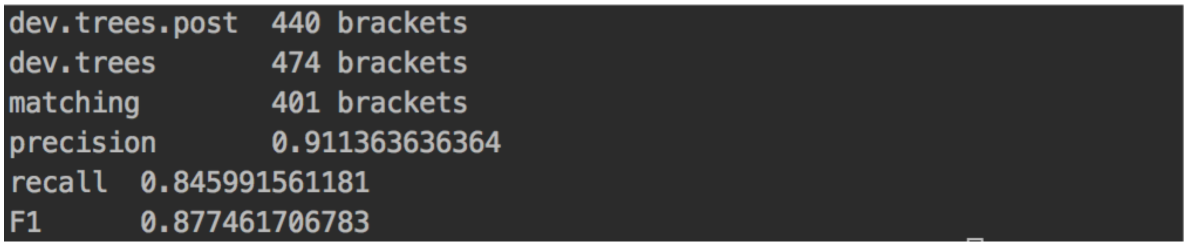
3480116405 Yuxin Zhou

- Modification 1 - Unary modification

The first modification I use is the unary modification mentioned in slide. that is, I keep the non-terminal unary rules and build [X, i, j] for every non- lexical rule R = X->Y and state [Y, i, j] if [X, i, j] does not exist. As for implement, I changed the preprocess.py and postprocess.py correspondingly, and did some modification to CKY.

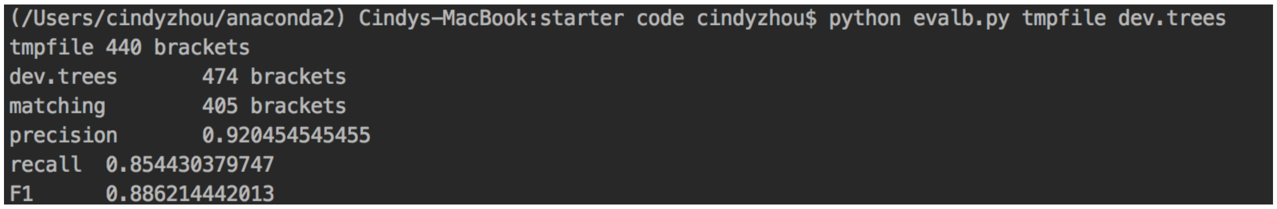
This modification makes the rules less specific, therefore I can parse more sentences. In the original version, 6 sentences are not parsed, but in this version, only 2 sentences are not parsed. However, since the rules are less specific, both the precision and F1 score drop, while the recall slightly increases.



```
dev.trees.post  440 brackets
dev.trees       474 brackets
matching        401 brackets
precision       0.911363636364
recall  0.845991561181
F1      0.877461706783
```

- Modification 2 – Case Insensitive

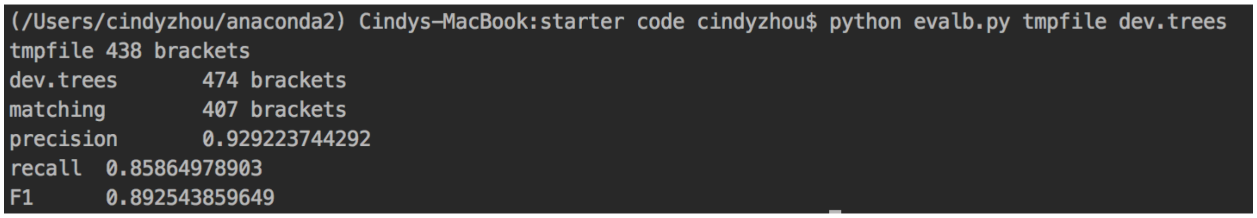
I modified the parser by making the words in sentence case insensitive. In this way, the F1 score raised to 88.0 to 88.6



```
(/Users/cindyzhou/anaconda2) Cindys-MacBook:starter code cindyzhou$ python evalb.py tmpfile dev.trees
tmpfile 440 brackets
dev.trees       474 brackets
matching        405 brackets
precision       0.920454545455
recall  0.854430379747
F1      0.886214442013
```

- Modification 3 – Smoothing

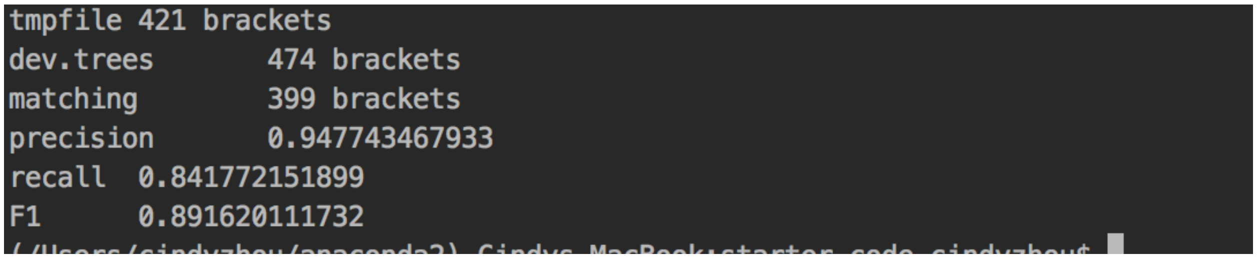
I added smoothing to my parser to raise the recall, that is I add an <unk> node to any non-terminal node if there hasn't been one. The F1 score raised from 88.6 to 89.2.



```
(/Users/cindyzhou/anaconda2) Cindys-MacBook:starter code cindyzhou$ python evalb.py tmpfile dev.trees
tmpfile 438 brackets
dev.trees       474 brackets
matching        407 brackets
precision       0.929223744292
recall  0.85864978903
F1      0.892543859649
```

• Modification 4 – Parent Annotation

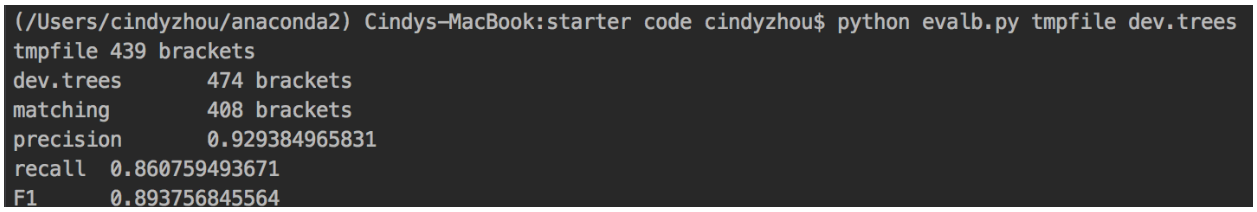
I tried the parent annotation but more sentences are unlikely to be parsed. I believe the reason lays in the sparsity of the data. Our training data is not large and the annotated rules are so specific that the probability of encountering unseen rules in testing increases.



```
tmpfile 421 brackets
dev.trees       474 brackets
matching        399 brackets
precision       0.947743467933
recall  0.841772151899
F1      0.891620111732
(/Users/cindyzhou/anaconda2) Cindys-MacBook:starter code cindyzhou$
```

• Modification 5

I have also tried a "tricky" method. Since the recall on test data seems to have more room for improvement, I try to construct an "illegal" parsing tree when the parsing fail. I linked the TOP node to one node on first row (representing for the subtree [0, k]) and one node on last column (representing for the subtree [k, n]) in DP table. That is, we construct a tree with a rule "TOP-> XX XX" that we have never seen in training data. In this way, one more sentence – the shortest one - can be parsed. But the F1 score is almost the same, increasing from 89.2 to 89.3, because the sentence is parsed partially correct.



```
(/Users/cindyzhou/anaconda2) Cindys-MacBook:starter code cindyzhou$ python evalb.py tmpfile dev.trees
tmpfile 439 brackets
dev.trees       474 brackets
matching        408 brackets
precision       0.929384965831
recall  0.860759493671
F1      0.893756845564
```

3480116405 Yuxin Zhou