Advanced Human Language Technologies

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Outline

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 a **ML classification** algorithm to solve the problem.

```
$ python3 ./ml-DDI.py data/Devel/
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.d211.s2|DDI-DrugBank.d211.s2.e0|DDI-DrugBank.d211.s2.e5|mechanism
...
```

Outline

Relation Extraction

- Relation Extraction is a NLP task, frequently required in Information Extraction applications.
- The goal of the task is to extract relations between entities (previously detected), expressed in the text. E.g.: `is_CEO_of(Person,Organization)`:
  - *Steve Jobs was the chairman, the chief executive officer (CEO), and a co-founder of Apple Inc., ...*
  - *During his career at Microsoft, Bill Gates held the positions of chairman, chief executive officer (CEO), president and chief software architect.*
  - *Mark Zuckerberg is known for co-founding Facebook, Inc. and serves as its chairman, chief executive officer, and controlling shareholder.*

Relation Extraction

Other examples:

- Medical domain:
  `caused_by(diagnose,drug)`
  `prescribed_for(drug,diagnose)`
  `drug_interaction(drug,drug)`
- Legal domain:
  `is_suing(Person/Org,Person/Org)`
  `is_representing(Person,Person/Org)`
  `is_sentenced_for(Person/Org,Crime)`
  `is_sentenced_to(Person/Org,Penalty)`
- Business/Economy:
  `is_CEO_of(Person,Organization)`
  `absorbed_by(Organization,Organization)`
- etc.

Relation Extraction

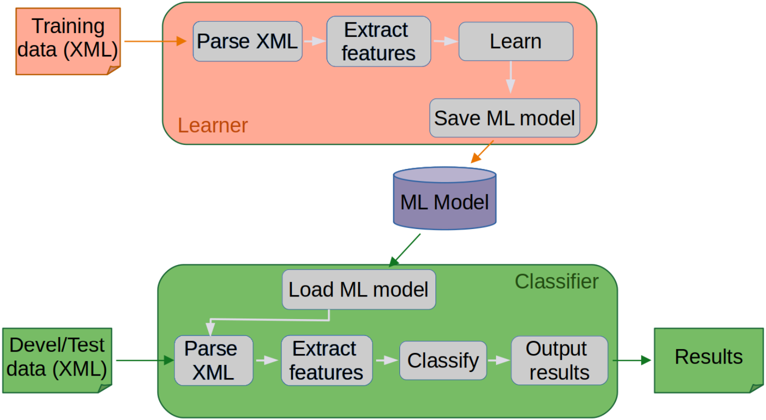
Relation Extraction can be approached as a classical ML classification task, where:

- The objects to be classified are a text fragment (sentence, paragraph...) plus a pair of target entities in it.
- Each object *(text,entity1,entity2)* is encoded as a feature vector.
- The output class is either `None`, or one relation type chosen among a *predefined list*.

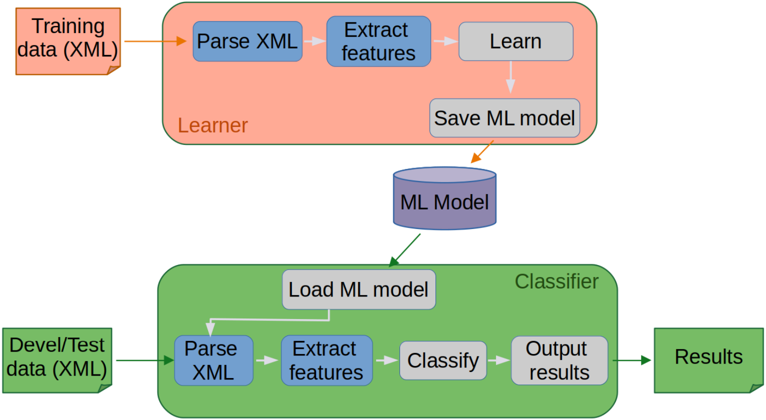
Informative enough features are crucial to get good results.

Outline

General Structure

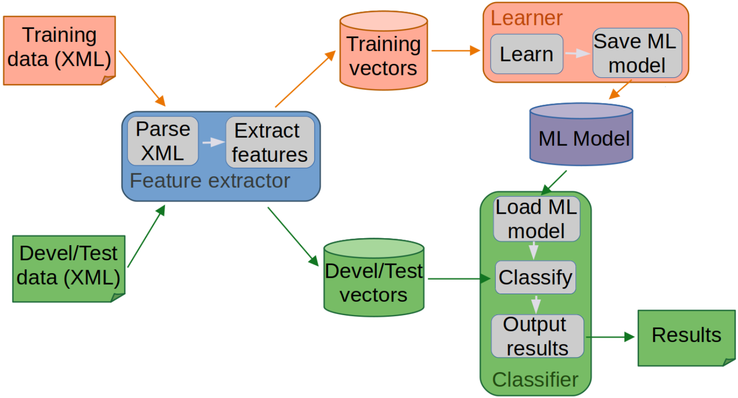


General Structure



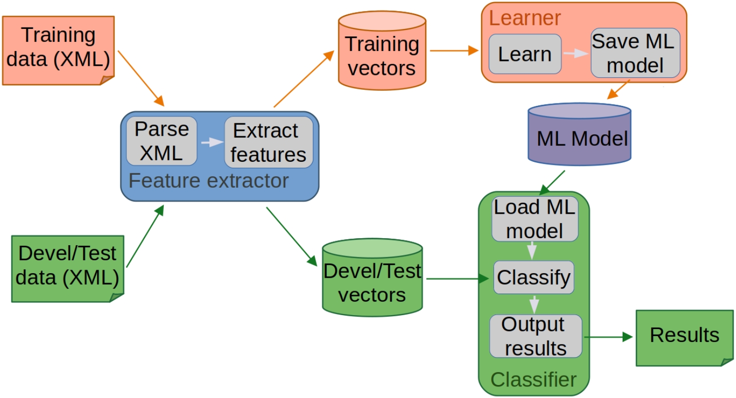
Extracting features is a costly operation, which we do not want to repeat for every possible experiment or algorithm parametrization.

General Structure



Feature extraction process is performed once, out of learning or predicting processes.

General Structure



Feature extraction process is performed once, out of learning or predicting processes.
Thus, we need to write not a single program, but three different components: feature extractor, learner, and classifier.

Outline

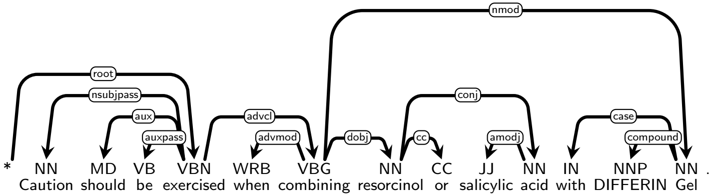
Resources

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)
...
```



Outline

Outline

Feature Extractor

The feature extractor:

- Independent program, separated from learner and classifier
- Receives as argument the directory with the XML files to encode.
- Prints the feature vectors to `stdout`

```
$ python3 ./feature-extractor.py data/devel > devel.feat
$ more devel.feat
DDI-DrugBank.d339.s0 DDI-DrugBank.d339.s0.e0 DDI-DrugBank.d339.s0.e1 null lib=elevated
wib=Elevated lpib=elevated_JJ la2=level wa2=levels lpa2=level_NNS la2=have wa2=have
lpa2=have_VBP la2=be wa2=been lpa2=be_VBN la2=report (...) lpa2=concomitantly_RB path1=NNP
path2=NNP\dep\nsubjpass\compound path=NNP\dep\nsubjpass\compound
DDI-DrugBank.d339.s0 DDI-DrugBank.d339.s0.e0 DDI-DrugBank.d339.s0.e2 null lib=elevated
wib=Elevated lpib=elevated_JJ wib=experience lpib=experience_NN la2=be wa2=is lpa2=be_VBZ
la2=administer wa2=administered lpa2=administer_VBN la2=concomitantly wa2=concomitantly
lpa2=concomitantly_RB path1=NNP path2=NNP\dep\advcl\advcl\nsubjpass
path=NNP\dep\advcl\advcl\nsubjpass
DDI-DrugBank.d339.s0 DDI-DrugBank.d339.s0.e1 DDI-DrugBank.d339.s0.e2 mechanism
lb1=carbamazepine wb1=Carbamazepine wb1=Elevated lpb1=elevated_JJ lib=level wib=levels
lpib=level_NNS lib=have wib=have lpib=have_VBP lib=be wib=been lpib=be_VBN lib=report
wib=reported lpib=report_VBN lib=postmarket wib=postmarketing lpib=postmarket_VBG
lib=experience wib=experience lpib=experience_NN la2=be wa2=is lpa2=concomitantly_RB
path1=compound/nsubjpass/VBN path2=VBN\advcl\advcl\nsubjpass
path=compound/nsubjpass/VBN\advcl\advcl\nsubjpass
...
```

Feature Extractor

```
# 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 ground truth entities
        entities = {}
        ents = s.getElementsByTagName("entity")
        for e in ents :
           id = e.attributes["id"].value
           entities[id] = e.attributes["charOffset"].value.split("-")
        # analyze sentence if there is at least a pair of entities
        if len(entities) <= 1: continue

        analysis = deptree(stext)
        # for each pair of entities, decide whether it is DDI and its type
        pairs = s.getElementsByTagName("pair")
        for p in pairs:
            # get ground truth
            ddi = p.attributes["ddi"].value
            dditype = p.attributes["type"].value if ddi=="true" else "null"
            # target entities
            id_e1 = p.attributes["e1"].value
            id_e2 = p.attributes["e2"].value
            # feature extraction
            feats = extract_features(analysis,entities,id_e1,id_e2)
            # resulting feature vector
            print(sid, id_e1, id_e2, dditype, "\t".join(feats), sep="\t")
```

Feature Extractor Functions - Extract features

```
def extract_features(tree, entities, e1, e2) :
  '''
  Task:
    Given an analyzed sentence and two target entities, compute a feature
     vector for this classification example.

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

  Output:
    A vector of binary features.
    Features are binary and vectors are in sparse representation (i.e. only
     active features are listed)

  Example:

  >>> extract_features(tree, {'DDI-DrugBank.d370.s1.e0':['43','52'],
                                'DDI-DrugBank.d370.s1.e1':['57','70'],
                                'DDI-DrugBank.d370.s1.e2':['77','88']},
                         'DDI-DrugBank.d370.s1.e0', 'DDI-DrugBank.d370.s1.e2')
     ['lb1=Caution', 'lb1=be', 'lb1=exercise', 'lb1=combine', 'lib=or', 'lib
     =salicylic', 'lib=acid', 'lib=with', 'LCSpos=VBG', 'LCSlema=combine',
     'path=dobj/combine\nmod\compound' 'entity_in_between']
  '''
```

Feature Extractor - Relevant Features

- Presence of certain *clue verbs* may be indicative of the interaction type.
- Clue verb position (before/inbetween/after) with respect to the target entities.
- Presence of other entities in between.
- Words, lemmas, PoS (or combinations of them) appearing before/inbetween/after the target pair.

Feature Extractor - Relevant Features

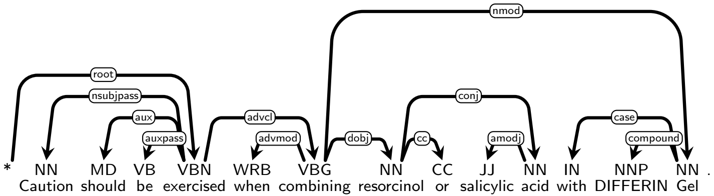
- Presence of certain *clue verbs* may be indicative of the interaction type.
- Clue verb position (before/inbetween/after) with respect to the target entities.
- Presence of other entities in between.
- Words, lemmas, PoS (or combinations of them) appearing before/inbetween/after the target pair.
- Features encoding information from the syntactic tree.

Feature Extractor - Relevant Features

- Presence of certain *clue verbs* may be indicative of the interaction type.
- Clue verb position (before/inbetween/after) with respect to the target entities.
- Presence of other entities in between.
- Words, lemmas, PoS (or combinations of them) appearing before/inbetween/after the target pair.
- Features encoding information from the syntactic tree.

Remember: All features are *binary features* (they are either present or not present in a given example) and examples are encoded as *sparse vectors* (lists feature names present in the example)

Feature Extractor - Path Features



Entities:

e0: `resorcinol` e1: `salicylic acid` e2: `DIFFERIN Gel`

Example path features:

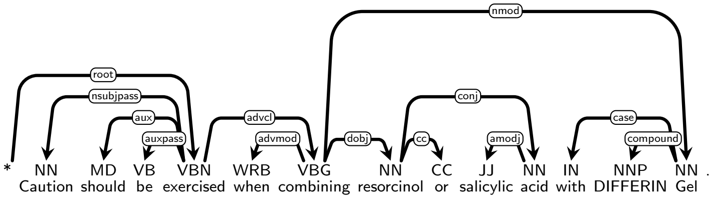
PAIR (e0,e1)

Tree fragment: e0 $\overset{conj}{\rightarrow}$ e1

(e1 is direct child of e0. The arc is labeled *conj*)

Feature name: `path=conj>`

Feature Extractor - Path Features



Entities:

e0: resorcinol   e1: salicylic acid   e2: DIFFERIN Gel

Example path features:

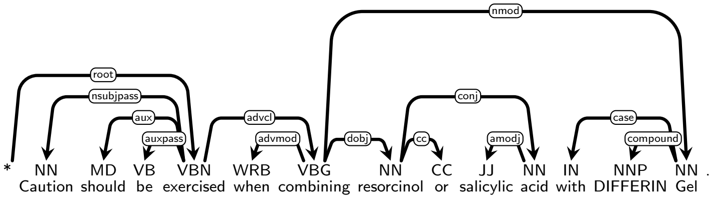
PAIR (e0,e2)

Tree fragment: e0 $\stackrel{dobj}{\leftarrow}$ combine $\stackrel{nmod}{\rightarrow}$ e2

(e0 is direct child of verb "combine" with label *dobj*, and e2 is direct child of the same verb, with label *nmod*)

Feature name: path=dobj<combine>nmod

Feature Extractor - Path Features



Entities:

e0: `resorcinol` e1: `salicylic acid` e2: `DIFFERIN Gel`

Example path features:

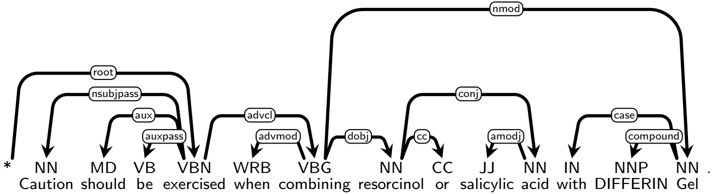
PAIR (e1,e2)

Tree fragment: e1 $\overset{conj}{\leftarrow}$ resorcinol $\overset{dobj}{\leftarrow}$ combine $\overset{nmod}{\rightarrow}$ e2

(e1 is *conj* child of "resorcinol", which is under verb "combine" with label *dobj*, and e2 is direct child of the same verb, with label *nmod*)

Feature name: path=conj<dobj<combine>nmod

Feature Extractor - Path Features



Entities:

e0: resorcinol   e1: salicylic acid   e2: DIFFERIN Gel

Example path features:

PAIR (e1,e2)

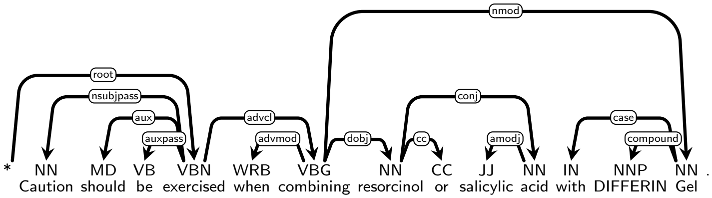
Tree fragment: e1 $\overset{conj}{\leftarrow}$ resorcinol $\overset{dobj}{\leftarrow}$ combine $\overset{nmod}{\rightarrow}$ e2

(e1 is *conj* child of "resorcinol", which is under verb "combine" with label *dobj*, and e2 is direct child of the same verb, with label *nmod*)

Also possible: path=conj<ENTITY/dobj<combine>nmod

Machine Learning DDI
Relation Extraction
General Structure
Resources
Detailed Structure
Feature Extractor
Core task
Goals & Deliverables

Feature Extractor - Path Features



Entities:

e0: `resorcinol`   e1: `salicylic acid`   e2: `DIFFERIN Gel`

Example path features:

PAIR (e1,e2)

Tree fragment: e1 $\xleftarrow{conj}$ resorcinol $\xleftarrow{dobj}$ combine $\xrightarrow{nmod}$ e2

(e1 is *conj* child of “resorcinol”, which is under verb “combine” with label *dobj*, and e2 is direct child of the same verb, with label *nmod*)

Also possible: `path=dobj*<combine>nmod`

Feature Extractor - Path Features

Path features may be build in different ways, encoding different information about the tree

- Node words
- Node lemmas
- Node PoS
- Edge labels
- Edge direction
- Direct/indirect dependencies
- ... or any combination of these ...

Feature Extractor - Path Features

Path features may be build in different ways, encoding different information about the tree

- Node words
- Node lemmas
- Node PoS
- Edge labels
- Edge direction
- Direct/indirect dependencies
- ... or any combination of these ...

Remember: All features are *binary features* (they are either present or not present in a given example) and examples are encoded as *sparse vectors* (lists of present feature names)

Outline

Learner - Option 1: Maximum Entropy

- Use megam to train a model as seen in class
- megam does not expect the extra information in the features file, thus the first 3 fields (*sent_id*, *ent_id1*, *ent_id*) must be removed:

```
python3 extract-features.py data/train | cut -f4- > train.feats
./megam-64.opt -nc -nobias multiclass train.feats >model.MEM
```

Learner - Option 2: Your choice

- Select a ML algorithm of your choice (DT, SVM, RF, ...) and a python library implementing it.
- Adapt the feature file format to the needs of the selected algorithm
- Train a classification model for the task of **classifying** entity pairs.

Note that the target task is a mere classification, not a sequence prediction. So, for a given sentence and pair of entities in it, the output is just **one** label, not a sequence. Thus, sequence labeling algorithms such as CRFs are overdimensioned (and probably not straightforward to apply).

Outline

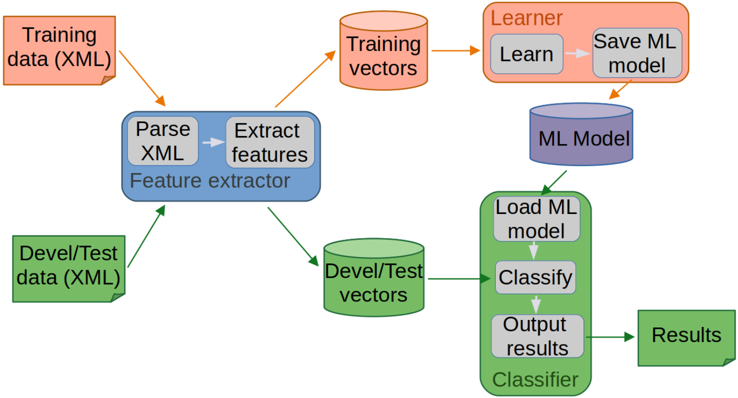
Classifier

```
# read each vector in input file
for line in sys.stdin:
    # split line into elements
    fields = line.strip('\n').split("\t")
    # first 4 elements are sid,e1,e2, and ground
     truth (ignored since we are classifying)
    (sid,e1,e2,gt) = fields[0:4]

    # Rest of elements are features, passed to the
     classifier of choice to get a prediction
    prediction = mymodel.classify(fields[4:])

    # if the classifier predicted a DDI, output it
     in the right format
    if prediction != "null" :
        print(sid,e1,e2,prediction,sep="|")
```

Outline

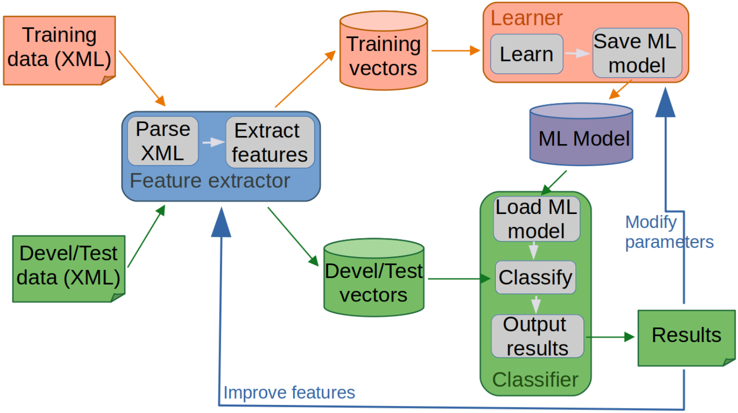
Build a good ML-based DDI detector

Strategy to follow:



Build a good ML-based DDI detector

Strategy to follow:



Detecting interactions: Possible features

Some feature possibilities to explore:

- Features detailing the position of words in the sentece (e.g. before E1, between E1 and E2, after E2)
- More general (or more specific) path features (with/without all elements in the path, lemma/tag of each, etc)
- Features encoding specific tree patterns (and maybe their combination with certain verbs). E.g. Features stating whether patterns used in Task 3 apply.
- Information about the LCS (PoS, lemma, ...)
- Features considering lists of relevant verbs.
- Type of entities in the pair
- Presence of a third entity (in the sentence, in between the target pair, in the path connecting the pair in the tree, ...)
- ...

Outline

Exercise Goals

What you should do:

- Work on your feature extractor. It is the component of the process where you have most control.
- Pay special attention to features encoding syntactic information.
- Experiment with different parameterizations of the chosen learner. You may try different learning algorithms if you feel up to. Note that the same feature vectors can be fed to different learners.
- Keep track of tried features and parameter combinations.

What you should **NOT** do:

- Use neural network learners. We'll do that later on the course.
- Alter the suggested code structure.
- Produce an overfitted model: If performance on the test dataset is much lower than on devel dataset, you probably are overfitting your model.

Exercise Goals

Orientative results

- Provided feature extractor uses 7 feature templates and gets a macroaverage F1 about 50%. Used information includes :
  - word forms, lemmas, and PoS tags (and combinations) appearing in between the target pair.
  - information on the path connecting both target entities: whole path, path from e1 to LCS, path from e2 to LCS.
  - information on the LCS lemma if one entity is under its subject and the other under its object.
- Extending feature repertorie with different pieces of information from the tree, and experimenting with learner parameters should raise macroaverage F1 on devel up to ~60%.

Deliverables

Write a report describing the work carried out in this exercise.
The report must be a **single self-contained PDF document**, under ~10 pages, containing:

- *Introduction:* What is this report about. What is the goal of the presented work.
- *Rule-based baseline*
  - *Ruleset construction:* What did you observe in the data exploration. Which rules did you wrote according to those observations.
  - *Code:* Include your `check_interaction` function (and any other function it may call), properly formatted and commented. **Do not include any other code.**
  - *Experiments and results:* Results obtained on the **devel** and **test** datasets, for different rule combinations you deem relevant.

Deliverables (continued)

- *Machine learning DDI*
  - *Selected algorithm:* Which classifier/s did you select or try. Reasons of the choice. Comparison if you tried more than one.
  - *Feature extraction:* Tried/discarded/used features. Impact of different feature combinations
  - *Code:* Include your `extract_features` function (and any other function it may call), properly formatted and commented. **Do not include any other code.**
  - *Experiments and results:* Results obtained on the **devel** and **test** datasets, for different algorithms, feature combinations, parameterizations you deem relevant.
- *Conclusions:* Final remarks and insights gained in this task.

Keep result tables in your report in the format produced by the evaluator module. Do not reorganize/summarize/reformat the tables or their content.