Mining Unstructured Data

Outline

Assignment

Improve a Relation Extraction system by trying different features from the original XML data. The program must use a **ML classification** algorithm to solve the problem.

```
$ python3 predict-sklearn.py model.joblib vectorizer.joblib < devel.cod
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

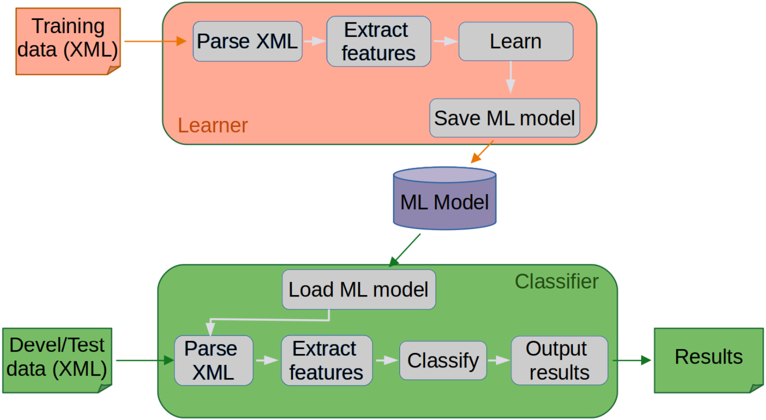
Relation Extraction can be approached as a classical ML classification task, where:

- 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*.

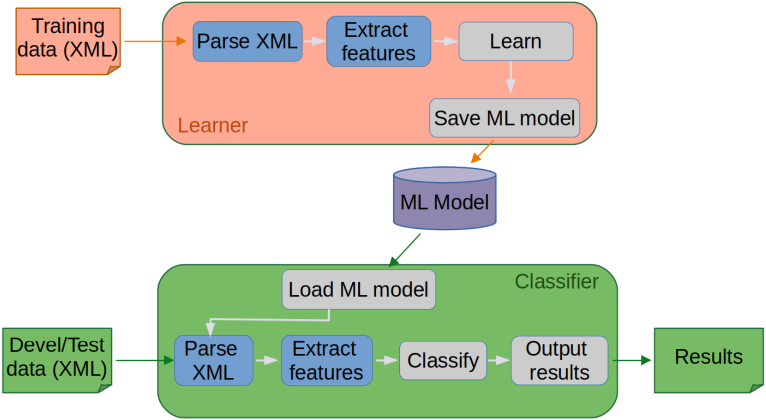
Informative enough features are crucial to get good results.

Outline

General Structure

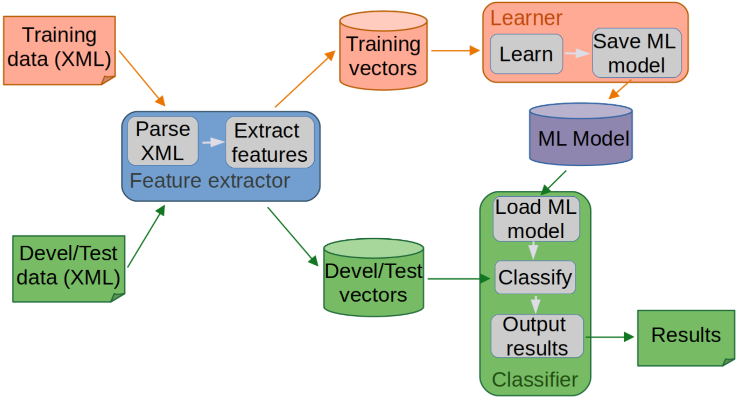


General Structure



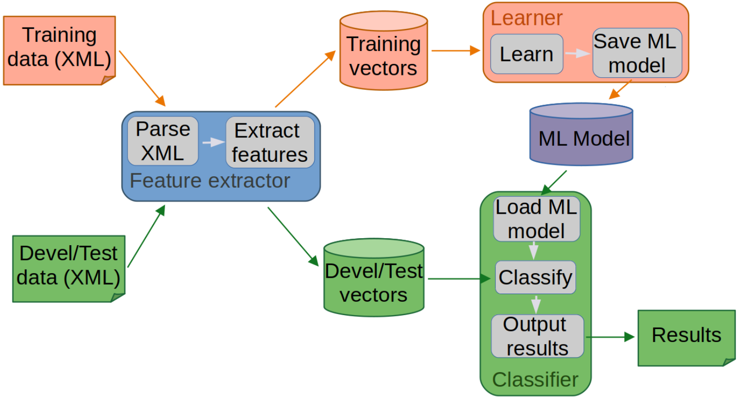
Extracting features is a costly operation, which we do not want to repeat for every possible experiment or algorithm parametrization.

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

Outline

Feature Extractor

The feature extractor:

- Must be an independent program, separated from learner and classifier
- Must get as argument the directory with the XML files to encode.
- Must print 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 : analysis = analyze(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 - Analyze text

```
def analyze(s) :
'''
Task:
  Given one sentence, sends it to CoreNLP to obtain the tokens, tags, and
   dependency tree. It also adds the start/end offsets to each token.

Input:
  s: string containing the text for one sentence

Output:
  Returns the nltk DependencyGraph (https://www.nltk.org/_modules/nltk/
  parse/dependencygraph.html) object produced by CoreNLP, enriched with
  token offsets.
```

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']
  '''
```

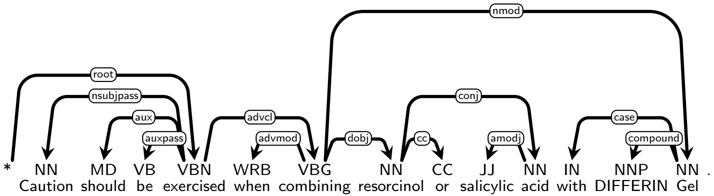
Feature Extractor - Relevant Features

- 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 - Path Features



Entities:

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

Example path features:

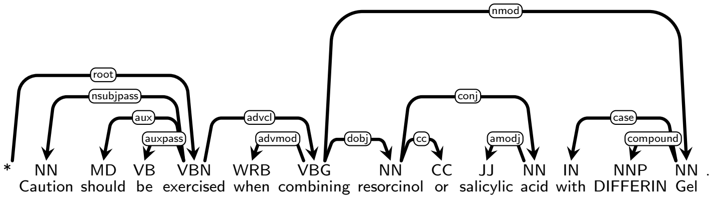
PAIR (e0,e1)

Tree fragment: e0 $\stackrel{conj}{\rightarrow}$ 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`

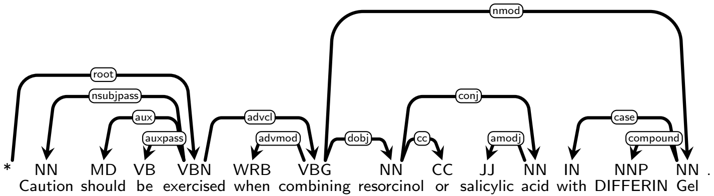
Example path features:

PAIR (e0,e2)

Tree fragment: e0 $\xleftarrow{dobj}$ combine $\xrightarrow{nmod}$ 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:

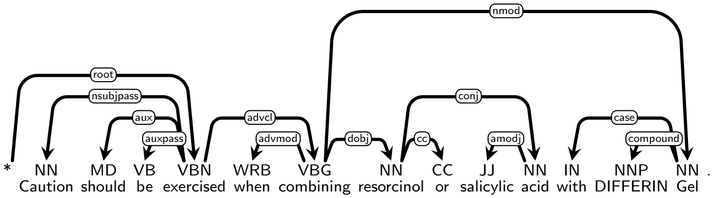
PAIR (e1,e2)

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:

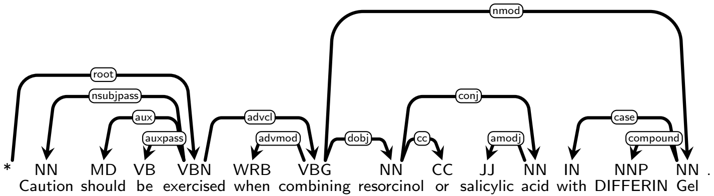
PAIR (e1,e2)

Tree fragment: e1 $\xleftarrow{conj}$ 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=conj<ENTITY/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 $\xleftarrow{conj}$ 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 ...

Outline

Learner - Option 1: Naive Bayes

- Install and import `skcit-learn`
  `$ pip install sckit-learn`
- Use provided train-sklearn.py to learn a model.
  `$ python3 train-sklearn.py model.joblib vectorizer.joblib < train.clf.feat`

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 algortihm
- 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

```
# load leaned model and DictVectorizer
    model = load(sys.argv[1])
    v   = load(sys.argv[2])

    for line in sys.stdin:

            fields = line.strip('\n').split("\t
")
            (sid,e1,e2) = fields[0:3]
            vectors = v.transform(
prepare_instances([fields[4:]]))
            prediction = model.predict(vectors)

            if prediction != "null" :
                    print(sid,e1,e2,prediction
[0],sep="|")
```

Classifier - Option 1: Naive Bayes

- Install and import `skcit-learn`
  `$ pip install sckit-learn`
- Use provided train-sklearn.py to learn a model.
  `$ python3 train-sklearn.py model.joblib vectorizer.joblib < train.cod.cl`

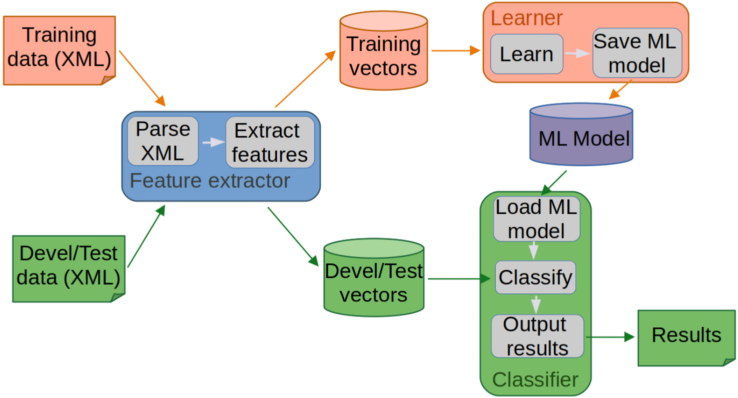
Classifier - Option 2: Your choice

- Write the necessary code to call your choice classifier and get a label for each vector in the dataset.

Outline

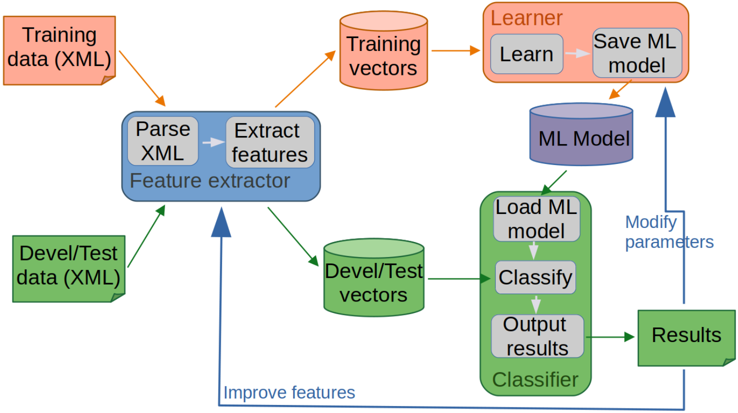
Build a good ML-based DDI detector

Strategy to follow:



Build a good ML-based DDI detector

Strategy to follow:



Outline

Evaluating Results

Use module evaluator provided in the lab project zip file to obtain performance statistics.

```
# extract features for train and devel datasets
python3 feature-extractor.py data/train/ | tee train.cod |
cut -f4- > train.cod.cl
python3 feature-extractor.py data/devel/ > devel.cod
# use train dataset to learn a model
python3 train-sklearn.py model.joblib vectorizer.joblib <
train.cod.cl
# annotate devel dataset using learned model
python3 predict-sklearn.py model.joblib vectorizer.joblib <
devel.cod > devel.out
# evaluate performance of the model
python3 evaluator.py DDI data/devel/ devel.out > devel.stats
```

Evaluating Results

- Repeat training – evaluation cycle on devel dataset to find out which is the best parameterization for the used algorithm.
- Repeat feature extraction – training – evaluation cycle on devel dataset to find out which features are useful.

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

- A set of 8 feature templates is enough to get a macroaverage F1 about 32%. Useful features information includes :
  - word forms, lemmas, and PoS tags (and combinations) appearing before, in between, and after the target pair.
  - information on the path connecting both target entities: whole path, path from e1 to LCS, path from e2 to LCS, PoS of the LCS, ...

Results much lower than these orientative scores is an indication that you are doing something wrong or not elaborated enough.

Other information worth trying

- Lists of relevant verbs for each class
- 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, ...)
- etc.

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.

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.