Relation Classification with Parameter-restricted Models

Profilierungsmodul II

December 20, 2019

In this homework you will implement training and prediction with models for relation classification. To make the task more interesting, we add the side-condition that the amount of **`trainable`** parameters of your models is limited. Models with fewer parameters usually need fewer data to train and can have several advantages such as more robustness, faster training or better adaptability.

You can implement any model of your choice based on the deep learning packages that we used in the course. In any case, make sure that input and output formats correspond to the specification.

Formalities

- Work in teams of 2 or 3 students on this exercise sheet.
- Hand in your solution (zipped `src` folder and zipped `predictions` folder) no later than Tuesday, February 4, 2019, 18:00.
- If you present your approach (Exercise 3), this will be on Wednesday, February 5, 2019.
- There is a total of 22.5 points for this exercise sheet.

Task Description

We define the task of relation prediction in the following way:
Decide which relation(s) (of a fixed set of given relations) hold between two selected entities in a sentence.

Some remarks:

- We consider 41 relations defined in the TAC KBP evaluations[1]. Some examples are:

[1] https://tac.nist.gov/

  - `per:employee_of`: Does (did) person X work for company Y?
  - `org:city_of_headquarters`: Is (was) company/organization X based in city Y?
  - `per:countries_of_residence`: Does (did) person X live in country Y?
  - `per:title`: Does (did) person X have the job title Y?

- In theory, several (or no) relations could hold between two entities. However, **our dataset is constructed in a way so that exactly one (or no) relation holds between the selected entity pairs** in the instances. Therefore, we can use a multiclass classifier (e.g. a softmax over all relations including a special output `no_relation`).

The relation classifier we build in this project could be used in a pipeline system after a named entity recognizer in order to find all relations in a sentence. Consider the following example sentence:

```
The last remaining assets of bankrupt Russian oil company Yukos - including
its headquarters in Moscow - were sold at auction for nearly 3.9 billion U.S.
dollars Friday .
```

A named entity recognizer might have found the following 4 entities: `Russian`, `Yukos`, `Moscow`, `U.S.`

We can then list all pairs of entities in the sentence (`Russian-Yukos`, `Russian-Moscow`, ...`U.S.-Moscow`) and solve the relation classification task for each pair as defined above.

Data Format

The data we use [2] contains as an instance a sentence with two pre-selected entites, and as a label the relation (or `no_relation`) that holds between them (according to human annotators). Download the data from the course homepage (use the password shared in class to extract `relation_project.zip`), and have a look at the training data (`relation_project/data/conll/train.conll`).

The data is stored in the CONLL-format, where all tokens of a sentence are listed line-by-line, and for each token different tab-separated pre-processing information is provided (e.g., part-of-speech, named-entity-tag, dependency grammar information). The two entities selected as relational subject (the first logical argument of a relation) and relational object (the second logical argument) are merked as well. Each instance starts with meta-information (identifiers and training label) and is separated from the next instance by an empty line. For your convenience, we have provided a helper method (in `relation_project/src/utils.py`) that reads files in the CONLL-format, and returns the tokens (replacing relational arguments by dummy values). Feel free to use or extend this method (or write your own).

[2] For more details about the provenance of the data see: `http://anthology.aclweb.org/D/D14/D14-1164.pdf`

1 Exercise: Parameter-restricted relation prediction without external resources

For this exercise, train and evaluate deep learning models, that *only use information contained in the training data* (such as the tokens, or the tags in the CONLL file). Do not use external resources, such as pre-trained word-vectors or models, and do not-use non-deep-learning tools such as parsers. When you are done with developing, provide scripts in the `src` directory that take three arguments:

```
train_predict_xx.py ../data/conll/train.conll ../data/conll/dev.conll
../predictions/xx.dev ../data/conll/test.conll ../predictions/xx.test
```

that train your model on `train.conll`, and `xx.test` (`xx.dev`) are the predicted labels, line-by-line for all instances, resulting for `test.conll` (`dev.conll`). Your script can use development data e.g., for early stopping. In order to discourage overfitting, we do not release the real test gold labels; just use the dev-data instead for your experimentation.

You can evalaute the resulting predictions using the provided evaluation script:

```
score.py ../data/gold/dev.gold ../predictions/xx.dev
```

1. Write a script `train_predict_noext_10k.py` that provides the functionality as described above, and uses not more than 10'000 trainable parameters.
2. Write a script `train_predict_noext_100k.py` that provides the functionality as described above, and uses not more than 100'000 trainable parameters.
3. Write a script `train_predict_noext_1m.py` that provides the functionality as described above, and uses not more than 1'000'000 trainable parameters.

Provide the `src` folder with your scripts, and the `predictions` folder. Your points are related to the performance (micro F1-score calculated by score.py) on dev data and (held-out) test data:

- `train_predict_noext_10k.py`:
  - Dev-set F1 >= 46%: +1 point
  - Dev-set F1 >= 49%: +0.5 point
  - Test-set F1 >= 44%: +0.5 point
  - Test-set F1 >= 47%: +0.5 point
- `train_predict_noext_100k.py`:
  - Dev-set F1 >= 51%: +1 point
  - Dev-set F1 >= 54%: +0.5 point
  - Test-set F1 >= 49%: +0.5 point
  - Test-set F1 >= 52%: +0.5 point
- `train_predict_noext_1m.py`:

  - Dev-set F1 >= 52%: +1 point
  - Dev-set F1 >= 55%: +0.5 point
  - Test-set F1 >= 50%: +0.5 point
  - Test-set F1 >= 53%: +0.5 point

2 Exercise: Parameter-restricted relation prediction

For this exercise, you can now use pre-trained word vectors as an external resource.

- We have provided GloVe[3] word vectors in the directory `/big/b/beroth/glove`, including a filtered version that contains only vectors for words in our relation extraction data set.
- For using BERT, you can uncomment the respective lines in `setup.sh` in the project folder (which uses BERT-as-a-service[4]).

Proceed as in the previous exercise, and limit the number of parameters accordingly.

1. Provide a script `train_predict_ext_10k.py` that provides the functionality as described above, and uses not more than 10'000 trainable parameters.
2. Provide a script `train_predict_ext_100k.py` that provides the functionality as described above, and uses not more than 100'000 trainable parameters.
3. Provide a script `train_predict_ext_1m.py` that provides the functionality as described above, and uses not more than 1'000'000 trainable parameters.

Your points are related to the performance on dev data and (held-out) test data:

- `train_predict_ext_10k.py`:
  - Dev-set F1 >= 54%: +1 point
  - Dev-set F1 >= 57%: +0.5 point
  - Test-set F1 >= 52%: +0.5 point
  - Test-set F1 >= 55%: +0.5 point
- `train_predict_ext_100k.py`:
  - Dev-set F1 >= 56%: +1 point
  - Dev-set F1 >= 59%: +0.5 point
  - Test-set F1 >= 54%: +0.5 point
  - Test-set F1 >= 57%: +0.5 point

[3] `https://nlp.stanford.edu/projects/glove/`
[4] `https://bert-as-service.readthedocs.io/en/latest/`

- `train_predict_ext_1m.py`:
  - Dev-set F1 >= 58%: +1 point
  - Dev-set F1 >= 61%: +0.5 point
  - Test-set F1 >= 56%: +0.5 point
  - Test-set F1 >= 59%: +0.5 point

3 Exercise: Experimental relation prediction

For this exercise, the idea is to experiment with creative ideas you might have for relation classification. Performance is not the key factor.

- Provide a script `train_predict_experimental.py`, that deviates substantially from the solutions you provided in the previous exercises and from the standard approaches for text classification we studied in the course. (+2.5 points)
- Provide a file `experimental.txt`, where you describe the main idea behind your approach ($\sim$ 2000 characters). (+2.5 points)
- Show the result of your experimental approach and how it compares to the other approaches in a small presentation ($\sim$ 10 minutes). (+2.5 points)

Remark

Each of your models must train within 60 minutes on a machine in the Antarktis CIP Pool. For your convenience we have provided a script `setup.sh` that installs relevant python packages and is tested on machines in the Antarktis CIP Pool. Do not forget to initialize all random seeds, if you want to have stable results. It is advised to test your code on a CIP machine to ensure that there are no problems when running the code in grading environment (CIP Pool).