Part of Speech Tagging of Chinese Sentences Using Genetic Algorithm

K T Lua
Department of Information Systems and Computer Science
National University of Singapore
Kent Ridge, Singapore 119260
email: luakt@iscs.nus.sg
http://www.iscs.nus.sg/~luakt

Abstract

This is a short report on the use genetic algorithm for part-of-speech tagging of Chinese sentences. Our preliminary work shows that we can correctly tag 93.8% of the sentences using bigrams and word-POS frequencies. The hit rate increases to 95.8% if tri-grams are used. Compared to the other approaches of tagging, such as rule-based, HMM, recurrent neural net, the GA tagging enjoys great flexibility for being able to incorporate all information (statistical and rule) into one integrated system. Therefore, the GA tagging would provide the highest possible hit rates. However, GA is very computational intensive. Our prototype GA written in FOXPRO spends more than 1 ms per word per generation in tagging. It takes several seconds to tag a sentence in about 1000 generations.

1 Introduction

The problem of tagging of Chinese sentences belongs to a class of general optimization problems. Every Chinese word has a number of possible tags associated to it. In a typical sentences of about 10 words, the total number of possible tagging is a huge number. For example, if each word has 5 possible tags, the number of ways of tagging will be $5^{10} \sim 10 \times 10^6$. Existing approaches solving this problem including HMM, Recurrent Neural Net, Annealing, Relaxation etc(Brill, Eric, 1992, Kupiec, Robust, 1992, Lin, M. Y. and W. H. Tsai, 1987, Lin, Y.C., T. H. Chiang and K.Y. Su, 1992, Liu, Shing-Huan, Ken-jiann Chen, Li-ping Chang and Yeh-Hao Chin, 1995, Pong T. Y. and J. S Chang, 1993). All these approaches are designed to identify the most probable tag sequence based the statistical occurrence of the tag n-grams and word-tag frequencies.

In a bigram HMM model, the probability of occurrence of the tag bigram, i.e., $P(\text{tag}_i | \text{tag}_{i+1})$ (I=inetgers) and the word-tag $P(\text{tag}_i | \text{word}_i)$ are used in the computation. The solution is obtained when we can maximize the expression:

$$P = \Pi P(\text{tag}_i | \text{tag}_{i+1}) P(\text{tag}_i | \text{word}_i)$$  \[1\]

We may also convert the above expression into logarithm. Consider:

$$I(\text{tag}_i | \text{tag}_{i+1}) = - \log_2 P(\text{tag}_i | \text{tag}_{i+1})$$  \[2\]
$$I(\text{tag}_i | \text{word}_i) = - \log_2 P(\text{tag}_i | \text{word}_i)$$  \[3\]
and we have

\[ I = ? I(\text{tag}_i \text{tag}_j) + ? I(\text{tag}_i \mid \text{word}_i) \]  

Again, we are looking for a sequence of tags that produce maximum value of I in [4]

**Limitations of HMM**

In a bigram model, HMM looks at only 2 inputs, it cares only the connection between a tag and its preceding one. We have to move from bigrams to tri-grams, 4-grams and more to include more tags into the consideration. Then the problem of the sparse data set or inadequate parameters emerges. For example, if we have a tag set of 100, and we wish to consider a tri-gram model, the number of unique tri-grams will be \(100^3\), or 1 000 000. For 4-gram, this number grows to, 100 000 000. In practice, we may have a big enough corpus for tri-grams. We will definitely have problems with 4-grams as the size of the corpus must be greater than 100 millions words. Similar problem occurs in all other statistical based approaches.

The second problem of an HMM is that it is purely statistical. The HMM always selects the tag sequence with maximum probability. This is fine if we have a corpus of infinity size and all possible combinations of n-grams have occurred. If this is not the case, we will have to rely on rules. However, at the present moment, incorporation of rules HMM is still quite difficult.

**Advantage of Genetic Algorithm**

The situation is quite different from the angle of genetic algorithm(GA, Fogel, David B., 1995). In GA, a fitness function is defined to guide the searching of the best answer. The function, its definition and computation, are totally independent to the process of the searching. We have the liberty of defining (and also changing) the fitness function the way we like. We can include an HMM or a complete RNN to be part of the fitness function. The only requirements on the fitness functions are:

1. The fitness function can be computed
2. We have some knowledge about the solution
3. The fittest can be defined

Once we get a definition of the fittest for a fitness function, knowing that it can be expressed and computed, derived or experimentally obtained, we are ready to apply GA in the search of an optimal solution. One every important point to note is that we have to suggest the solution as GA will not generate a solution for us. It simply assess our suggestions and help us in the search for the optimized solution.
Heuristic Search

The search of an optimal solution in GA is heuristic by its nature. Possible solutions are suggested and fitness values obtained for the solutions. Then GA, through generations of evolution, provides the possible optimal solutions. In this way, the time complexity of $O(p^n)$ is reduced to $O(P*f(n))$. $f(n)$ gives a dependency between the number of possible selections and the generations needed to provide an optimal solution. $n$ is the number of entities in the solution (it is equal to the number of genes in the chromosome). For a sentence of 10 words each with 5 possible tags, in GA, the computation effort will $10 \times 20 \times 1000 = 200,000$, given a population size of 20 and 1000 generations. This compares very favorably to $5^{10}$ times of computations in an exhaustive approach.

Advantage of Genetic Algorithm

GA’s search for an optimal solution is very much longer that those of HMM and RNN. In later cases, the time consuming part is the training. After training, they normally provide the optimal solution in one round of computation.

Typically, a HMM tags more than 100 sentences in one second. It is thus at least 100 times faster than the GA tagging. However, we have to consider the long training time for HMM and RNN. Sometimes, it takes several days and nights to train a RNN. In GA, after obtaining the necessary parameters (such as n-grams and word-tag frequency data), it is ready to provide the solution. There is no training in GA at all. GA is therefore readily adaptable to the changing inputs whereas, in RNN, it is more difficult to make them adaptable to the change of inputs. HMM can sometimes be as fast, but it is not as flexible as GA in including rules and functions into its computations.

This short paper is our first paper on the use GA in Chinese NLP. We use it to perform POS tagging on a corpus of medium size of about 200,000 characters obtained from Tsing Hua university. We want to verify that it can be used to tag Chinese sentences to the same degree of accuracy of HMM and RNN. We also wish to add rules to the system to improve the hit rates in the near future.

In the below sections, we will provide a brief description on the corpus and tagset we used. In Section 3, we provide a detailed description on the GA program we developed for tagging; In Section 4, the performance of our GA Tagger is given and in the last section, we bring in our conclusion.

2. Corpus and Tagset

We obtained a POS hand tagged corpus from a researcher from Tsing Hua University. The corpus contains 179,159 words and tokens such as punctuation marks, begin and end of sentence marks. The words are tagged manually using the following tagset (see Table 1):
3. Our Genetic Algorithm

To evaluate the performance of GA, we start by developing a GA in its simplest form. Our GA consists of only the following necessary parts: (1) preliminary data setup, (2) population generation, (3) fitness computation and evaluation (4) mutation.
3.1 Preliminary Data Setup

As this is a bigram and tri-gram POS tagging, we have to prepare the following 3 tables: 1. POS bigrams, 2. POS tri-grams and 3. Word-POS frequencies.

The POS bigrams and tri-grams are extracted from the corpus, stored into a file, sorted and counted. The probabilities of occurrence are then computed. We find this is the only way of computing n-gram probabilities. Using an array could speed up the computation a bit for bigrams. But the array size will grow to a very huge one that no computer can handle if we to calculate n-gram data for n > 4.

3.2 Population Generation

We use a chromosome of the same word length as the sentence. Each word is given a value according its possible tag. This number is between of 1 and 94 as there are 94 POS tags. An array is created for the chromosome, chrom[i] with 1 ≤ chrom[i] ≤ 94 and 1 ≤ i ≤ word length of sentence. A random number generator is called to generate the preliminary population set. So,

\[ \text{chrom}[i] = \text{INT}(94 \times \text{RAND}() + 1) \]

Under Visual FOXPRO Ver 3.0, \( \text{RAND}() \) takes 9 \( \mu s \) to execute on a 120 MHz, Pentium PC. The total computational time for a sentence of 10 words and 1000 generations is 9 \( \mu s \times 20 \times 1000 = 0.18 \) sec.

3.3 Fitness Computation

In this preliminary work, we include only the bigram(or tri-gram) probabilities and word-POS frequencies in the fitness function. The fitness function is defined as:

\[ \text{FITNESS} = ? \text{I}(\text{tag}_i, \text{word}_i) + ? \text{I}(	ext{tag}_i, \text{tag}_j) \]

We convert multiplication to addition by using the entropy values directly from the tables. However, even with this approach, the computation time taken is about 5 ms. So, to tag a sentence in 1000 generations, the total computation will several seconds. See detailed timing in table 2.

<table>
<thead>
<tr>
<th>Computation</th>
<th>Time in ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>mutation</td>
<td>2.7</td>
</tr>
<tr>
<td>Evaluation</td>
<td>0.9</td>
</tr>
<tr>
<td>Fitness(bigram)</td>
<td>4.9</td>
</tr>
<tr>
<td>Fitness(monogram)</td>
<td>5.1</td>
</tr>
<tr>
<td>Whole</td>
<td>15.8</td>
</tr>
</tbody>
</table>
About 2/3 of the computation time are on the computation of the fitness values. In this work, we select the minimum entropy values for the optimal solutions, i.e., min(FITNESS).

3.4 Mutation

We have experimented on several approaches of mutation. Finally, we find that there is no need to keep all the chromosomes except the most fitted one. We therefore keep the most fitted one and use it to generate another 19 new ones by mutating randomly one of its genes. The gene that is selected to mutate is assigned a random number between 1 to 94. Remember we represent the tags in numbers.

We have also experimented on the number of gene to be mutated. Our conclusion is that there is no need to mutate more than one gene. Perhaps this is because the word length of the sentence that we are tagging are about 10 words or fewer. For longer sentences, we may have to allow more genes to mutate. Otherwise the number of generations producing an optimal solution will be too many. We need to experiment more in this area.

3.5 GA Algorithm

The complete GA program functions as below

\[
\text{do while fitness} \leq \text{a pre-determine value or no more improvement}
\]
\[
\quad \text{generate a population of size 20}
\]
\[
\quad \text{compute fitness and select the fittest chromosome}
\]
\[
\quad \text{Mutation}
\]
\[
\text{enddo}
\]

4. Results of POS Tagging

We have only preliminary work shows that the POS tagging with bigram and word-POS frequencies has a hit rate of 93.8%. Addition of POS tri-grams into the calculation increases the hit rate by only 2% to 95.8%.

Figure 1 shows the hit rates versus generations for sentences of 5 words and 10 words respectively. For shorter sentences, maximum hit rates is achieved earlier, at 600th generation. For longer sentences, the system converges much later, at 900th generation. Also, longer sentences provide higher hit rates. We do not under this and more experiments are needed.
5 Conclusion

Our preliminary work shows that we can use GA to POS tag a Chinese corpus with POS bigrams (or tri-grams) and the word-POS frequencies. The hit rates are about the same as those applying traditional approaches such as HMM and RNN.

The advantage is that GA allows the inclusion of more attributes. We can always add in new attributes to the system without altering the basic structure of the system. The effect of the new attributes can thus be examined with accurately. Such a property allows selection and experimenting of the attributes that may have an effect on the tagging. Furthermore, rules can also be added when necessary. We can build a real hybrid system.

The problem of GA tagging is that it is very slow. HMM normally tags few hundred sentences in one second. This is hundred or even thousand times faster than the GA tagging, which requires several second completing a sentence. We should therefore work harder solving this problem.

Reference


