Language Technology - EDAN20
Assignment 3 - Fall 2020

Extracting Noun Groups Using Machine Learning Techniques

Hicham Mohamad, hi8826mo-s
hsmo@kth.se

august 30, 2020

1 Objectives

The objectives of this assignment are to:

- Write a program to detect partial syntactic structures
- Understand the principles of supervised machine learning techniques applied to language processing
- Use a popular machine learning toolkit: scikit-learn
- Write a short report of 1 to 2 pages on the assignment

2 Organization and location

The third lab session will take place on September 24 and 25.

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 and train a machine-learning program to detect noun groups.
- Evaluate the results and comment them briefly.

3 Programming

Choosing a training and a test sets

1. As annotated data and annotation scheme, you will use the data available from **CoNLL 2000**.
2. Download both the training and test sets (the same as in the previous assignment) and decompress them. (Local copies available here: [train.txt] [test.txt].)
3. Be sure that you have the **scikit-learn** package: Start it by typing `import sklearn` in Python.

Baseline

Most statistical algorithms for language processing start with a so-called baseline. The baseline figure corresponds to the application of a minimal technique that is used to assess the difficulty of a task and for comparison with further programs.

1. Read the baseline proposed by the organizers of the **CoNLL 2000 shared task** (In the *Results Sect.*). What do you think of it?

2. Implement this baseline program. You may either create a completely new program or start from an existing program that you will modify [Program folder].

   - Complete the train function so that it computes the chunk distribution for each part of speech. You will use the train file to derive your distribution and you will store the results in a dictionary. Below, you have an excerpt of the expected results:

     ```
     {'JJR':
     {'I-ADVP': 17, 'I-ADJP': 45, 'I-NP': 204, 'B-ADVP': 63,
     'B-PP': 2, 'B-ADJP': 111, 'B-NP': 382, 'B-VP': 2,
     'I-VP': 11, 'O': 16},
     'CC':
     {'B-ADVP': 3, 'O': 3676, 'I-VP': 104, 'B-CONJP': 6,
     'I-ADVP': 30, 'I-UCP': 2, 'I-PP': 24, 'I-ADJP': 26,
     'I-NP': 1409, 'B-ADJP': 2, 'B-NP': 18, 'B-PP': 70,
     'I-PRT': 1, 'B-VP': 1},
     'NN':
     {'B-LST': 2, 'I-INTJ': 2, 'B-ADVP': 38, 'O': 37,
     'I-ADVP': 11, 'B-INTJ': 1, 'I-UCP': 2, 'B-UCP': 2,
     'I-VP': 77, 'B-PRT': 2, 'I-ADJP': 41, 'I-NP': 24456,
     'B-ADJP': 44, 'B-NP': 5160, 'B-PP': 15, 'B-VP': 257},
     ...
     ```

   - For each part of speech, select the most frequent chunk. In the example above, you will have (`NN, I-NP`)
   - Using the resulting associations, apply your chunker to the test file.
   - You will store your results in an output file that has four columns. The three first columns will be the input columns from the test file: word, part of speech, and gold-standard chunk. You will append the predicted chunk as the 4th column. Your output file should look like the excerpt below:

     ```
     Rockwell NNP B-NP I-NP
     International NNP I-NP I-NP
     Corp. NNP I-NP I-NP
     's POS B-NP B-NP
     Tulsa NNP I-NP I-NP
     unit NN I-NP I-NP
     said VBD B-VP B-VP
     it PRP B-NP B-NP
     ```

     The CoNLL 2000 evaluation script will use these two last columns, chunk and predicted chunk, to compute the performance.

3. Measure the performance of the system. Use the **conlleval.txt** evaluation program used by the CoNLL 2000 shared task.

   - conlleval.txt is the official CoNLL Perl script. It expects the two last columns of the test set to be the manually assigned chunk (gold standard) and the predicted chunk.
   - Start it like this

     ```
     $ conlleval.txt <out
     ```

     where the out file contains both the gold and predicted chunk tags. `conlleval.txt` is a Perl script.
   - Perl is installed on most Unix distributions. If it is not installed on your machine, you need to install it. Make also sure that you have the execution rights. Otherwise change them with:

     ```
     $ chmod +x conlleval.txt
     ```

   - The `conlleval.txt` script expects the new lines to be `\n` as in Unix. If you run your Python program on Windows, your new lines will be `\r\n`. To have the correct new lines, add this parameter to `open()`: `newline='\n'` like this:

```
f_out = open('out', 'w, newline='\n')
```

- The complete description of the CoNLL 2000 evaluation script is available here: `https://www.clips.uantwerpen.be/conll2000/chunking/output.html`.

Using Machine Learning

In this exercise, you will apply and explore the `ml_chunker.py` program. You will start from the original program you downloaded and modify it so that you understand how to improve the performance of your chunker. You will not add new features to the feature vector.

The program that won the CoNLL 2000 shared task (Kudoh and Matsumoto, 2000) used a window of five words around the chunk tag to identify, c_i . They built a feature vector consisting of:

- The values of the five words in this window: $wi-2$, w_{i-1} , w_i , w_{i+1} , w_{i+2}
- The values of the five parts of speech in this window: t_{i-2} , t_{i-1} , t_i , t_{i+1} , t_{i+2}
- The values of the two previous chunk tags in the first part of the window: c_{i-2} , c_{i-1}

The two last parameters are said to be dynamic because the program computes them at run-time. Kudoh and Matsumoto trained a classifier based on support vector machines. Read **Kudoh and Matsumoto's paper** and the **Yamcha** software site.

1. What is the feature vector that corresponds to the `ml_chunker.py` program? Is it the same Kudoh and Matsumoto used in their experiment?
2. What is the performance of the chunker?
3. Remove the lexical features (the words) from the feature vector and measure the performance. You should observe a decrease.
4. What is the classifier used in the program? Try two other classifiers and measure their performance: decision trees, perceptron, support vector machines, etc.. Be aware that support vector machines take a long time to train: up to one hour.

Improving the Chunker

Implement one of these two options, the first one being easier.

1. Complement the feature vector used in the previous section with the two dynamic features, c_{i-2} , c_{i-1} , and train a new model. You will need to modify the `extract_features_sent()` and `predict()` functions.

   In his experiments, your teacher obtained a **F1 score** of 92.65 with logistic regression and a `lbfgs` solver and automatic multiclass;

2. If you know what **beam search** is, apply it using the probability output of logistic regression or the score if you use support vector machines.

   With the same classifier and a beam diameter of 5, your teacher obtained 92.87.

4 Reading

You will read the article, *Contextual String Embeddings for Sequence Labeling* by Akbik et al. (2018) and you will outline the main differences between their system and yours. A LSTM is a type of recurrent neural network, while CRF is a sort of beam search. You will tell the performance they reached on the corpus you used in this laboratory.

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] Pierre Nugues, Assignment 3: `https://github.com/pnugues/edan20/blob/master/notebooks/3-language_detector.ipynb`

[5] Scikit-learn, Feature Extraction: `https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.DictVectorizer.html`

[6] Scikit-Learn, Neural Network Models `https://scikit-learn.org/stable/modules/neural_networks_supervised.html`

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

[8] CLD3 - Compact Language DEtector v3: `https://github.com/google/cld3`

[9] 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`