Assignment 3

Name: Mengxi Nie
GID: G48739076
Gmail: eva_happylife@gwu.edu

Q1:

1> Commands ------ Get the different training size:

- *1.a*:
  ```
  cat en-universal-train.conll | awk '{printf "%s#END#", $0}' |perl -pe 's/#END##END#/\n\n/g' |head -8 |perl -pe 's/#END#/\n/g' >en-4-train.conll
  ```
- *1.b*:
  ```
  cat en-universal-train.conll | awk '{printf "%s#END#", $0}' |perl -pe 's/#END##END#/\n\n/g' |head -80 |perl -pe 's/#END#/\n/g' >en-40-train.conll
  ```
- *1.c*:
  ```
  cat en-universal-train.conll | awk '{printf "%s#END#", $0}' |perl -pe 's/#END##END#/\n\n/g' |head -800 |perl -pe 's/#END#/\n/g' >en-400-train.conll
  ```
- *1.d*:
  ```
  cat en-universal-train.conll | awk '{printf "%s#END#", $0}' |perl -pe 's/#END##END#/\n\n/g' |head -8000 |perl -pe 's/#END#/\n/g' >en-4000-train.conll
  ```

2> Commands ------ Get the training model - "parser-part" & Apply to Dev set:

-
  ```
  java -jar maltparser-1.9.1.jar -c parser-part -i NLP_data/en-4X-train.conll -m learn
  ```

-
  ```
  java -jar maltparser-1.9.1.jar -c parser-part -i NLP_data/en-universal-dev.conll -m parse -o dev.out.ex.1.X.conll
  ```

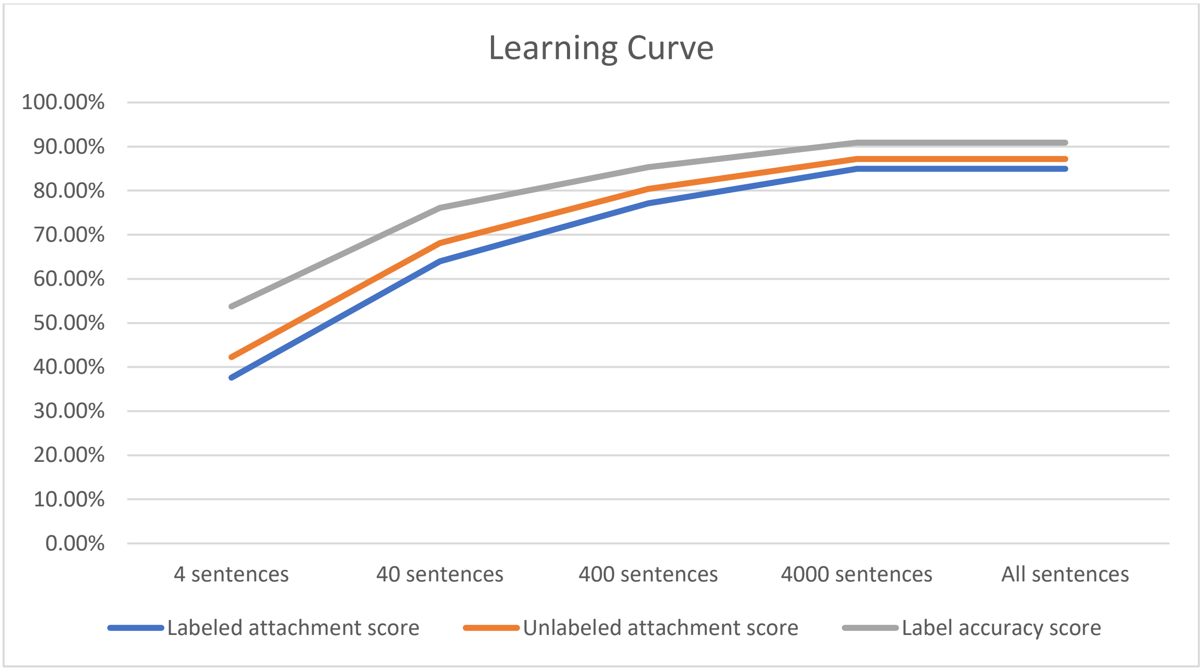
3> Commands ------ Evaluation:

-
  ```
  perl conll-eval.pl -q -g en-universal-dev.conll -s dev.out.ex.1.X.conll
  ```

4> Results:

Training set size	Labeled attachment score	Unlabeled attachment score	Label accuracy score
4 sentences	37.59%	42.25%	53.74%
40 sentences	63.99%	68.15%	76.14%
400 sentences	77.11%	80.39%	85.32%
4000 sentences	84.96%	87.19%	90.89%
All sentences	84.96%	87.19%	90.89%

5> Learning curve table ------ Showing performance on the development data set



6> Analysis:
The curve shows the performance on the development dataset for each training size. It is obviously to see that with the growth of training dataset size, the labeled attachment score(LAS) increases. However, it reaches a peak at about 4000 sentences and becomes stable after that point. I think the accuracy is 84.96% for 400,000 sentences because 4,000 sentences, 39,832 sentences and 400,000 sentences are all not very large number of sentences. As for 4,000,000 sentences, I think the accuracy will increase to about 90% because the size of training size is enormous, and the model will include more structures and words. Therefore, the accuracy will increase when applied to development dataset.

Q2:

1> Commands ------ Get full training model – "parser-all" using different parameters:

- *2.a*:
  java -jar -Xmx5000m -Xms5000m maltparser-1.9.1.jar -c parser-all -m learn -i NLP_data/en-universal-train.conll -a nivrestandard -ne false -nr false
- *2.b*:
  java -jar -Xmx5000m -Xms5000m maltparser-1.9.1.jar -c parser-all -m learn -i NLP_data/en-universal-train.conll -a nivreeager -ne false -nr false
- *2.c*:
  java -jar -Xmx5000m -Xms5000m maltparser-1.9.1.jar -c parser-all -m learn -i NLP_data/en-universal-train.conll -a stackproj -ne false -nr false
- *2.d*:
  java -jar -Xmx5000m -Xms5000m maltparser-1.9.1.jar -c parser-all -m learn -i NLP_data/en-universal-train.conll -a stackeager -ne false -nr false
- *2.e*:
  java -jar -Xmx5000m -Xms5000m maltparser-1.9.1.jar -c parser-all -m learn -i NLP_data/en-universal-train.conll -a stacklazy -ne false -nr false

- *2.f*:
  java -jar -Xmx5000m -Xms5000m maltparser-1.9.1.jar -c parser-all -m learn -i NLP_data/en-universal-train.conll -a nivrestandard -ne true -nr false
- *2.g*:
  java -jar -Xmx5000m -Xms5000m maltparser-1.9.1.jar -c parser-all -m learn -i NLP_data/en-universal-train.conll -a nivrestandard -ne false -nr true
- *2.h*:
  java -jar -Xmx5000m -Xms5000m maltparser-1.9.1.jar -c parser-all -m learn -i NLP_data/en-universal-train.conll -a nivrestandard -ne true -nr true

2> Commands ------ Apply to Dev set:

- java -jar -Xmx5000m -Xms5000m maltparser-1.9.1.jar -c parser-all -m parse -i NLP_data/en-universal-dev.conll -o dev.out.ex.2.X.conll

3> Commands ------ Evaluation:

- perl conll-eval.pl -q -g en-universal-dev.conll -s dev.out.ex.2.X.conll

4> Results:

Algorithm number	Labeled attachment score	Unlabeled attachment score	Label accuracy score
2.a	88.79%	90.62%	93.34%
2.b	88.81%	90.71%	93.27%
2.c	88.71%	90.54%	93.35%
2.d	88.68%	90.54%	93.32%
2.e	88.68%	90.54%	93.32%
2.f	88.79%	90.62%	93.34%
2.g	88.69%	90.53%	93.35%
2.h	88.69%	90.53%	93.35%

5> Analysis:
Because we only use labeled attachment score(LAS), the best performance is 2.b and the command is as follows when training dataset:
java -jar -Xmx5000m -Xms5000m maltparser-1.9.1.jar -c parser-all -m learn -I NLP_data/en-universal-train.conll -a nivreeager -ne false -nr false
The parameters are: -a nivreeager -ne false -nr false

Q3:

1> Command ----- Get part of development data set GOLD file as new gold file – "dev_tagger.conll" & Get part of training set as new training set – "train-tagger.conll":

- cat en-universal-dev.conll | awk '{print $2"\t"$4"-"$5}' | perl -pe 's/\s-//g' > dev_tagger.conll
- cat en-universal-train.conll | awk '{print $2"\t"$4"-"$5}' | perl -pe 's/\s-//g' > train-tagger.conll

2> Command ------ Get the training model – "mytagger.model" using different parameters:

- *3.a:*
  cat train-tagger.conll |./hunpos-train -e3 -f5 mytagger.model
- *3.b:*
  cat train-tagger.conll |./hunpos-train -t1 -f15 mytagger.model
- *3.c:*
  cat train-tagger.conll |./hunpos-train -t3 -f15 mytagger.model
- *3.d:*
  cat train-tagger.conll |./hunpos-train -t4 -e3 -f15 mytagger.model
- *3.e:*
  cat train-tagger.conll |./hunpos-train -t3 -e3 -f15 mytagger.model
- *3.f:*
  cat train-tagger.conll |./hunpos-train -t3 -e3 -s20 -f20 mytagger.model
- *3.g:*
  cat train-tagger.conll |./hunpos-train -t3 -s20 -f20 mytagger.model
- *3.h:*
  cat train-tagger.conll |./hunpos-train -t3 -s25 -f25 mytagger.model
- *3.i:*
  cat train-tagger.conll |./hunpos-train -t3 -s15 -f15 mytagger.model
- *3.j:*
  cat train-tagger.conll |./hunpos-train -t4 -s20 -f20 mytagger.model
- *3.k:*
  cat train-tagger.conll |./hunpos-train -t4 -e3 -s20 -f20 mytagger.model
- *3.l:*
  cat train-tagger.conll |./hunpos-train -t4 -s25 -f25 mytagger.model
- *3.m:*
  cat train-tagger.conll |./hunpos-train -t5 -s20 -f20 mytagger.model

3> Command ------ Make predictions:

- cat en-universal-dev.conll | awk '{print $2}' > dev_word.conll
- ./hunpos-tag mytagger.model <dev_word.conll > dev.out.3.X.conll

4> Command ------ Evaluation:

- perl pos-eval.pl dev_tagger.conll dev.out.3.X.conll

5> Result:

Algorithm number	POS Accuracy
3.a	96.17%
3.b	95.81%
3.c	96.30%
3.d	96.26%
3.e	96.22%
3.f	96.22%
3.g	96.31%
3.h	96.30%
3.i	96.30%

3.j	96.36%
3.k	96.27%
3.l	96.35%
3.m	96.22%

6> Analysis:
The best algorithm is *3.j* and its parameters are *-t4 -s20 -f20*. The POS accuracy of *3.j* is 96.36% and it is the highest one among these algorithms. I make *t = 4* and this makes the word you want to tag has a stronger association with words appearing before. Besides, *-s = 20 and -f = 20* extents the max length of suffix and makes more words with low frequency get a precious tag. I am surprised with the result. *-t* means tag-order and I think the accuracy will go up with the growth of *t* because of the stronger relationship between words. However, when *t = 5*, the accuracy decreases. Sometimes, adding emission order can decrease the accuracy, too. When comes to a unknow word, we can recognize its suffix from morphology to analyze this word's tag. To my surprise, when increasing *s* and *f*, the result is not so good.

Q4:

1> Command ------ Processing predicted POS tags in Q3
- cat dev.out.ex.3.j.conll | awk '{sub(/-/, "\t", $2)};1' >dev.3.j_new.conll

2> Command ------ Construct new GOLD file
- awk -v OFS='\t' 'FNR==NR {a1[NR]=$2;a2[NR]=$3;next}{$4=a1[FNR];$5=a2[FNR]}1' dev.3.j_new.conll en-universal-dev.conll | perl -pe 's/\t{4}//g' >dev.new.ex.2_gold.conll

3> Command ------ Get the training model using different parameters (The same parameter settings in Q2)
- *4.a*:
  java -jar -Xmx5000m -Xms5000m maltparser-1.9.1.jar -c parser-all -m learn -i NLP_data/en-universal-train.conll -a nivrestandard -ne false -nr false

4> Command ------ Get the predicted files and evaluate using new GOLD file
- java -jar -Xmx5000m -Xms5000m maltparser-1.9.1.jar -c parser-all -m parse -i Q4/dev.new.ex.2_gold.conll -o Q4/dev.out.ex.4.X.conll
- perl conll-eval.pl -q -g Q4/dev.new.ex.2_gold.conll -s Q4/dev.out.ex.4.X.conll

5> Result:

Algorithm number	LAS1**(Q2)**	LAS2**(Q4)**
4.a	88.79%	86.06%
4.b	88.81%	86.22%
4.c	88.71%	85.99%
4.d	88.68%	85.92%
4.e	88.68%	85.92%
4.f	88.79%	86.06%

4.g	88.69%	85.94%
4.h	88.69%	85.94%

6> Analysis:

The command is as follows:

java -jar -Xmx5000m -Xms5000m maltparser-1.9.1.jar -c parser-all -m learn -i NLP_data/en-universal-train.conll -a nivreeager -ne false -nr false

java -jar -Xmx5000m -Xms5000m maltparser-1.9.1.jar -c parser-all -m parse -i Q4/dev.new.ex.2_gold.conll -o Q4/dev.out.ex.4.b.conll

The best performance setting is the same as setting in Q2. There are some predict tag errors in the best result of Q3. When I change the gold pos tag with the predicted tag, the wrong tags have a bad effect on the dependencies and parse result. Therefore, the labeled attachment score is lower than Q2. Besides, the best parameters in Q2 are suitable in this question because the best result in Q3 is accurate though it isn't 100%.

Q5:

1> Command ------ Get the first 20 sentences of GOLD file and predicted file

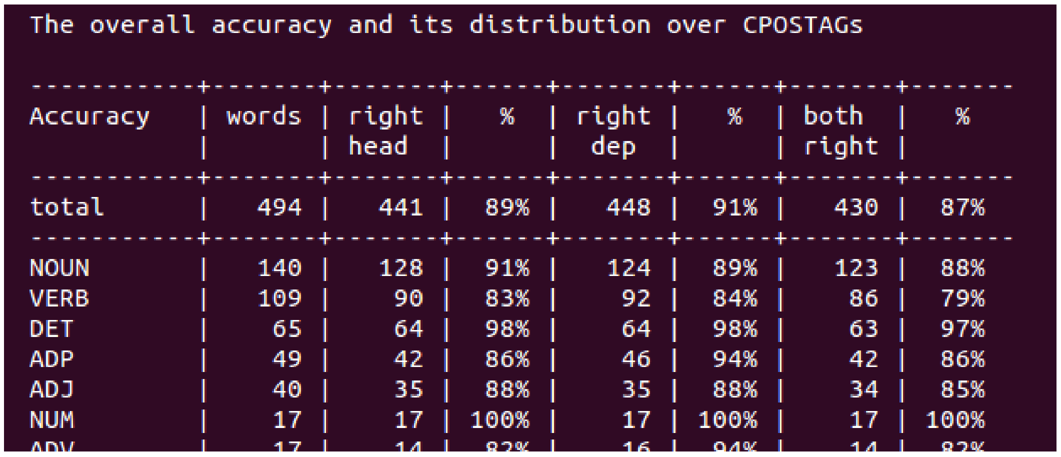
- ➢ cat Q4/dev.new.ex.2_gold.conll | awk '{printf "%s#END#", $0}' |perl -pe 's/#END##END#/\n\n/g' |head -40 |perl -pe 's/#END#/\n/g' >dev.out.ex.5.a.conll
- ➢ cat Q4/dev.out.ex.4.b.conll | awk '{printf "%s#END#", $0}' |perl -pe 's/#END##END#/\n\n/g' |head -40 |perl -pe 's/#END#/\n/g' > dev.out.ex.5.b.conll

2> Evaluation:

- ➢ perl conll-eval.pl -g dev.out.ex.5.a.conll -s dev.out.ex.5.b.conll

3> Analysis:

a. Take NOUN for example. There are 140 noun words and the distribution of their right head is 91%. Besides, the right dependencies are 124 and the distribution is 89%. However, VERB has a lower accurate rate. The right head of a verb only has 83% and the right dependency of a verb is 84%. Other types of words have fewer errors than *verb*. Among these words, NUM has the highest accuracy. The system can give its right head and dependency.

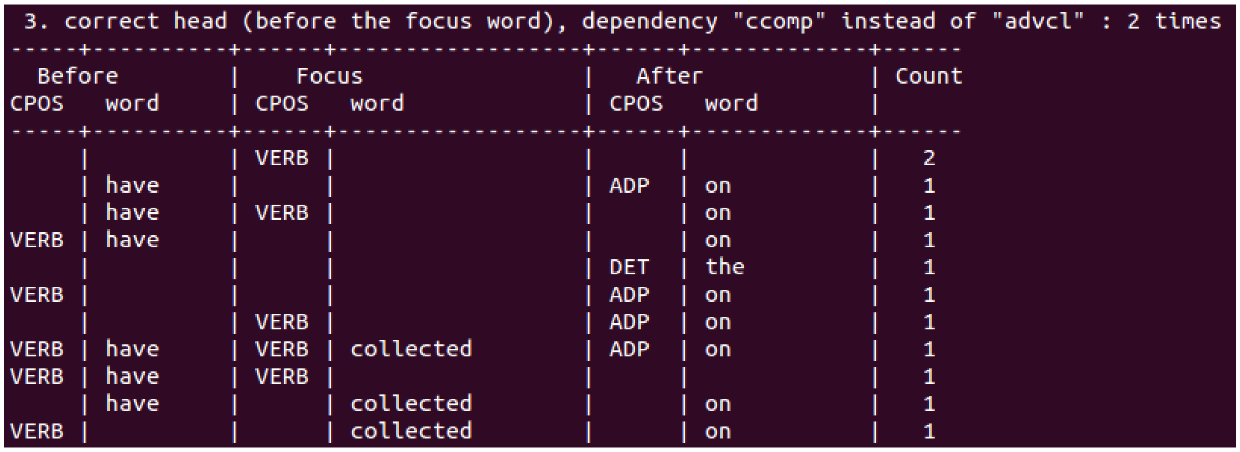


```
The overall accuracy and its distribution over CPOSTAGs

-----------+-------+-------+------+-------+------+-------+-------
Accuracy   | words | right |   %  | right |   %  | both  |   %
           |       | head  |      |  dep  |      | right |
-----------+-------+-------+------+-------+------+-------+-------
total      |   494 |   441 |  89% |   448 |  91% |   430 |  87%
-----------+-------+-------+------+-------+------+-------+-------
NOUN       |   140 |   128 |  91% |   124 |  89% |   123 |  88%
VERB       |   109 |    90 |  83% |    92 |  84% |    86 |  79%
DET        |    65 |    64 |  98% |    64 |  98% |    63 |  97%
ADP        |    49 |    42 |  86% |    46 |  94% |    42 |  86%
ADJ        |    40 |    35 |  88% |    35 |  88% |    34 |  85%
NUM        |    17 |    17 | 100% |    17 | 100% |    17 | 100%
ADV        |    17 |    14 |  82% |    16 |  94% |    14 |  82%
```

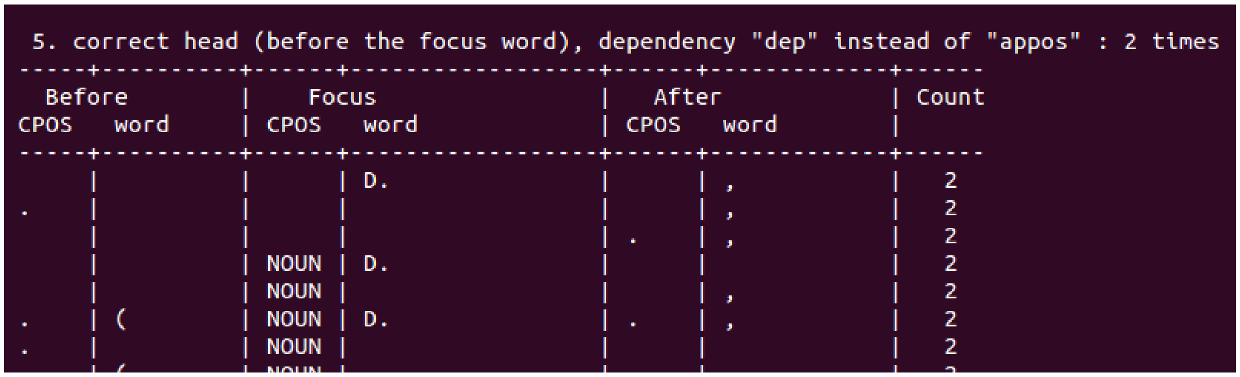
Additionally, there are some kinds of specific errors, six of them are dependencies errors, I will analyze three of them:

- The focus word has the correct head, but the dependency should be *ccomp*, but the evaluate script has the *advcl* dependency *(ccomp: clausal complement, advcl: adverbial clause modifier).*
  Like "*have collected on*" → The original sentence is: *"We would have to wait until we have collected on those assets before we can move forward," he said.*



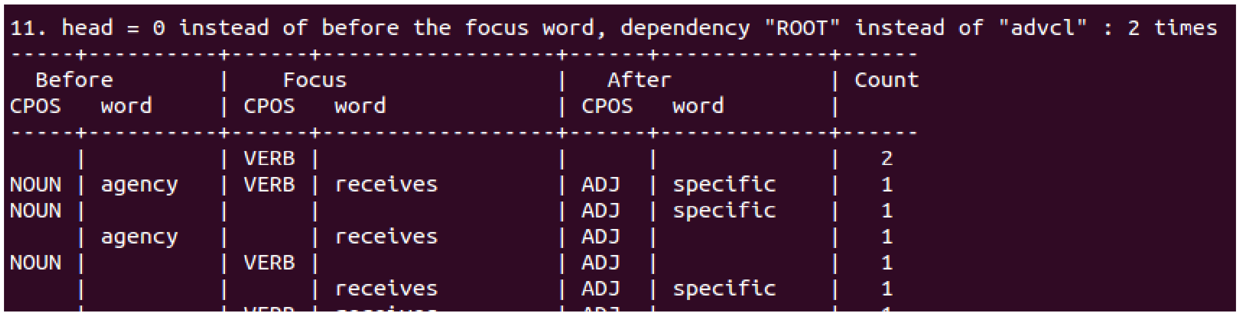
```
 3. correct head (before the focus word), dependency "ccomp" instead of "advcl" : 2 times
-----+----------+------+-----------------+------+------------+------
  Before         |    Focus               |   After          | Count
CPOS   word      | CPOS   word            | CPOS   word      |
-----+----------+------+-----------------+------+------------+------
     |           | VERB |                 |      |           |   2
     | have      |      |                 | ADP  | on        |   1
     | have      | VERB |                 |      | on        |   1
VERB | have      |      |                 |      | on        |   1
     |           |      |                 | DET  | the       |   1
VERB |           |      |                 | ADP  | on        |   1
     |           | VERB |                 | ADP  | on        |   1
VERB | have      | VERB | collected       | ADP  | on        |   1
VERB | have      | VERB |                 |      |           |   1
     | have      |      | collected       |      | on        |   1
VERB |           |      | collected       |      | on        |   1
```

- The focus word has a correct head, but the dependency should be *dep*, not *appos (dep: dependent, appos: appositional modifier)*
  Like "(D.," → The original sentence is: *The bill, whose backers include Chairman Dan Rostenkowski (D., Ill.), would prevent the Resolution Trust Corp. from …*



```
 5. correct head (before the focus word), dependency "dep" instead of "appos" : 2 times
-----+----------+------+-----------------+------+------------+------
  Before         |    Focus               |   After          | Count
CPOS   word      | CPOS   word            | CPOS   word      |
-----+----------+------+-----------------+------+------------+------
     |           |      | D.              |      | ,         |   2
.    |           |      |                 |      | ,         |   2
     |           |      |                 | .    | ,         |   2
     |           | NOUN | D.              |      |           |   2
     |           | NOUN |                 |      | ,         |   2
.    | (         | NOUN | D.              | .    | ,         |   2
.    |           | NOUN |                 |      |           |   2
```

- If the head of the focus word is 0, this word should be *root*, however, the dependency is *advcl* in development set.
  The original sentence is: *The bill intends to restrict the RTC to Treasury borrowings only, unless the agency receives specific congressional authorization.*



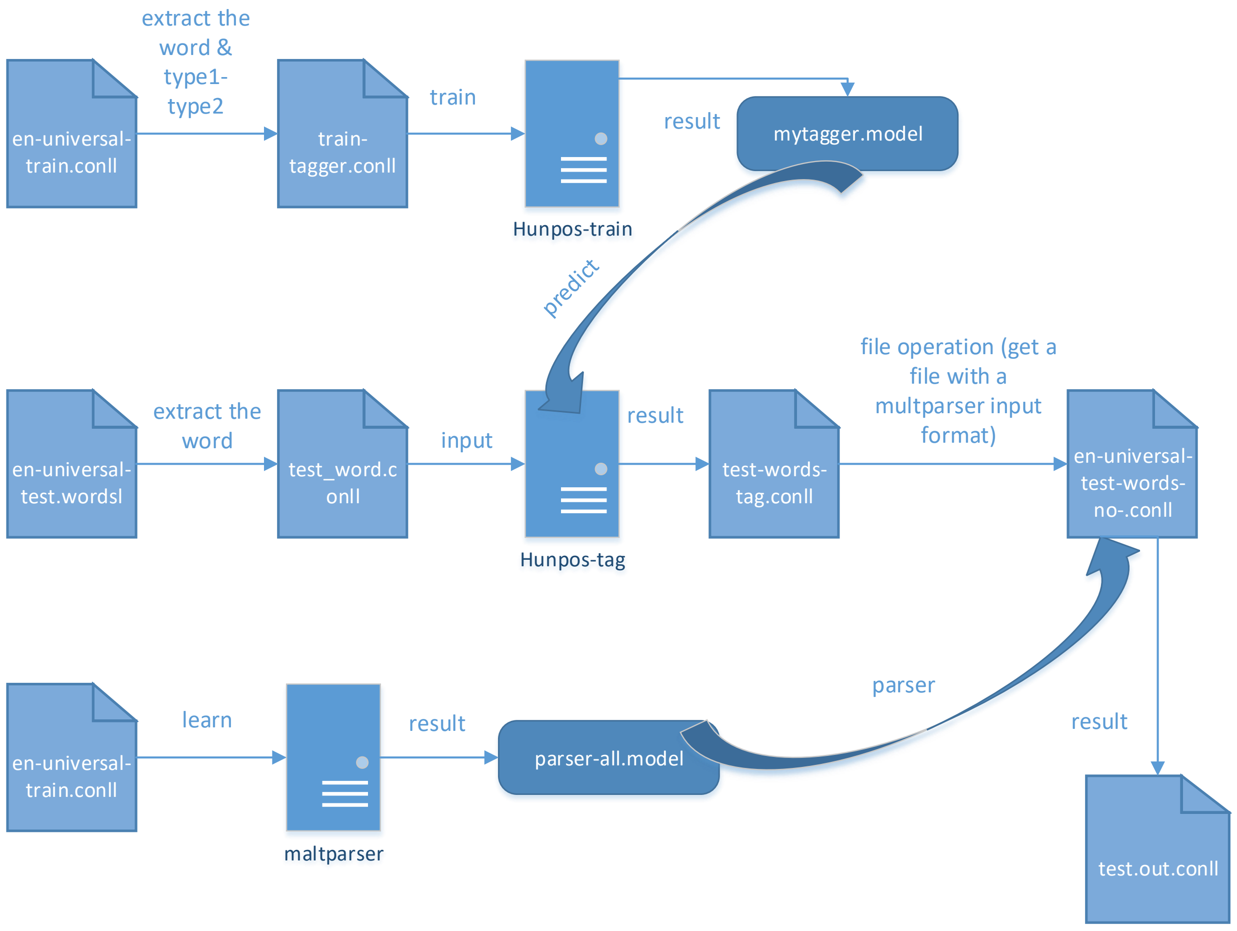
```
11. head = 0 instead of before the focus word, dependency "ROOT" instead of "advcl" : 2 times
-----+----------+------+-----------------+------+------------+------
  Before         |    Focus              |   After            | Count
CPOS   word      | CPOS   word           | CPOS   word        |
-----+----------+------+-----------------+------+------------+------
     |           | VERB |                |      |            |   2
NOUN | agency    | VERB | receives       | ADJ  | specific   |   1
NOUN |           |      |                | ADJ  | specific   |   1
     | agency    |      | receives       | ADJ  |            |   1
NOUN |           | VERB |                | ADJ  |            |   1
     |           |      | receives       | ADJ  | specific   |   1
```

b. 1）Use re-annotation method in grammar rule. Re-annotating each node with the category of its parent category in treebank is able to improve parsing performance. For example, to first error, if the category of "wait" is re-annotated, the error will dismiss because the different between ccomp and advcl is that advcl is modifier for verb and ccomp is an object of the verb.
2) Apply more training data. For second error, the reason causing this error is the model from training cannot identify the meaning of "D.". More training data will solve this problem. Through training more data, the model will know more about abbreviation and the model will get a more accurate result.

Q6:

1> I extract the training set word and tagger as a new training data file in hunpos-train.exe, then getting the predicted tags of test words by using hunpos-tag.exe. After that, by adding some columns, I get a new test file with Maltparser input format. I use Multparser to train the training set and get a parser-all model. At last, applying the model to the new test file and get the result.
The commands are as follows:

- cat en-universal-train.conll | awk '{print $2"\t"$4"-"$5}' | perl -pe 's/\s-//g' >train-tagger.conll
- cat train-tagger.conll | ./hunpos-train -t4 -s20 -f20 mytagger.model
- cat en-universal-test.words | awk '{print $2}' > test_word.conll
- ./hunpos-tag mytagger.model <test_word.conll > test-words-tag.conll
- cat test-words-tag.conll | awk '{sub(/-/,"\t",$2)};1' >test-words-tag_new.conll
- awk -v OFS='\t' 'FNR==NR {a1[NR]=$2;a2[NR]=$3;next}{$3=a1[FNR];$4=a2[FNR]}1' test-words-tag_new.conll en-universal-test.words | perl -pe 's/\t{4}//g' >en-universal-test-words.conll
- cat en-universal-test-words.conll | awk -F"\t" '{gsub("-","\t",$3)}{print $1"\t"$2"\t_\t"$3"\t"$4}' | perl -pe 's/\t{2}_\t{2}//g' >en-universal-test-words-no-.conll
- java -jar -Xmx5000m -Xms5000m maltparser-1.9.1.jar -c parser-all -m learn -i en-universal-train.words -a nivreeager -ne false -nr false
- java -jar -Xmx5000m -Xms5000m maltparser-1.9.1.jar -c parser-all -m parse -i en-universal-test-words-no-.conll -o test.out.conll



2> The parser result of test locates in Q6/test.out.conll