Report of Final project of ML, Out-of-Domain PoS tagging

Xinneng XU et Wei HUANG

April 13, 2019

Abstract

The goal of this project is to design a PoS tagging which can well solve the problem of Out-of-Domain PoS tagging. And evaluate this PoS tagging on the different combination of train and test sets by considering: 1.its precision over the whole test set, 2.this precision over the ambiguous words, 3.its precision on Out-of-Vocabulary(OOV) words.For now, the PoS tagger we designed gives a 95.53% accuracy on all the test dataset of corpus 'fr.ftb'. Finally, we implement and evaluate the different classifiers HMM Tagging which gives a 94.63% accuracy.

Keywords: Part-of-Speech; Out-of-Domain PoS tagging; Machine Learning; NLP; Classifier; HMM

1 Introduction

In traditionnal grammar, a PoS is a category of words which have similar grammatical properties. PoS tagging can be considered as a multi-class classification problem, and with very simple features achieve human-comparable performance. However, the PoS tagger performance decrease with the increasing of the percentage of sentences that depart from training dataset. We aim to evaluate the impact of changes in domain and develop the features and models that are robust to changes in domain.

In this report we describe a little about the different french corpus we used in our work in the aspects of the size of the different of the train and test sets, the differences between UGC corpora (denoted as 'in-domain' in thefollowing) and 'canonical' corpora (denoted as 'out-domain' in the following) etc. Then we describe the Perceptron, a part-of-speech (POS) tagger which uses the principal notion of Multi-Classification and the classical Hidden Markov Model (HMM) tagger which is described in (Charniak, Hendrickson... 1993). After that we describe the features we extracted from the corpus, such the suffix of the token, the shape features. Finally we show the evaluations of the different classifiers and describe the results obtained of our experiments.

2 Experimental french corpus

In this section, we describe the similarity between the different datasets and the differences the train and test sets.

There are 6 different corpus in our sources : 1.fr.ftb, 2.fr.gsd, 3.fr.partut, 4.fr.pud, 5.fr.sequoia, 6.fr.spoken. These corpus are given by the professor Guillaume, we find that perhaps they are come from the french newspaper or journal.

2.1 Size of Corpus

The table 1 shows the size (number of sentences and words) of the different of the train and test set. In which the number of sentences and words in corpus 'fr.ftb' is the largest, 14759 and 44228 respectively in train_set.

Table 1: The size of the different corpus

	train_set		test_set	
Corpus	#Sentences	#Words	#Sentences	#Words
fr.ftb	14759	442228	2541	75073
fr.gsd	14450	345009	416	9742
fr.partut	803	23324	110	2515
fr.pud	803	23324	1000	24138
fr.sequoia	2231	49173	456	9740
fr.spoken	1153	14952	726	10010

2.2 Differences between UGC and Canonical corpus

In this subsection, we focus on the noisiness of a corpus (or the similarity between train and test set). **Why?** We interested in the similarity because if we get a very good precision in the corpus which has large similarity between train and test set, this can not indicate that our model(or algorithm) is good. Thinking that if we use the train set as the test set, obviously we can get a good precision, but perhaps the model hasn't a good precision on test set which is very different from train set. In contrast, if the model execute well in test set which is different from train set, we can mostly get that the model is good enough. The noisiness of a corpus can be measured in the 3 following methods.

• The percentage of Out-of-Vocabulary (OOV) words (words appearing in the test set that are not contained on the train set). Why? We interested in oov because in general, the word which is not in train set can not be learned, as a consequence, the model has less probability to predict the tag correctly. So normally the more oov exist, the less accuracy we get. In contrast, the less oov in the test set, the better accuracy we get, but we if the In the section of Experimental results we will show the performance of the classifiers in the data set of oov.

• The KL divergence of 3-grams characters distributions estimated on the train and test sets (Hector M, Djame S,... 2016). TODO: explain KL

• Perplexity on the test set of a (word level) Language Model estimated on the test set. TODO: explain

Table 2: The result of 3 metrics

Corpus	#OOV	P(OOV)	KL	Perplexity
fr.ftb	2529	21.06%	TODO	TODO
fr.gsd	576	17.84%	TODO	TODO
fr.partut	293	27.73%	TODO	TODO
fr.pud	**6540**	**72.56%**	TODO	TODO
fr.sequoia	899	29.12%	TODO	TODO
fr.spoken	1783	60.51%	TODO	TODO

• **Metric.** The table 2 show the different metric for the different combination of train and test sets. (*i*) We can see that the corpus 'fr.pud', has the largest number of word which is in test set but not in train set. And the oov words has a large percentage in all test set. This means for the corpus 'fr.pud', it has the less similarity and similarly the corpus 'fr.spoken'. But for corpus 'fr.gsd', it has the most similarity. (*ii*) in the column KL we can get that the corpus 'fr.partut' has the largest KL which means that it has the lest similarity of the composition of words between train set and test set. Oppositely the corpus 'fr.ftb' has the smallest KL so it has a most similarity of the composition of words between train set and test set. (*iii*) TODO

3 Perceptron

In this section we describe the principal of the Perceptron which is a multiple class classifier actually.

3.1 Features extraction

In this subsection we talk about the features extraction of a word with its context for the input for Perceptron. Note that for the feature name fn and the feature value corresponding fnv_i of a word w_i, all feature values used by Perceptron will be defined by $fn +' \_' + fnv_i$ if w_i has fnv_i, do nothing if not. As showed in figure 1, is all the feature values extracted by the following methods for the word 'aller'.

Window features. The tag of a word usually has a relation with the word last and following. For example, the word 'de', if it follows by the word 'mecanismes' whose tag is 'NOUN', its tag will be 'ADP', but if it follow by the word 'imposer' whose tag is 'VERB', its tag will be 'DET'. The word following impact the tag analogously. In addition, the word two previous also has influence to this tag. However, if we choose to many words previous or following, this

will occupy to much memory, as a result, in the experiment, for the word w_i in the its context we use a window of size $l = 2$ around the word w_i : $(w_i - l, \ldots, w_i, \ldots, w_i + l)$ to extract the window features for the word w_i.

Suffix features. The suffixs of a word is useful because basic morphology rules are the same in different domains. In the experiment, we use the suffix in a length of $l = 3$ of a word w_i : $(w_i[-l:], \ldots, w_i[-1:])$.

Shape featuers. Each word is mapped to a bit string encompassing 10 binary indicators that correspond to different orthographic (e.g., does the word contains a digit, hyphen, capital character, starts with capital character, only has capital characters, has digit, is the word composed not only by alpha or digit) and morphological (e.g., does the word end in -ment or -tion) features. We note that the shape features we used are for the French and perhaps not really the same for English.

Distributional features. Firstly for the word w_i we calculate (i) its left neighbors with the counts of left neighbors, (ii) its right neighbors with the counts of right neighbors. Then based on the these, TODO: WEI will write it.

```
['aller',
 'win_i-1pour',
 'win_i+1du',
 'win_i-2bon',
 'win_i+2CRDT',
 '-1-th_suffix_r',
 '-2-th_suffix_er',
 '-3-th_suffix_ler',
 '0-th_freq_left_pour',
 "1-th_freq_left_d'",
 '2-th_freq_left_y',
 '3-th_freq_left_aimé',
 '4-th_freq_left_pu',
 '5-th_freq_left_plus',
 '6-th_freq_left_pas',
 '7-th_freq_left_là',
 '8-th_freq_left_toujours',
 '9-th_freq_left_un'],
```

Figure 1: all feature values for word 'aller'

3.2 notion of multiple class and Perceptron

• Each observation x is associated to a label $y \in \mathcal{Y}$ where $\mathcal{Y}$ = finite of set of all possible labels in train set.

• Then we define the 0/1 loss for the multi-class loss :

$$l^{multi}(y_1, y_2) = \begin{cases} 0 & if\ y_1 == y_2 \\ 1 & otherwise \end{cases} \tag{1}$$

• Decision function. The most common we do is one weight vector u_y for each label y. By this we get the decision function :

$$y^* = argmax_{y \in \mathcal{Y}}(x * u_y) \tag{2}$$

We multiply the observation x with each weight vector u_y of each label y, and then we choose the label y which gives the largest product. However, if we use one-hot encoding for the features mentioned above, the feature matrix will be very sparse (too many 0 and little 1), this will take many memory and not easy to train the model.

For Perceptron : So here we use the feature encoding mentioned in **3.1**. And each feature value gets its own weight vector with one weight for each possible label. As showed in figure 2 is the weights for feature value 'win_i-1pour', in which the feature name is 'win_i-1' and the value for a word is 'pour'. In the figure, all possible labels associated with feature value 'win_i-1pour' has a weight, there are the labels not in the dictionary has the weight value 0.

```
'win_i-1pour': defaultdict(float,
                {'ADJ': -0.499,
                 'VERB': 1.531,
                 'SCONJ': 0.699,
                 'PRON': -1.913,
                 'ADV': 0.704,
                 'NUM': -0.704,
                 'DET': 1.244,
                 'ADP': -1.0,
                 'PROPN': -0.061}),
```

Figure 2: weights of feature 'win_i-1pour'

We suppose U as the dictionary of weights for all feature values FV, for word w_i, we can get all the feature values FV_i. For each feature value $fv \in FV_i$, we can get the weight vector $u_{fv} = U[fv]$, in which, for each label $y \in \mathcal{Y}$ associated with a weight $u_y = u_{fv}[y]$. The decision function will be:

$$y^* = argmax_{y \in \mathcal{Y}} \sum_{fv \in FV_i} u_{fv}[y] = argmax_{y \in \mathcal{Y}} \sum_{fv \in FV_i} U[fv][y] \tag{3}$$

• Update rule. If $l^{multi}(y^*, truth) = 0$ then we do nothing and if $l^{multi}(y^*, truth) = 1$ then for each feature value $fv \in FV_i$ update the weights:

$$\forall fv \in FV_i,\ U[fv][y^*] -= 1;\ U[fv][truth] += 1 \tag{4}$$

• Training and Prediction. We use the same feature extraction in the train set of a corpus (not all corpus) and the same decision function as the training mentioned before. However, the difference between training and prediction is that in training we find the prediction is not correct, we use the update rule to update the weights for reduce the loss, but when we in prediction, we just return the prediction predicted by the model and do nothing about the update.

4 HMM

Just as tomorrow's weather condition has a probability to be influenced by today's weather condition, the PoS tag of word also has a probability to be influenced by the PoS tag of last word. In this project, the state-of-the-art statistical tagger is trained : Hidden Markov Model (HMM) tagger (Charniak, Hendrickson... 1993).

4.1 Notation

Giving a sentence of words $W = (w_i, \ldots, w_i, \ldots, w_n)$, we aim to find a chain of labels (tags) $Y = (y_1, \ldots, y_i, \ldots, y_n)$, with each y_i associated with a word w_i, which can maximiser $P(Y|W)$: $Y^* = \arg\max_Y P(Y|W)$. Thinks to Bayesian formula, $Y^* = \arg\max_Y P(W|Y) * P(Y)$. The HMM model will make predictions according to formula :

$$Y^* = \underset{Y, y_i \in Y, w_i \in W}{\arg\max} \prod_{i=1}^{n} P(w_i|y_i) * P(y_i|y_{i-1}) \tag{5}$$

In the formula, $P(w_i|y_i)$ shows that the word w_i present depends on the its possible associated label.$P(y_i|y_{i-1})$ shows that the current label type depends on which its last label was.

4.2 HMM tagger

In this project we use the HiddenMarkovModel in the Pomegranate library to build a hidden Markov model for the PoS tagging.

The state. Suppose $\mathcal{W}$ as the set of all possible word in the train set and $\mathcal{Y}$ as the set of all possible labels in train set. Each hidden state $y \in \mathcal{Y}$ gives the conditional probability $P(w|y)$ of observing a given word $w \in \mathcal{W}$. The emission distribution for the word w of the state y is :

$$P(w|y) = \frac{Count(y, w)}{Count(y)}$$

where $Count(y, w)$ is the number of pair of word w and its label y, $Count(y)$ is the number of appearance of label y.

The start and end transition. We also have to estimate the starting probability distribution (the probability of each label being the first label of a

sentence) and the ending probability distribution (the probability of each label being the last label of a sentence) from the train set :

$$P(y|start) = \frac{Count(start, y)}{Count(start)}$$

$$P(y|end) = \frac{Count(end, y)}{Count(end)}$$

where $Count(start, y)$ is the number of the label y being the start of a sentence and $Count(start)$ is the number of sentences, similar in the case of end.

The transition between two state. The transition probability $P(y_i|y_{i-1})$ gives a the conditional probability of transition from label y_{i-1} to y_i. From the train set we can learn the probabilities of all transitions from all state $y_i \in \mathcal{Y}$ to each state y_j :

$$P(y_j|y_i) = \frac{Count(y_i, y_j)}{Count(y_i)}$$

where $Count(y_i, y_j)$ is count of pair (y_i, y_j).

• Training and Prediction. We estimate all the terms $P(w|y)$, $P(y|start)$, $P(y|end)$, $P(y_j|y_i)$ based on a training corpus which is a set of couples (W, Y) but not from all the corpus. We don't use any data in the test set. After all the terms estimated, we build the HiddenMarkovModel by adding all the states and transition mentioned before, and finally we use the model built to predict a sentence of labels for a sentence of input words.

5 Evaluation

The experimental results of classifier Perceptron and HMM in the six corpus are presented in the Table 3 . The precision over the whole test set (i.e. the percentage of labels that have been correctly predicted) is presented in the column 'ALL'. The precision over the ambiguous words (i.e. the precision computed only on words that appear with more than one label in the train set) is presented in the column 'Ambiguous'. And the precision over OOV is presented in the column 'OOV'

Table 3: The results

	Perceptron			HMM		
Corpus	ALL	Ambiguous	OOV	ALL	Ambiguous	OOV
fr.ftb	**95.53%**	95.14%	**82.87%**	**94.60%**	92.93%	48.24%
fr.gsd	94.85%	**95.61**%	81.84%	92.51%	93.82%	46.67%
fr.partut	92.36%	89.66%	72.72%	91.73%	91.90%	53.26%
fr.pud	84.43%	93.44%	67.60%	77.68%	91.90%	51.87%
fr.sequoia	92.61%	91.64%	75.19%	93.73%	**94.22%**	**57.82%**
fr.spoken	89.23%	90.97%	78.57%	86.54%	90.34%	51.10%

5.1 For Perceptron

We trained the Perceptron classifier on the training set. A word in the test set is classified by Perceptron by building the feature values using the same extraction of features as the train set.

Table 4: Ambiguous words

	fr.ftb	fr.gsd	fr.partut	fr.pud	fr.sequoia	fr.spoken
Ambiguous words	212787	151557	7243	7243	16096	4151
Total train words	442228	345009	23324	23324	49173	14952
Ambi/Total	48%	44%	31%	31%	33%	28%

• For ALL. From the results of precisions on ALL on all the corpus we can get the perceptron has a maximum precision 95.53% on the corpus 'fr.ftb', and then 95.85% on 'fr.gsd'. The precisions on the others are less accurate because the corpus 'fr.ftb' and 'fr.gsd' has much larger train data set than others and this gives much more training on the Perceptron.

5.2 For HMM

References