Master in Artificial Intelligence

Advanced Human Language Technologies

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Outline

Session 1 - NERC baseline

Assignment

Write a python program that parses all XML files in the folder given as argument and recognizes and classifies drug names. The program must use simple heuristic rules to carry out the task.

```
$ python3 ./baseline-NER.py data/Devel/
DDI-DrugBank.d278.s0|0-9|Enoxaparin|drug
DDI-DrugBank.d278.s0|93-108|pharmacokinetics|group
DDI-DrugBank.d278.s0|113-124|eptifibatide|drug
DDI-MedLine.d88.s0|15-30|chlordiazepoxide|drug
DDI-MedLine.d88.s0|33-43|amphetamine|drug
DDI-MedLine.d88.s0|49-55|cocaine|drug
DDI-MedLine.d88.s1|82-95|benzodiazepine|drug
...
```

Outline

General Structure

Your main function will look like:

```
def nerc(inputdir, outputfile) :
   for file in inputdir :
      tree = parseXML(file)
      for sentence in tree :
         (id, text) = get_sentence_info(sentence)
         token_list = tokenize(text)
         entities = extract_entities(token_list)
         output_entities(id, entities, outputfile)

   evaluate(inputdir,outputfile)
```

Outline

Resources

You will need to use:

- An XML parser: we recommend `xml.dom.minidom` (`https://docs.python.org/3.7/library/xml.dom.minidom.html`)
- A tokenizer for English text: We recommend `nltk.tokenize` (check `https://www.nltk.org/install.html` if you don't have it installed)

Outline

Functions - Tokenize text

```
def tokenize(s) :
```

- Input: Receives a sentence text `s`, calls nltk.tokenize to split it in tokens, and adds to each token its start/end offset in the original sentence.
- Output: Returns a list of tuples `(word, offsetFrom, offsetTo)`.
- Example:

```
>>> tokenize("Ascorbic acid, aspirin, and the common
cold.")
[("Ascorbic",0,7), ("acid",9,12), (",",13,13),
("aspirin",15,21), (",",22,22), ("and",24,26),
("the",28,30), ("common",32,37), ("cold",39,42),
(".",43,43)]
```

Functions - Extract entities

`def extract_entities(s) :`

- Input: Receives a tokenized sentence s (list of triples `(word, offsetFrom, offsetTo)`.
- Output: Returns a list of entities. Each entity is a dictionary with the keys *name*, *offset*, and *type*.
- Example:

```
>>> extract_entities([("Ascorbic",0,7), ("acid",9,12),
(",",13,13), ("aspirin",15,21), (",",22,22),
("and",24,26), ("the",28,30), ("common",32,37),
("cold",39,42), (".",43,43)])
[{"name":"Ascorbic acid", "offset":"0-12",
"type":"drug"}, {"name":"aspirin", "offset":"15-21",
"type":"brand"}]
```

Functions - Output entities

```
def output_entities(id,ents,outf) :
```

- Input: Receives a sentence id, a list of extracted entities (i.e .a list of dictionaries with keys *name*, *offset*, and *type*), and an open output file object.
- Output: Prints on `outf` the entities in the right format: one line per entity, fields separated by '|', field order: *id*, *offset*, *name*, *type*.
- Example:

```
>>> output_entities("DDI-DrugBank.d553.s0",
[{"name":"Ascorbic acid", "offset":"0-12",
"type":"drug"}, {"name":"aspirin", "offset":"15-21",
"type":"brand"}], sys.stdout)
DDI-DrugBank.d553.s0|9-12|Ascorbic acid|drug
DDI-DrugBank.d553.s0|15-21|aspirin|brand
```

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 (created by `output_entities`).
- Output: Prints statistics about the predicted entities in the given output file.
- Code:

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

Note: `outputfile` must match the pattern: `task9.1_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

Extracting entities - Choosing rules

Function `extract_entities` will implement our rule-based extractor. 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)
- Example observations that may lead to some rules:
  - Tokens fully capitalized (e.g. KERASTICK, DILAUDID, LEVSIN) are usually drug names. Also, two out of three of them are of type *brand*.
  - Non-capitalized words that are drugs often have particular suffixes (e.g. -azole, -idine, -amine, -mycin, etc). Words with these endings are typically drug names and most frequently of type *drug*.

Extracting entities - Multi-token entities

- Many drug names in DDI corpus are multi-token drug names (e.g. *beta blockers*, *calcium channel antagonists*, *angiotensin converting enzyme inhibitors*, etc).
- So far, we check each token to decide whether it is an entity or not, so, we miss multi-token drug names.
- Improve your function `extract_entities` to glue toghether in a single entity consecutive tokens that may form a unique drug.

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 0.5 on **Devel** dataset using **only** information from the training dataset.

Goal 2:
Get an overall F_1 score of at least 0.6 on **Devel** dataset using information from external knowledge sources.

Deliverables

Prepare a report containing:

For Goal 1 (Rule-based, no external knowledge):

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

For Goal 2 (Rule-based, using external knowledge):

- Final version of `extract_entities` 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.