Spoken Language Understanding: slot tagging project

Christian Raymond

6 février 2019

Table des matières

1 Introduction

The project objective is to build taggers for the ATIS (Air Travel Information System) domain. This project is running over 3 sessions of 2 hours and will be evaluated. You will work by group of 2 students. Two different taggers have to be implemented : the first one should be based on CRF (Conditional Random Fields), the choice of the second is free. It could be implemented with any classification algorithms, I suggest to use the python scikit-learn library in this case : `http://scikit-learn.org/stable/`. In the case you choose to build a neural network base tagger, I suggest you use Keras library `https://keras.io/`. The project will follow the rules of a evaluation campaign : I will provide to you training data to train and tune your systems. At the end your tagger will be evaluated on an unseen part of the corpus. The performance of your systems will be a part of the notation. You will also provide a small pdf document summarizing the conception of your taggers.

2 Provided data

2 files are provided at the beginning of the project :

1. atis.train is a collection of about 5000 dialog utterances whose aim to train and tune your systems ;
2. evaluation.pl is the script to compute the performances of your systems, it provides concepts error statistics as well as the global statictics (measure to tune is global FB1).

3 Expected output

We expect you to provide the predictions of your two systems applied on the test corpus revealed only at the end of the project. The test corpus look like the training corpus except that the last column of the ground truth is missing, thus it contains only one column : the words. You should provide predictions files that contains the tag predicted by your system, these file must contains the same number of lines as the test file. the following table illustrates the two files

test file	prediction file
I	O
want	O
to	O
go	O
to	O
Chicago	arrival.city_name
from	O
Atlanta	departure.city_name
The	O
...	...

At the end you may provide up to 6 files (at least 1 one each system) to test several configuration of tuning/model. I will keep the best ones for evaluation.

4 Guide to build a basic CRF tagger

Use the software « wapiti », `https://wapiti.limsi.fr/`, it should already installed. Create a template file, let's name it « template ». This file should contains, the templates to generate feature observation for the tagger. Let's start by the obvious ones, the observation of the current word to decide of the current label in the sequence, the corresponding template information to put in the file is :

```
U0:%x[0,0]
```

It means : generate all feature functions that look at the feature at position 0 in the sequence (current word in our atis.train) and column 0 in atis.train. This template language with examples is depicted at `https://taku910.github.io/crfpp/`. you can now train a first CRF tagger with :

```
wapiti train -p template -t 4 atis.train atis.crfmodel
```

t is a flag to use 4 threads to speed up the learning, and atis.crfmodel is the name of the model file produced. You can observe what the CRF has learnt using the following command :

```
wapiti dump atis.crfmodel modellisible.txt
```

Now in modellisible.txt you have all features fonctions that have been generated by the template scored by the CRF, more the weight computed is high more importance the features has to model a phenomenum.

To use the CRF tagger, you have to use the following command :

```
wapiti label -m atis.crfmodel atis.train > predictions
```

The file predictions contains the atis.train + a new column : the predictions that the tagger did. So to evaluate the performance of your tagger, you have just to compare the two last columns of the file (ground truth and prediction) using the scrypt evaluate.pl.

```
cat predictions | evaluate.pl
```

Now you have built a first system, it is up to you to add new informations (for example list of classes (like city names)) et test over template sets to come up with a strong tagger.

5 Guide to build an other tagger

Both scikit-learn or Keras use numpy matrices to store data. I will provide some tips to build the input of the algorithm. To use classifier or design the typology of your neural network, you can start from example provided in my course.

5.1 Destructuring sequences and use of any algorithm

For a system that destructure sequences : one training sample for one position in a sequence, you will need to create as many training samples as words (nb utterances*average length of an utterance). The observation have to be encoded as a 2D numpy scalar matrix (nbsamples*sizeofrepresentation). In atis.train you will have 52170 samples. If you choose to use as observations words inside a windows $[-1, 1]$ (3 words : current word, previous word and next word), and if you represent each word by an embedding of size 100, then the representation will be of size 300.

The predictions (groundtruth) needed to train the network have to be encoded as a vector of tag index.

5.2 Deep learning tagger : keeping the sequence structure

The observation have to be encoded as a 3D numpy scalar matrix (nbsamples*sizeof longuestsequence*representationofthecurrentobservation). In ATIS you would have :

1. nbsamples is 4978 (number of utterances)
2. the longest sequence is size 86 (you must pad the shorter sequences with 0)
3. the representation is the vector coding the current observation in the sequence (in general the word itself at least)
   (a) you can create your own representation vector encoding the information you want and use it as third dimension (use zeros vector to pad)
   (b) if you plan to use an embedding layer (to compute a word embedding together with the full network), you can just use a 2D numpy matrix (nbsamples*sizeof longuestsequence) fulled with the words (coded as an integer), use integer 0 to pad : the embedding layer will produce the third dimension.

The predictions (groundtruth) needed to train the network have to be encoded as a matrix[3] : (nbsamples*sizeoflonguestsequence*sizeofthetagcoding). There is two ways to code the tag :

— dense : you need to code the output value of each output neuron (*e.g.* $[0110]$ for a NN with 4 output neurons where only the second and the third are active : useful for multilabels classification : predict several tag for a sample. Use loss $categorical_crossentropy$ with danse tag coding.
— in our case, and multiclass problem where only 1 output neurons can be active, you should just give the index of the active neurons (*e.g.* [5] to active the neuron 6) : Use loss $sparse_categorical_crossentro$ with sparse tag coding.

6 Tips

— remember in ML one key is to find the best compromise between under and overfitting
  — split training data provided in train/validation
  — use the validation split to evaluate your system and tune your system
— keep in mind that you may encounter unknown words in the test set : deal with that
— you may use a fast GPU server to train neural network : raoh.educ.insa