evaluate your model. Use the `jtheval.py` variant of spaCy's `evaluate`. This script does the usual evaluation that spaCy performs, just with some extra debugging output turned on; copy it into your home directory from the instructor area on GACRC:

```
cx-y$ cp /work/ling6570/instructor_data/jtheval.py $HOME
```

Please make sure the environment variable `PYTHONIOENCODING` is set to `UTF-8` to avoid problems printing out this debugging information. Then run it as a script, passing in the final best model from the training step you did earlier

```
cx-y$ export PYTHONIOENCODING=UTF-8
cx-y$ python jtheval.py models/model-final ewt-json/en_ewt-ud-test.json
```

There should be lots of debugging output. Lines that start with `F` show falsely-proposed dependency arcs that shouldn't be there according to the gold standard. Lines that start with `M` show missed arcs that should have been there according to the gold standard. You can capture that information in a file using the unix `tee(1)` command like this

```
cx-y$ python jtheval.py models/model-final ewt-json/en_ewt-ud-test.json | tee report
```

Find interesting errors

Pick a specific sentence on which the spaCy dependency parser makes some errors. Sift through the detailed `report` using `sed`[1] to print just the lines between the sentence that you want and the first line that begins with something other than `F` or `M` .

```
$ sed -n '/Would love for you to join us ./,/^[^FM]/p' report
Would love for you to join us .
F for case you
F you obl love
F join xcomp love
M you nsubj join
M for mark join
M join ccomp love
```

Visualize

Now that you have isolated some interesting error, invoke the spaCy dependency parser on this mistake-inducing sentence. You will ultimately view the output through your web browser, but in order to make that work at the teaching cluster, we need to first set up two SSH tunnels.

1. Randomly pick one of the interative nodes on the teaching cluster, i.e. c2-$\{4, 5, 11, 12\}$. Remember your choice!
2. Log in to the teaching cluster as usual. You will need to type your UGA MyID password and authenticate with ArchPass Duo. This creates what we will call Terminal #1

[1] For more on the stream editor `sed` see chapters 4, 5 and 6 of Dougherty and Robbins 2010 which is available through the UGA library.

3. Set up an SSH tunnel between the teaching cluster's login node and the node that you randomly picked. You will need to remember the IP address of your selected compute node, as well as a port number for the tunnel. Port 8888 is a classic choice.

   For example, if you chose compute node c2-4, you can isolate that node's IP number with the `host` utility like this.

   ```
   teach$ IPUSED=$(host c2-4  | sed -nr 's|.*address (.*)|\1|p')
   teach$ PORTUSED=8888
   teach$ echo "IPUSED is" $IPUSED
   teach$ echo "PORTUSED is" $PORTUSED
   ```

   Using these values, the actual tunnel can be set up using ssh's -L flag:

   ```
   teach$ ssh -L $PORTUSED:localhost:$PORTUSED $IPUSED
   ```

4. Then, in a separate window on your local machine, create a second terminal — Terminal #2. Use this second terminal to set up another SSH tunnel between the teaching cluster's login node and the machine you are sitting in front of by issuing a command of this form:

   ```
   laptop$ ssh -L PORTUSED:IPUSED:PORTUSED myID@teach.gacrc.uga.edu
   ```

   Replace each of the boxed elements in the template above with an appropriate value that is specialized to your situation. The port number and IP address are the ones that you saw in the output of the echo commands above (Step 3). The element in front of the at-sign is just your UGA myID. Log in yet again with your MyID password and ArchPass Duo.

5. Now back in Terminal #1, visualize the output of the retrained dependency parser by applying it to the sentence that you located in the debugging output. Serve up a visualization of spaCy's answer like this:

   ```
   >>> import spacy
   >>> dp = spacy.load("models/model-final")
   >>> analysis = dp('Would love for you to join us . ') # sentence on which you observed errors
   >>> from spacy import displacy
   >>> displacy.serve(analysis,style="dep",port=8888)    # or appropriate value of PORTUSED
   ```

6. Browse http://localhost:8888/ (or whichever `PORTUSED` you chose) to see the dependency graph of spaCy's `analysis`. Type control-C in Terminal #1 to shut down the display server.

Compare against the dependency treebank

Compare what you see to the offical annotation by using a query tool like Grew-match To find my chosen sentence, I selected `UD_English-EWT` and entered this query, which is modeled on the example given in that site's help menu under "Search for a 4-gram of words"

```
pattern { N1 [form="Would"]; N2 [form="love"];  N3 [form="for"]; N4 [form="you"]; N1 < N2; N2 < N3; N3 < N4 }
```

The query specifies that each of the four words should appear in exactly the order they show up in in the debugging output that we produced earlier. This query suffices to isolate the sentence I want to examine, but you may be able to get away with fewer constraints.

Keeping the browser window for the Grew-match result near the displaCy window, and considering just the non-blue arcs (i.e. not "enhanced"), go through each of the `F` and `M` entries from your error report and look up the relevant syntactic relation in the table of universal dependencies.

Take time to reflect.

Turn in a write-up explaining what relations the spaCy dependency parser got wrong. Contrast your linguistic interpretation of both the gold-standard treebank annotations and the system's errorful output. Use your web browser's Print function to turn in PDFs of the two dependency graphs along with your writeup.