Assignment Report:

Named Entity Recognition

Name:汤济之
ID:1300012124
E-mail: tjz427@sina.cn
School of Electronics Engineering and Computer Science, Peking University

Abstract

The first assignment of the course is to deal with the Chinese named entity recognition problem. I trained a NER tagger using structured perceptron with Viterbi algorithm. And in order to improve the performance of the tagger, I've tried several combination of the features. Some tricks are added to the training procedure to prevent overfitting. The model achieves an F1-measure of over 85% (85.92% for the best time).

1. Assignment Introduction

Named Entity Recognition (NER) is a central task in natural language processing. It aims to recognize information units like name, including person, organization and location names, and numeric expressions including time, date, money and percent expressions. This information is usually important and useful for further analysis like sentence analysis and textual analysis. NER can also provide some semantic information.

A decent approach for NER is to treat this problem as a tagging task. Namely try to train a tagger. Every time the tagger reads a character, it gives that character a tag, which indicates whether that character is in a named entity or not, or whether that character is the beginning of a named entity. For example, "O" stands for "not a named entity", "B-PER" stands for "the beginning character for a named entity, which represents a person". With pre-tagged training set, we can easily train a NER tagger.

The first assignment for this course is to train a tagger for Chinese Named Entity Recognition. And to try every feasible method to improve the model's performance.

2. Data and Evaluation

Training data and test data are given.

Training data contains 22334 pre-tagged Chinese sentences. There are 7 types of tags, their meanings are listed below:

TAG	MEANING
O	Not a named-entity
B-PER	Beginning of the name of a person
I-PER	The rest of the name of a person
B-LOC	Beginning of the name of a location
I-LOC	The rest of the name of a location
B-ORG	Beginning of the name of an organization
I-ORG	The rest of the name of an organization

Each sentence are separated by an

empty line. In a sentence, each line contains a Chinese character and corresponding tag, separated by a space.

The test data, containing 5484 untagged sentences, has the same format with training data, with tags removed.

And an evaluation program is also given to us. This program doesn't calculate the precision or recall of the tag, but the named entity which can be segment out by the tags. This evaluation method makes this task more difficult, because even if you just bring one wrong tag in a named entity, the other correct tag become worthless.

3. Method

3.1 Cross Validation

Because I need to evaluate my model's performance, I should split the training data into two part: one for training (training dataset) and one for evaluation (validation dataset). Validation dataset is important for it can let you to test you model while training phase, give an insight on how good the model is, whether the training is converged, and whether the model is overfitting.

Because the size of available tagged data is quite small. Simply partitioning the data into two sets is not appropriate. So I use 10-fold cross validation: dividing the data into 10 parts. Every time pick 1 part to be the validation set and rest 9 parts together to be the training set. As a result, 10 models trained on 10 partially different training set are generated. In this case, the training set contains 20100 sentences and the validation set contains 2234 sentences.

3.2 Random Sampling

Training in a fixed order may result in overfitting. Because a latent feature are induced in the training process —— the order of the data. One method to overcome this shortcoming is to do random shuffling upon the whole training set before every training epoch.

A more radical way is to randomly pick one sentence in every training step, like the pure stochastic gradient descent (SGD). In this way, because of the unpredictability of the training order, the overfitting problem caused by the order can be completely eliminated. And experiments show that the speed of training is much higher than batch-based training process.

Eventually I decided to use the total random sampling method due to its high training speed.

3.3 Structured Perceptron

As we learned in the class, structured perceptron is a powerful tool to solve sequence tagging problem. I am not going to talk about the principle of this algorithm here because Mr. Feng has taught it thoroughly in class.

There are two points to note here.

The first is, I use dictionary (a data structure in python, the same as hash table) rather than a vector to store the weight vector. This improvement can remarkably increase the speed of training. Consider that you have a weight vector with 10000 dimensions, and the feature vector has 10000 dimensions as well. Every time you calculate the score of a possible tag sequence, you need to get the dot product of the 2 vectors. Therefore, the machine

should do multiplication and addition for 10000 times for every possible tag sequence. But the problem is, most of the multiplications and additions are meaningless, due to the feature vector is usually very sparse, namely only few ones exist in the feature vector, and rest of them are all zero. And zero has no contribution to the score. Using hash table to store feature-weight pair can deal with this problem efficiently. The complexity of one single query is O(1), and the number of queries is the same as the valid features of a possible tag sequence. Thus, there are no redundant multiplication and addition in score calculation.

Second, I use Viterbi algorithm to decode the sequence. The Viterbi algorithm has been discussed in class as well, so I'm not going to talk about it here. The reason why I use Viterbi algorithm rather than beam decoder is that I haven't find proper global features till now. I will discuss this problem in detail in the next section.

3.4 Feature Selection

There two kinds of features: local feature and global feature.

A local feature is a feature that defined on a substructure, and may occur multiple times in the whole structure. In this tagging problem, the model structure is quite similar to HMM, but not the same. In HMM, the probability of a hidden state depends and only depends on the previous hidden state, and the probability of a visible output depends and only depends on the corresponding hidden state. But in this model, things get more complicated, and the model get more powerful. The probability of each hidden state can be calculated by the training history, namely the whole sentence and the state before current position, in which the most probable tag has been figured out. The local features that Michael Collins chose in the NP chunking task[1] can be a great reference. After feature combinations and experiments, I finally choose these local features:

Current character	w_i	$\&T_i$
Previous character	w_{i-1}	$\&T_i$
Word two back	w_{i-2}	$\&T_i$
Word three back	w_{i-3}	$\&T_i$
Next word	w_{i+1}	$\&T_i$
Word two ahead	w_{i+2}	$\&T_i$
Word three ahead	w_{i+3}	$\&T_i$
Bigram features	$w_{i-3}+w_{i-2}$	$\&T_i$
	$w_{i-2}+w_{i-1}$	$\&T_i$
	$w_{i-1}+w_i$	$\&T_i$
	w_i+w_{i+1}	$\&T_i$
	$w_{i+1}+w_{i+2}$	$\&T_i$
	$w_{i+2}+w_{i+3}$	$\&T_i$
Trigram features	$w_{i-3}+w_{i-2}+w_{i-1}$	$\&T_i$
	$w_{i-2}+w_{i-1}+w_i$	$\&T_i$
	$w_{i-1}+w_i+w_{i+1}$	$\&T_i$
	$w_i+w_{i+1}+w_{i+2}$	$\&T_i$
	$w_{i+1}+w_{i+2}+w_{i+3}$	$\&T_i$
Previous tag	T_{i-1}	$\&T_i$

This feature combination listed above is currently the best combination, which can achieve the highest overall F1-score.

A global feature is defined on the whole model structure. For example, the count of a specific tag. But for NER tagging, the global feature is not that easy to find. There is no such rule like "there must be at least 1 PERSON in a sentence" or something like that. Actually I had done some analysis upon the whole data. The minimum number of each type of named entity in one sentence

[1] Collins M. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms[C]//Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10. Association for Computational Linguistics, 2002: 1-8.

is the same: zero. And the maximum number of each type of named entity in one sentence is 46, 125, 22 respectively. In fact, any number of any type of named entity exist in one sentence is syntactically possible. (I have done some unsuccessful experiment, see the Discussion section)

But I did have found some kind of features, maybe not looks similar to the traditional global features, but quite useful. The only three I have found are:

1. Any tag starts with "I" cannot be in the first position in one tag sequence.
2. Any tag starts with "I" must follow the tags that in the same entity type. For example, "I-PER" can only follow these two tags: "I-PER" and "B-PER".
3. Some punctuation, like comma and period, must be tagged with "O"

These features can be easily integrated into Viterbi algorithm by initialize the invalid feature weight to minus infinite. So it is not necessary to use beam search and re-ranking.

3.5 Averaging Parameters

Averaging the weights over all training steps has been proved to be a good way to overcome overfitting. And I have added this trick into my model as well

3.6 Other Configuration

I set 10000 training steps to be one epoch. Usually after 3 epochs the training speed slows down, and tend to converge after 5 epochs. After then, the precision and recall fluctuates around a specific value. So I set 10 epochs for every training. Maybe 5 or 6 performs better (can avoid fluctuation), but after averaging, the improvement is quite slight and can be ignored.

4. Result

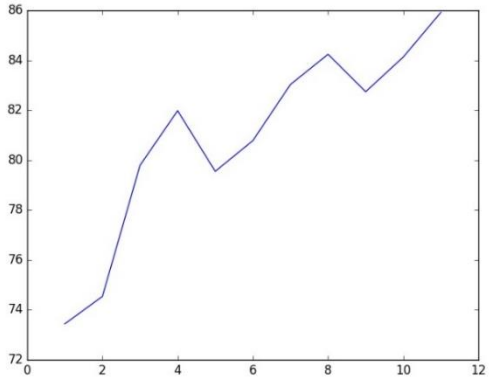
The result of 10-fold cross validation using the model described in the last section are listed below (after parameter averaging):

	Overall F1
1	83.87
2	84.73
3	85.03
4	84.97
5	85.04
6	85.34
7	85.35
8	85.92
9	85.06
10	85.53
Average	85.08
Max	85.92
Min	83.87

Best result is shown below for authenticity:

```
9050 ALL:    precision: 90.17  recall: 85.05  f1_score: 87.53
9051 PER:    precision: 91.73  recall: 85.87  f1_score: 88.71
9052 LOC:    precision: 91.23  recall: 87.88  f1_score: 89.52
9053 ORG:    precision: 83.35  recall: 76.00  f1_score: 79.51
9054 macro_precision: 88.77
9055 nmacro_recall:   83.25
9056 nmacro_f1:       85.92
```

Figure below shows the training process of training No.8. The horizontal axis represents the training step, while the vertical axis represents the overall F1-score. Note that the 11th step is averaging the weight, not a real training step.



5. Discussion

The performance of my model is acceptable, but not good. A mature NER tagger that can be used in practice should have F1-score at least 90%.

These may be some reasons that causing my model doesn't work as well as I have expected. Here are two that I firmly believed have great influence but have no ability to validate:

1. The training set is too small.
2. No word segmentation process. The character-based NER tagger has weak ability to detect the boundaries of a word. This usually make mistake.

And I also did some experiments to find how to improve the performance.

Batch Learning VS Online Learning

Batch learning means update the weights after predicting the tag sequences of a batch of sentences, while online learning means update the weights immediately after predicting the tag sequence one single sentence. Online learning runs really fast, and can escape from saddle point or local optimum easily. But near the convergence, the fluctuation is inevitable. In contrast, a mini batch that randomly selected from whole training data can effectively represent the properties of the while data, and thus can smoothly converge. But the cost is the relative low training rate.

I have tried to use mini batch leaning. The result is, I spent over 10 folds of time, and got a similar performance.

Bigram Tag Feature VS Unigram Tag Feature

When using the training history to generate the local feature, it is strange to use only unigram tag feature like what I described in the Method section. Actually I've tried to add "tag two back" (T_{i-2}) feature, and "bigram tag" ($T_{i-2}+T_{i-1}$) feature, and their combination. The performance got lower finally without exception (F1 around 80%. if more complicated features added, it can be worse). I cannot explain this phenomenon well at present. Maybe the unbalanced tag distribution ("O" accounted for over 90%) gives a harmful bias to the relative unimportant features.

Beam Decoder VS Viterbi Decoder

Actually I had spent some effort to try to embed beam search method into my model. As I had discussed in the "feature selection" part, the result is that I hadn't found any proper global feature is suitable for re-ranking.

What I have done is to calculate the percentage of sentences, in which the number of a specific type of named entity appears is in a specific range. And I found that most of the sentences are in the range listed below:

0 <= loc <= 8	0.9962
0 <= per <= 5	0.9957
0 <= org <= 5	0.9961

And then I built a beam decoder to get the most probable 4 tag sequences. And check each sequence to see whether the number of "LOC" in it is more than 8, whether the number of "PER" is more than 5, etc. If so, decrease the score of that sequence and re-rank. This method caused a slightly decrease of F1-score.