POS-Tagging Project Report

Yang An 1300012204

1. **Introduction**

In this project, I implemented an automatic pos-tagger based on structured average perceptron. About 14000 features were mined from given corpus and their weight parameters were trained iteratively. Using this perceptron model, the pos-tag for each word of the development and test set were automatically predicted in a reasonable way.

1. **Model and algorithm**

I built and trained the model mainly based on perceptron algorithm. The pos-tagging task is actually a sequence tagging problem, in which each word in the input corpus should be given a classification tag. Obviously the content and morphology of a word may have effect on its tag value, thus we can use a classifier using word’s history(content and morphology information) and a candidate tag as input and a score indicating how likely this candidate is the true tag as output. Furthermore, the global score for a tag sequence predicted for a sentence can be defined as the sum of local scores of component words. Tag sequence with maximum global score is used as pos-tagging output for the given sentence.

Structured perceptron is a linear classifier model. In this model, the input history and tag pair are represented as a vector , where is a binomial function indicating whether obeys the corresponding rule. According to the definition of global score and linear assumption, the global representation of a sentence given the candidate tag sequence is .

Almost 14000 rules are derived from training corpus, which contains 4 categories:

|  |  |
| --- | --- |
| Rules | Example |
| Using tags of last 2 words. | When last tags are (“NN”,”VE”) ,this tag will be ”JJ”. |
| Using last word. | When last word is “远程”, this tag will be “NN”. |
| Using next word. | When next word is “提供”, this tag will be “NN”. |
| Using last tag and this word. | When this word is “严重” and last tag is “NN”, this tag will be “VA”. |

Each feature(rule) is given a weight parameter indicating its importance. A weight vector need to be trained. The local and global score can be calculated using and . The training process is iterative and uses Viterbi algorithm to decide the best tag sequence under current . The training algorithm is given below.

**Algorithm: Average Perceptron**

Input: A training corpus with sentences {,…,} together with their correct tag sequences {,…,}

Initialization: Max iteration time=, , ,

For

For

Using Viterbi to predict tag sequence for :

If not equals

Else

Update average weight vector

Output: as trained weight vector

The **GEN** for a given word is based on its tag set in train corpus. So if a word presented only one tag in train corpus we will infer this word doesn’t need to be predicted. If an unseen word is met in test corpus, the **GEN** set will be ['M','NN','NR','VV'], which is based on analysis of this specific test corpus. Though this trick may introduce in some bias, it indeed reduced the running time of the program.

1. **Implementation of the model**

My code used NumPy to do vector calculation. The model is built in a python module package called “postagger”, which contains 4 submodules:

**tagger:** This module implements a ChineseTagger class, which contains functions including reading training data, training parameters and predicting tag sequences.

**decoder:** This module implements Viterbi algorithm used by tagger module.

**indicator:** This module calculates and for input or pair.

**history:** This module defined a class recording content and morphology information for a word. Last 2 tags (containing placeholder “START”), last 2 words (containing placeholder “BeginFlag”), next 2 words (containing placeholder “EndFlag”) and the word itself will be recorded.

Some scripts are used to grab rules (**trigram.py**, **lastword\_notspecific.py**, etc.). The rules I used are listed in **rules.txt**, in which a rule is presented in a line using (thisword, lasttag, last2tag, lastword, last2word, nextword, next2word, tag) format splitted by “\t”. The training and testing process is achieved by **main.py**.

1. **Evaluation and conclusion**

In the final run, I set . The whole process took about a day on a computing server. I tagged both develop and test sets. The precision is about 87.5% on development set, which is below the expectation. However, for the reason of my code they are tagged in a little different ways due to some random settings, thus the correctness of development result may be inferior to test set.

In this project, I experimented using perceptron to handle POS-tagging task, which is effective and quite easy to understand. However, finding approaches to reduce computation and increase converge speed is a matter still need to be seriously considered.