CMPT 825: Natural Language Processing **Spring 2006**

Lecture 16 — 17, Mar 7-9, 2006

Lecturer: Anoop Sarkar *Scribe: Javier Thaine*

16.1 Required readings

[1] J. Goodman. *Parsing Algorithms and Metrics*, Proc. ACL 34th Annual meeting, Association for Computation Linguistics, New Brunswick, NJ, 1996, pp. 177-183.
[2] M. Collins. *Three Generative, Lexicalised Models for Statistical Parsing.* Proceedings of the 35th Annual Meeting of the ACL (jointly with the 8th Conference of the EACL), Madrid, 1997.
[3] E. Charniak. *Statistical parsing with a context-free grammar and word statistics.* Proceedings of the Fourteenth National Conference on Artificial Intelligence AAAI Press/MIT Press, Menlo Park (1997).

We begin by discussing the Goodman paper and the evaluation metrics for parsing that it presents. While Collins and Charniak both try to find the best trees, Goodman talks about taking the measure of a parse for comparison purposes.

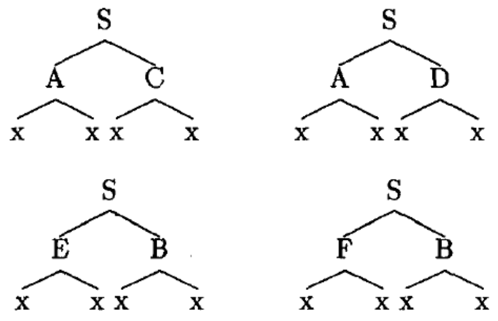
16.2 Labelled Recall Rate (LR)

Suppose we have the following grammar:[1]

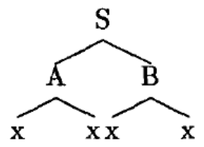
$$\begin{array}{rcll} \text{S} & \rightarrow & \text{A C} & 0.25 \\ \text{S} & \rightarrow & \text{A D} & 0.25 \\ \text{S} & \rightarrow & \text{E B} & 0.25 \\ \text{S} & \rightarrow & \text{F B} & 0.25 \\ \text{A, B, C, D, E, F} & \rightarrow & \text{x x} & 1.0 \end{array}$$

This grammar generates the following trees:

[1]Most images are taken from [1].

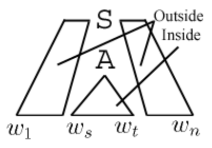


Note that A occurs twice in the trees and C just once. Suppose you are building a tree from scratch. No matter what you do, you always get S right. Half the time you get A as a correct choice. To maximimize the probability of each node being correct, you could independently pick the constituents that occur the most frequently:



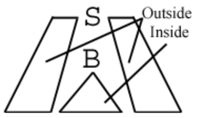
“The probabilities of each node being correct are S: 100%; A: 50%; and B: 50%. [...] The expected value of [the LR] for this tree is 2.0, the highest of any tree. This tree therefore optimizes the Labelled Recall Rate.” [1] The LR is the number of correct labelled constituents in the test parse tree over the number of nodes in the correct parse tree. In other words, we don’t know exactly which is the “correct” parse we are aiming for, but we do know that it must have S at the top, and it has a 50% chance of having A as S’s left constituent and a 50% chance of having B as S’s right constituent.

Suppose we want to find the best labelled bracket for each constituent. The basic idea is to divide a tree as follows:



We want to pick the A that occurs in the overall best tree. One way is to run a PCFG, find all the parses, compute the probabilities and find the best tree. Goodman suggests a slightly different idea. Suppose A derives some string of words $w_s ... w_t$ (denoted as w_s^t). What is the "inside" probability? Just the probability of tree A. When you compute it, you can think of it as $P(A \overset{*}{\Rightarrow} w_s^t)$ (where the * means that the expansion from the LHS to the RHS of the rule takes place in 0 or more steps). The outside probability is $P(S \overset{*}{\Rightarrow} w_1^{s-1} A w_{t+1}^n)$.

Now pick a different node and treat it independently:



B might be the best choice for one constituent, A the best for another, even though there is no rule that combines the two. This is very different from what you would look for to compute the best parse: you would start from S, find all the trees produced by the PCFG rules, and pick the best one.

Using these ideas, the tree T_G with the highest expected value for the LR can be found using [1]:

$$T_G = \arg\max_T \sum_{(s,t,X) \in T} \frac{P(S \overset{*}{\Rightarrow} w_1^{s-1} X w_{t+1}^n) P(X \overset{*}{\Rightarrow} w_s^t)}{P(S \overset{*}{\Rightarrow} w_1^n)}$$

Goodman provides a dynamic programming algorithm to compute such optimal LRs, and explains how to modify the algorithm to generate the best parse.

16.3 Other common evaluation metrics

The Labelled Precision Rate (LP) is the number of correct labelled constituents in the test parse tree over the number of nodes in the test parse tree itself. This makes it a much less robust measure than LR, since you can, for example, get 100% precision by just guessing the top node S. Hence LP is not a very good measure by itself; LR is also needed. Some people just measure LR - it is harder to fool it. To get 100% recall no matter what the reference (i.e. "correct") tree is (bracketing being important), the trick is not to return a tree but a forest of all possible parses, which contains constituents of all sizes.

Other commonly used metrics are: Bracketed Precision (BP), Bracketed Recall (BR), CB (the average number of cross-brackets per sentence), 0CB (the percentage of sentences with zero cross-brackets), and 2CB (the percentage of sentences with 2 or fewer cross-brackets).[3] Cross-brackets are defined in [4] as "the number of crossed brackets (e.g. the number of constituents for which the treebank has a bracketing such as ((A B) C) but the candidate parse has a bracketing such as (A (B C)))." Typically people report all these measures. The evalb tool (used in homework 7) returns all these values.

16.4 Collins Model 1

We now discuss the Collins Paper [2]. Collins presents three generative models for parsing (homework 7 is related to Model 3). The models are *lexicalised* - from doing PP-attachment using PCFGs it was learned that *words are important* (in fact, it seems like if you have a new and radical method to do PP-attachment, it is always possible to use it to improve accuracy). The task is, given a Penn Treebank input sentence, to compute the parse tree and compare it to the gold standard. We seek the best tree for a given sentence, $T_{best} = \arg\max_T \mathcal{P}(T|S)$ where, $\mathcal{P}$ is a probability distribution modeled on the treebank training data. We have that

$$\arg\max_T \mathcal{P}(T|S) = \arg\max_T \mathcal{P}(T,S)/\mathcal{P}(S) = \arg\max_T \mathcal{P}(T,S)$$

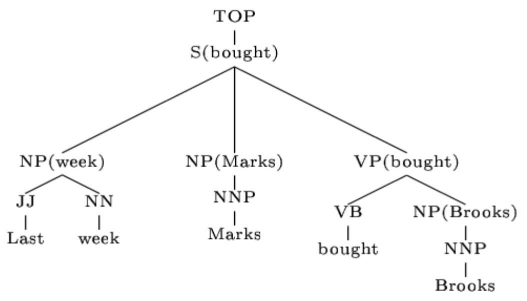
since $\mathcal{P}(S)$ is fixed because the sentence is fixed. The model should come with some way of enumerating all the trees (to find the best one).

For a set of CFG rules of the form $LHS_i \rightarrow RHS_i$, we have that $\mathcal{P}(T,S) = \prod_i \mathcal{P}(RHS_i|LHS_i)$. Note that the following must hold for these rules: given a particular LHS which maps to one or more RHS, e.g. $A \rightarrow \alpha|\beta|...|\gamma$, we must have that $\sum_{RHS} \mathcal{P}(A \rightarrow RHS) = 1$ where $\mathcal{P}(A \rightarrow RHS) = \mathcal{P}(RHS|A)$.

Here is a Treebank tree. We want to use a generative model that produces trees like this.[2] The idea is to use a context-free parser to find the best set

[2]Image taken from [2].

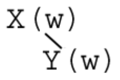
of rules, compute for each tree the probability using the rules, and find the best one.



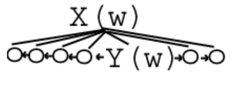
`TOP`	`->`	`S(bought)`	
`S(bought)`	`->`	`NP(week)`	`NP(Marks)`
`NP(week)`	`->`	`JJ(Last)`	`NN(week)`
`NP(Marks)`	`->`	`NNP(Marks)`	
`VP(bought)`	`->`	`VB(bought)`	`NP(Brooks)`
`NP(Brooks)`	`->`	`NNP(Brooks)`	

Here is the generative story presented by Collins. For each of the non-terminals, generate all the words. There are several choices for each word, but you generate one at a time. Suppose that in a particular tree, node X has n children. This is similar to an $(n-1)$-gram, i.e. we can generate the nth child Y of X using the probability $\mathcal{P}(Y|$ *the other* $n-1$ *children of* $X)$. How big does this n get? If $n = 3$, a trigram model, we can deal with it. If n is large (e.g. 20), then you have the same sparse data problems as with n-grams. Generating the words then becomes a big problem.

Collins solves this problem by changing the generative story a bit. Every node has a special child node called the head node. The word associated with the head node is called the head word and is shared with its parent (in the following, w is the head word):



Starting from the head node, we generate nodes to the left and right, treating it as a tagger:



You could use a bigram model to generate these, or do as Collins suggests: generate each node based on its relation to the head. Furthermore you must have a left stop and a right stop symbol to halt the generation process. More precisely, we make the following 0^{th} order Markov assumptions [2]:

$$\mathcal{P}_l(L_i(l_i)|H,P,h,L_1(l_1)...L_{i-1}(l_{i-1})) = \mathcal{P}_l(L_i(l_i)|H,P,h)$$

$$\mathcal{P}_r(R_i(r_i)|H,P,h,R_1(r_1)...R_{i-1}(r_{i-1})) = \mathcal{P}_r(R_i(r_i)|H,P,h)$$

where $\mathcal{P}_l$ is the probability distribution used for the generation of nodes to the left of the head node H, L_i is the ith node to the left of H, l_i is the word associated with L_i and h is the head word associated with H, and P is the parent word (from whom h was inherited). Similarly for $\mathcal{P}_r$, etc. on the right side of the head word.

For example, in the tree shown earlier, "the probability of the rule S(bought) → NP(week) NP(Marks) VP(bought) would be estimated as" [2]

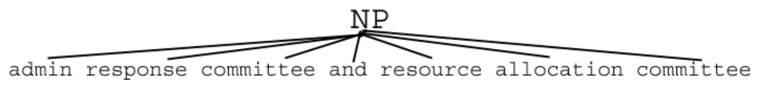
$$\mathcal{P}_h(\text{VP}|\text{S,bought})\times\mathcal{P}_l(\text{NP(Marks)}|\text{S,VP,bought})\times\mathcal{P}_l(\text{NP(week)}|\text{S,VP,bought})\times$$

$$\mathcal{P}_l(\text{STOP}|\text{S,VP,bought}) \times \mathcal{P}_r(\text{STOP}|\text{S,VP,bought})$$

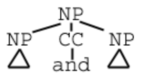
(The head node has its own probability $\mathcal{P}_h$ of being generated. A stop symbol is included so the model stops generating left or right nodes when it is generated.) Here we are taking a top-down generative approach, i.e. we are assuming the top-level nodes are generated and lexicalised first. In actual computation we would follow a bottom-up approach.

We assumed each node is generated independently of its neighbours. In some cases we might want some more context. For example, if the tree is generated depth-first then one could as additional context use child nodes of the neighbour of the node being generated. Collins came up with pretty robust contextual features, e.g. distance (an interesting way to record context). The Charniak paper [3] has a different way of dealing with the sparse data problem.

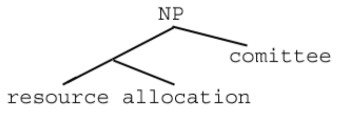
By accident the Collins model deals with another problem in the Penn treebank. Noun phrases are often left flat, as in:



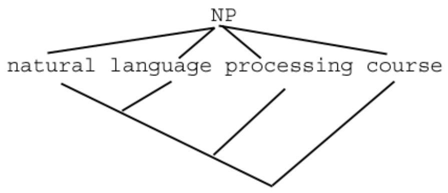
We can use rules such as the following to help parse NPs. This rule will be quite rare (it might appear once in the corpus):



We can also make the argument that an NP subtree is always binary branching, as in:



However even with such aides it sometimes is hard to tell exactly what the meaning is. We could have parses such as

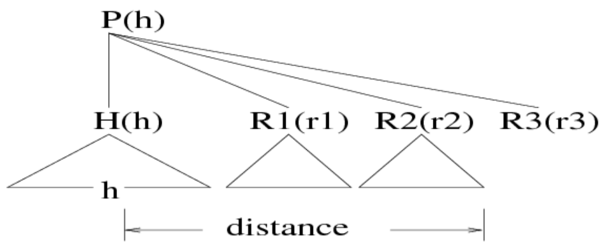


There are many other ways of parsing that sentence. Some people in the mid 90's (e.g. Lauer) tried to figure out exactly how to parse NPs. However, since you don't get rewarded for this in evaluation, nobody does it. In fact, one is penalized in precision and recall by doing this. However, it might matter in an application.

In NPs we don't know which word is the head. In practice the last one is the head, and the rest are generated using a bigram context. It doesn't pay to do this for the S node, but it does for NP nodes.

16.5 Distance as context

Thus far we gave the model the ability to do 0^{th}-order Markov Chains. We want to take back some of this freedom. This is done by introducing the notion of distance (figure 2 of [2]):



"Figure 2: The next child, $R_3(r_3)$, is generated with a probability

$$\mathcal{P}(R_3(r_3)|P, H, h, distance_r(2))$$

The *distance* is a function of the surface string from the word after h to the last word of R_2, inclusive. In principle the model could condition on any structure dominated by H, R_1 or R_2." [2]

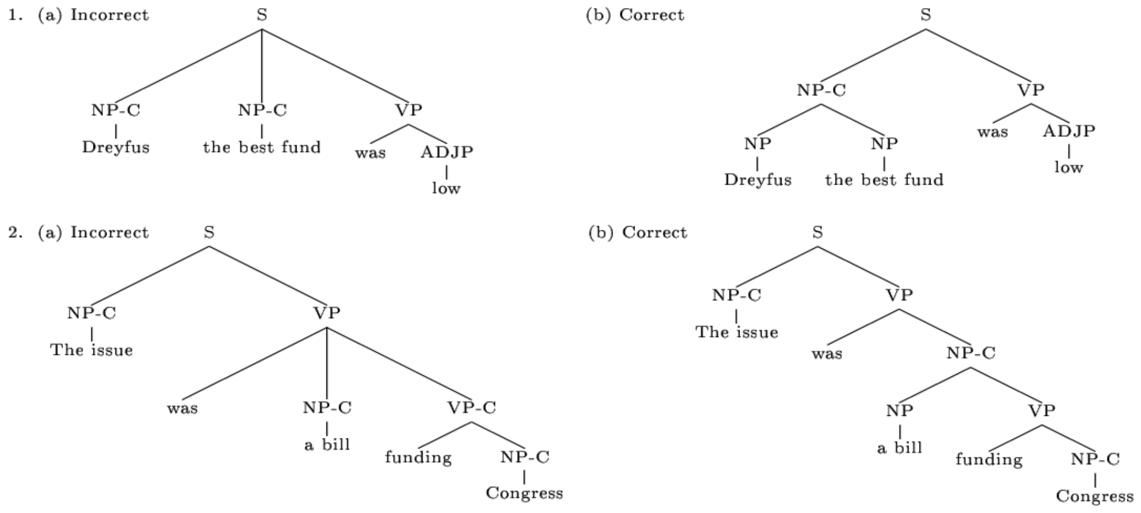
This simple distance function is a way of encapsulating the context and is fairly straightforward to add to the model. It was invented by Collins in this probabilistic sense. It takes into account any verbs and punctuation in the way. [2] does not explain exactly how this works, but we do know that the parser uses punctuation to get a more accurate parse. However, Collins lies a bit: he got his accuracies on other things not in the paper.

When trying to improve accuracy in parsing, the distance is often not enough. We could get a tree where at the left of the VP there is nothing. This is not good, since in English you typically have something before the verb. In English you must put a subject first - we can't just have a sentence such as "Snowing." Collins simplified the process: use the rules in section 2.2.1 of [2] to determine if a word is a complement, and if so add a -C to its part-of-speech tag. This distinction between adjuncts and complements is necessary for proper processing of subcategorisation frames, and may help improve parsing accuracy. This is what Model 2 is concerned with.

An important thing to keep in mind is that tagging something as a verb is itself ambiguous. It is probabilistic, *not* deterministic. In fact, we don't always know if a string contains a verb.

16.6 Model 2 and subcategorisation frames

Subcategorisation frames specify how many complements of each kind a word requires, and is divided into left and right frames for specifying the complements that must appear to the left and right of the word. Model 2 uses subcategorisation frames as part of the generative process in order to avoid bad parses in which two complements are deemed to be independent of one another. Figure 4 of [2] demonstrates this:



In (1a), the two NP-C's are independently generated as subjects to "was," which is incorrect. In (1b), using the knowledge from the subcategorisation frame for "was" that it most probably requires one left complement (or modifier), the correct parse is generated. Similarly, in (2b) we use the knowledge that "was" most probably needs one right complement to produce the correct parse.

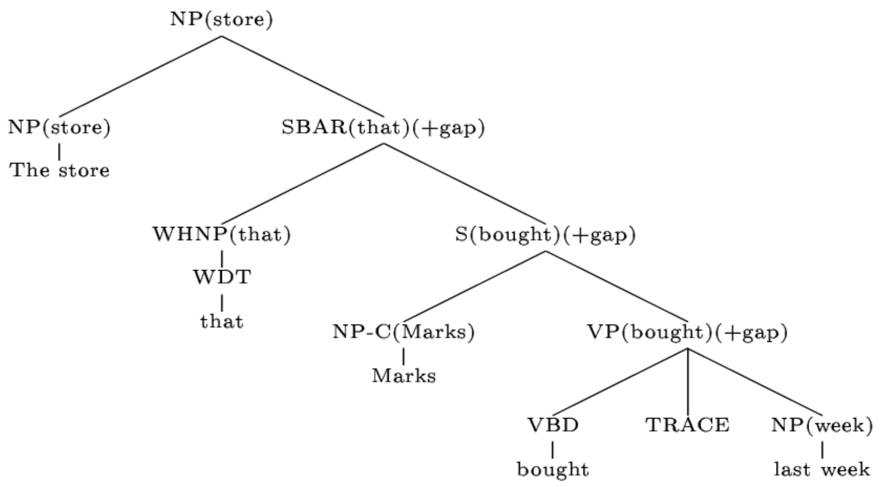
Here is an example of a probabilistic use of subcategorisation frames. Refer to the parse of "Last week Marks bought Brooks." Suppose the word "bought" has a 90% probability of having an NP as a left complement, and 10% chance of having no left complement. We denote this as:

$$P_{lc}(\{NP\}|S, VP, bought) = 0.9$$
$$P_{lc}(\{\}|S, VP, bought) = 0.1$$

The curly braces denote the use of a multiset, i.e. a set in which duplicates are permitted. Thus the probability of generating the ith word (and its corresponding part of speech) to the left of "bought" would be denoted $P_l(L_i, l_i|H, P, h, dist, \{NP\})$, i.e. it would depend on the left subcategorisation frame. If we predict an NP at this position, then for the remaining left words, everything else is the same except that the NP is removed from the multiset since it has been used already. To generate the next word, we would use $P_l(L_{i+1}, l_{i+1}|H, P, h, dist, \{\})$, i.e. we are now taking into account the probability that the subcategorisation frame is empty.

16.7 Model 3

Model 3 of [2] takes into account wh-movement through the use of traces. Most people doing parsing, including Charniak, ignore this. This is linguistically important. For example, the relationship between a complement and the verb it modifies can be stretched over a great distance (e.g. "Marks [...] bought"). A dependency that is arbitrarily far away does not fit well with our previous models, so we use a *trace* to keep track of it. In order to produce a trace at the right position, a +gap is added when building the tree which follows the route down the tree taken by "store" to "bought." In linguistics this is called "movement" - it is the relationship between "store" and "bought", and goes through these steps:



Most computer scientists think linguistics is a hindrance because you spend a long time thinking about it and it doesn't give you more than a 0.5% performance increase, since you don't get evaluated on traces, for example. The Charniak parsers don't use them at all. However these can be important in applications.

Table 1 of [2] shows that, as you condition on less information, chances are your model becomes more biased and reliable. Note that the smoothing used by Collins is exactly the same as Jelinek-Mercer Smoothing: a linearly interpolated backoff smoothing model.

Table 2 in [2] presents accuracy results. It is standard in parsing literature to report results for sentence lengths of ≤ 40 words and ≤ 100 words. For some models this is not a good way to present results since they do not scale well: with an increase in the number of words, results gets worse. It's an engineering feat to get a model to work well for 100 word sentences.

16.8 Charniak's model

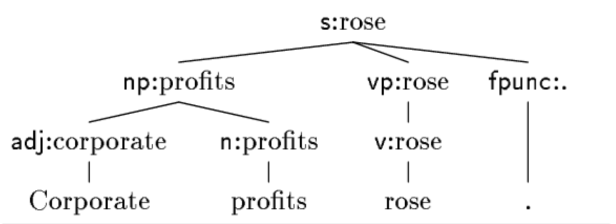
Charniak [3] uses a PCFG and words just like Collins. The core of the model is equations (2) and (3), which are backoff smoothing models. Understand these and you understand the model:

$$p(s|h,t,l) = \lambda_1(e)\hat{p}(s|h,t,l) + \lambda_2(e)\hat{p}(s|\mathbf{c}_h,t,l) + \lambda_3(e)\hat{p}(s|t,l) + \lambda_4(e)\hat{p}(s|t)$$

$$p(r|h,t,l) = \lambda_1(e)\hat{p}(r|h,t,l)+\lambda_2(e)\hat{p}(r|h,t)+\lambda_3(e)\hat{p}(r|\mathbf{c}_h,t)+\lambda_4(e)\hat{p}(r|t,l)+\lambda_5(e)\hat{p}(r|t)$$

$p(s|h,t,l)$ is the probability that s is the head-word of a node of type t (e.g. NP) whose parent is of type l and is headed by word h. p is a probability while the $\hat{p}$'s are estimates obtained from the training data. $\mathbf{c}_h$ is the cluster (word class) h belongs to. The $\lambda_i(e)$ are interpolation parameters where e is "an estimate, given the amount of training data used, of how often one would expect the particular concurrence of events", i.e. given the amount of training data used, how often we would see that particular s as the head of a node of type t under a node of type l headed by word h.[3] This is not described further in [3] - the reader is instead referred to [5] for further details. $p(r|h,t,l)$ is similar to $p(s|h,t,l)$; r is the grammar rule used to expand the constituent t. That is, $p(r|h,t,l)$ is the probability that rule r is used to expand t given also h and l. Note that $\hat{p}(r|t)$ is the standard PCFG probability. Charniak's parser uses a CFG parser as a component of this model.

We clarify this further with an example from [3]. Suppose we have the following parse and probability estimates:



	$p(\text{prf} \mid \text{rose}, \mathsf{np}, \mathsf{s})$	$p(\text{crp} \mid \text{prf}, \mathsf{adj}, \mathsf{np})$
$\hat{p}(s \mid h, t, l)$	0	0.2449
$\hat{p}(s \mid \mathbf{c}_h, t, l)$	0.00352223	0.0149821
$\hat{p}(s \mid t, l)$	0.0006274	0.00533
$\hat{p}(s \mid t)$	0.000556527	0.004179

The probability estimate of "profits" being the head of an NP given the head of the sentence "rose" is 0 because such a case was never encountered in the training data. However, if we replace "rose" by its cluster, the latter was encountered in the training data. After that, if we condition on less information, we get lower probability estimates. This can be interpreted as follows, starting from $\hat{p}(s|t)$ and working our way up. It is not very likely for any particular NP to be headed by "profits." It is somewhat more likely, though, that an NP is headed by "profits" when that NP's parent is the top (or sentence) node. It is even more likely for NP to be headed by "profits" if on top of this, the head word of the sentence is in the same cluster as the word "rose." The example on the right column of the table has a similar interpretation, and in this case the specific instance of "corporate profits" with corporate being and adjective and "profits" being the head of the NP "corporate profits" is found in the training data, so that probability estimate is nonzero.

Note that $\hat{p}(s|h, t, l)$ is different from the probability used in Collins because it is based on parent/child relationships as opposed to siblings. Unlike in the Collins parser, we don't decompose rules. Why? This is mysterious since for some sentences you might not get a parse.

Additional References

[4] D. Jurafsky and J. H. Martin. *Speech and Language Processing*. Prentice Hall, 2000.

[5] E. Charniak. *Expected-Frequency Interpolation*. Department of Computer Science, Brown University, Technical Report CS9637, 1996.