Mining Unstructured Data
Outline

1. Machine Learning DDI

2. Relation Extraction

3. General Structure

4. Detailed Structure
   - Feature Extractor
   - Learner
   - Classifier

5. Core task

6. Evaluating Results

7. Goals & Deliverables
Assignment

Write a python program that 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

1. Machine Learning DDI
2. Relation Extraction
3. General Structure
4. Detailed Structure
   - Feature Extractor
   - Learner
   - Classifier
5. Core task
6. Evaluating Results
7. Goals & Deliverables
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 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 \texttt{None}\, or one relation type chosen among a predefined list.

Informative enough features are crucial to get good results.
Outline

1. Machine Learning DDI
2. Relation Extraction
3. General Structure
   - Feature Extractor
   - Learner
   - Classifier
4. Detailed Structure
5. Core task
6. Evaluating Results
7. Goals & Deliverables
Extracting features is a costly operation, which we do not want to repeat for every possible experiment or algorithm parametrization.
Extracting features is a costly operation, which we do not want to repeat for every possible experiment or algorithm parametrization.
Feature extraction process is performed once, out of learning or predicting processes.
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

1. Machine Learning DDI
2. Relation Extraction
3. General Structure
4. Detailed Structure
   - Feature Extractor
   - Learner
   - Classifier
5. Core task
6. Evaluating Results
7. Goals & Deliverables
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 lpb1=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=N

DDI-DrugBank.d339.s0 DDI-DrugBank.d339.s0.e0 DDI-DrugBank.d339.s0.e2 null lib=elevated wib=Elevated lpb1=elevated JJ wib=experience lpb=experience.NN la2=be wa2=is lpa2=be_VBZ lpa2=administer la2=administered lpa2=administer.VBN la2=concomitantly wa2=concomitantly lpa2=concomitantly_RB path1=NNP path2=NNP lpa2=concomitantly_RB path1=NNP path2=NNP lpa2=concomitantly_RB path1=NNP path2=NNP ...

...
Feature Extractor

```python
# 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**

```python
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.
```
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 \(\xrightarrow{conj} e1\)
(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 \textit{conj} \textit{resorcinol} \textit{dobj} \textit{combine} \textit{nmod} e2 (e1 is \textit{conj} child of “resorcinol”, which is under verb “combine” with label \textit{dobj}, and e2 is direct child of the same verb, with label \textit{nmod})

Feature name: \textit{path}={\textit{conj}<\textit{dobj}<\textit{combine}>\textit{nmod}}
Feature Extractor - Path Features

Entities:
- e0: resorcinol
- e1: salicylic acid
- e2: DIFFERIN Gel

Example path features:
PAIR (e1,e2)

Tree fragment: e1 \( \text{conj} \) resorcinol \( \text{dobj} \) combine \( \text{nmod} \) e2
(e1 is \text{conj} child of “resorcinol”, which is under verb “combine” with label \text{dobj}, and e2 is direct child of the same verb, with label \text{nmod})
Also possible: path=\text{conj}<\text{ENTITY}/\text{dobj}<\text{combine}>\text{nmod}
Feature Extractor - Path Features

**Entities:**
- e0: resorcinol
- e1: salicylic acid
- e2: DIFFERIN Gel

**Example path features:**
PAIR (e1, e2)

**Tree fragment:**
- e1 \(\xleftarrow{\text{conj}}\) resorcinol \(\xleftarrow{\text{dobj}}\) 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

1. Machine Learning DDI
2. Relation Extraction
3. General Structure
4. Detailed Structure
   - Feature Extractor
   - Learner
   - Classifier
5. Core task
6. Evaluating Results
7. Goals & Deliverables
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, so:
  - Remove the first 3 fields (`sent_id, ent_id1, ent_id`):
    ```
    cat feats.dat | cut -f4- > megam.dat
    ```
  - Alternatively, you can modify the `print` command producing the vectors to directly produce two versions of the feature file, one with the extra information, and one without.
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

1. Machine Learning DDI
2. Relation Extraction
3. General Structure
4. Detailed Structure
   - Feature Extractor
   - Learner
   - Classifier
5. Core task
6. Evaluating Results
7. Goals & Deliverables
# read each vector in input file
for line in sys.stdin:
    # split line into elements
    fields = line.strip('
').split('	')
    # 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="|")
Classifier - Option 1: Maximum Entropy

- Follow examples (and reuse code) for MaxEnt classifiers seen in class to get a label for each vector in the dataset.
Write the necessary code to call your choice classifier and get a label for each vector in the dataset.
Outline

1. Machine Learning DDI
2. Relation Extraction
3. General Structure
4. Detailed Structure
   - Feature Extractor
   - Learner
   - Classifier
5. Core task
6. Evaluating Results
7. Goals & Deliverables
Build a good ML-based DDI detector

Strategy to follow:

- Training data (XML)
- Parse XML
- Extract features
- Feature extractor
- Training vectors
- Learner
- ML Model
- Load ML model
- Classify
- Output results
- Classifier
- Results

Build a good ML-based DDI detector

Strategy to follow:

- Training data (XML)
- Parse XML
- Extract features
- Feature extractor
- Training vectors
- Learner
- ML Model
- Load ML model
- Classify
- Output results
- Classifier
- Results
Build a good ML-based DDI detector

Strategy to follow:

- Training data (XML)
  - Parse XML
  - Extract features
  - Feature extractor
- Devel/Test data (XML)
- Training vectors
- Learner
  - Learn
  - Save ML model
- ML Model
- Classifier
  - Load ML model
  - Classify
  - Output results
  - Results
- Improve features
- Modify parameters

Goals & Deliverables
Build a good ML-based DDI detector
Outline

1. Machine Learning DDI
2. Relation Extraction
3. General Structure
4. Detailed Structure
   - Feature Extractor
   - Learner
   - Classifier
5. Core task
6. Evaluating Results
7. Goals & Deliverables
Evaluating Results

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

```bash
# extract features for train and devel datasets
python3 feature-extractor.py data/train/ > train.feat
python3 feature-extractor.py data/devel/ > devel.feat
# use train dataset to learn a model
python3 learner.py mymodel train.feat
# annotate devel dataset using learned model
python3 classifier.py mymodel devel.feat > devel.out
# evaluate performance of the model
python3 evaluator.pyc DDI data/devel/ devel.out
```

- 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

1 Machine Learning DDI
2 Relation Extraction
3 General Structure
4 Detailed Structure
   ■ Feature Extractor
   ■ Learner
   ■ Classifier
5 Core task
6 Evaluating Results
7 Goals & Deliverables
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 15 feature templates is enough to get a macroaverage F1 about 55%. Used 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.
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.