University of Trento

Department of Information Engineering and Computer Science

Language Understanding Systems

first project

Roberto Zen

April 21, 2015

Chapter 1

Introduction and Data Analysis

The report is structured as follows. In this chapter a data analysis of the given dataset is done. Chapter 2 describes how I used the *FST* and *GRM* tools for training and testing sequence labeling based on POS-Tagging. In addition, it shows the results of applying these tools with different Language Model (LM) parameters. Chapter 3 describes as Chapter 2 sequence labeling and results but using the *CRF++* tool instead. Chapter 4 shows how text classification is made using *Naïve Bayes*. The last section states the results and the conclusion of the conducted work.

The given data set is composed in train and test files. The train files are usually bigger than the test files. More precisely, the train file used in the sequence labeling with FST have 21453 tokens and 3337 sentences. Both the train and the set files contain lemmas which are tokens that might be used in order to work with a grammar less complex (e.g different verb forms are treated as the same). For the sequence labeling in CRF, the train set is composed by 21453 words and 3338 phrases. For the text classification part the train set is composed by 3338 sentences while the test set is composed by 1084. Figure shows train and test set for the sequence labeling in CRF.

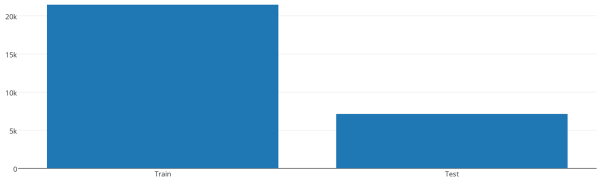

Figure 1.1: The word count for the train and the test dataset.

For each part of this work (FST, CRF, *Naïve Bayes*), a tool named **CoNLL**[1] has been used to evaluate the results of conducted tests. The level of parameters such as accuracy, precision and recall are used to compare LMs which might use different feature sets or tool configurations.

[1] `https://github.com/tpeng/npchunker/blob/master/conlleval.pl`

Chapter 2

Sequence labeling with FST

The LM I trained and tested for this part is based on *Finite State Transducers* (FST). In fact, it is based on a composition of them. For each sentence that belongs to the test file[1], I built a FST in which each transition is labbeled with a pair word-word. Figure 2.1 shows the FST built for the sentence "who plays luke on star wars".

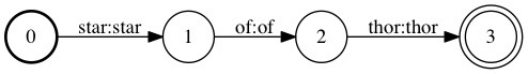

Figure 2.1: The FST built for a sentence that belongs to the train set.

Each of these FSTs is then composed with another one, in which there is only one final state and transitions that loop in it. Each transition is made by a pair word-tag and a weigth which is calculated as reported in the following expression. Note that unknown words has been considered as well.

$$W_{ML}(w_i|tag_j) = \begin{cases} -\log\left(\frac{C(w_i|tag_j)}{C(tag_j)}\right) & \text{if } w_i \text{ is a knwon word} \\ -\log\left(\frac{1}{|tagj|}\right) = -\log\left(\frac{1}{49}\right) & \text{if } w_i \text{ is an unknown word} \end{cases}$$

For the train part, I tried different approaches. First of all, I created the lexicon file based on the words, the tokens and also the lemmas written in the train file. Afterwards, I trained the model first considering lemmas as knonw words and then not. I got better results testing the model in the second case. Then, I experimented various train models with different ngramcount complexity orders. I started from the option **–order** set to value 1 and then I sequentially incremented it up to 6. Finally I tested the LM for each model configuration. Results for each parameter are shown in Table 2.1.

	accuracy	precision	recall	FB1
simple unigram-based POS-Tagger	84.91%	83.20%	84.61%	83.90
ngramcount –order=1	91.26%	89.97%	89.58%	89.77
ngramcount –order=2	93.37%	91.98%	92.45%	92.21
ngramcount –order=3	93.80%	92.51%	93.04%	92.77
ngramcount –order=4	94.13%	92.87%	93.29%	93.08
ngramcount –order=5	93.97%	92.65%	93.07%	92.86
ngramcount –order=6	93.97%	92.77%	93.01%	92.89

Table 2.1: Test results with FST.

[1] **NLSPARQL.test.feats.txt**

The values shown in Table 2.1 are depicted in Figure 2.2. As can be seen, although from order 1 to 2 there is a huge gap, the level of accuracy and of all the other parameters seems to stabilize with an ngram count value greather equal than 3. Hence, when the model counts trigrams or bigger n-grams. This mean that training the model with bigrams rather than unigrams we reach a very high level of accuracy. However, we reached better results training the LM with trigrams and ngrams.

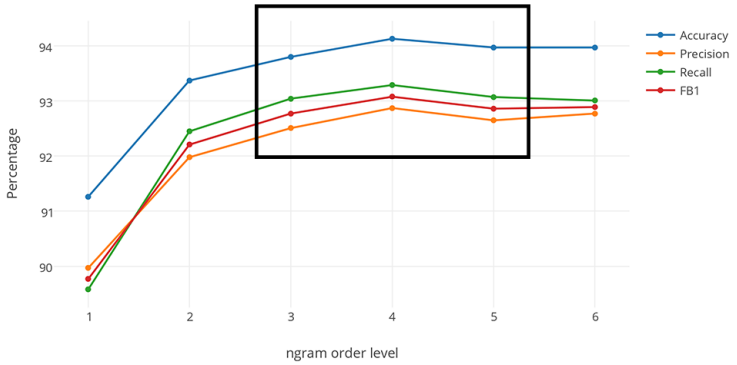

Figure 2.2: Test results of the LM which use different ngram count order's levels.

As conclusion, I think that training the LM using trigrams rather than complex n-grams is the best choice because we reached a high precision and accuracy level maintaining the training task as simple as possible.

Chapter 3

Sequence labeling with CRF++

I used Conditional Random Fields (CRFs) for the sequence labeling starting from training a LM based on the train file[1] written in IOB notation. I trained the LM using various template definition, however only four templates which gave me better results are written in Table 3.5. These templates differ one each other for relevant reasons: the first two use only unigrams while the others use both unigrams and bigrams. Template A is the simplest one, its window has size 5 while Template B's window has size 7. Template C is defined as 7 unigrams and 3 bigrams while template D is the most complicated one, because it is made of 5 unigrams and 4 bigrams.

Table 3.1: Temp. A

U00:%x[-2,0]
U01:%x[-1,0]
U02:%x[0,0]
U03:%x[1,0]
U04:%x[2,0]

Table 3.2: Temp. B

U00:%x[-3,0]
U01:%x[-2,0]
U02:%x[-1,0]
U03:%x[0,0]
U04:%x[1,0]
U05:%x[2,0]
U06:%x[3,0]

Table 3.3: Temp. C

U00:%x[-2,0]
U01:%x[-1,0]
U02:%x[0,0]
U03:%x[1,0]
U04:%x[2,0]
U05:%x[-1,0]/%x[0,0]
U06:%x[0,0]/%x[1,0]
U07:%x[1,0]/%x[2,0]
B00:%x[-1,0]
B01:%x[0,0]
B02:%x[1,0]

Table 3.4: Temp. D

U00:%x[-1,0]
U01:%x[0,0]
U02:%x[1,0]
U03:%x[-1,0]/%x[0,0]
U04:%x[0,0]/%x[1,0]
U05:%x[1,0]/%x[2,0]
B00:%x[-2,0]
B01:%x[-1,0]
B02:%x[0,0]
B03:%x[1,0]
B04:%x[2,0]

Table 3.5: CRF Templates.

Regarding tool parameters, I trained the LM with different levels of *cut-off* using the *crf_learn* command with the -f option value set initially to 1, then 2 and finally 3. As can be seen in Table 3.6, I got better results with the default value[2].

	accuracy	precision	recall	FB1	#features
Template A	93.59%	74.74%	73.24%	73.98	304958
Template B	93.48%	78.01%	74.15%	76.03	399873
Template C	94.28%	84.79%	78.19%	81.35	8804873
Template D	94.34%	86.13%	78.55%	82.17	12974573

Table 3.6: Test results with CRF++.

[1] NLSPARQL.train.data

[2] As written in `http://taku910.github.io/crfpp`, the default cut-off value for crf_learn is 1.

As can be seen in Figure 3.1, there is an exponential increase of the number of feaures with respect to the level of precision.

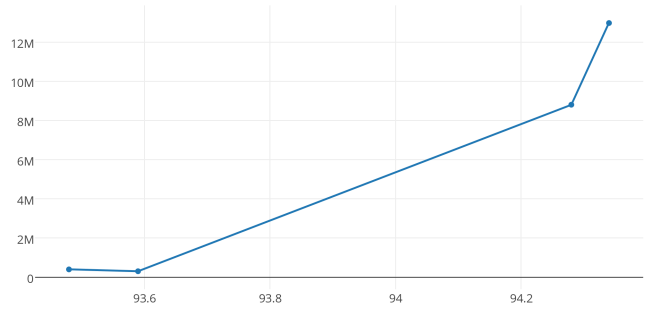

Figure 3.1: The exponential increase of the number of features.

As a consequence, to the best of my knowledge it is often more convenient to use simple models as the one defined by the Template C.

Chapter 4

Text classification with Naïve Bayes

The goal of the text classification part is to predict labels starting from sentences. I trained a Naïve Bayes LM based on Maximum Likelihood hypothesis. I tried the same LM starting first starting from the initial dataset and then replacing words by lemmas.

Both the train[1] and the test[2] dataset has been modified before starting to train the LM. In fact, in both files multiple labels occur for some sentences. As a consequence, bad situations might occur as follows.

1. During the training, the probability of having **multiple labels** for same sentences has to be treated in a differant way.

2. During the test, the LM predicts only one label for each sentence. So it is necessary to **check** if the predicted label is in the set of the expected ones.

In order to solve both the previous problems, I considered sentences that are labelled by more than once. I used a script which writes each sentence a number of times equal to the number of labels. Afterwards, I calculated the probability which can be expressed by the following expression.

$$P(w_i|tag_j) = \frac{C(w_i|tag_j)}{C(w_i)}$$

Once that the probability script execution is finished, I trained the LM as follows. For each sentence written in the train file, I predict its label usign the Maximum Likelihood hypothesis formulae shown in the followin expression.

$$h_{ML} = argmax_{h \in H} P(D|h)$$

The test results written in Table 4.1 refear to a test guided on a dataset in which there were no sentences labelled with multiple labels: the LM has been trained and tested with modified version of files as described above.

accuracy	precision	recall	FB1
72.14%	72.14%	72.14%	72.14

Table 4.1: Test results with Naïve Bayes using Maximum Likelihood hypothesis.

I first trained and then tested the same model using lemmas, so replacing words by lemmas but I got lower scores.

[1] NLSPARQL.train.tok
[2] NLSPARQL.test.tok

Conclusion

This report has described the first project of the Language Understanding Systems course. It is focused on sequence labeling and text classification of a dataset which refers to the movie domain. The sequence labeling part has been computed in two ways using respectively two different tools: the first one was based on Finite State Transducers (FST) while the second one was based on Conditional Random Fields (CRF++). The text classification part has been done using Naïve Bayes with Maximum Likelihood. Different train techniques has been experimented and test results for each of these has been reported. As result of the sequence labeling with FST, the high score I got is 94.13% of accuracy and 92.87% of precision. To reach these values I used a LM trained without lemmas and with a maximum n-gram order count equal to 4. With CRF++ the highest score I reached is 94.34% of accuracy and 86.13% of precision (Template D). However, these values came with 12974573 features, which is very high respect to 8804873 features which allowd me to reach 94.28% of accuracy and 84.79% of precision (Template C). In the text classification part, using based on Maximum Likelihood hypothesis I got 72.14% both on accuracy and precision.