

6.864, Fall 2009: Problem Set 2

Total points: 140 points

Due date: 5pm EST, October 19 (Monday)

Submit writeup to Harr Chen in 32-G494. Submit code via e-mail with subject "6.864 PSET 2".

Late policy: 3 allowed late days over the course of the semester.

Question 1 (20 points)

A probabilistic context-free grammar $G = (N, \Sigma, R, S, P)$ in Chomsky Normal Form is defined as follows:

- N is a set of non-terminal symbols (e.g., NP, VP, S etc.)
- Σ is a set of terminal symbols (e.g., *cat*, *dog*, *the*, etc.)
- R is a set of rules which take one of two forms:
 - $X \rightarrow Y_1 Y_2$ for $X \in N$, and $Y_1, Y_2 \in N$
 - $X \rightarrow Y$ for $X \in N$, and $Y \in \Sigma$
- $S \in N$ is a distinguished start symbol
- P is a function that maps every rule in R to a probability, which satisfies the following conditions:
 - $\forall r \in R, P(r) \geq 0$
 - $\forall X \in N, \sum_{X \rightarrow \alpha \in R} P(X \rightarrow \alpha) = 1$

Now assume we have a probabilistic CFG G' , which has a set of rules R which take one of the two following forms:

- $X \rightarrow Y_1 Y_2 \dots Y_n$ for $X \in N, n \geq 2$, and $\forall i, Y_i \in N$
- $X \rightarrow Y$ for $X \in N$, and $Y \in \Sigma$

Note that this is a more permissive definition than Chomsky normal form, as some rules in the grammar may have more than 2 non-terminals on the right-hand side. An example of a grammar that satisfies this more permissive definition is as follows:

S	→	NP	VP	0.7	
S	→	NP	NP	VP	0.3
VP	→	Vt	NP	0.8	
VP	→	Vt	NP	PP	0.2
NP	→	DT	NN	NN	0.3
NP	→	NP	PP	0.7	
PP	→	P	NP	1.0	

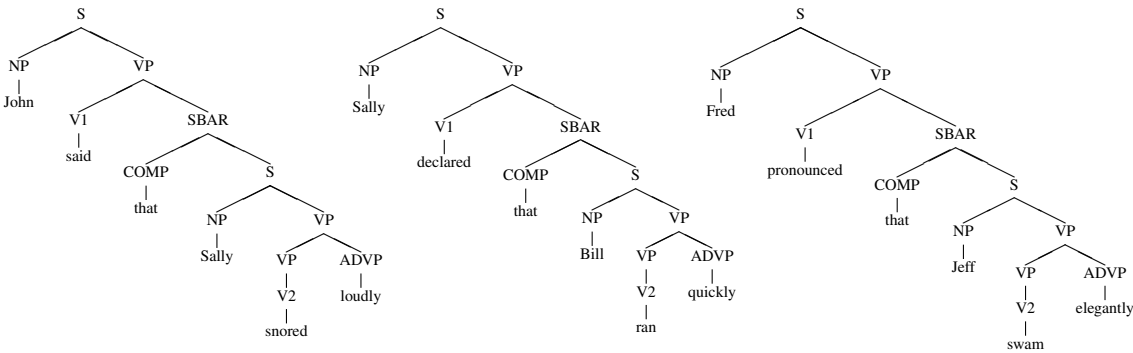
Vt	→	saw	1.0
NN	→	man	0.7
NN	→	woman	0.2
NN	→	telescope	0.1
DT	→	the	1.0
IN	→	with	0.5
IN	→	in	0.5

(a) Describe how to transform a PCFG G' , in this more permissive form, into an “equivalent” PCFG G in Chomsky normal form. By equivalent, we mean that there is a one-to-one function f between derivations in G' and derivations in G , such that for any derivation T' under G' which has probability p , $f(T')$ also has probability p . One reason we’d want to perform this transformation is that we can then apply the dynamic programming parsing algorithm, described in lecture, to the transformed grammar. (Hint: think about adding new rules with new non-terminals to the grammar.)

(b) Show the resulting grammar G after applying your transformation to the example PCFG shown above.

Question 2 (20 points)

Nathan L. Pedant decides to build a treebank. He finally produces a corpus which contains the following three parse trees:



Clarissa Lexica then purchases the treebank, and decides to build a PCFG, and a parser, using Nathan’s data.

(a) Show the PCFG that Clarissa would derive from this treebank.

(b) Show two parse trees for the string “Jeff pronounced that Fred snored loudly”, and calculate their probabilities under the PCFG.

(c) Clarissa is shocked and dismayed that “Jeff pronounced that Fred snored loudly” has two possible parses, and that one of them—that Jeff is doing the pronouncing loudly—has relatively high probability, in spite of it having the *ADVP loudly* modifying the “higher” verb, *pronounced*. This type of high attachment is never seen in the corpus, so the PCFG is clearly missing something. Clarissa decides to fix the treebank, by altering some non-terminal labels in the corpus. Show one such transformation that results in a PCFG that gives zero probability to parse trees with “high” attachments. (Your solution should systematically refine some non-terminals in the treebank, in a way that slightly increases the number of non-terminals in the grammar, but allows the grammar to capture the distinction between high and low attachment to VPs.) Note that the desired solution should preprocess the treebank, *not* alter the PCFG after it has been learned.

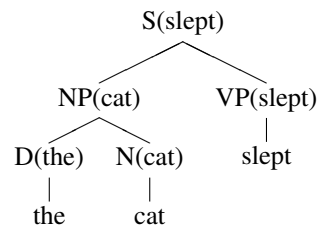
Question 3 (40 points)

We will refer to a “lexicalized PCFG” in Chomsky normal form, as a PCFG $G = (N, \Sigma, R, S, P)$ similar to that in question 1, where each of the rules in R takes one of the following three forms:

- $X(h) \rightarrow Y_1(h) Y_2(w)$ for $X \in N$, and $Y_1, Y_2 \in N$, and $h, w \in \Sigma$.
e.g., $\text{NP}(\text{man}) \rightarrow \text{NP}(\text{man}) \text{PP}(\text{with})$.
- $X(h) \rightarrow Y_1(w) Y_2(h)$ for $X \in N$, and $Y_1, Y_2 \in N$, and $h, w \in \Sigma$.
e.g., $\text{S}(\text{snores}) \rightarrow \text{NP}(\text{man}) \text{VP}(\text{snores})$.
- $X(h) \rightarrow h$ for $X \in N$, and $h \in \Sigma$
e.g., $\text{NP}(\text{man}) \rightarrow \text{man}$.

Here the symbols in the grammar rules are of the form $X(h)$, where X is a symbol such as NP, VP, etc., and h is a lexical item such as *man*, *snores*, etc.

In addition, for any symbol of the form $X(h)$, there is a probability $P_S(X(h))$ which is the probability of $X(h)$ being chosen as the root of a parse tree. As one example, the tree



would have probability

$$\begin{aligned}
 &P_S(\text{S}(\text{slept})) \times \\
 &P(\text{S}(\text{slept}) \rightarrow \text{NP}(\text{cat}) \text{VP}(\text{slept}) | \text{S}(\text{slept})) \times \\
 &P(\text{NP}(\text{cat}) \rightarrow \text{D}(\text{the}) \text{N}(\text{cat}) | \text{NP}(\text{cat})) \times \\
 &P(\text{D}(\text{the}) \rightarrow \text{the} | \text{D}(\text{the})) \times \\
 &P(\text{N}(\text{cat}) \rightarrow \text{cat} | \text{N}(\text{cat})) \times \\
 &P(\text{VP}(\text{slept}) \rightarrow \text{slept} | \text{VP}(\text{slept}))
 \end{aligned}$$

(a) Describe and give pseudo-code for a dynamic programming algorithm, similar to the one in lecture, which finds the highest scoring parse tree under a grammar of this form. Your algorithm should make use of a dynamic programming table $\pi[i, j, k, X]$ where

$$\begin{aligned}
 \pi[i, j, k, X] = & \text{highest probability for any parse tree whose root is the symbol } X(w_k), \\
 & \text{and which spans words } i \dots j \text{ inclusive}
 \end{aligned}$$

For example, if the sentence being parsed is $w_1, w_2, \dots, w_6 = \text{the cat sat on the mat}$, then $\pi[4, 6, 4, \text{PP}]$ would store the maximum probability for any parse tree whose root is $\text{PP}(\text{on})$, and which spans the string *on the mat*.

Your algorithm should also allow recovery of the parse tree which achieves the maximum probability.

(b) What is the running time of your algorithm?

Question 4 (60 points)

In class, we discussed a Bayesian model for unsupervised morphological segmentation. This model is parameterized by a *lexicon*, which lists a set of allowed morphemes along with their generation probabilities. Given such a lexicon, the model assumes that words in the observed corpus was created by generating sequences of independent morphemes according to the probabilities of the lexicon. More precisely, if we have a lexicon of morphemes m along with their probabilities $\Theta(m)$, and a corpus of words $w_1 \dots w_n$ where the length of the i^{th} word (in morphemes) is l_i and its j^{th} morpheme is m_{ij} , then we can write the likelihood of the corpus as:

$$P(w_1, \dots, w_n) = \prod_{i=1}^n \prod_{j=1}^{l_i} \Theta(m_{ij})$$

In this question, we will be exploring maximum a posteriori (MAP) estimates in relation to this model.

IMPLEMENTATION NOTES:

- When computing log-probabilities use natural log.
- $\log n! = \sum_{i=1}^n \log i$
- $\Gamma(i) = (i - 1)!$ for integer i
- Hyperparameter values: $\alpha = 5, \beta = 1, \lambda = 1, h = 0.5$
- The reference implementation is about 100 lines of Python code. As sanity checks, the highest log-prob for part (c) should be around -558563.12, for part (e) using the Gamma prior around -957567.88, and highest f-measure in (f) is 0.696154. You should get similar numbers.

(a) Briefly (two sentences) describe the general idea of a MAP estimate, and use Bayes Rule to display its relationship to a maximum likelihood estimate.

(b) Given a morphologically segmented corpus, what would be the *maximum likelihood* estimate under this model for the morpheme probabilities? Under this maximum likelihood estimate, what would be the log-likelihood of the segmented corpus?

Now suppose that we have a corpus which is *unsegmented* and we wish to simultaneously induce the most likely segmentation and lexicon. We can search through the space of possible segmentations, find each segmentation's maximum likelihood lexicon, and compute the likelihood of the segmentation under that lexicon. When we're satisfied that we searched enough, we can output the (segmentation, lexicon) pair with highest likelihood.

Download the file `segmentation.tgz`, which contains `segmentation1` through `segmentation6`. Unzip and take a look at these files. Notice that they are different possible segmentations of the same corpus of English words.

(c) Write a program that will compute the maximum likelihood lexicon for each of these segmentations. and report the log-likelihood values of each segmentation in descending order. Do you notice anything peculiar about the highest ranked segmentation? Why do you think this one scored best?

Until now we've only been considering the *likelihood* of the corpus given a lexicon. Now let's also define a *prior* on the lexicon. To be formal, let's say that the lexicon consists of M morpheme types. The number of characters in the i^{th} morpheme will be denoted l_i and the j^{th} character will be denoted c_{ij} . The probability of generating a token of the morpheme (in a corpus) will be denoted by Θ_i , and the total number of morpheme tokens to be produced in a corpus will be N . As we saw in lecture, we write the prior probability as:

$$P(\text{lexicon}) = M! \cdot P(M, N) \cdot \prod_{i=1}^M \left[P(l_i) \cdot \left[\prod_{j=1}^{l_i} P(c_{ij}) \right] \cdot P(\Theta_i | N) \right]$$

The first factor ($M!$) is required to account for all the different orders in which the morphemes in the lexicon could have been generated. The next factor $P(M, N)$ gives the probability that the number of morpheme types (in the lexicon) is M and the number of morpheme tokens (in a corpus generated from the lexicon) will be N . Let's assume that $P(M, N)$ is a constant for any reasonable values of M and N and ignore it.

Now, for each morpheme type included in the lexicon, our first prior assigns a probability to its length l using a *Gamma Distribution*:

$$P_1(l) = \frac{\beta^\alpha l^{\alpha-1} e^{-\beta l}}{\Gamma(\alpha)}$$

α and β are *hyperparameters*. The Gamma distribution peaks at $\frac{\alpha-1}{\beta}$. Let's assume that we have reason to believe that the most frequent English morpheme length is 4. Accordingly we will set $\alpha = 5$ and $\beta = 1$ (β controls how peaked the distribution is).

There is nothing special about using the Gamma distribution to model morpheme length — it just conveniently captures our *prior intuition* that morpheme lengths should roughly peak around 4, and decay gracefully around that. As an alternative, imagine using an exponential distribution as the prior for l , that is:

$$P_2(l) = \lambda e^{-\lambda l}$$

λ here is a hyperparameter; let's set it to 1. This prior assigns rapidly decaying probability mass as morphemes get longer — and we'll see why that's a Bad Thing shortly.

Once the length of the morpheme type is determined, the prior assigns a probability to its actual sequence of characters. Let's estimate the probability of each of the 28 characters from the corpus itself:

$$P(c) = \frac{\text{count of } c}{\text{count of all characters}}$$

Finally, the we assign a prior probability to the probability Θ associated with each morpheme in the lexicon:

$$P(\Theta_i | N) = (\Theta_i \cdot N)^{\log_2(1-h)} - (\Theta_i \cdot N + 1)^{\log_2(1-h)}$$

Since N is the total number of morpheme tokens to be produced in a corpus, we can expect the total number of appearances of morpheme type m_i to be $\Theta_i \cdot N$. This formula says that the frequencies of the morph

types are distributed according to a *Zipfian* distribution. With probability h , a morph type will be expected to occur only once in the corpus. With drastically falling probability, the morph may occur with greater frequency. We will set $h = 0.5$. Note that the logs in the exponents are base 2.

(d) Write down and simplify the *log* of the prior for both choices of the morpheme length distribution (that is, derive two equations, one by plugging in $P_1(l)$ and $P(\Theta|M)$, and the other using $P_2(l)$ and $P(\Theta|M)$; you can drop $P(M, N)$, which we have assumed to be a constant).

Now we will re-rank the candidate segmentations, taking the prior into account. As before, we will use the maximum likelihood lexicon associated with each segmentation (i.e. setting the Θ_i 's to the relative frequencies and N to the total number of morpheme tokens in the candidate segmentation). This time, however, we will record the *log-posterior* of the (segmentation, lexicon) pair – i.e., the log-likelihood plus the log-prior.

(e) Write a program to report the log-posterior of each candidate segmentation (using its maximum likelihood lexicon), for both choices of morpheme length prior $P_1(l)$ and $P_2(l)$. List the log-posteriors in descending order for each. Are these rankings different than the one you produced in part (c)? For the prior using the Gamma distribution for morpheme length, does the top-ranked segmentation seem better now? What about the exponential prior? Why do the different priors lead to different results?

Lo and behold, the true gold-standard segmentation is `segmentation2`! Now let's compute how close the other candidate segmentations come to it. We'll use three standard measures used in NLP: precision, recall, and F-measure. Precision will measure how many of our predicted morpheme boundaries *actually were* morpheme boundaries. Recall will measure how many of the true morpheme boundaries were *included* in our predictions. F-measure is the harmonic mean of precision and recall.

$$p = \frac{\text{true positive predictions}}{\text{all positive predictions}} \quad r = \frac{\text{true positive predictions}}{\text{all true positives}} \quad f = \frac{2pr}{p+r}$$

(f) Using `segmentation2` as a gold standard, write a program to compute the precision, recall, and f-measure of each of the other candidate segmentations (note that for this purpose, define $0/0 = 1$). Report these scores in descending order of f-measure.