Frequenty Asked Questions about the project

Gideon Maillette de Buy Wenniger
Phong Le

University of Amsterdam
`gemdbw - at - gmail - dot -com`
`P - dot - Le - at - uva - dot -nl`

1 Relevant questions asked in previous years

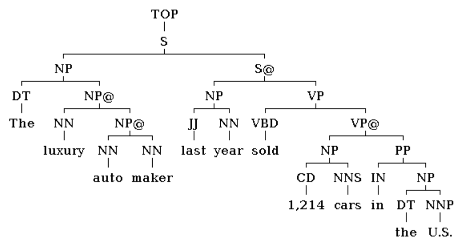
Question 1: There seems to be a discrepancy between the labels that occur in the training data and those in the test set. Particularly in the test set the labels are enriched by extra annotations in the form of -"ExtraAnnotation" e.g. NP-SBJ. How to deal with this?

Answer: Indeed the test set contains information that is not present in the training data. This is something that happens often in our domain. The methodological right way to deal with is, is to keep the test set as is but adapt the evaluation metric. This is exactly what is done in the evaluation program you can use (EVALB). In the readme it also says:

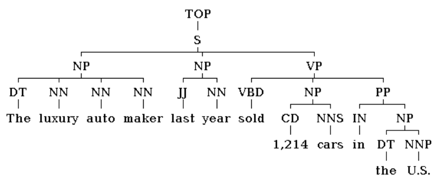
The scorer also removes all functional tags attached to non-terminals (functional tags are prefixed with "-" or "=" in the treebank). For example "NP-SBJ" is processed to give "NP", "NP=2" is changed to "NP".

So in other words, EVALB will take care of this discrepancy for you, by doing the normalization to the less specific format of the training data automatically. (Had this not been the case, and you would be writing your own evaluation, then you would either have to normalize the test set yourself, or - which is better - write an evaluation metric that does the normalization internally without touching the actual test set).

Question 2: The special binary rules containing the @ symbols sometimes can generate a loop, for example when an np@ is followed by a set of children containing again an np@, this seems to be sub-optimal?



(a) A binary tree



(b) The de-binarized tree

Figure 1: Illustration of the de-binarization proces

Answer:

The node labels containing @ are artificial and have to be removed from the output of the parser before evaluation. In principle usage of the same label e.g.NP@ followed again by NP@ might not be optimal for performance (other labeling schemes are imaginable), but it will not give problems in parsing, since under this scheme at least all trees in the treebank will be derivable from the generated grammar (so while another binarization label scheme might lead to even better results, this one certainly works). Regarding these labels also:

If you think that repeating the same label recursively is suboptimal, then you can apply your own renaming *but* you have to explain what you did in the report and justify your choice (some of the choice could affect your parsing precision/recall positively but other negatively). Also the generated trees will have to be de-binarized by flattening the parts of the tree containing the @s. This means putting all the children added starting from the first @ (see definition of project) directly under the parent node which had the first NODE@ in the sequence as a child.

For example, for the fourth tree in the train corpus, we have:
(TOP (S (NP (DT The) (NP@ (NN luxury) (NP@ (NN auto) (NN maker)))) (S@ (NP (JJ last) (NN year)) (VP (VBD sold) (VP@ (NP (CD 1,214) (NNS cars)) (PP (IN in) (NP (DT the) (NNP U.S.)))))))) (see Figure 1a).

Which should be de-binarized as:
(TOP (S (NP (DT The) (NN luxury) (NN auto) (NN maker)) (NP (JJ last) (NN year)) (VP (VBD sold) (NP (CD 1,214) (NNS cars)) (PP (IN in) (NP (DT the) (NNP U.S.)))))) (see Figure 1b).

Question 3: My parser returns a most likely derivation (tree) that does not start with “TOP”, is this OK?

Answer: No. "TOP" is the dedicated start symbol in our grammar, which means all trees must start from top. To be more precise, any coverage of a full sentence that does not start from "TOP" is not complete/legal : it could not have been generated under the grammar, as that requires all productions to be started from "TOP". However, this also means that you cannot just *"add TOP as a special last step to make things complete"*. The production of "TOP" is a normal production and must follow the same rules of all other productions, i.e. when $TOP \rightarrow Something$ is added this may only be done when the rule $TOP \rightarrow Something$ is present as a rule in the grammar and $Something$ has already been inferred (added to the chart) for the chart entry that covers the full sentence.

Furthermore as top itself always only occurs as the root of trees in the treebank, it will only occur in the left side of rules and will never itself be produced by another symbol. So when top is added for chart entries that only cover part of the sentence - which could in principle be possible - these labels will never be seen back in a parse of the full sentence.

Questions 4:

Questions 4.1: *I was reading your slides about CYK parsing and found myself not fully understanding the algorithm. So for example, this is in the initialization part:*
//Add a node to the Inferred nodes table ; Add Node(Nj) to Chart[i][i].Nodes
//Add inference of the node from the word ai to added
node;
Chart[i][i].Nodes.get(Nj).Add(Inferrence(ai , null))
The first part (Add Node(Nj)) i do understand, but what does the second part mean?
To be more precise, what is the inference part?

Answer: The inference part serves to store the multiple ways a certain node/label can be *inferred* : build from different splits and different combinations of two (or one) label(s) that are combined. As there are possibly many rules starting from the same label on the left side of the rule, for any entry in the chart a label may potentially be build up in many different ways. These different ways have to be kept track of, as they correspond to different (partial) derivation trees and for finding the most likely parse/tree to different probabilities. It is not enough for the Viterbi algorithm (for finding the maximum likelihood tree) in step 3 to have just one inference for a combination (chart-Entry,label), a list of inferences for the label for that chart entry is kept as well as an indication of the one that is best and its probability. This may be more clear from the Standford slides, were the computation of the probabilities is already included.

Questions 4.2: *My second question is about my understanding of the algorithm. I thought that the algorithm starts with the words of the sentence and try to find the labels learned in the learning phase. It then places these labels on the diagonal entries of the chart. But what happens then? For what I belief, you then take different spanlenghts? So, 2 words, 3 words until all the words?*

Answer: You are right, the algorithm starts bottom up from the words. It then loops over increasingly different spanlengths and try to infer/"proof" labels for them using the rules available in the grammar and the labels that have already been added/proved for smaller chart entries. You can understand this intuitively: the rules of the grammar always rewrite a symbol into 2 (or 1 for terminal rules) symbols. In other words, the length of what is inferred by applying more rules always becomes bigger, never smaller.

The algorithm is constructed in such a way that when it finishes we have a guarantee that all labels that could have been inferred have been, and all trees that can be build while inferring those labels have been (*completeness requirement*). When starting bottom up with the smallest span length, and adding all labels/nodes that can be inferred for that span length, then incrementally increasing the span length and repeating, this assures that nothing is missed and the *completeness requirement* mentioned before can be guaranteed. The reason being that any node in a chart entry can only be build up from parts that consist of nodes in smaller chart entries that partition that bigger chart entry. In other words, the structure of the grammar and the implicit assumptions it makes (e.g. with respect to the fact that a label on the left side is always rewritten as a consecutive/adjacent list of labels on the right side - no interruptions allowed) assures that such a bottom up parsing algorithm will be correct. Of course there are other (i.e. top down) parsing algorithms (e.g. the Earley parsing algorithm) that are also correct, they achieve the same correctness properties by similar mechanisms, but they are not the topic of this lab.

Questions 4.2:
After reading your mail, I have tried to implement the cyk algorithm. So far I think I understand the basic concept of of the algorithm, however the result was not quite as I would expected.

So what did I do: Step 1: initialize the chart with spanlenght = 1.
Step 2: for each chart entry, take the permutation of the lower 2 chart entries and find the left-hand-side of each 2 label combination. Put this result in the chart entry.

OK, so at this point everything goes well. However this result in the last chart entry consisting of labels which are not all S, so also ADJ etc. To deal with this I have thought about a solution to then go top down trough the chart and see which entries results in a real tree. In other words which goes from TOP to words. Now I was wondering if this is a correct approach or that I should reconsider my implementation?

Answer:

You are on the right track. You are right about the fact that there is a unary production $TOP \rightarrow SOMELABEL$ at the start of the trees in the treebank. This means that the first labels added that cover the full sentence, are not necessary the final starting symbol of the trees. To deal with this, as a special case you have to reapply the rules where possible in a second round for the top chart entry. This will add the productions/inferences $TOP \rightarrow SOMELABEL$, making the trees complete. Notice that this is a sort of special case of reapplying the rules for the biggest chart entry, similar to the one that appears at the leaf level/initialization (chart entries of length 1). There you have (sometimes) first a POS-tag producing the word, and then another unary (one child) production that produces this POS-tag. In a similar way to cover that, you have to perform the initialization step in effect 2 times (but of course you should on the second step only add inferences producing proper labels, not again add productions of the words). A similar question has been posted also before and you can check it in the FAQ: see question 3.

Your proposed solution is in the right direction in other words, only it is not really "going top down" what is required, just another round of chart building for the chart entry covering the full sentence.

Notice that there is also a more "generic way" to deal with unaries, you can find this in the standford CYK slides listed in the lab site. This comes down to adding unary productions as long as there are still new ones added after adding the binary productions for every chart entry. That also assumes there are no (combinations of) unary rules that can lead to cycles, but this is the case for our treebank.

Questions 5:

Questions 5.1: *What does XP stand for in the task of the second exercise ($TOP \rightarrow XP$)? Does it represent a non-terminal which was stored in the parsing table (after applying CYK) at position (1, N), where N is the length of the sentence? Or does it represent a tree without the "TOP" node, a tree that is producing our test sentence?*

Answer: XP stands for any non-terminal that you find under TOP in the treebank. In fact the original treebank did not contain TOP but has root nodes labeled with e.g. S, SQ, FRAG and the like. I added TOP to make life easier for you since an SCFG needs one start nonterminal. I hope this answers your question.

Questions 5.2: *In the beginning of the task, it is said that the test sentences are limited to 15 words or less. However, I saw that some sentences have more than 15 words in the test file. Is this the correct one we should work with?*

Answer: Yes, what you need to do is to filter out the long sentences yourself (your parser can skip all sentences of length ¿ 15 and write out an empty tree (TOP (POS word1) (POS word2)....(POS wordn)) for every such sentence, where POS is a fake POS tag category). You do this because when you evaluate your parser using EvalB (or EvalC) you'll need exactly the same sentences in parser output as in the gold test set (the correct trees).

Questions 5.3: *Should the parse-forest for e.g. sentence U under G, if this sentence U is from the training data, contain also the original training tree for this sentence?*
Answer: Well, this is a question to you, actually. If you've extracted the productions from the training trees, would this CFG be able to generate the original trees in the chart, among others?

Question 6: Following discussions had with several colleagues, Im writing to ask for clarifications regarding the unaries in the training data set.

In the data description file posted on Blackboard, it is stated that: Unaries have been pulled out for you: an unary production X Y in the tree is now marked as a single node Xfile is suggestively named wsj.02-21.training.nounary. Moreover, during the previous lab, I asked the TA if we should pay attention/implement the handle unaries part from the Stanford slides on CYK, and he confirmed that this is not the case, since the unaries have been extracted.

However, this doesnt seem to be the case. The unaries seem to have been pulled out, except for the last 2 levels of the trees. The last level (containing the terminals) is normal to have only unaries (ie the words), but many times, the previous level is unary as well. There are plenty of examples in the training data set. Here is one of them, sentence number 2 (see: (NP (NNP Elianti))):

(TOP (S (NP (NNP Ms.) (NNP Haag)) (S@ (VP (VBZ plays) (NP (NNP Elianti))) (. .))))

If we ignore unaries, and since CYK builds a parse forrsts assuming a binarized tree, than the parse forest produced by our parser for this sentence after training will not contain the original tree depicted above. This however contradicts what you suggested to one of my colleges, that the parse forest should contain the original tree (ie should contain non-terminal unaries). Looking forward to your clarification.

Answer:

Indeed the unaries $A \rightarrow B$ where both A and B are nonterminals that dominate other nonterminals have been removed from the data to avoid cycles. The case of B being a pre-terminal POS-tag like your example of $NP \rightarrow NNP$ remains in the treebank: this is a much simpler case than the general case and will not lead to cycles. That said, if you wish to simplify further, the remaining "POS tag level unaries" can also be removed (but that could harm your results later). I think that remaining two kinds of unaries (TOP level and above POS level), and the knowledge that the remaining unaries do not create chains of nonterminals longer than one level, should simplify the algorithm (no need to worry about the risk of cycles).

I hope this answers your question.

Question 7: I am not sure about what the output of step 2 should look like. My understanding is to print out all the productions that cover one possible parse tree for each sentence. So what I got for the first sentence of the test data is :

```
TOP  S
S DT VP
DT No
VP , VP@
, ,
VP@ NP NP
NP PRP NP
PRP it
NP VBD NPVBD was
NPRB n't
JJ Black
NP NNP .
NNP Monday
```

Is it correct? Should the output be in a tree format as training data?

Answer:
In the assignment for step 2 it reads

" Output: All productions $TOP \rightarrow XP$ that cover the whole sentence U (entry 0, n ,where n is number of words in U) according to your CYK parser. In the process, the CYK parser builds a parse-forest $CYK(U)$ for input U ."

what this means is that your parser should produce implicitly the trees consistent with the sentences in the form of a parse forest. In your output you have to only report the productions at the top of this parse forest i.e. $TOP \rightarrow Something$. The reason we ask for this and not for an enumeration of the trees is that such an enumeration could become prohibitively large for many sentences. (On the other hand wihou the Viterbi algorithm, which is asked only from step 3, a choice for one best parse can not be made yet.)

What you seem to produce now is all productions over all chart entries , or more likely just all productions for one tree. This is close but not exactly what is asked for step 2. The parser should implicitly build all the trees in the form of the parse forest for each sentence, and you report the $TOP \rightarrow Something$ productions for these parse forests, i.e. the productions contained in the entry [0,n] of the chart that start with TOP.