Advanced Human Language Technologies

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Outline

Session 3 - DDI baseline

Assignment

The main program 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|null
DDI-MedLine.d88.s0|DDI-MedLine.d88.s0.e0|DDI-MedLine.d88.s0.e1|null
DDI-MedLine.d88.s0|DDI-MedLine.d88.s0.e0|DDI-MedLine.d88.s0.e2|null
DDI-MedLine.d88.s0|DDI-MedLine.d88.s0.e1|DDI-MedLine.d88.s0.e2|null
DDI-DrugBank.d398.s0|DDI-DrugBank.d398.s0.e0|DDI-DrugBank.d398.s0.e1|effect
DDI-DrugBank.d398.s0|DDI-DrugBank.d398.s0.e0|DDI-DrugBank.d398.s0.e2|effect
DDI-DrugBank.d398.s0|DDI-DrugBank.d398.s0.e2|DDI-DrugBank.d398.s0.e3|null
DDI-DrugBank.d398.s1|DDI-DrugBank.d398.s1.e0|DDI-DrugBank.d398.s1.e1|null
DDI-DrugBank.d211.s2|DDI-DrugBank.d211.s2.e0|DDI-DrugBank.d211.s2.e5|mechanism
DDI-DrugBank.d211.s2|DDI-DrugBank.d211.s2.e1|DDI-DrugBank.d211.s2.e2|null
...
```

Outline

General Structure - Main function

```
# process each file in directory
for f in listdir(datadir) :
   # 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 :
          eid = e.attributes["id"].value
          entities[eid] = e.attributes["charOffset"].value.split("-")

      # Tokenize, tag, and parse sentence
      analysis = deptree(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
          ddi_type = check_interaction(analysis, entities, id_e1, id_e2)
          if ddi_type != None :
             print(sid+"|"+id_e1+"|"+id_e2+"|"+ddi_type], file=outf)

# get performance score
evaluator.evaluate("DDI",inputdir,outputfile)
```

Outline

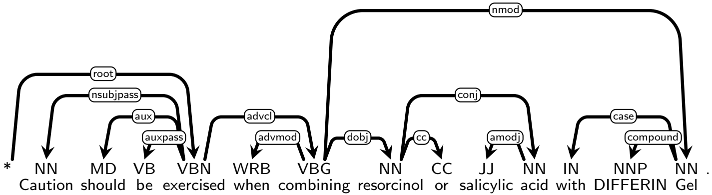
Resources

We will use Stanford CoreNLP dependency parser, which can be called via `nltk`, and integrates a tokenizer, a part-of-speech tagger, and a dependency parser.

- Download and uncompress Stanford CoreNLP.
- Provided class `deptree.py` will handle calling the parser and access the resulting structure

Functions - Analyze text


```
text = 'Caution should be exercised when combining resorcinol or
        salicylic acid with DIFFERIN Gel'
analysis = deptree(text)
analysis.get_word(tk)
analysis.get_lemma(tk)
analysis.get_tag(tk)
analysis.get_rel(tk)
analysis.get_parent(tk)
analysis.get_children(tk)
analysis.get_ancestors(tk)
analysis.get_up_path(tk1,tk2)
...
```



Functions - Check for Drug-Drug Interactions

```
def check_interaction(analysis,entities,e1,e2) :
'''
Task:
  Decide whether a sentence is expressing a DDI between two drugs.

  Input:
    analysis: a DependencyGraph object with all sentence information
    entites: A list of all entities in the sentence (id and offsets)
    e1,e2: ids of the two entities to be checked.

  Output: Returns the type of interaction ('effect', 'mechanism', 'advice
   ', 'int') between e1 and e2 expressed by the sentence, or 'None' if no
    interaction is described.
'''

 # get head token for each gold entity
 tkE1 = tree.get_fragment_head(entities[e1]['start'],entities[e1]['end'])
 tkE2 = tree.get_fragment_head(entities[e2]['start'],entities[e2]['end'])

 p = patterns.check_LCS_svo(tree,tkE1,tkE2)
 if p is not None: return p

 p = patterns.check_wib(tree,tkE1,tkE2,entities,e1,e2)
 if p is not None: return p

 ## add more patterns to improve performance
 # p = patterns.check_XXXX(tree,tkE1,tkE2,...)
 # if p is not None: return p

 return "null"
```

Predefined Functions - Evaluation

Performance evaluation functions are available in module evaluator, so you need to add to your main program

```
from evaluator import evaluate
```

The behaviour and parameters of `evaluate` are:

```
def evaluate(task, datadir, outfile) :
'''
Task:
  Compare results in outfile with ground truth annotations in datadir, and
    produce performance statistics.

Input:
  task: string with the name of the task to evaluate: "NER" or "DDI"
  datadir: directory containing original XML files with the ground truth
  outfile: filename containing the entities produced by your system

Output:
  Stats table about the predicted entities in the given output file
'''
```

Outline

Detecting interactions - First baseline

Function `check_interaction` will implement our *simple* rule-based interaction detector.

- It is a *baseline*, i.e. a lower bound for the performance of ML systems that we will build later.
- It must consist of an *if-then-else* cascade of *simple* rules. Do not implement statistical approaches.

Detecting interactions - Choosing the rules

- Examine (by hand or collecting simple statistics) the **train** dataset and try to infer general rules that are right in most cases, even if they seldom apply (high precision, low recall).
- Express your observations in terms of *if-then-else* rules. Note that the order in which the rules are checked will alter the results, so you should apply first those rules with higher precision.
- Use the **train** dataset to make your observations. Use provided program `explore.py` to implement a new rule and get an idea of its potential utility

Detecting interactions - Initial rules

The initial version if `baseline-DDI.py` has two rules, based on the following explorations:

- When the two target entites are one under the subject (`nsubj`) and the other under the direct object (`obj`) of a verb, then it is likely that there is an interaction. Some verbs will be indicative of certain relations.

  Pattern `check_LCS_svo` in `patterns.py` spots trees with this structure, and check for some relevant verbs

- Tree structure is not matched often. A more relaxed pattern could be checking whether certain words appear in between both entities.

  Pattern `check_wib` in `patterns.py` checks whether certain words appear in between the two target entities.

Detecting interactions - Improving the rules

Suggestions to improve performance

- Use program `explore.py` to get possible words to add to the lists in existing patterns `check_LCS_svo` and `check_wib`.
- Identify weak classes in your predictor (e.g. class `advise` has very low recall) and try to identify patterns that could help improving them.

  E.g. Using `explore.py` to find examples of class `advise`, we see that the main verb is very often modified with *should*. Try adding a pattern in `patterns.py` that checks whether the LCS of both entities is a verb with a *should* child, and return 'advise' if it is.
- Identify other weak spots (e.g. `mechanism` has low recall) and try to figure out some patterns that could help improving that class.

Outline

Exercise Goals

What you should do:

- Derive simple rules from data observation (using `explore.py` may be helpful)
- Create a *simple* baseline for DDI using an *if-then-else* sequence of patterns rules.

What you should **NOT** do:

- Apply statistical algorithms or weighted information combination that are not a cascade of *if-then-else* rules.
- Try too hard: The goal is to calibrate the difficulty of the task, seeing how far can we get with *little effort*.
  However: This task is not as easy as NER, and the code checking the applicability of the rules will be more complex. Maintain the structure of provided `check_interaction` function, just adding new patterns (which should be in `patterns.py`)

Exercise Goals

Orientative results:

- Provided initial version achiecves 30% macro average F1 on devel with two rules.
- Improve/extend the rule set to achieve 35%-40% macro average F1 on devel (see suggestions on previous slides)

Deliverables

- You'll be expected to produce a report at the end of DDI task.
- By now, just keep track of the information you'll need later:
  - Observations drawn from the data
  - Rules build from those observations
  - Rules tried and discarded/kept
  - Performance results on devel and test corpus using different rule combinations