Language Understanding Systems - Final Project

Maria Pia Natale (189160)

University of Trento
mariapia.natale@studenti.unitn.it

Abstract

The following document explains the work done for the final project of the Language Understanding System course. The aim of this project was to develop discriminative models for Spoken Language Understanding and to compare them with the results obtained with generative method (Weighted Finite State Transducers). Conditional Random Fields (CRF++) and Recurrent Neural Network (RNN) have been both used as discriminative methods.

1 Introduction

This project has been focused on the development of discriminative models for SLU on movie domain. The generative model used for this purpose was the one developped for the first project of the course, the Weighted Finite State Transducer. These transducers associate each word of the sentence in the dataset to a predicted concept tag using the IOB notation.
Conditional Random Fields (CRF++) and Recurrent Neural Network (RNN) have been chosen to develop the discriminative models.
The following section describes CRFs and focuses on how features and/or different templates can change performances, improving or wasting them. In the section 3, RNN are treated paying particular attention on the test cases performed and all the features used with the aim of improving the results.

2 Conditional Random Fields

The generative approach models the complete **joint probability** $P(X, Y)$ which becomes $P(X, Y) = P(Y|X)P(X)$ where $P(Y|X)$ is the probability of Y given X and $P(X)$ is the prior distribution of X. In the transducers created during the mid-term project, X is a random variable over the word of a sentence in the dataset and Y is a random variable over the movie domain IOB tags.
While, the discriminative approach aims at modelling the **conditional distribution** $P(Y|X)$ without considering the prior distribution $P(X)$.
Conditional Random Fields (Chen, 2015) are a type of discriminative model to perform sequence labeling defined by a dependency graph G and a set of features F. Each feature function takes in as input:

- a sentence x;
- the position i of a word in the sentence;
- the label y_i of the current word;
- the label y_{i-1} of the previous word.

To each feature function F_i a *weight* λ_i is assigned.
The conditional probability is defined as:

$$p(y|x;\lambda) = \frac{exp \sum_{j=1}^{J} \lambda_j F_j(x, y)}{Z(x, \lambda)} \quad (1)$$

where F_j is the feature function, $Z(x, \lambda)$ is a normalization factor described as:

$$Z(x, \lambda) = \sum_{y \in Y} exp \sum_{j=1}^{J} \lambda_j F_j(x, y) \quad (2)$$

In CRF, the feature function is computed as:

$$F_j(X, Y) = \sum_{i=1}^{n} f_j(y_{i-1}, y_i, x, i) \quad (3)$$

The output of a feature function is usually binary, 1 if there is a match, 0 otherwise.
The tool used to implement conditional random

fields is CRF++, a simple, customizable, and open source implementation of Conditional Random Fields (CRFs) for segmenting/labelling sequential data (CRF, 2015).

2.1 Test cases

The tool CRF++ allows for the creation of different templates in order to analyse the changing of performances in adding, removing or managing words and features. The template file describes a feature per line with the syntax $\%x[row, column]$ where row and $column$ are both integer numbers. The row is the position of the word taking in consideration with respect to the current word, while the $column$ indicates the feature we want to analyse.

2.1.1 Baseline

The first template created was a simple window taking into consideration only the current word:

Unigram
U00:%x[0,0]

Table 1: Baseline template

The results obtained have been used as baseline for the other tests.

Accuracy	Precision	Recall	FB1
87.59%	44.99%	60.49%	51.60

Table 2: Baseline results

2.1.2 Base features

The first tests made took into consideration the words, without considering any extra feature. I started to increase the window size by including one word at a time, both previous words and next words. After a few tests, the best result I obtained (`FB1: 76.51`) was with a window size starting from -3 (`U00:%x[-3,0]`) to 2 (`U05:%x[2,0]`).
With the tool CRF++, it is possible to create two different kinds of templates: unigram and bigram, distinguishable by the initial letter in the template rule (U and B, respectively). After adding other rules to manage bigrams, the template that gave me the best result is the following:

Unigram	Bigram
U00:%x[0,0]	B00:%x[0,0]
U01:%x[-1,0]	B01:%x[-1,0]
U02:%x[-2,0]	B02:%x[-2,0]
U03:%x[1,0]	B03:%x[1,0]
U04:%x[2,0]	
U05:%x[3,0]	

Table 3: Best result with words and bigram

with the following performance:

Accuracy	Precision	Recall	FB1
94.03%	87.53%	77.18%	82.03

Table 4: Best bigram window size score

2.1.3 Adding features

At this point of the work, I decided to add extra features with the aim to improve the performances obtained in the previous tests. In order to have better models, I enhanced the training set with the Part-Of-Speech tags (POS tags) and lemmas related to each word in the dataset. Starting from the template giving the best result obtained early, I added more rules in order to improve the performances taking into consideration the features and concatenating them to the words. The following template shows the rules that gave me the highest values:

Unigram	Bigram
U00:%x[-2,0]	B00:%x[-2,0]
U01:%x[-1,0]	B01:%x[-1,0]
U02:%x[0,0]	B02:%x[0,0]
U03:%x[1,0]	B03:%x[1,0]
U04:%x[2,0]	B04:%x[2,0]
U05:%x[0,1]	B05:%x[0,1]
U06:%x[0,0]/%x[0,1]	B06:%x[0,0]/%x[0,1]
U07:%x[0,0]/%x[0,2]	B07:%x[0,0]/%x[0,2]
U08:%x[1,0]/%x[1,1]	B08:%x[1,0]/%x[1,1]

Table 5: Template with words, lemmas and POS-tags

In this template we have the same rules both for unigram and bigram. Here, we are considering the words in the range $n \in [-2, 2]$ and the lemma associated to the current word (`U05:%x[0,1]`);

then we consider the concatenation of the current word with its lemma, the current word and the POS-tag associated to it and the word and the lemma of the following word. With this template, I obtained the following results:

Accuracy	**Precision**	**Recall**	**FB1**
94.60%	87.50%	79.56%	83.34

Table 6: Results with words, lemmas and POS-tags

2.1.4 Prefix and suffix

At this point, I added the prefix and suffix of each word as an additional feature for the training set. After a few tests, I decided to use a prefix and suffix of length 3 because this gave me best results. I tried to use the previous template (Table 5), adding prefix, suffix and both. In the table 7 the changing of results is shown.

	Prefix	**Suffix**	**Both**
83.34	83.21	83.39	83.21

Table 7: Adding prefix and suffix

After this test, I have found that concatenating prefix and suffix leads to the best result in all the tests:

Accuracy	**Precision**	**Recall**	**FB1**
94.76%	87.91%	80.66%	84.13

Table 8: Best results

As a last test, I took this template and I tried to learn it by changing the hyperparameter.

Value	**FB1**
0.5	83.27
0.75	83.58
1	**84.13**
1.25	84.03
1.50	84.01
1.75	83.93
2	83.83

Table 9: Best results with different hyperparameter

3 Recurrent Neural Network

Recurrent Neural Networks (Wildm, 2015) are a class of Neural Networks where there is a directed cycle between units. Unlike normal NN, where inputs and outputs are independent, here the connection between units allows for the exploitation of sequential information. RNNs are called ”recurrent” because they perform the same task for each element in a sequence, with the output being dependent on the previous computation. Typically, an RNN is unrolled into a full network. This unrolling simply means writing the network for the complete sequence. The computation of a RNN is governed by the following parameters:

- x_t that represents the input at each step,
- s_t that is the hidden state at time step t, i.e. the memory of the network calculated based on the previous hidden state and the current input
- o_t is the output at step t.

During the Language Understanding Systems course, we have seen two different types of RNNs: the Elman RNN and the Jordan RNN. The first one models the posterior probability of a concept at time t taking as input the word at position t and output of the previous hidden layer at time $t-1$. While, the Jordan-type RNN models the posterior probability taking as input the word at position t and the output of the previous layer.

3.1 Validation set

Before starting the training and the testing of the RNNs, I created a validation set to be used during the training phase in order to avoid overfitting the training set. The validation set is created by taking the 20% of the total sentences in the training set randomly.

3.2 Test cases

The code for the two RNNs has been provided by Dr. Ali Orkan Bayer. With this tool, to train and test RNNs of Elman and Jordan types, there is a configuration file that allows to change:

- **lr:** learning rate
- **win:** number of words in the window
- **bs:** batch size

- **nhidden:** number of hidden units in the hidden layer
- **seed:** the random seed used to shuffle the data
- **emb_dimension:** dimension of word embedding space
- **nepochs:** number of epochs to execute

In the following tables, the different files of configuration are shown:

Config0
lr: 0.1
win: 9
bs: 5
nhidden: 100
seed: 3842845
emb_dimension: 100
nepochs: 25

Table 10: Configuration file: config0

Config1
lr: 0.1
win: 9
bs: 5
nhidden: 200
seed: 3842845
emb_dimension: 100
nepochs: 25

Table 11: Configuration file: config1

Config2
lr: 0.1
win: 9
bs: 5
nhidden: 100
seed: 3842845
emb_dimension: 50
nepochs: 25

Table 12: Configuration file: config2

In the beginning, the tests were done by using the basic dataset, which only contains the words. Subsequently, I concatenated them to the basic set different features as it has been done for the midterm project:

Config3
lr: 0.2
win: 9
bs: 5
nhidden: 100
seed: 3842845
emb_dimension: 50
nepochs: 25

Table 13: Configuration file: config3

- *word_lemma*: to each word in the dataset is associated the corresponding lemma
- *lemma_pos*: the corresponding lemma is associated to each word in the dataset
- *word_pos*: creates a concatenation between each word with the POS tag
- *word_lemma_pos*: all the features are concatenated and used for the training

3.2.1 Elman-type RNN

The following tables show the results obtained with different training set using the four configuration files described in 3.2 for the Elman-type RNNs.

Config	Best Result	Test set performance
0	81.63	**79.27**
1	**82.34**	77.50
2	79.27	75.69
3	80.12	78.80

Table 14: Results with *words*

Config	Best Result	Test set performance
0	80.77	**79.76**
1	**82.66**	77.90
2	81.89	78.80
3	74.86	71.98

Table 15: Results with *word_lemma*

3.2.2 Jordan-type RNN

As in the previous section, in the following tables results for Jordan-type RNNs are described highlighting the best.

Config	Best Result	Test set performance
0	78.66	73.94
1	**79.84**	**75.64**
2	76.33	73.73
3	74.63	70.16

Table 16: Results with *lemma_pos*

Config	Best Result	Test set performance
0	78.82	76.23
1	**80.76**	**77.16**
2	78.5	73.68
3	74.17	71.56

Table 17: Results with *word_pos*

Config	Best Result	Test set performance
0	76.76	73.38
1	**78.20**	**76.35**
2	78.07	75.33
3	73.60	69.77

Table 18: Results with *word_lemma_pos*

Config	Best Result	Test set performance
0	82.11	79.02
1	**82.35**	78.56
2	**82.35**	**79.54**
3	76.81	71.68

Table 19: Results with *words*

Config	Best Result	Test set performance
0	80.58	75.30
1	**82.69**	78.94
2	80.79	**79.34**
3	77.31	72.87

Table 20: Results with *word_lemma*

Config	Best Result	Test set performance
0	79.68	**78.21**
1	**80.34**	77.16
2	78.99	76.08
3	75.36	72.50

Table 21: Results with *lemma_pos*

4 Conclusions

During the mid-term and the final projects, it was possible to compare two different kind of approach to model Spoken Language Understanding: the generative and the discriminative. With the generative approach several Weighted Finite-State Transducer were created and evaluated, while for the discriminative one Conditional Random Fields and Recurrent Neural Network have been used. As explained in the report of the mid-term project, the best result was obtained with the transducer that associates each word in the dataset with an IOB tag using an Absolute smoothing of order 2 or Witten_bell smoothing with order 2. With the CRFs, the best result was obtained using all the features taken in consideration (word, lemma, POS, prefix and suffix). Considering the test set performance, the Elman-type RNN gave the best result using the Config0 10 and *word_lemma*, while with the Jordan-type was obtained using Config2 12 and the set without features.

Config	Best Result	Test set performance
0	**80.19**	**78.70**
1	79.81	77.18
2	79.27	77.63
3	75.70	72.55

Table 22: Results with *word_pos*

Config	Best Result	Test set performance
0	**79.78**	**78.29**
1	79.14	76.81
2	79.56	78.14
3	74.00	73.18

Table 23: Results with *word_lemma_pos*

	Accuracy	Precision	Recall	FB1
WFST	92.69%	78.51%	74.34%	76.37
CRF	94.76%	87.91%	80.66%	84.13
Elman RNN	95.34%	81.39%	78.19%	79.76
Jordan RNN	95.06%	83.55%	75.89%	79.54

Table 24: Results with *word_lemma_pos*

As can be seen in the table above, after all the tests, the best result was obtained with the CRF tool.

References

Edwin Chen. 2015. CRF introduction to conditional random fields. http:

//blog.echen.me/2012/01/03/introduction-to-conditional-random-fields/.

CRF. 2015. CRF++: Yet another crf toolkit. https://taku910.github.io/crfpp/.

Wildm. 2015. RNN recurrent neural networks tutorial. http://www.wildml.com/.