

QUESTION 1:

How many rules are there in your grammar? What is the most frequent rule, and how many times did it occur?

No. of rules in grammar: 752

Most common rule: PUNC -> '.'

Number of occurrences of most common rule: 346 times

```
Terminal
+ C:\Users\Neah_New\Desktop\CSCI 544 - NLP\Homework 4\hw4>python viterbi_parser.py
X No. of rules in grammar: 752
  Most common rule: PUNC -> '.'
  Number of occurrences of most common rule: 346 times

C:\Users\Neah_New\Desktop\CSCI 544 - NLP\Homework 4\hw4>
```

QUESTION 2:

Show the output of your parser on the _rst line of dev.strings , along with its log-probability (base 10).

Sentence: The flight should be eleven a.m tomorrow .

Log Probability: -17.7452979891

Parse tree before postprocess:

(TOP (S (NP (DT The) (NN flight)) (VP (MD should) (VP (VB be) (NP (NP* (CD eleven) (RB a.m)) (NN tomorrow))))) (PUNC .))

Parse Tree after postprocess:

(TOP (S (NP (DT The) (NN flight)) (VP (MD should) (VP (VB be) (NP (CD eleven) (RB a.m) (NN tomorrow))))) (PUNC .))

```
train.post
with open("dev.... > for line in fp > else

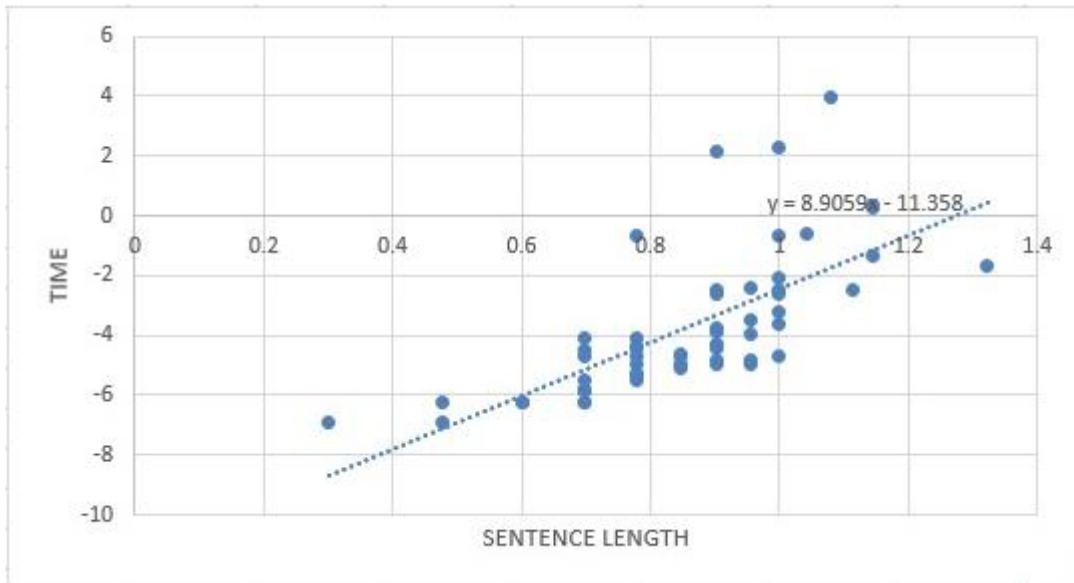
Terminal
+ C:\Users\Neah_New\Desktop\CSCI 544 - NLP\Homework 4\hw4>python viterbi_parser.py
- No. of rules in grammar: 752
  Most common rule: PUNC -> '.'
  Number of occurrences of most common rule: 346 times

  The flight should be eleven a.m tomorrow .
  Log Probability: -17.7452979891
  Parse tree before postprocess: (TOP (S (NP (DT The) (NN flight)) (VP (MD should) (VP (VB be) (NP (NP* (CD eleven) (RB a.m)) (NN tomorrow))))) (PUNC .))
```

```
learn_pcfg.py x grammar.py x dev.parses.post x grammar_dict.txt x grammar.txt x viterbi_parser.py x preprocess.py x
(TOP (S (NP (DT The) (NN flight)) (VP (MD should) (VP (VB be) (NP (CD eleven) (RB a.m) (NN tomorrow))))) (PUNC .))
(TOP (S (S (NP (PRP I)) (VP (MD would) (VP (VB like) (NP (PRP it)) (X (TO to)) (VP (VBP have) (NP (NP (DT a) (NN stop
New) (NNP York))))) (CC and) (S (NP (PRP I)) (VP (MD would) (VP (VB like) (NP (NP (DT a) (NN flight)) (SBAR (WHNP
(VB? sarvesh) (ADVP (PR breakfast)))))))) (PUNC .))
```

QUESTION 3:

Show a plot of parsing time (y axis) versus sentence length (x axis). Use a log-log scale. Estimate the value of k for which $y \propto x^k$ (you can do a least-squares fit or just guess). Is it close to 3, and why or why not?



This is the log – log curve that has been generated for sentence length vs time values.

To calculate slope, the following points are chosen: $y_2 = -1.8$, $y_1 = -3$, $x_2 = 1.18$, $x_1 = 0.78$

Slope = $k = 3.0$

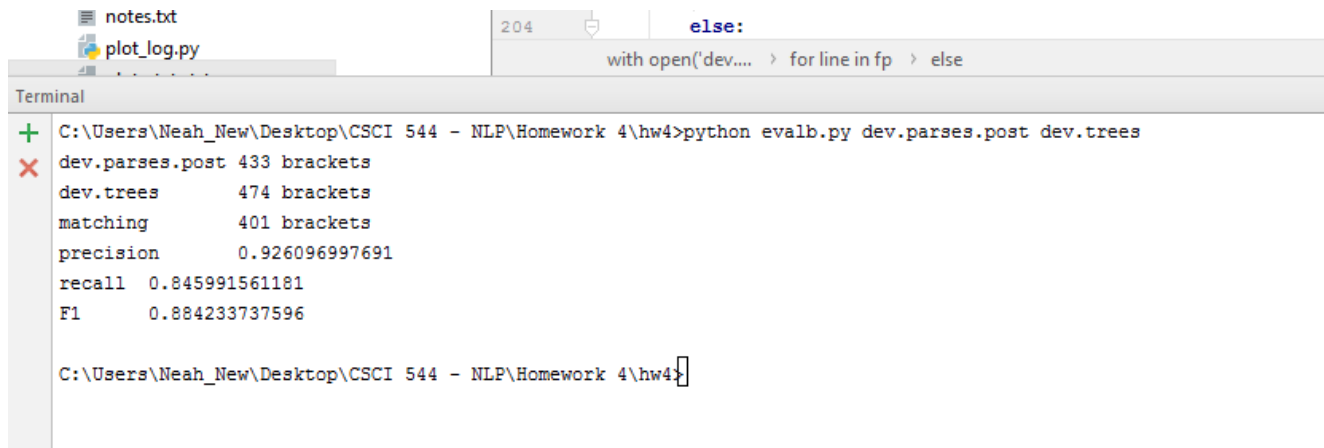
Here, since the complexity of probabilistic CKY is n^3 (for a sentence of length n), hence the slope of the line is 3.

QUESTION 4:

Run your parser output through `postprocess.py` and save the output to 'dev.parses.post'. Evaluate your parser output against the correct trees in 'dev.trees' using the command: `python evalb.py dev.parses.post dev.trees`

Show the output of this script, including your F1 score.

Evaluation of the parser output against the correct trees in dev.trees:



The screenshot shows a terminal window with the following content:

```
notes.txt
plot_log.py
204
else:
with open('dev.... > for line in fp > else

Terminal
+ C:\Users\Neah_New\Desktop\CSCI 544 - NLP\Homework 4\hw4>python evalb.py dev.parses.post dev.trees
X dev.parses.post 433 brackets
dev.trees 474 brackets
matching 401 brackets
precision 0.926096997691
recall 0.845991561181
F1 0.884233737596

C:\Users\Neah_New\Desktop\CSCI 544 - NLP\Homework 4\hw4>
```

QUESTION 5:

Describe your modifications and report your new F1 score on dev.strings . What helped, or what didn't? Why?

The modifications experimented with to try to improve the F1 score of the parser are as follows:

1. Making the grammar and dev.strings case insensitive

While creating the productions of the grammar, all words were converted to lowercase. All words of lines from dev.strings while being inputted to the parser were made lowercase.

The F1 score increased from 88.42 to 89.37%.

There were several rules in the original case sensitive grammar that contained the same left and right hand sides, but only differed by their case. Hence, by making the case uniformly lower, the probabilities of the several rules which only differed by case increased significantly. The file dev.trees (saved as dev2.trees) was also converted to lowercase for testing the performance of the parser after this modification was made.

A screenshot of the F1 score of the parser after converting the grammar, dev.strings and also the dev.trees (saved as dev2.trees) to lowercase is shown below:

```
Terminal
+
X C:\Users\Neah_New\Desktop\CSCI 544 - NLP\Homework 4\hw4>python evalb.py dev.parses2.post dev2.trees
dev.parses2.post      448 brackets
dev2.trees            474 brackets
matching              412 brackets
precision              0.919642857143
recall 0.869198312236
F1      0.893709327549

C:\Users\Neah_New\Desktop\CSCI 544 - NLP\Homework 4\hw4>
```

2. Parent Annotation

In addition to binarizing the tree and concatenating unary rules in the trees, each node label is parent annotated. Theoretically, the parent annotation is supposed to refine the probabilities of productions by adding some context.

However, this modification failed to improve the F1-measure and dropped it drastically to 68.32%.

Due to annotating each node with its parent, the rules tend to become much more specific. Hence, I think the F1 score failed to improve.

A screenshot of the F1 score of the parser after applying parent annotation is shown below:

```
Terminal
+
X C:\Users\Neah_New\Desktop\CSCI 544 - NLP\Homework 4\hw4>python evalb.py dev.parses.post dev.trees
dev.parses.post 451 brackets
dev.trees       474 brackets
matching        316 brackets
precision        0.70066518847
recall 0.666666666667
F1      0.683243243243

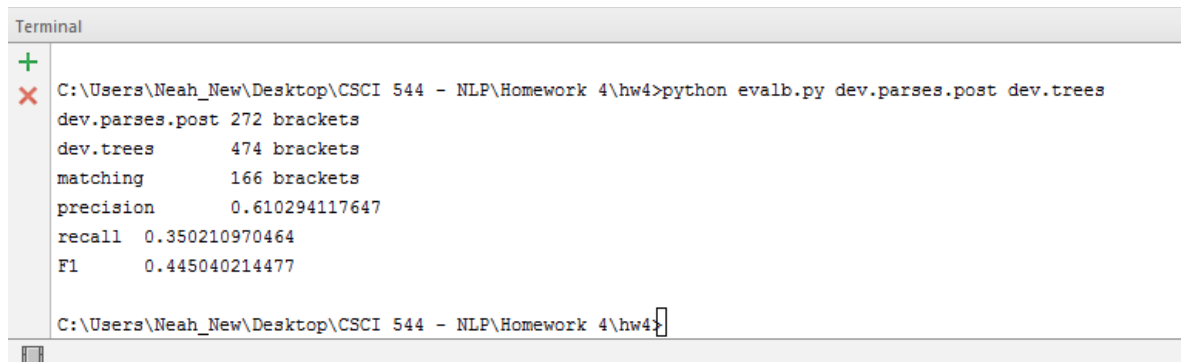
C:\Users\Neah_New\Desktop\CSCI 544 - NLP\Homework 4\hw4>
```

3. Grandparent Annotation

Annotating each node with its parent and grandparent was tested with the intent of establishing additional context.

However, this modification failed to improve the F1-measure and dropped it drastically to 44.50%. Due to annotating each node with its parent and their grandparent, the rules tend to become much more specific and hence the specificity outweighed the detailed context. Hence, think the F1 score failed to improve.

A screenshot of the F1 score of the parser after applying grandparent annotation is shown below:



```
Terminal
+
X C:\Users\Neah_New\Desktop\CSCI 544 - NLP\Homework 4\hw4>python evalb.py dev.parses.post dev.trees
dev.parses.post 272 brackets
dev.trees      474 brackets
matching       166 brackets
precision      0.610294117647
recall 0.350210970464
F1 0.445040214477

C:\Users\Neah_New\Desktop\CSCI 544 - NLP\Homework 4\hw4>
```

4. Creating partial parse trees for sentences when no parse trees are returned

By observing the pattern of strings in dev.strings, the following are observable:

Sentences starting with question words like 'What', 'Which', 'Where' end with a punctuation and have a syntactic structure like "(TOP (SBARQ (WHNP (first_word)))(PUNC ?))".

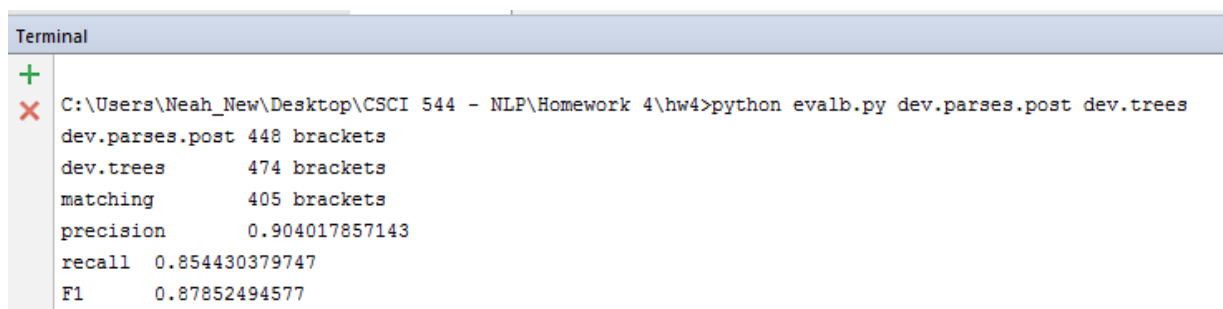
Similarly, for declarative sentences starting with 'The', the syntactic structure is similar to "(TOP (S (NP (DT The)))... (PUNC .))".

Such a pattern in the parses generated by the parser were observed, and for sentences that the parser failed to generate a parse, the above heuristics were used to predict atleast a portion of the parse tree with the punctuation at the end.

However, the F1 score dropped to 87.85% after applying this modification.

Most likely, since only a part of the parse tree was predicted, many brackets were matched, however several brackets of the sentence were still a mismatch. Hence, it did not make a difference.

A screenshot of the F1 score after this change was made is shown below:



```
Terminal
+
X C:\Users\Neah_New\Desktop\CSCI 544 - NLP\Homework 4\hw4>python evalb.py dev.parses.post dev.trees
dev.parses.post 448 brackets
dev.trees      474 brackets
matching       405 brackets
precision      0.904017857143
recall 0.854430379747
F1 0.87852494577
```

QUESTION 6:

Modify the runme batch script in vocareum such that all the pre- and post-processing needed to train your grammar on an arbitrary training corpus and evaluate it on an arbitrary test corpus are done (the included batch script will run preprocess.py and postprocess.py , will generate an artificial grammar, and will run a trivial parser; replace the dummy code with your own code!). When you submit your batch script will report a run that trains on train.trees and decodes dev.strings , evaluating against dev.trees , so you can verify proper behavior. After the submission window closes, your parser will be trained on a corpus similar to train.trees and then evaluated on a hidden test.strings . Your score on this question (maximum 10 points out of 120) will depend on your F1 score relative to the rest of the class.

```
#!/usr/bin/env bash
set -e
```

```
SCRIPTDIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"
TRAINING=$1;
INPUT=$2;
OUTPUT=$3;
python $SCRIPTDIR/preprocess.py < $TRAINING | $SCRIPTDIR/unknown.py > $SCRIPTDIR/train.trees.pre.unk
python $SCRIPTDIR/viterbi_parser.py -i $INPUT -o $OUTPUT < $SCRIPTDIR/train.trees.pre.unk >
$SCRIPTDIR/temp
python $SCRIPTDIR/postprocess.py < $SCRIPTDIR/temp > $OUTPUT
# $SCRIPTDIR/evalb.py < $OUTPUT $SCRIPTDIR/dev.trees
```