Language Technology - EDAN20
Assignment 5 - Fall 2020

Dependency Parsing

Hicham Mohamad, hi8826mo-s
hsmo@kth.se

October 22, 2022

1 Objectives

The objectives of this assignment are to:

- Know what a dependency graph is
- Understand the principles of Nivre's parsing mechanisms
- Extend Nivre's parser with a guiding predicate that parses an annotated dependency graph
- Extract features to learn parsing actions from an annotated corpus
- Write a short report on your results
- In this assignment, you will only generate the machine-learning models from the extracted features. You will complete the parser and apply it in the next assignment.

2 Organization and location

The fifth lab session will take place on October 8th and 9th. There can be last minute changes. Please always check the official times here: `https://cloud.timeedit.net/lu/web/lth1/ri1Q5006.html`

You can work alone or collaborate with another student.

Each group will have to:

- Write a program that parses a sentence when the dependency graph is known
- Extract features from the parsing actions.

3 Programming

This assignment is inspired by the shared task of the Tenth conference on computational natural language learning, **CONLL-X**, and uses a subset of their data. The conference site contains a description of multilingual dependency parsing, reference papers, training and test sets for a variety of languages, as well as evaluation programs. See also **CONLL 2007**, on the same topic.

Please note that the original CoNLL-X site is down. To access the pages, use the Archive.org site:

`https://web.archive.org/web/20161105025307/http://ilk.uvt.nl/conll/` and to download the data sets, use the local copies.

In this session, you will implement a dependency parser for Swedish. Should you want to use another corpus, please tell me in advance.

Choosing a training and a test sets

The CONLL-X annotated corpora and annotation scheme are available **here**. The Swedish corpus called *Talbanken* was originally collected and annotated in Lund and modified by Joakim Nivre. You can read details on the corpus and references **here**.

1. In this assignment, you will use the CONLL-X Swedish corpus. Download the tar archives containing the training and test sets for Swedish and uncompress them: [**data sets**]. Local copies: [**training set**] [**test set**] [**test set with answers**].

Nivre's parser

For each sentence with a projective dependency graph, there is an action sequence that enables Nivre's parser to generate this graph. Gold standard parsing corresponds to the sequence of parsing actions, left-arc (la), right-arc (ra), shift (sh), and reduce (re) that produces the manually-obtained, gold standard, graph.

Using an annotated corpus, we can derive all the action sequences producing the manually-parsed sentences (provided that they are projective). We can then train a classifier to predict an action from a current parsing context. To be able to predict the next action from a given parsing state, gold standard parsing must also extract feature vectors at each step of the parsing procedure. The simplest parsing context corresponds to words' part of speech on the top of the stack and head of the input list (the queue).

Once the data collected, the training procedure will produce a 4-class classifier that you will embed in Nivre's parser to choose the next action. During parsing, Nivre's parser will call the classifier to choose the next action in the set la, ra, sh, re using the current context.

1. Run the `dparser.py` program [**1**]. You will have to edit the data paths so that they fit your configurations.
2. Understand from the slides and the program how Nivre's parser is extended to carry out a gold standard parsing. Given a manually-annotated dependency graph, what are the conditions on the stack and the current input list – the queue – to execute left-arc, right-arc, shift, or reduce? Start with left-arc and right-arc, which are the simplest ones.
3. The parser can only deal with projective sentences. In the case of a nonprojective one, the parsed graph and the manually-annotated sentence are not equal. Examine one such sentence and explain why it is not projective. Take a short one (the shortest).
4. You will use three feature sets to build your models:
   - The top of the stack and the first word of the input list (word forms and parts of speech);
   - The two first words and POS on the top of the stack and the two first words and POS of the input list;
   - A feature vector that you will design that will extend the previous one with at least two features. You can read **this paper** (Table 6) to build your vector. In this paper, Sect. 4 contains the description of the feature codes: LEX, POS, fw, etc.
5. Nivre's parser sets constraints to actions. Name a way to encode these constraints as features. Think of Boolean features.

3.1 Parsing functions

Using the actions in the set {`la, ra, sh, re`} produces an unlabelled graph. It is easy to extend the parser so that it can label the graph with grammatical functions. In this case, we must complement the actions la and ra with their function using this notation for example: `la.++`, `la.+A`, `la.+F`, `la.AA`, `la.AG`, etc. where the prefix is the action and the suffix is the function.

Extracting features (I)

The final goal is to parse the Swedish corpus in CoNLL-X and produce a labelled dependency graph. You will show the parsing results at the end of the 6th assignment. In this assignment, you will only generate the scikit-learn models.

You will consider three feature sets and you will train the corresponding logistic regression models using scikit-learn:

1. The first set will use the word and the part of speech extracted from the first element in the stack and the first in the queue,
2. the second one will use two elements from the stack and two from the input list.
3. For the third model, you will extract at least two more features, one of them being the part of speech and the word form of the word following the top of the stack in the sentence order.

These sets will include two additional Boolean parameters, "can do left arc" and "can do reduce", which will model constraints on the parser's actions. In total, the feature sets will then have six, respectively ten and 14, parameters.

This means that the purpose of this assignment is to generate three scikit-learn models for the labelled graphs.

To carry this out:

1. Create a Python module (program) named `features.py` with a

   ```
   def extract(stack, queue, graph, feature_names, sentence):
       ...
       return features
   ```

   function that will return the features in a dictionary format compatible with scikit-learn. You have a code example of feature encoding in this format in the chunking program.
2. Parse the annotated corpus using the reference parser and collect the features in a matrix (X) and the transitions in a vector (y).
3. Generate the three scikit-learn models using the code models from the chunking labs. You will evaluate the model accuracies (not the parsing accuracy) using the classification report produced by scikit-learn and the correctly classified instances. This is done with the training set.

The first lines of your features for the 4 parameters (**x**) and labelled actions (y) should look like the excerpt below, where the columns correspond to `stack0_POS`, `stack1_POS`, `stack0_word`, `stack1_word`, `queue0_POS`, `queue1_POS`, `queue0_word`, `queue1_word`, `can-re`, `can-la`, and the transition value:

```
x = ['nil', 'nil', 'nil', 'nil', 'ROOT', 'NN', 'ROOT', 'Äktenskapet', False
    , False], y = sh
x = ['ROOT', 'nil', 'ROOT', 'nil', 'NN', '++', 'Äktenskapet', 'och', True,
    False], y = sh
x = ['NN', 'ROOT', 'Äktenskapet', 'ROOT', '++', 'NN', 'och', 'familjen',
    False, True], y = sh
x = ['++', 'NN', 'och', 'Äktenskapet', 'NN', 'AV', 'familjen', 'är', False,
    True], y = la.++
x = ['NN', 'ROOT', 'Äktenskapet', 'ROOT', 'NN', 'AV', 'familjen', 'är',
    False, True], y = ra.CC
x = ['NN', 'NN', 'familjen', 'Äktenskapet', 'AV', 'EN', 'är', 'en', True,
    False], y = re
x = ['NN', 'ROOT', 'Äktenskapet', 'ROOT', 'AV', 'EN', 'är', 'en', False,
    True], y = la.SS
x = ['ROOT', 'nil', 'ROOT', 'nil', 'AV', 'EN', 'är', 'en', True, False], y =
    ra.ROOT
x = ['AV', 'ROOT', 'är', 'ROOT', 'EN', 'AJ', 'en', 'gammal', True, False], y
    = sh
```

4 Complement (Optional)

You can read an historical source to transition-based parsing: *An Efficient Algorithm for Projective Dependency Parsing*, by Joakim Nivre (2003) [pdf].

5 Appendix

References

[1] Pierre Nugues, *Language processing with Perl and Prolog*, 2nd edition, 2014, Springer.

[2] P. Nugues, Language Technology - EDAN20, Lectures notes: `https://cs.lth.se/edan20/`

[3] D. Jurafsky, J. Martin, *Speech and Language Processing*, 3rd edition. Online: `https://web.stanford.edu/~jurafsky/slp3/`

[4] VanderPlas, A Whirlwind Tour of Python. O'Reilly 2016. Online reading: `https://jakevdp.github.io/WhirlwindTourOfPython/`

[5] Pierre Nugues, Assignment 5: `https://github.com/pnugues/edan20/blob/master/notebooks/5-triples.ipynb`

[6] Universal Dependencies: `https://universaldependencies.org/`

[7] Graphical representation of sentences, conllu.js: CoNLL-U format library for JavaScript: `http://spyysalo.github.io/conllu.js/`

[8] Language Pipelines. `http://vilde.cs.lth.se:9000/#/`

[9] J Fan, D Ferrucci and al. (2010), PRISMATIC: Inducing Knowledge from a Large Scale Lexicalized Relation Resource. IBM Watson Research Lab: `https://www.aclweb.org/anthology/W10-0915.pdf`

[10] Geron, Jupyter notebook on linear algebra from Machine Learning and Deep Learning in Python using Scikit-Learn, Keras and TensorFlow: `https://github.com/ageron/handson-ml2`