

Master in Artificial Intelligence

Advanced Human Language Technologies

NERC
Baseline

General
Structure

Resources

Detailed
Structure

Core task

Goals &
Deliverables



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Facultat d'Informàtica de Barcelona



Outline

- 1 NERC Baseline
- 2 General Structure
- 3 Resources
- 4 Detailed Structure
- 5 Core task
- 6 Goals & Deliverables

NERC
Baseline

General
Structure

Resources

Detailed
Structure

Core task

Goals &
Deliverables

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/ result.out
```

```
$ more result.out
```

```
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
```

```
...
```

NERC
Baseline

General
Structure

Resources

Detailed
Structure

Core task

Goals &
Deliverables

Outline

- 1 NERC Baseline
- 2 General Structure**
- 3 Resources
- 4 Detailed Structure
- 5 Core task
- 6 Goals & Deliverables

NERC
Baseline

General
Structure

Resources

Detailed
Structure

Core task

Goals &
Deliverables

General Structure - Main function

The program expects two arguments: Directory with XML files to process, and name for the output file

```
datadir = sys.argv[1]
outfile = sys.argv[2]
```

Then, it reads all files in the given directory, and for each sentence, extracts drug mentions, if any.

```
# 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
        # tokenize text
        tokens = tokenize(stext)
        # extract entities from tokenized sentence text
        entities = extract_entities(tokens)

        # print sentence entities in format requested for evaluation
        for e in entities :
            print(sid+"|"+e["offset"]+"|"+e["text"]+"|"+e["type"], file=outf)

# print performance score
evaluator.evaluate("NER", datadir, outfile)
```

NERC
Baseline

General
Structure

Resources

Detailed
Structure

Core task

Goals &
Deliverables

Outline

- 1 NERC Baseline
- 2 General Structure
- 3 Resources**
- 4 Detailed Structure
- 5 Core task
- 6 Goals & Deliverables

NERC
Baseline

General
Structure

Resources

Detailed
Structure

Core task

Goals &
Deliverables

Resources

The program uses:

- An **XML parser**: `xml.dom.minidom` (<https://docs.python.org/3.7/library/xml.dom.minidom.html>, included in python standard library)
- A **tokenizer** for English text:
`nltk.tokenize.word_tokenize` (check <https://www.nltk.org/install.html> if you don't have it installed)
- The **evaluator** module to compute performance scores (provided in the lab project zipfile).

NERC
Baseline

General
Structure

Resources

Detailed
Structure

Core task

Goals &
Deliverables

Outline

- 1 NERC Baseline
- 2 General Structure
- 3 Resources
- 4 Detailed Structure**
- 5 Core task
- 6 Goals & Deliverables

NERC
Baseline

General
Structure

Resources

Detailed
Structure

Core task

Goals &
Deliverables

Functions - Tokenize text

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

Task:

Given a sentence, calls `nltk.tokenize` to split it in tokens, and adds to each token its start/end offset in the original sentence.

Input:

s: string containing the text for one 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)]  
,,,
```

NERC
Baseline

General
Structure

Resources

Detailed
Structure

Core task

Goals &
Deliverables

Functions - Extract entities

```
def extract_entities(s) :  
    , , ,
```

Task:

Given a sentence, tokenize it and identify which tokens (or groups of consecutive tokens) are drugs.

Input:

s: A sentence text

Output:

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"  
    "}]  
    , , ,
```

NERC
Baseline

General
Structure

Resources

Detailed
Structure

Core task

Goals &
Deliverables

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  
    '''
```

NERC
Baseline

General
Structure

Resources

Detailed
Structure

Core task

Goals &
Deliverables

Outline

- 1 NERC Baseline
- 2 General Structure
- 3 Resources
- 4 Detailed Structure
- 5 Core task**
- 6 Goals & Deliverables

NERC
Baseline

General
Structure

Resources

Detailed
Structure

Core task

Goals &
Deliverables

Extracting entities - First baseline

Function `extract_entities` will implement our *simple* rule-based extractor.

- 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.
- *Hint:* start classifying each token separately (even if that means failing to extract some drugs). Later we'll address the multi-word drug names.

NERC
Baseline

General
Structure

Resources

Detailed
Structure

Core task

Goals &
Deliverables

Extracting entities - 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 the **devel** dataset to test each rule combination and decide whether a new rule is worth keeping, or which is the best order.
- Never get insights from the **devel** or **test** datasets (but you can run the rules on the **train** dataset to get information from the errors made by the extractor)

NERC
Baseline

General
Structure

Resources

Detailed
Structure

Core task

Goals &
Deliverables

Extracting entities - Choosing the rules

- For example, looking at the data, we observe that:
 - 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*.
- With these two observations, we could write the following rules to classify a token:

```
if word.isupper() : return "brand"
elif word[-5:] in ['azole', 'idine', 'amine', 'mycin'] :
    return "drug"
else : return "NONE"
```

Extracting entities - Multi-token entities

Next step is dealing with multi-token drug names.

- 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 together in a single entity consecutive tokens that may form a unique drug. (it is a simple heuristic, but *remember*: We are building a *simple baseline* to evaluate how high can we get with a very small development cost.)

NERC
Baseline

General
Structure

Resources

Detailed
Structure

Core task

Goals &
Deliverables

Outline

- 1 NERC Baseline
- 2 General Structure
- 3 Resources
- 4 Detailed Structure
- 5 Core task
- 6 Goals & Deliverables

NERC
Baseline

General
Structure

Resources

Detailed
Structure

Core task

Goals &
Deliverables

Exercise Goals

What you should do:

- Derive simple rules from data observation (via direct visual inspection or using very simple frequency information).
- Create a *simple* baseline for NER using *if-then-else* cascaded rules.

What you should **NOT** do:

- Apply statistical algorithms or weighted information combination that are not a cascade of *if-then-else* rules.
- Alter the provided code structure.
- Try too hard: The goal is to calibrate the difficulty of the task, seeing how far can we get with *little effort*.

NERC
Baseline

General
Structure

Resources

Detailed
Structure

Core task

Goals &
Deliverables

Exercise Goals

Orientative results:

- Provided initial version achieves 36% macro average F1 on **devel** with 3 simple rules.
- Improve/extend the rule set to achieve at least 45% macro average F1 on **devel**.

Information that may result helpful:

- Longer lists of suffixes, differentiated by class
- capitalization
- presence of numbers
- presence of dashes
- ...

NERC
Baseline

General
Structure

Resources

Detailed
Structure

Core task

Goals &
Deliverables

Deliverables

NERC
Baseline

General
Structure

Resources

Detailed
Structure

Core task

Goals &
Deliverables

- You'll be expected to produce a report at the end of NER 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