NLP 2016/17 - Homework 1 Report

Marco Favorito
favorito.1609890@studenti.uniroma1.it

May 4, 2017

1 System oveview

I implemented the system in Python 3.5, using the APIs of the suggested library, `sklearn_crfsuite`. You can see some functionalities through three demos:

- `demos/demo_01.py`, which shows how performances change varying the percentage of train data used;
- `demos/demo_02.py`, which executes a grid search over two parameter space for `epsilon` (the condition of convergence for the averaged perceptron algorithm) and `max_iterations` (the maximum number of iterations executed by the averaged perceptron);
- `demos/demo_03.py` similar to `demo_02.py`, but adding the bigger corpus from the `crowd-sourced-annotations.txt` file.

You can use the system from command line, just calling the `main.py` in a straightforward way (please, look at the `README.txt` for further details).

1.1 How I implemented the model

As I said, I used the APIs of the library `sklearn_crfsuite`, creating instances of `CRF` class and using the averaged perceptron as algorithm and the options `all_possible_transitions` and `all_possible_states` in order to achieve better performances. For the grid search, I used the `GridSearchCV` class, and the development set for the validation. The `param_grid` is defined in `settings.py`, where I preferred lower values of `max_transitions` in order to avoid overfitting on the train data. Moreover, I needed to define my own scorer, defined in `metrics/my_scorer.py`, in order to implement your evaluation system. That module relies on the core functions for compute H, I, D, implemented in `metrics/evaluation.py`

1.2 How I encoded the features

In the file `features/features_utils.py`[1]there are most of the functions used for extract features from pre-processed data. In particular, the `extract_paper_features` does exactly what the paper explain; I build a dictionary of pairs ("feature", "activation value") from each sample, depending on the hyperparameter δ, that feeds the training process. NOTICE: before training the model, I always compute the length of the longest word from the training set, which is an obvious upper bound for δ.

2 Result and analysis

After executing `demo_01.py`, in `output/demo_01/report.txt` you will find all the evaluation metrics obtained fitting the model using only a certain percentage of the train set (for increasing values: 10%, 20% ...).

An important observation is that, in every test, the *Recall* score is lower than the *Precision*, which means that the classifier is a bit "shy" about inserting breaks in the word, and tries to guess only if it is quite confident (in other words, the *false negatives* are more than the *false positives*). As expected, increasing the number of train samples improve slightly the performance (the lowest on test set is $\sim$ 65%), but just after the 50% of the set we reach the maximum F-score, and we have not more improvements[2] (F-score on the development set $\sim$ 86%; on test set $\sim$ 84%). The file shows only the delta with which we obtained the best F-score, which **on average**[3] are quite high ($\delta \geq$ 6-7). For lower values of delta (1,2,3) we have a sort of underfitting, which is also as expected. What is interesting is that, given a tuning session for δ (i.e. fixed the percentage of train set used), only for the first values of δ we improve performances, but seems that, after δ = 4-5, we don't improve so much the model (sometimes it worsens!).

With the `demo_02.py`, I performed a grid search, as described above. Here, performances improve greatly, achieving an F-score of $\sim$ 91% on the test set. You will found all the details in the `output/demo_02` folder. I took the best model from the output and I put it into the `task_1/model` folder.

3 Extra points

Executing `demo_03.py`, similarly to `demo_02.py`, there is a grid search over the same parameter grid. In this case, we found a slight increase of performance: max F-score on the test set is $\sim$ 92%.

[1]You may notice that, in the `features` folder, there are other variants for extract the features, but since I do not obtain consistent improvements, and for lack of space in this document, I cannot comment.

[2]In this demo I did not executed the grid search; otherwise, almost surely performances would be higher.

[3]It seems that the library, when using the averaged perceptron algorithm, introduces some random element that prevent the model to be deterministic; indeed, on each run, evaulations can change slightly.