Project One: Semantic Role Labeling

1500012938

Abstract

The goal of this project is labeling semantic role of verbs in given sentences while given the syntax roles of words in the sentences. About 8800 examples with labeled semantic role are given. What I'm supposed to do in this project is extracting information from the given examples and build a parser according that information. I did't use any typical machine learning technique in the project because the task is quite different from traiditional 'feature to classificaton' machine learning pattern.

1 Training Set Extraction

1.1 Building Path Tree

At first, I parse paths of verbs in example sentences to its arguments. The path is as sequence that records the steps a verb stepping to one of its argument in the syntax tree. In this project, there are three direction in the path which are:

1. To Father

2. To Child

3. To Brother

'To Brother' is actually the combination of 'To Father' and 'To child', but infomation between brothers is important, if we simply treat 'To Father' then 'To child' seperately, some decisive infomation may be lost. Apart from direction, brothers of the next step is also recorded in the paths, if the node doesn't have left or right brother, then the brother tag is 'NA' In 'To Child' step, the index of the child which is the next step is recorded. In 'To Brother' step, the distance to the brother is recorded. The distance can be negative, if the brother is at left side.

Then build the path tree, which substantially merge same profixes of paths, from these paths. The root node of path tree is literally called 'root'. nodes of first generation of root are verbs, recording type of verb and the brothers of verb. Leaf nodes consist of the argument tag and counts of the paths they represent(each leaf represent a unique, integreted path)

1.2 Count the Paths

In previous section, paths that actually are argument paths of some verbs are count and saved in the path tree. In this section, after the tree is build, I count the paths that exist in example sentences. There are many cases that a path in one sentence is the path of a verb argument but in another sentence there is excatly the same path in its syntax tree but the end of path is not an argument of the verb. For each path, I have these two counts, so I can calculate the *rate* of the path being a argument path while it exist in a syntax tree, which is a inportant criteria for choosing paths among possible paths.

2 Argument Labeling

2.1 Getting Syntax Trees

My path-searching method is based on the syntax tree of each sentence. Syntax structures are not given in the project material, so I use Berkerly Parser 1.7 version with grammar file trained by train.tree file to parse each sentence's syntax tree, using given POS in *.text files.

2.2 Determining Argument

Now I have the path tree and syntax tree of each target sentence, I'm able to get sets of argument paths of verbs. For each verb V in a sentence S, There are a different possible argument paths according to path tree. Candidate argument path are

$path_i$, where $i = 0, 1, 2, ...a - 1$. each path has three attributes: argument tag, position in the sentence and *rate*. First, I select the path who has highest rate among all paths of the sentence, then remove all paths that has overlap position or same tag with the selected path. Doing this operation iteratively untill no candidated left.

2.3 Psudo Code

```
input : set of argument path $P = \{path_i\}$
output: set of selected argument path $SP$

$SP \leftarrow \emptyset$
do
    select $p_{max} \in P$ ,
    $p_{max}.rate = max\{path.rate|path \in P\}$
    $P = P \setminus \{p_{max}\}$ $SP = SP \cup \{p_{max}\}$ foreach $path \in P$ do
        if $path.pos$ overlap $p_{max}.pos$ or $path.tag == p_{max}.tag$ then
            $P = P \setminus \{path\}$
        end
    end
while $P \neq \emptyset$;
return $SP$;
```

3 Files and Codes

* *Parse.py* read syntax tree from STDIN and output in *.prop format to STDOUT. Redirect input and output to read from and write to files.

* *Train.py* generate *PathTree*, which is essential for *Parse.py*. *Train.py* read file 'trn/trn.syntax' and 'trn/trn.props'.

* *Tools.py* consists of important functions in *Train.py* and *Parse.py*.

* *.syntax files are generated by Berkerly Parser.

* Result files are in 'result' folder .