Chinese Event Extraction

Ping Xuanhuang 15307130283

December 8, 2017

Contents

1 Question Preface

The task is to use sequence labelling models to do 'Chinese event extraction'.

The efficiency of HMM will be influenced by the algorithm('Viterbi algorithm'/'Forward algorithm'), smoothing method, Markov model(bigram/trigram). So I separate the task into basic experiment and further experimental. In further experiment, I will apply different methods(model) on the model to see the difference.

Let's have a look at the evaluation index. Type_accuracy marks whether the Argument/Trigger label is right or not. Precision and recall is closely relate to the result when we labelled a word as O, so accuracy and precision(recall) are some kind conflicting: if I give less Argument/Trigger label, less wrong will make on Argument/Trigger label, but the result of precision and recall may decrease, as a result, F1, which related to recall and precision will decrease.

2 Basic Experiment

I will use bigram model and Viterbi algorithm to do the task.

In the Training phase(File: Train.second.py), by the conclusion of MLE, I can accomplish it by counting the number of 'label-label' pair... Then calculate the log-probability of each 'label-label' pair and 'label-word' pair. I simply use the 'Add α smoothing method', and set α as 0.002.

In the Labelling progress(File: Basement.py), I apply Viterbi algorithm on the model. I use two matrix. One records the maximum probability when a word is going to be labelled as some kind of tag; another one records the previous tag (the path) which contributes to the maximum probability .

I use the 'transmission matrix' and 'emission matrix' to calculate the maximum probability. Because not every word have appeared in the training set,

when come across unseen 'label-word' pair, I suppose the probability of the word to be labelled as any kind of label is equal (This hypothesis is very strong) In practice, I set the log-probability of this kind of pair as '0'.

Result as follow:

	Trigger	Argument
type_correct	0.967	0.4059
Precision	0.8154	0.7015
Recall	0.6642	0.5646
F1	0.7321	0.6257

3 Further experiment

Smoothing method

There exist two kind of 0-problem: the first appears during the training process, the second exists in the test process. In the basic experiment, I cope with both of these two problems by 'add α smoothing method'.

Test process 0 problem

Firstly, I tried to deal with the words which have never been seen. In previous experiment, I simply suppose 'the probability it belongs to each label is equal'. However, a more scientific thought is supposing 'the probability it belongs to each label is equal to the frequency of this label'. So I record the probability of each label in the training process (training.second_2.py) and store these probability as '$label_k$ - \$: $probability\ of\ label_k$ '(\$ marks the unseen word) in the emission matrix. Result as follow:

	Trigger	Argument
type_correct	0.967	0.4167
Precision	0.8154	0.722
Recall	0.6642	0.4714
F1	0.7321	0.5704

The result on Trigger has no change, because every words of Trigger have appeared in the training set. The recall of argument decreased, it means more words are labelled as O ,because O appears most frequently in the training set. And, of course, type_correct will increases, because less words are been labelled as argument, and the wrong made on it will decrease. Based on both theory and the result of type_correct, this smoothing method is more scientific than previous method. So in continuing part I will still use this method, and in subsequent part I will call this method as 'frequency method'.

Train process 0 problem

I tried to use good-Turing method to deal with the '0 problem' on train process: I use the number of one-occurrence word to indicate the occurrence times of 0 word. Result as follow:

	Trigger	Argument
type_correct	0.967	0.3438
Precision	0.8154	0.5909
Recall	0.6642	0.537
F1	0.7321	0.5626

The indicate occurrence frequency of 0-occurrence as follow:

	Trigger	Argument
Label-label	0.01351	0.02385
Label-word	0.05990	0.02755

The result is very poor, all data decreased seriously except Recall, so we can suppose the model has labelled much more 'Argument' . We could also find these indicate data is much bigger than 0.002, so it will change the original model easily, and we can say the data of training set is not sparse at all, which means that 'good-Turing method' is not appropriate in this case.

From above, in subsequent experiments, I will use 'add smoothing method' in training process and use 'frequency method' in test process.

Forward algorithm

I uses a matrix to record the probability of every labels when one word is going to be labelled as them, but there is no need for another matrix to store the path. And the probability of each label under each word changes from maximum probability to the sum of probability.Result as follow:

	Trigger	Argument
type_correct	0.5158	0.1483
Precision	0.1612	0.4024
Recall	0.2774	0.4115
F1	0.2039	0.4069

In this experiment, the result is variable, because in the last step we tried to find the tag with biggest probability under each word, but this kind of label may exist more than one. And Generally speaking, the result of 'Forward algorithm' is worse than 'Viterbi algorithm'.

In theory, Viterbi algorithm can find out the global best sequence, but it can also hinder the sequence which may generate a better result. But the forward algorithm maybe worse, in one hand, it is unstable, in another hand, the interpretability of this algorithm is bad.

Trigram model

First I trained the trigram model in the similar way of bigram model. However, because I will use 'backoff smoothing method', there is no need to deal with the '0 problem' in training progress.

In the labelling process, I used 'backoff smoothing method', it means if the 'label-label-label' pair doesn't exist in the transmission matrix, I will use bigram model. Because I use 'Viterbi algorithm' on this task, I set a structure:'step' to record the previous 2 steps of each label on a path. And I need to find the maximum probability based on the previous two label. Result as follow:

All the result on Trigger and the type_correct on Argument is better than ever before, it is because more information is available in trigram model. The decrease on Recall means more words are wrongly labelled as O. Considering the efficiency of 'frequency method'(more word will be labelled as O), I changed the smoothing method to the basic method(Trigram_2.py).

we could see both precision, recall and F1 on argument is better than before, because less wrong O are been labelled. But the type_correct decreased, because the information about the frequency of each label have lost.

	Trigger	Argument
type_correct	0.967	0.4221
Precision	0.8169	0.7149
Recall	0.6642	0.4577
F1	0.7327	0.5581

Figure1: Trigram(Frequency smoothing)

	Trigger	Argument
type_correct	0.967	0.4019
Precision	0.8169	0.6913
Recall	0.6642	0.5803
F1	0.7327	0.6309

Figure2: Not Frequency smoothing

4 CRF experiment

To acquire more futures, I firstly did the 'part of speech tagging'(Part of speech tagging.py) on all files. Then I test different ways to extract features, result as follow, more details could be find in (result.CRF.xlsx)

one-dimension						
	word:x[0,0]		x[-1,0]/x[0,0]		x[-1,0]/x[0,0]/x[1,0]	
	Trigger	Argument	Trigger	Argument	Trigger	Argument
type_correct	0.9783	0.5061	0.9796	0.5003	0.9798	0.5026
Precision	0.9529	0.7642	0.9526	0.7644	0.9547	0.7665
Recall	0.4039	0.3675	0.3582	0.314	0.3129	0.2602
F1	0.5673	0.4964	0.5207	0.4451	0.4713	0.3885
	Part of speech:x[0,1]		x[0,1]/x[-1,1]		x[0,1]/x[-1,1]/x[1,1]	
	Trigger	Argument	Trigger	Argument	Trigger	Argument
type_correct	0.8489	0.4144	0.7712	0.3986	0.7172	0.389
Precision	0.9565	0.7149	0.958	0.71	0.9556	0.7077
Recall	0.3258	0.3187	0.3334	0.339	0.3424	0.3445
F1	0.486	0.4409	0.4947	0.4589	0.5042	0.4634
Multi-dimension						
	x[0,0] ; x[-1,0]/x[1,0]		x[-1,0] ; x[0,0]/x[1,0]		x[-1,0]/x[0,0]/x[1,0]	
	Trigger	Argument	Trigger	Argument	Trigger	Argument
type_correct	0.7522	0.4028	0.7667	0.4088	0.7863	0.4159
Precision	0.9537	0.7174	0.9567	0.7129	0.9544	0.7186
Recall	0.3546	0.3564	0.3486	0.3499	0.3563	0.3568
F1	0.517	0.4762	0.511	0.4694	0.5189	0.4769
	x[-1,1]/x[0,1]/x[-1,1]		x[-1,1]/x[0,1]x[-1,1]		x[0,1]x[-1,1]x[1,1]x[-	
	Trigger	Argument	Trigger	Argument	Trigger	Argument
type_correct	0.7458	0.4046	0.7115	0.3989	0.6853	0.3922
Precision	0.9508	0.7168	0.9485	0.7165	0.9429	0.7172
Recall	0.3649	0.3673	0.3723	0.3732	0.3807	0.383
F1	0.5273	0.4857	0.5347	0.4908	0.5424	0.4993

Word&Part of speech Multi-dimension						
	x[0,0]x[0,1]x[-		x[0,0]x[0,1]x[-		x[0,1]/x[0,0]	
	Trigger	Argument	Trigger	Argument	Trigger	Argument
type_correct	0.7025	0.3965	0.7277	0.3977	0.7428	0.4
Precision	0.9407	0.7212	0.9376	0.7252	0.9369	0.7254
Recall	0.3978	0.3934	0.4255	0.4028	0.4287	0.4038
F1	0.5592	0.5091	0.5854	0.5179	0.5882	0.5188
	x[0,0]/x[0,1]x[0,1]x[-		x[0,0]/x[0,1]x[0,0]x[0.		x[0,0]/x[-	
	Trigger	Argument	Trigger	Argument	Trigger	Argument
type_correct	0.7503	0.4019	0.7595	0.4041	0.7671	0.4067
Precision	0.9359	0.7288	0.9347	0.7322	0.9339	0.7354
Recall	0.4391	0.4094	0.4501	0.4159	0.4595	0.4207
F1	0.5978	0.5243	0.6076	0.5305	0.616	0.5352
	x[-		c 2		c 1	
	Trigger	Argument	Trigger	Argument	Trigger	Argument
type_correct	0.7724	0.408	0.7815	0.41	0.7771	0.4092
Precision	0.9337	0.7381	0.9327	0.7425	0.9335	0.7405
Recall	0.4674	0.4241	0.4821	0.4304	0.4746	0.4273
F1	0.623	0.5387	0.6356	0.545	0.6293	0.5419
Parameter of CRF++						
	c 3		a CRF -1			
	Trigger	Argument	Trigger	Argument		
type_correct	0.7854	0.411	0.7878	0.4105		
Precision	0.9319	0.7443	0.931	0.7444		
Recall	0.4893	0.4338	0.4967	0.44		
F1	0.6417	0.5481	0.6478	0.5531		

There are some finding on the result:

1. Using words as features will make model tent to label entity as O, because Type_accuracy and precision will increase when use words as feature. And this tendency will become significant as the dimension of words increase.
2. On the contrary, using part of speech as features will make model tents to label as Argument/Trigger, and the more dimensions the feature are, the stronger the tendency is;
3. When only use words as feature, it's better not use multi-feature, which means it is better to use single word as feature, and the result will be better as the number of words decrease;
4. Contrast to words feature, the more part of speech were taken into consideration, the better the result will be;
5. From above, to get best result, it's better to use only the word itself as feature and use much more part of speech as features;
6. multi-feature can significantly increase the efficiency of model, because it can efficiently increase the comentropy of model(word feature works well in multi-feature)
7. The result will be better when the model match the data better(adjust the parameter of CRF++)