Master in Artificial Intelligence

Advanced Human Language Technologies

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Outline

Task 1.1 - NERC using machine learning

Assignment

The main program parses all XML files in the folder given as argument and recognizes and classifies drug names, calling a sequence-tagging machine learning algorithm.

```
$ python3 ./ml-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
...
```

Outline

Sequence tagging: the B-I-O approach

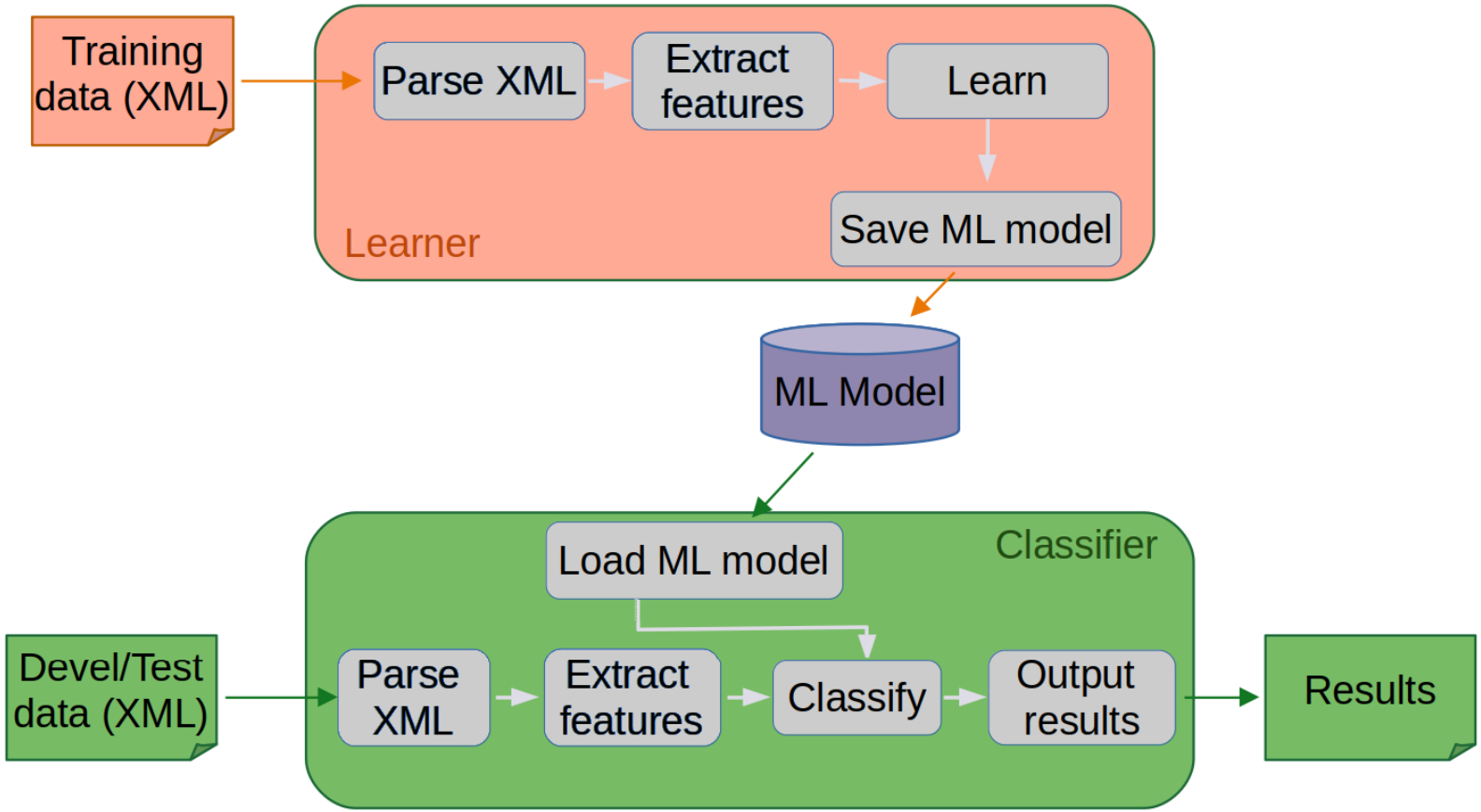
- We want to detect *subsequences* in a sentence (e.g. drug names).
- To approach this as a ML classification problem, we will classify each token.
- The classes predicted by the classifier must allow the later reconstruction of the target subsequences.
- B-I-O schema: mark each token as **B**egin of a subsequence, **I**nside a subsequence, or **O**utside any subsequence.
- If we not only want to recognize the subsequences, but also classify them, we use more informative B-I-O classes:

Ascorbic	**acid**	**,**	**aspirin**	**,**	**and**	**the**	**common**	**cold**	**.**
B-drug	I-drug	O	B-brand	O	O	O	O	O	O

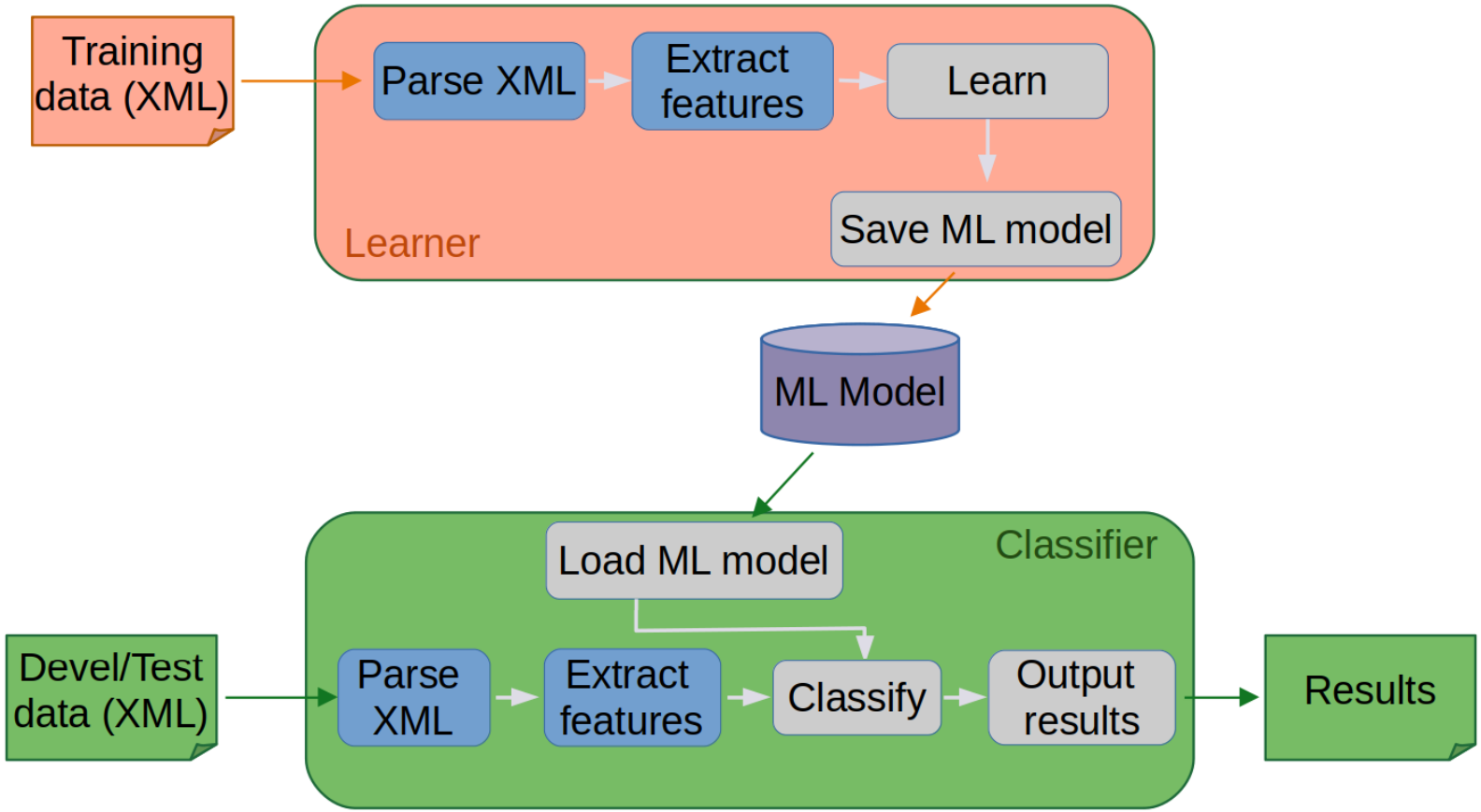
- Different variations of this schema exist: BIO, BIOS, BIOES (aka BILOU)

Outline

General Structure

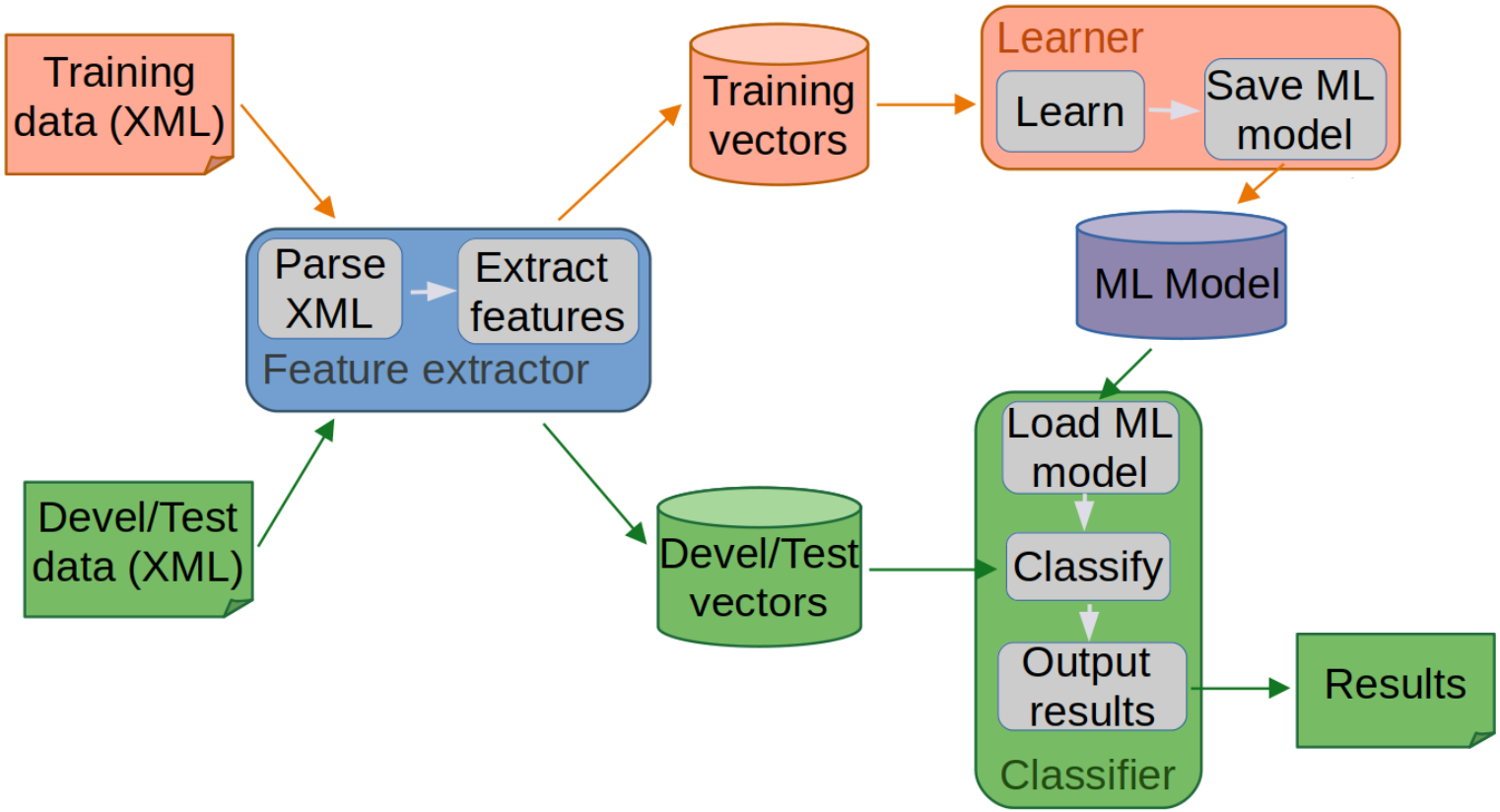


General Structure



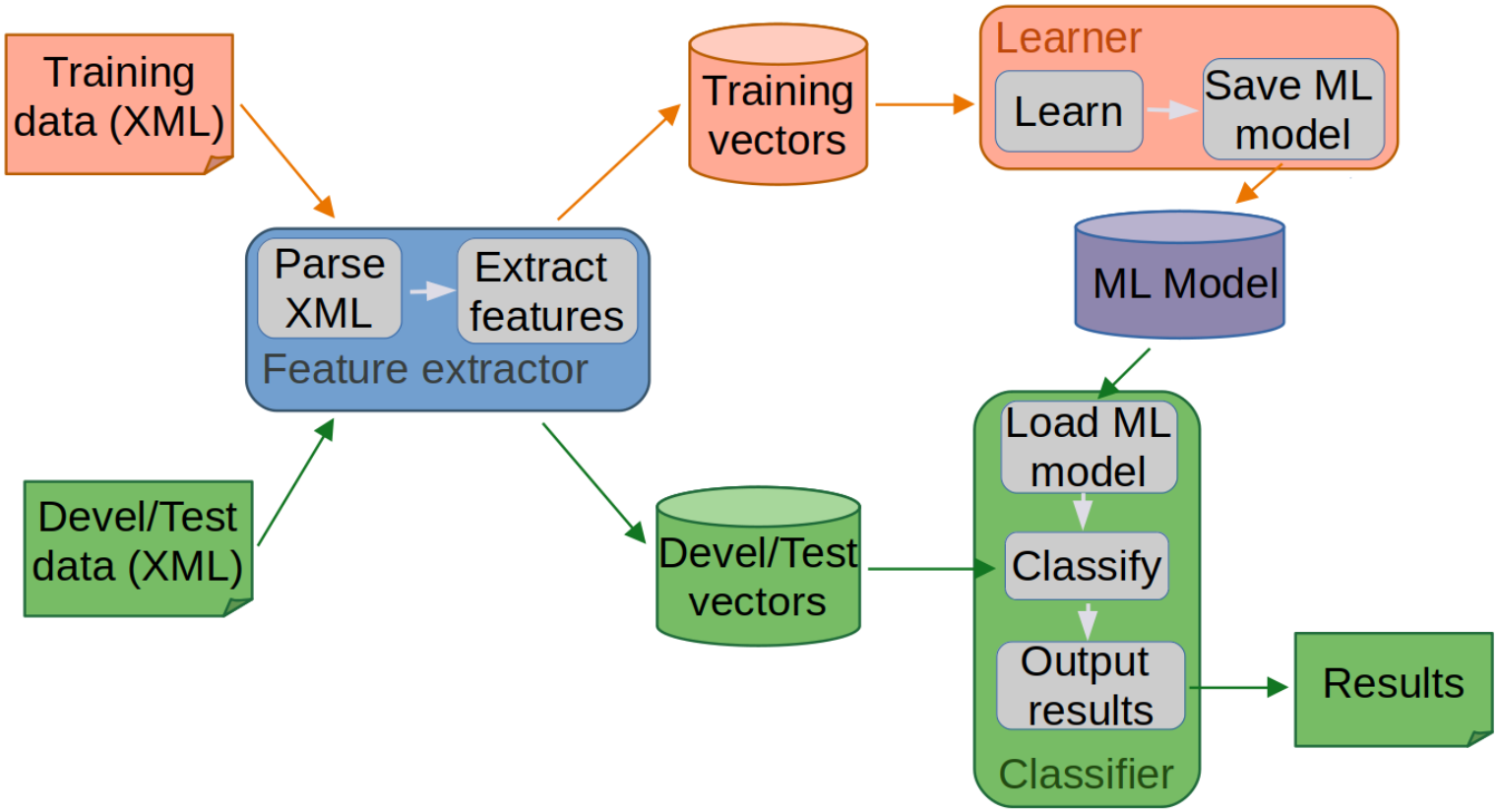
Extracting features is a costly operation, which we do not want to repeat for every possible experiment or algorithm parametrization.

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:

- 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 ./extractor_features.py devel.xml devel.feat
$ more devel.feat
DDI-DrugBank.d658.s0 When 0 3 O form=When formlower=when suf3=hen
suf4=When isTitle BoS formNext=administered
formlowerNext=administered suf3Next=red suf4Next=ered
DDI-DrugBank.d658.s0 administered 5 16 O form=administered
formlower=administered suf3=red suf4=ered formPrev=When
formlowerPrev=when suf3Prev=hen suf4Prev=When isTitlePrev
formNext=concurrently formlowerNext=concurrently suf3Next=tly
suf4Next=ntly
...
```

Feature extraction for NLP

- In most ML applications, the feature space is finite and known (e.g. credit scoring, medical diagnose prediction, churn prevention, fraud detection, etc).
- Also, most of the used features are numerical or categorial (income, age, sex, colestherol level, number of receipts returned, etc.)
- Thus, in these ML applications, feature vectors are usually *exhaustive* lists of pairs feature-value.

Feature extraction for NLP

- In most ML applications, the feature space is finite and known (e.g. credit scoring, medical diagnose prediction, churn prevention, fraud detection, etc).
- Also, most of the used features are numerical or categorial (income, age, sex, colestherol level, number of receipts returned, etc.)
- Thus, in these ML applications, feature vectors are usually *exhaustive* lists of pairs feature-value.

BUT...

Feature extraction for NLP

- In most ML applications, the feature space is finite and known (e.g. credit scoring, medical diagnose prediction, churn prevention, fraud detection, etc).
- Also, most of the used features are numerical or categorial (income, age, sex, colestherol level, number of receipts returned, etc.)
- Thus, in these ML applications, feature vectors are usually *exhaustive* lists of pairs feature-value.

BUT...

- In most NLP applications, features are related to appearing words, suffixes, prefixes, lemmas, etc.
- Thus, the feature space is huge.

Feature extraction for NLP

- Example: We want to encode in a feature the word form for a token.
- We could use a feature (i.e. a single column) `word_form` with thousands of possible values.
- But no ML algorithm is able to deal with categorical features with that amount of possible values, and not enough data will ever be available to learn the behaviour of that feature.
- Thus, features in NLP are typically *binary-valued* (e.g. `word-form-is-THE`, `word-form-is-SOME`, `word-form-is-DOG`, etc.).
  That is, a column for each possible value, with boolean value.
- This creates feature spaces with tens or hundreds of thousands of features, but each feature has only two possible values. This is manageable by many ML algoritmhs.

Feature extraction for NLP

- Thus, in NLP applications, feature vectors are usually *intensive* lists of strings (i.e. listing the names for features with value `true`, and ommiting those with value `false`), and are stored as *sparse vectors*.
- But, feature spaces of hundreds of thousands of dimensions can not be stored in a table or a matrix.
- However, feature vectors are usually very sparse: Each example has only a reduced number of non-zero columns (e.g. even there are thousands of columns like `word-form-is-XXX`, only one of them will be `true` for a given example).

Sparse vector representation: CSR matrix

- CSR matrix representation consists of storing the coordinates and the value for non-zero elements in the matrix.
- The matrix elements are indexed by their (row,column) coordinates.

$$\begin{pmatrix} 0 & 0 & 3 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 6 \end{pmatrix} \Longrightarrow \begin{pmatrix} \mathit{row} & \mathit{col} & \mathit{val} \\ 0 & 2 & 3 \\ 1 & 0 & 2 \\ 3 & 2 & 1 \\ 3 & 4 & 6 \end{pmatrix}$$

- Each row (of the full matrix) corresponds to a training example, and each column to a feature.
- The CSR matrix is a compressed representation of the same information.

- In the case of NERC, all features are binary (either 0 or 1) so the third column in CSR matrix may be ommited, saving more space.
- A feature index is needed to keep the correspondence between feature names and their column number.

Executing the feature extractor


```
$ python3 run.py extract
```

Will process `train.xml` and `devel.xml` and produce `train.feat` and `devel.feat` in directory `predicted`.

If the function `extract_sentence_features` is modified (to add or remove features), the datasets must be processed again.

Time to have a look to the `extract_sentence_features` and to the `train.feat` file.

Outline

Learner

The B-I-O approach consists of a classifier that assigns a label to each token, so any ML algorithm may be used

- Local classifiers: Each token is classified independently.
  - Maximum Entropy Classifiers (MEM)
  - Support Vector Machines (SVM)
  - ...

  May lead to inconsistencies (e.g `III` sequences without a `B` at the beggining)
- Global Classifiers: The whole sequence is taken into account for a global decision. Features are factorized to get an affordable cost.
  - Maximum Entropy Markov Models (MEMM)
  - Conditional Random Fields (CRF)
  - ...

Learner: MEM (a.k.a. Logistic Regression)

- Install and import `scikit-learn`
  `$ pip install scikit-learn`
- Use provided `train.py` to learn a MEM model.
  `$ python3 run.py train MEM`

  You may modifiy learner parameters (loss function, thresholds, learning rates, etc).
- Check performance on development data.
  `$ python3 run.py predict MEM`
- `scikit-learn` implementation of `LogisticRegression` (MEM), admits sparse matrix representations such as CSR (Compressed Sparse Row) matrix.

Learner: SVM

- Install and import `scikit-learn`
  `$ pip install scikit-learn`
- Use provided `train.py` to learn a SVM model.
  `$ python3 run.py train SVM`

  You may modifiy learner parameters (loss function, thresholds, learning rates, etc).
- Check performance on development data.
  `$ python3 run.py predict SVM`
- `scikit-learn` implementation of `SVC` (SVM), admits sparse matrix representations such as CSR (Compressed Sparse Row) matrix.

Learner: CRF

- Install and import `pycrfsuite`
  `$ pip install python-crfsuite`
- Use provided `train.py` to learn a CRF model.
  `$ python3 run.py train CRF`

  You may modifiy learner parameters (loss function, thresholds, learning rates, etc).
- Check performance on development data.
  `$ python3 run.py predict CRF`
- `python-crfsuite` admits sparse vectors of boolean features represented as lists of names of the columns (i.e. features) that are true for each example.

Outline

Classifier

You can apply the learned models to new data and evaluate the results (stored in directory `results`):

```
$ python3 run.py predict MEM devel
$ python3 run.py predict SVM devel
$ python3 run.py predict CRF devel
```

Once a best configuration is selected experimenting with different features, algorithms, and hyperparameters, it can be applied to `test` data to evaluate generalization ability of the model.

```
$ python3 run.py extract test
$ python3 run.py predict MEM test
$ python3 run.py predict SVM test
$ python3 run.py predict CRF test
```

Outline

Improve provided ML system

Starting point:

- Prefix-tree baseline reaches about 58% macro F1.
- Provided ML system reaches about 65%-67%, depending on the algorithm

Task:

- Improve the performance of the provided system by:
  - Adding new features
  - Experimenting with different hyperparameters for the ML algorithms
  - Get insights from training partition. Use devel partition only to evaluate system behaviour.

The goal is not to reach the highest F1 score but to learn through experimenting with different strategies.

Choosing useful features

- Used models are *token* classifiers, so there is a feature vector *per token*.
- Features about a token should allow its classification, so they should encode information about *both* the token itself and its context (i.e. nearby words).
- Feature names must be *unambiguous*.
  E.g., a feature named `sufx:azole` may not be enough if one wants to encode also suffixes for nearby words. In that case, different feature names are needed (e.g.: `sufx:azole` for the focus word, plus e.g. `sufx-2:azole`, `sufx-1:azole`, `sufx+1=azole`, `sufx+2=azole` for nearby words).

Choosing useful features

- IMPORTANT: Feature names such as `sufx=azole` or `sufx:azole` are not key-value pairs (i.e. not a `sufx` feature with value `azole`), but just a string naming a binary (true/false) feature.
  The feature name could be any other (`sufx-is-azole`, `word-ends-in-azole`, ...) as long as it is active (i.e. present in the sparse vector) only when that property holds.
- IMPORTANT: Features are boolean, and only those with value `true` are listed. So, a feature not present in the list is an *existing* column with value `false`.

  Thus, it does not make sense to have e.g. a boolean feature `capitalized:true` and another feature `capitalized:false`, since the absence of the former is equivalent to the presence of the latter and viceversa.

Outline

Exercise Goals

What you should do:

- Extend your feature extractor.
- Experiment with different algorithms and parameterizations.
- Keep track of tried features and parameter combinations, and results produced by each.

What you should **NOT** do:

- Use neural network learners. We'll do that later on the course.
- Get insights about errors from `devel` dataset. You have `train` for that. `devel` is only used to evaluate the performance of a given configuration.
- Select features or hyperparameters based on system performance on `test` dataset. You have `devel` for that. `test` is only used to evaluate model generalization ability once the best configuration has been chosen.

Deliverables

At the end of the NERC task, you will need to deliver a single report on the work carried out on NERC-ML, NERC-NN, and NERC-LLM systems

So, during the development and experimentation on NERC-ML:

- keep track of tried/discarded features
- keep track of used algorithms and parameterizations.
- Record obtained results in the different experiments, and compile the information you'll later need to elaborate the report.