Master in Artificial Intelligence

Advanced Human Language Technologies

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Outline

Session 3 - DDI baseline

Assignment

Write a python program that parses all XML files in the folder given as argument and classifies drug-drug interactions between pairs of drugs. The program must use simple heuristic rules to carry out the task.

```
$ python3 ./baseline-DDI.py data/Devel/
DDI-DrugBank.d278.s0|DDI-DrugBank.d278.s0.e0|DDI-DrugBank.d278.s0.e1|0|null
DDI-MedLine.d88.s0|DDI-MedLine.d88.s0.e0|DDI-MedLine.d88.s0.e1|0|null
DDI-MedLine.d88.s0|DDI-MedLine.d88.s0.e0|DDI-MedLine.d88.s0.e2|0|null
DDI-MedLine.d88.s0|DDI-MedLine.d88.s0.e1|DDI-MedLine.d88.s0.e2|0|null
DDI-DrugBank.d398.s0|DDI-DrugBank.d398.s0.e0|DDI-DrugBank.d398.s0.e1|1|effect
DDI-DrugBank.d398.s0|DDI-DrugBank.d398.s0.e0|DDI-DrugBank.d398.s0.e2|1|effect
DDI-DrugBank.d398.s0|DDI-DrugBank.d398.s0.e2|DDI-DrugBank.d398.s0.e3|0|null
DDI-DrugBank.d398.s1|DDI-DrugBank.d398.s1.e0|DDI-DrugBank.d398.s1.e1|0|null
DDI-DrugBank.d211.s2|DDI-DrugBank.d211.s2.e0|DDI-DrugBank.d211.s2.e5|1|mechanism
DDI-DrugBank.d211.s2|DDI-DrugBank.d211.s2.e1|DDI-DrugBank.d211.s2.e2|0|null
...
```

Outline

General Structure - Main function


```
# process each file in directory
for f in listdir(inputdir) :
   # parse XML file, obtaining a DOM tree
   tree = parse(datadir + "/" + f)
   # process each sentence in the file
   sentences = tree.getElementsByTagName("sentence")
   for s in sentences :
      sid = s.attributes["id"].value # get sentence id
      stext = s.attributes["text"].value # get sentence text

      # load sentence entities into a dictionary
      entities = {}
      ents = s.getElementsByTagName("entity")
      for e in ents :
         id = e.attributes["id"].value
         offs = e.attributes["charOffset"].value.split("-")
         entities[id] = offs

      # Tokenize, tag, and parse sentence
      analysis = analyze(stext)

      # for each pair in the sentence, decide whether it is DDI and its type
      pairs = s.getElementsByTagName("pair")
      for p in pairs:
         id_e1 = p.attributes["e1"].value
         id_e2 = p.attributes["e2"].value
         (is_ddi,ddi_type) = check_interaction(analysis, entities, id_e1, id_e2)
         print("|".join([sid, id_e1 id_e2, is_ddi, ddi_type]), file=outf)

# get performance score
evaluate(inputdir,outputfile)
```

Outline

Resources

You will need to use a tokenizer, a part-of-speech tagger, a dependency parser.
We recommend Stanford CoreNLP dependency parser, which can be called via `nltk`, and integrates all three steps. (You'll need to add token spans, though):

1. Download and uncompress Stanford CoreNLP.
2. Launch a CoreNLP server:

```
cd stanford-corenlp-full-2018-10-05
java -mx4g -cp "*" edu.stanford.nlp.pipeline.StanfordCoreNLPServer
```

3. In your python program:

```
# import nltk CoreNLP module (just once)
from nltk.parse.corenlp import CoreNLPDependencyParser
# connect to your CoreNLP server (just once)
my_parser = CoreNLPDependencyParser(url="http://localhost:9000")
# parse text (as many times as needed)
mytree, = my_parser.raw_parse(mytext)
```

Outline

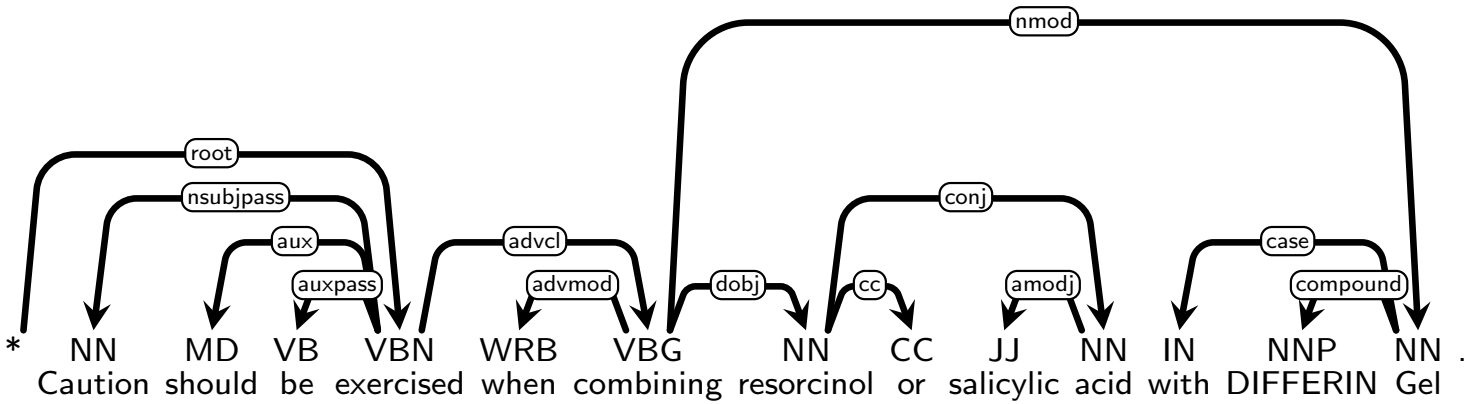
Functions - Analyze text

`def analyze(s) :`

- Input: Receives a sentence text `s`, and sends it to CoreNLP to obtain the tokens, tags, and dependency tree. It also adds the start/end offsets to each token
- Output: Returns the nltk DependencyGraph object produced by CoreNLP, enriched with token offsets.
- Example:

```
>>> analyze("Caution should be exercised when combining resorcinol or salicylic acid
with DIFFERIN Gel")
{0:{'head':None,'lemma':None,'rel':None,'tag':'TOP','word':None},
1:{'word':'Caution','head':4,'lemma':'caution','rel':'nsubjpass','tag':'NN','start':0,'en
2:{'word':'should','head':4,'lemma':'should','rel':'aux','tag':'MD','start':8,'end':13},
3:{'word':'be','head':4,'lemma':'be','rel':'auxpass','tag':'VB','start':15,'end':16},
4:{'word':'exercised','head':0,'lemma':'exercise','rel':'ROOT','tag':'VBN','start':18,'en
5:{'word':'when','head':6,'lemma':'when','rel':'advmod','tag':'WRB','start':28,'end':31}
6:{'word':'combining','head':4,'lemma':'combine','rel':'advcl','tag':'VBG','start':33,'en
7:{'word':'resorcinol','head':6,'lemma':'resorcinol','rel':'dobj','tag':'NN','start':43,'
8:{'word':'or','head':7,'lemma':'or','rel':'cc','tag':'CC','start':54,'end':55},
9:{'word':'salicylic','head':10,'lemma':'salicylic','rel':'amod','tag':'JJ','start':57,'e
10:{'word':'acid','head':7,'lemma':'acid','rel':'conj','tag':'NN','start':67,'end':70},
11:{'word':'with','head':13,'lemma':'with','rel':'case','tag':'IN','start':72,'end':75},
12:{'word':'DIFFERIN','head':13,'lemma':'DIFFERIN','rel':'compound','tag':'NNP','start':7
13:{'word':'Gel','head':6,'lemma':'gel','rel':'nmod','tag':'NN','start':86,'end':88},
14:{'word':'.','head':4,'lemma':'.','rel':'punct','tag':'.','start':89','end':89}}
```

Functions - Analyze text



Functions - Check for Drug-Drug Interactions

`def check_interaction(analysis,entities,e1,e2) :`

- Input: Receives a DependencyGraph object with all sentence information, a list of all entities in the sentence (`id` and offsets), and the `ids` of the two entities to be checked.
- Output: Returns a 0/1 value indicating whether the sentence states an interaction between entities `e1` and `e2`, and the type of interaction (`null` if there is none).

Functions - Evaluation

`def evaluate(inputdir, outputfile) :`

- Input: Receives a data directory and the filename for the results to evaluate. `inputdir` is the folder containing original XML (with the ground truth). `outputfile` is the file name with the entities produced by your system.
- Output: Prints statistics about the predicted entities in the given output file.
- Code:

```
os.system("java -jar eval/evaluateDDI.jar "
          + inputdir + " " + outputfile)
```

Note: `outputfile` must match the pattern: `task9.2_NAME_NUMBER.txt` (where `NAME` may be any string and `NUMBER` any natural number).
You can use this to encode the program version that produced the file.

Outline

Detecting interactions - Choosing rules

Function `check_interaction` will implement our rule-based interaction detector.
Strategy to follow:

- Examine the train data set and try to infer general rules that are right in most cases, even if they seldom apply (high precision, low recall).
  - Look at the text data directly (less useful)
  - Write small scripts that perform some kind of data exploration to find out features that distinguish drug names (more useful)

Detecting interactions - Some hints

Example observations that may lead to some rules:

- Pairs with an interaction of type *effect* often have clue words like *administer*, *potentiate*, *prevent*, etc. between e1 and e2.
- Pairs with an interaction of type *mechanism* often have clue words like *reduce*, *increase*, *decrease*, etc. between e1 and e2.
- Pairs with an interaction of type *int* often have clue words like *interact*, *interaction*, etc. between e1 and e2.
- Checking for the clue word position (before e1, between e1 and e2, after e2) is a pretty naive heuristics. Better results may be acheved if properties of the dependency tree are used (e.g. e1 is under e2, both e1 and e2 are under the same verb, e1 is [under] the subject of certain verbs (*enhance*,*reduce*,...), etc.)

Outline

Evaluating Results

After each change or new rule added, you must check whether it improves the performance of the system. We will evaluate performance using SemEval-2013-Task9 official evaluator.
Evaluation goals:

- Find out whether the added rule is useful or damaging
- Find out the weaknesses of our system to decide the target of new rules

Rule-based Systems Development Methodology

1. Start with a simple set of rules.
2. Use **Train** dataset to get insights about possible rules:
   - Extract statistics or data analysis from **Train** dataset to find patterns that may be good rules.
   - Run the rules on the **Train** dataset and check system errors and performance statistics to get hints of what needs improvement.
3. Create one (or a few) new rules
4. Run the new set of rules on the **Devel** dataset. Record the score and save the rules that produced it.
5. If the score is better, keep the new rules. If it is worse, back off to best rule set so far. Go to step 2 (or stop when the score is good enough or when no improving rules are found)
6. Once a satisfactory set of rules has been established, apply them to **Test** dataset, and record the score.

Rule-based Systems Development Methodology

- NEVER look at the **Devel** or **Test** dataset.
- **Train** dataset is used extract information that can be generalized to create new rules.
- **Devel** dataset is used only to obtain a score and decide whether newly added rules are useful or not.
- **Test** dataset is used only to obtain a final score on unseen data.

Exercise Goals

Goal 1:

Get an overall F_1 score of at least 0.15 on **Devel** dataset.

Deliverables

Prepare a report containing:

For Goal 1:

- Final version of `check_interaction` function (and any other subsidiary function used by it).
- Evaluator output for this version on **Devel** and **Test** datasets.

All code must be properly commented. Self-contained Jupyter notebooks are acceptable.