CS134 (Fall 2012): Programming Assignment #4
Transition-based dependency parsing

Chuan

December 15, 2012

Brief description

In this assignment, I have mainly implemented three parts of the general transition based dependency parser: Gold-standard parser(feather)MaxEnt and NaiveBayes classifier training and decoding parser and also developed some classifier optimization.

Part I Gold standard parser and feature extraction

In this part, I utilize the gold standard parsing algorithm to represent the data instance as a list of configuration and transition pairs. Then extract the 6 features as basicly required in the assignment:

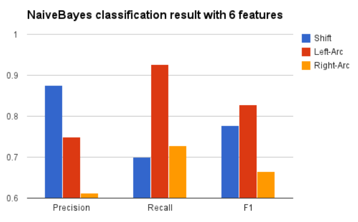
- Front of queue
- Top of stack
- Stack/Buffer bi-gram
- POS variants

And I store the these features as (featuretype,featurevalue) pairs.

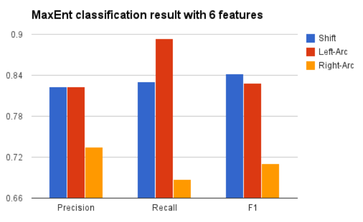
Part II MaxEnt and NaiveBayes classifier training

Here I have trained my own MaxEnt classifier and the nltk NaiveBayes classifier with the same featuerset provided above. And the classifier evaluation is provided in Figure 1.

In the figure, we can see that ME has good precision and F1 score but



(a) NaiveBayes classifier with 6 basic features



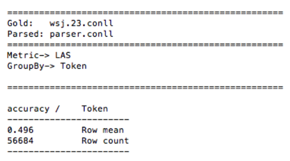
(b) MaxEnt classifier with 6 basic features

Figure 1: Two classification result with basic 6 features

somehow suffer from recall.

Part III Decoding

In this step, I tried to predict the test sentences' dependency arcs set by using the above trained MaxEnt classifier (Intuitively, I should do the classifier optimization part and then do the decoding part. However, I made some mistakes about how to arrange the experiments). By replacing the oracle function with classification decision, I manage to rebuild the dependency parse with .conll format. Then I compare this with the gold standart test file by using the evaluation tool MaltEval. And the result is shown in Figure 2. From the result, we can see that although classification performance is

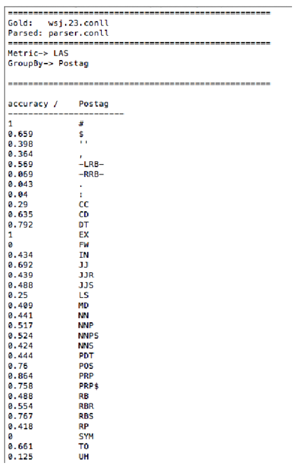


```
=====================================================
Gold:   wsj.23.conll
Parsed: parser.conll
=====================================================
Metric-> LAS
GroupBy-> Token

=====================================================

accuracy /    Token
-----------------------
0.496         Row mean
56684         Row count
-----------------------
```

(a) groupby token

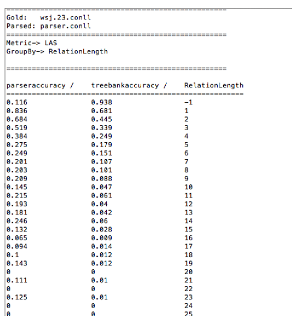


```
=====================================================
Gold:   wsj.23.conll
Parsed: parser.conll
=====================================================
Metric-> LAS
GroupBy-> Postag

=====================================================

accuracy /    Postag
-----------------------
1             #
0.659         $
0.398         ''
0.364         ,
0.569         -LRB-
0.069         -RRB-
0.043         .
0.04          :
0.29          CC
0.635         CD
0.792         DT
1             EX
0             FW
0.434         IN
0.692         JJ
0.439         JJR
0.488         JJS
0.25          LS
0.409         MD
0.441         NN
0.517         NNP
0.524         NNPS
0.424         NNS
0.444         PDT
0.76          POS
0.864         PRP
0.758         PRP$
0.488         RB
0.554         RBR
0.767         RBS
0.418         RP
0             SYM
0.661         TO
0.125         UH
```

(b) groupby POS tagging



```
=====================================================
Gold:   wsj.23.conll
Parsed: parser.conll
=====================================================
Metric-> LAS
GroupBy-> RelationLength

=====================================================

parseraccuracy /    treebankaccuracy /    RelationLength
--------------------------------------------------------
0.116               0.938                 -1
0.836               0.681                 1
0.684               0.445                 2
0.519               0.339                 3
0.384               0.249                 4
0.275               0.179                 5
0.249               0.151                 6
0.201               0.107                 7
0.203               0.101                 8
0.209               0.088                 9
0.145               0.047                 10
0.215               0.061                 11
0.193               0.04                  12
0.181               0.042                 13
0.246               0.06                  14
0.132               0.028                 15
0.065               0.009                 16
0.094               0.014                 17
0.1                 0.012                 18
0.143               0.012                 19
0                   0                     20
0.111               0.01                  21
0                   0                     22
0.125               0.01                  23
0                   0                     24
0                   0                     25
```

(c) group by arc-length

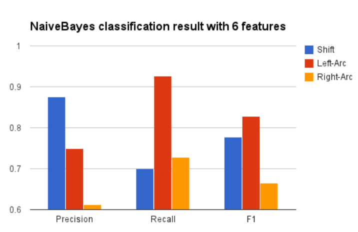
Figure 2: dependency parsing eval result with different aspects

eligible, the performance of the decoder is much worse. The overall average accuracy is only 50%. And we also can see by grouping the result with POS tag and arc-length. The arc-length is even worse than the POS tag accurracy.

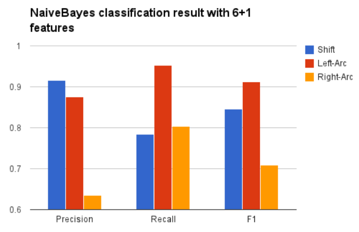
Part IV Classifier Optimization

In this step, I add one more (topofStackPOS,topofBufferPOS,bufferChildrenPOS) feature into the 6 basic featureset, which makes the new 6+1 featureset. And I train the NaiveBayes classifier(since it will be more faster). And this one additional feature indeed help a lot. The result is shown in Figure 3 and Figure 4.

we can see that the improvements is very obvious. And this one more feature has some linguistic meaning which is very helpful for the classification.



(a) NaiveBayes classifier with 6 basic features



(b) NaiveBayes classifier with 6+1 features

Figure 3: Two classification result with different featuresets

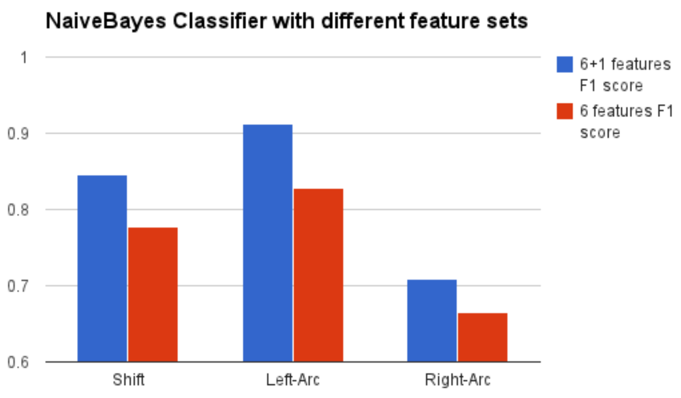


Figure 4: F1 score comparison