University of Trento

Department of Information Engineering and Computer Science

Language Understanding System

FIRST PROJECT

Zamboni Luca

April 21, 2015

1 Sequence lebeling

In this section will be analyzed and discussed the performance of 2 different methods for the sequence lebeling task. Final state transducer (FST) based on POS-Tagging and Conditional random fields (CRF) based on IOB notation.

1.1 FST

1.1.1 Data analysis

The training set and the test set are composed with 3 columns: token POS-tag lemma. The token is a word of the sentence with his relative POS-tag and his lemma. Every sentence is separated with a empty line. Token are 21453 for a total of 3337 sentences. Lemmas are a generalization of the relative words. For example the plural and the singular of a given word have the same lemma. The distribution of POS-tags among the dataset is described in Figure 1. As we can see POS-tags are not equally distributed and there are few tags that take almost all probability.

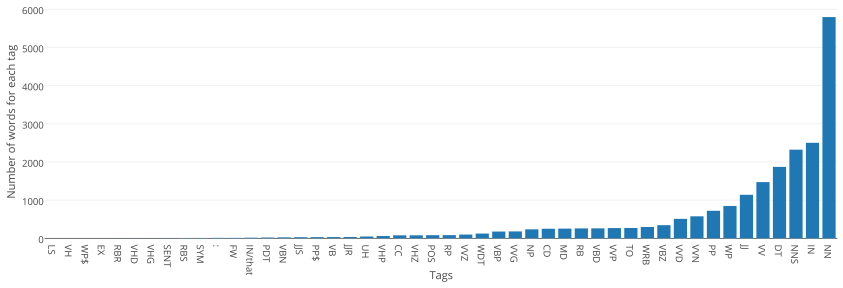

Figure 1: Bar chart of the distribution of number of words for each tag.

1.1.2 Tools

The tools used for the train and test:

- OpenFst used to create and compose fsts for unigram
- OpenGrm used to automatically train models with n-grams

1.1.3 Training

First of all I created a file with the following structure

$$word \qquad tag \qquad -log(P(w_i|t_i))) \tag{1}$$

Where $P(w_i|t_i)$ is the probability of a word given a tag and is computed with the following formula $\frac{C(w_i,t_i)}{C(t_i)}$. $C(w_i,t_i)$ is the count of the word with a particular tag and $C(t_i)$ is the numer of appearance of a tag in the training test.

I used minus log of the probability because in this way higer probability take a lower number than minus log of a a low probability and this will be needed later. For each tag is added a line that take in consideration unknown.

$$< unk > \qquad tag \qquad \frac{1}{|tag|} \tag{2}$$

Where $|tag| = 49$. Then I created the transducer for the model in the following way:

$$\begin{array}{cccc} 0 & 0 & word & tag & -log(P(w_i|t_i)) \\ & & & ... & \\ 0 & 0 & word & tag & -log(P(w_i|t_i)) \end{array}$$

This transducer is our model based on unigrams. Now we want consider also the probability of a tag given the predecessor. To do this I used the Open-grm tool that automatically creates the model of n-grams of token. The command that I used are the following:

farcompilestrings –unknown_symbol='<unk>' input.txt > far.far
ngramcount –order=N –require_symbols=false far.far > input.cnt
ngrammake –method=witten_bell input.cnt > ngrammodel.lm

The final output is a fst that rappresent out model of n-gram POS-tag. Now that I trained my model I can test it.

1.1.4 How to test

To test a sentence I created a fst for each sentence in this way:

$$\begin{array}{cccc} 0 & 1 & word1 & word1 \\ 1 & 2 & word2 & word2 \\ & & ... & \\ (N-1) & N & wordN & wordN \end{array}$$

If a word is not present in the train file it is substituted with '<unk>'. Now this fst is composed with the fst that represent our unigram-tag model.

fstcompose input-sentence.fst unigram-model.fst > word-tagged-unigram.fst

Now we have a simple tagger based on unigrams. To use also the model based on n-gram of tags we just compose the previous tagger with this.

fstcompose simple-tagger.fst n-gram-model.fst > word-tagged-n-gram.fst

And after to get the best tagger we just run the command fstshortestpath.

We can use the command shortest path because is not the real probability but the minus log o probability and in this way with this command we get the most probable sequence of tag. All command togheter in order to get a sequance of POS-tag of a sentence:

fstcompose input-sentence.fst unigram-model.fst > simple-tagger.fst
fstcompose simple-tagger.fst n-gram-model.fst > word-tagged-n-gram.fst
fstrmepsilon word-tagged-n-gram.fst | fstshortestpath > final-tag.fst

The result is a fst that take as input a sentence and gives the best possible sequence of POS-tag like the one in Figure 2

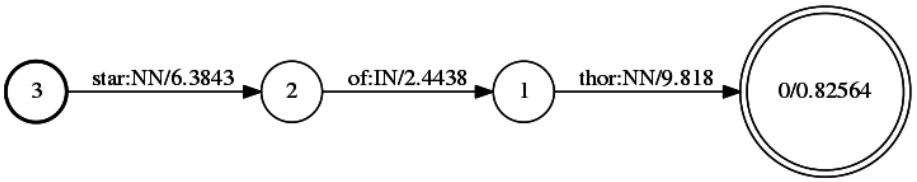

Figure 2: Final result of POS-tagger

1.1.5 Results of testing

The test set is structured in the same way of the training set. This set has 7117 words each with the expected POS-tag. First I runned the test set with a simple tagger without n-grams of tag and after with. I also runned a train and test with lemmas. To evaluate the test was used a script in perl conlleval.pl.

	Accuracy	**Precision**	**Recall**	**FB1 Score**
Unigram word POS-tag	84.91%	83.20%	84.61%	83.90
2-gram POS-tag	93.37%	91.98%	92.45%	92.21
3-gram POS-tag	93.80%	92.51%	93.04%	92.77
4-gram POS-tag	94.13%	92.87%	93.29%	93.08
5-gram POS-tag	93.97%	92.65%	93.07%	92.86
3-gram with lemma	78.22%	75.80%	74.09%	74.94

Table 1: Result of test

The best result is in the model based on 4-gram. As we can see in the table, the simple POS-tagger is less precise of the tagger based on n-gram. This because using we do not consider only probability as word but also the previous word and the previous tag. As we can se from the table if we train our model considering also the lemma accuracy goes down a lot this because maybe in enghlish lemma doesn't count too much. Perhaps they can be useful in other language like Italian where the grammar is more complex. We conclude that models based on 4-grams are the best in term of accuracy but they require much more computational power than the simpler ones. If we want more accuracy we can use models based on 4-grams if we want performance we can use models based on 2-grams because the difference of accuracy is lower than 1%.

1.2 CRF

1.2.1 Data analysis

The training set is composed with 2 column: one for the token and the other with the relative tags in IOB-notation. Words are 21453 for a total of 3338 phrases. The distribution of the tags is described in the Figure 3.

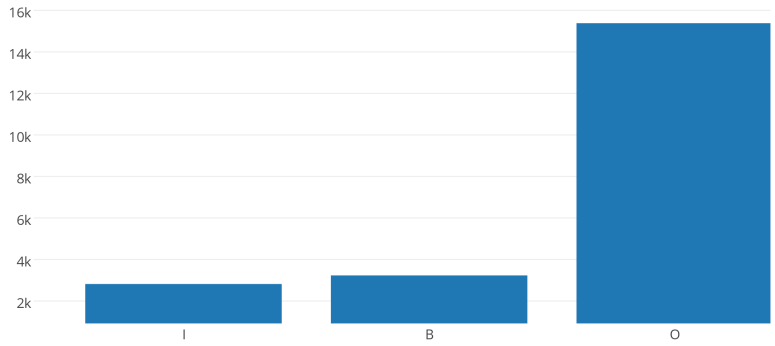

Figure 3: Distribution of I O B.

1.2.2 Tool

The tools used for the train id CRF++. It is a tollkit to automatically train and test conditional random fields.

1.2.3 Training

To train with this tool we need a template for feature extracting. An example of template is:

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

With this template you consider 3 words at time as feature.
The command to train is

crf_learn template_file train_file model_file

The model_file is our trained model in output.

1.2.4 Testing

To test we must simply run the command

crf_test -m model_file test_files

It outputs a file with token, expected tag and tag predicted by the model. After that to calculate statistics I used a perl file conneval.pl that calculate automatically: accuracy, precision, recall and FB1 score. I runned different train with different template file and result are described in the following table with relative template below. As we can see from the tables template 4 is the one

	Accuracy	**Precision**	**Recall**	**FB1 Score**
Template 1	87.59%	44.99%	60.49%	51.60
Template 2	93.59%	74.74%	73.24%	73.98
Template 3	93.76%	76.97%	74.43%	75.68
Template 4	94.23%	86.35%	78.28%	82.12

Table 2: Result of test

Template 1	**Template 2**	**Template 3**	**Template 4**
U02:%x[0,0]	U00:%x[-2,0]	U00:%x[-2,0]	U00:%x[-2,0]
	U01:%x[-1,0]	U01:%x[-1,0]	U01:%x[-1,0]
	U02:%x[0,0]	U02:%x[0,0]	U02:%x[0,0]
	U03:%x[1,0]	U03:%x[1,0]	U03:%x[1,0]
	U04:%x[2,0]	U04:%x[2,0]	U04:%x[2,0]
		U05:%x[-1,0]/%x[0,0]	U05:%x[-1,0]/%x[0,0]
		U06:%x[0,0]/%x[1,0]	U06:%x[0,0]/%x[1,0]
			B00:%x[-2,0]
			B01:%x[-1,0]
			B02:%x[0,0]
			B03:%x[1,0]
			B04:%x[2,0]

with higher performance. Template 1 and 2 are very semple and they have a very little window. Template 3 and 4 have a template more complex and they have a similar accuracy but Template 4 have a greater FB1 Score. In conclusion I can say that template 4 is the one with best performance because it is more complex because it consider also bigrams.

2 Text classification

2.1 Naive Bayes

2.1.1 Goal

Train a model with sentences and relative tag in order to output a tag given a sentence.

2.1.2 Data analysis

The training set is composed by sencences with relative tag. Sencences are 3338. The distribution of tags is described in Figure 4. As we can see "movie is the tags with more probability.

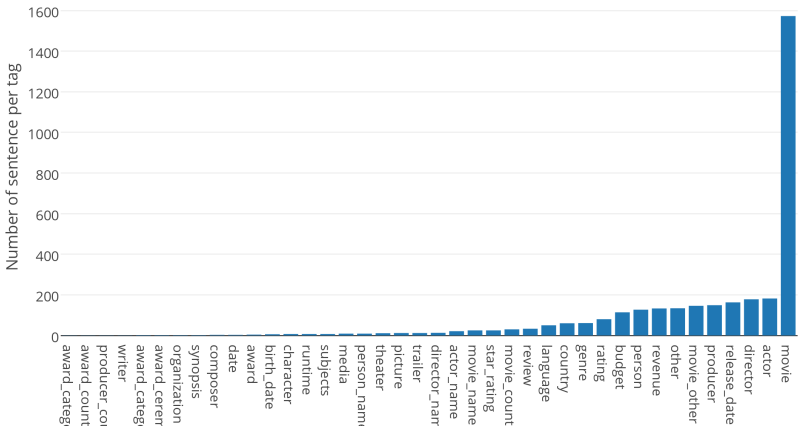

Figure 4: Distribution of tags.

2.1.3 Implementation

Naive bayes algorithm is very simple. It uses the bayes rule

$$P(tag|D) = \frac{P(D|tag)P(tag)}{P(D)}$$

From this we can build a naive Bayes classifier with the following formula:

$$tagOutput = \arg\max_{tag \in D} P(tag) \prod_{i=1}^{N} P(w_i|tag)$$

Where $P(w_i|tag)$ is the probability that the $word_i$ have that particular tag. To compute this probability we use the following formula $\frac{C(w_i|tag)}{C(w_i)}$. Where $C(w_i|tag)$ is the count of the word that appers with a particular tag and $C(w_i)$ is the count of that word in the dataset.

2.1.4 Testing

I did some test. First one with a simple naive classifier. After considering also bigrams of words as unigrams like $word_i\_word_i + 1$. In this way the phrase "star of thor" becomes the set $\{star, of, thor, start\_of, of_thor\}$. This way also for the trigrams. It has been considered also POS-tag and removing english stop word. For the smoothing for a unknown word I used simple +1 on every count. I have

	Stop word	POS-Tag	Accuracy	Precision	Recall	FB1
S	Yes	No	66.51%	66.51%	66.51%	66.51
S B	Yes	No	79.52%	79.52%	79.52%	79.52
S T	Yes	No	78.69%	79.13%	78.69%	78.91
S	No	No	67.34%	67.34%	67.34%	67.34
S B	No	No	71.86%	71.86%	71.86%	71.86
S T	No	No	71.96%	71.96%	71.96%	71.96
S	Yes	Yes	68.73%	68.73%	68.73%	68.73
S B	Yes	Yes	78.23%	78.30%	78.23%	78.26
S T	Yes	Yes	77.86%	78.37%	77.86%	78.11
S	No	Yes	68.54%	68.54%	68.54%	68.54
S B	No	Yes	72.88%	72.88%	72.88%	72.88
S T	No	Yes	72.97%	72.97%	72.97%	72.97
S NP	Yes	No	69.10%	69.10%	69.10%	69.10
S B NP	Yes	No	79.43%	79.43%	79.43%	79.43

Table 3: S = Simple , B = Bigrams , T = Tringrams , NP = No Prior

done some test also not considering the prior. The result are described in the table 3. As we can see the best model is based on Simple naive bayes classifier considering also bigrams of 2 word of the sentence. The results shows that if we remove stop word Accuracy gows down and if we consider also pos tag as feature the result doesn't change a lot. Instead bigrams and trigrams gives a lot of help to get the real tag, in fact accuracy grow by a $\sim 10\%$. Not considering the prior does'n change a lot.