

Problem Set 3

Due at the beginning of class on Wednesday, February 25

Collaboration is allowed in groups of at most three students, but you must submit separate writeups. Please also write the names of all your collaborators on your submissions. If you are working alone, we will drop the problem with the lowest score.

Problem 1. (25 points) CRFs

(For this problem, make sure you review the lecture on CRFs. It may also be helpful to read the “CRF introduction” available in PDF at the course web site).

In this problem we shall consider a CRF with N states (S_1, \dots, S_N) and K features (f_1, \dots, f_K). Let $\boldsymbol{\pi} = \pi_1 \pi_2 \dots \pi_L$ be a parse of a sequence \mathbf{x} of length L . A feature is a function $f_j(\pi_i, \pi_{i-1}, i, \mathbf{x})$ that depends on the current state π_i , the previous state π_{i-1} , the current position i and the whole sequence \mathbf{x} , and returns either 0 or 1 in constant time. Each feature f_j is scaled by some weight w_j . The score $Score(\boldsymbol{\pi}, \mathbf{x})$ of a parse $\boldsymbol{\pi}$ together with a sequence \mathbf{x} is calculated by the contribution of each feature at each position:

$$Score(\boldsymbol{\pi}, \mathbf{x}) = \exp\left(\sum_{i=1}^L \sum_{j=1}^K w_j f_j(\pi_i, \pi_{i-1}, i, \mathbf{x})\right) = \exp\left(\sum_{j=1}^K w_j F_j(\boldsymbol{\pi}, \mathbf{x})\right)$$

where $F_j(\boldsymbol{\pi}, \mathbf{x}) = \sum_{i=1}^L f_j(\pi_i, \pi_{i-1}, i, \mathbf{x})$ denotes the total contribution of feature f_j in the parse $\boldsymbol{\pi}$ of \mathbf{x} (that is, the number of times feature f_j occurs in $(\boldsymbol{\pi}, \mathbf{x})$). [Note: For completeness, let π_0 always denote a dummy start state of the CRF, so that f_j is well defined for $i=1$].

For a sequence \mathbf{x} , the partition function $Z(\mathbf{x})$ is the sum of the scores over all possible parses (recall that a parse is a sequence of states; since each π_i can be any of S_1, \dots, S_N , there are obviously N^L possible parses for a given sequence \mathbf{x}):

$$Z(\mathbf{x}) = \sum_{\boldsymbol{\pi}} Score(\boldsymbol{\pi}, \mathbf{x}) = \sum_{\boldsymbol{\pi}} \exp\left(\sum_{i=1}^L \sum_{j=1}^K w_j f_j(\pi_i, \pi_{i-1}, i, \mathbf{x})\right)$$

(a) (9 points) Inspired by the way we calculated $P(\mathbf{x})$ (the probability of a given emitted sequence \mathbf{x}) for HMMs, give an algorithm to compute $Z(\mathbf{x})$ in $O(N^2L)$ time and $O(NL)$ space. Ignore K in your complexity analysis.

(b) The conditional probability of a parse $\boldsymbol{\pi}$ given a sequence \mathbf{x} is defined as follows:

$$P(\boldsymbol{\pi}|\mathbf{x}) = \frac{Score(\boldsymbol{\pi}, \mathbf{x})}{Z(\mathbf{x})} = \frac{\exp\left(\sum_{j=1}^K w_j F_j(\boldsymbol{\pi}, \mathbf{x})\right)}{\sum_{\boldsymbol{\pi}'} \exp\left(\sum_{j=1}^K w_j F_j(\boldsymbol{\pi}', \mathbf{x})\right)}$$

To train the CRF, suppose we have exactly one sequence \mathbf{x} and a true parse $\boldsymbol{\pi}$. We would like to set the weights w_j such that $P(\boldsymbol{\pi}|\mathbf{x})$ is maximized. In practice, this is done using numerical optimization methods such as gradient descent. As with HMMs, it is normally more convenient to work in log space and instead maximize $T = \log[P(\boldsymbol{\pi}|\mathbf{x})]$.

(i) (6 points) First, we will analytically calculate the gradient $\nabla T = \nabla \log P(\boldsymbol{\pi} | \mathbf{x})$. Prove the following:

$$\frac{\partial T}{\partial w_j} = F_j(\boldsymbol{\pi}, \mathbf{x}) - \frac{\sum_{\boldsymbol{\pi}'} F_j(\boldsymbol{\pi}', \mathbf{x}) \text{Score}(\boldsymbol{\pi}', \mathbf{x})}{Z(\mathbf{x})}$$

(ii) (2 points) For CRFs, we can calculate the posterior probability of some event E in a way similar to what we did for HMMs. All we need to do is take the sum of the scores of all $(\boldsymbol{\pi}', \mathbf{x})$ for which the event is true, and divide by the partition function:

$$P(E | \mathbf{x}) = \frac{\sum_{\boldsymbol{\pi}' \text{ such that } E \text{ is true}} \text{Score}(\boldsymbol{\pi}', \mathbf{x})}{Z(\mathbf{x})}$$

For example, to calculate the posterior probability of the event " $\pi_3' = S_j$ ", we would sum the $\text{Score}(\boldsymbol{\pi}', \mathbf{x})$ of all parses whose third state π_3' is S_j . Now, consider the event "the j -th feature evaluated at position i of the parse is 1", or in other words " $f_j(\pi_i', \pi_{i-1}', i, \mathbf{x}) = 1$ ". Prove the following:

$$P(f_j(\pi_i', \pi_{i-1}', i, \mathbf{x}) = 1 | \mathbf{x}) = \frac{\sum_{\boldsymbol{\pi}'} f_j(\pi_i', \pi_{i-1}', i, \mathbf{x}) \text{Score}(\boldsymbol{\pi}', \mathbf{x})}{Z(\mathbf{x})}$$

(iii) (2 points) Using the results from the two previous steps, prove the following:

$$\frac{\partial T}{\partial w_j} = F_j(\boldsymbol{\pi}, \mathbf{x}) - \sum_{i=1}^L P(f_j(\pi_i', \pi_{i-1}', i, \mathbf{x}) = 1 | \mathbf{x})$$

(iv) (6 points) Inspired by formula (3.19) of the Durbin book, give an algorithm to compute the posterior probabilities $P(f_j(\pi_i', \pi_{i-1}', i, \mathbf{x}) = 1 | \mathbf{x})$ (and therefore the gradient ∇T) in $O(N^2L)$ time. Once again, ignore the factor K in your analysis.

Problem 2. (25 points) *Modeling the genome sequencing process*

We would like to sequence the human genome, of length G , using a whole-genome shotgun approach. We shear many copies of the genome into N pieces, all of equal length L , distributed randomly and uniformly from across the genome. The sequencing depth of coverage is defined as $C = NL / G$. You may assume that the length of the genome G is much greater than the length of a read fragment L , in particular this means you may ignore end effects and, for example, assume that the left ends of the reads are distributed in the range $[0, G]$.

For the following problems, it may help to approximate the selection of read locations as a Poisson process along the length of the genome. You are welcome to review the Lander-Waterman model at: http://www.genetics.wustl.edu/bio5488/lecture_notes_2005/Lander.htm

However, please show all steps of your derivations and justify approximation assumptions you make.

- (a) (i) (7 points) What is the expected proportion of the bases in the genome covered by at least one read, in terms of C ? (hint: your initial derivation may involve N , L , and G , but using the fact that $L \ll G$ you should be able to express the final answer using only C)
- (ii) (2 points) If we would like our expected proportion of genome bases covered to be 99%, what depth of coverage C do we need?
- (iii) (2 points) What about if we want an expected 99.9% to be covered?

(b) (8 points) We would like to assemble these sequence reads into *contigs*, or contiguous sequence. Suppose that whenever two reads overlap or even touch (i.e. when one read's left end starts at position x and another starts at position $x+L$), we can merge these two reads into a longer contig. This way, contigs extend through contiguous positions in the genome that are covered by at least one read. Compute the approximate number of contigs and average contig size in our assembly, in terms of N, C , and G .

Big hint: one way of deriving this is the following:

- (i) If you scan the genome from left to right, assume the incidence of read start locations can be modeled as a Poisson process, which is memory-less.
- (ii) For a particular read i , compute the probability that read $i+1$ will start at a position greater than L bases from the start of read i .
- (iii) Using the above result, compute the expected number of gaps.
- (iv) Now, explain approximately what the number of contigs and what the average contig size should be.

(c) (6 points) Suppose that reads must overlap each other by at least K bases before the overlap can be detected and they can be merged together. Again, using the results from the previous problem, compute the approximate number of contigs and average contig size of our assembly.

Problem 3. (25 points) *Fragment Assembly*

For the following problem, we would like to assemble our N sequence reads of length L from the previous problem into large contigs. Let \mathbf{R} be the set of reads, and we construct the directed overlap multigraph of \mathbf{R} , denoted $OM(\mathbf{R})$. The nodes in the graph are the reads themselves, and we have an edge $E(a, b)$ from node a to node b with weight w whenever the w -letter suffix of a is the same as the w -letter prefix of b , in other words, read b extends read a to the right with an overlap of w . Note that this is a multigraph and there may be more than one edge between two nodes if there is more than one possible overlap alignment.

(a) (7 points) Suppose we are only interested in edges with weight at least K , i.e. the reads overlap by at least K bases. Describe the main ideas behind an algorithm to construct $OM(\mathbf{R})$ restricted to these edges that does not need to perform all possible $N(N-1)/2$ alignments between pairs of reads. You may review the class lecture on fragment assembly for an idea on how to do this.

(b) (6 points) A Hamiltonian path through $OM(\mathbf{R})$, or a path that visits each node exactly once, represents an assembly of the reads into a supersequence. Minimizing the length of this supersequence is equivalent to finding the Hamiltonian path with maximum weight, where the weight of a path is defined as the sum of the weights of the edges that make up the path. Unfortunately, the simpler task of deciding whether or not a graph has a Hamiltonian path is in general NP-complete.

Dr. Fratki suggests a greedy approach to a slightly different problem to maximize the weights as follows: First, start the path by picking the heaviest edge in the entire graph. Then, incrementally add the next edge to the path by taking all the remaining edges that start at the last node in our path and lead to a node not already in our path, and picking a maximal weight edge. Assume that our graph has been made complete by adding in zero-weight edges between nodes that have no edge. Show that this greedy algorithm does not always find the optimal, maximal-weight path.

(Hint: you should be able to construct a very simple example with perhaps 3 or 4 reads such that Dr. Fratki's algorithm finds a supersequence that is longer than the shortest possible supersequence).

(c) Consider the following set of sequence reads \mathbf{R} :

A: WeCanAllAgreeThatThe
B: TheCS262TeachingAssistants
C: TheStudents
D: ntsLoveThe
E: nts, Right?

(i) (4 points) Draw the directed overlap multigraph $OM(\mathbf{R})$ for this set of reads, including edge weights.

(ii) (4 points) Write out all possible single-contig assemblies of this read data, i.e. supersequences formed by Hamiltonian paths through the graph in part (i), and give their total path weights.

(iii) (4 points) Suppose we obtain one more sequence read. Draw the new multigraph as well as all possible single-contig assemblies with the addition of this read:

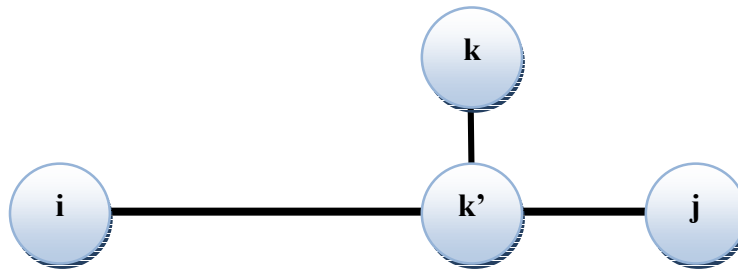
F: Assistants,R

Problem 4. (25points) *Tree-building*

Neighbor-joining is the most widely used algorithm for tree construction, but it has $O(n^3)$ running time which may be problematic for some large data sets. In fact, this is a problem in many real world cases. In this problem we will ask you to come up with a simpler and faster alternative algorithm.

Assume that you are given a matrix of pair-wise distances $T(i,j)$, which gives the path length between each pair of nodes i and j .

(a) (4 points) Demonstrate that given any two nodes (i, j) it is possible to find, in $O(1)$ time, the position along this path where a third node k branches off. In other words, determine distances (i, k') , (j, k') , and (k, k') in the figure below.



(b) (8 points) Given the insight from problem 4(a), devise a divide-and-conquer algorithm for reconstructing the entire tree. In other words, this means finding locations of all the internal nodes. This algorithm should have a worst-case running time of $O(n^2)$.

(c) (7 points) Given noisy data in the matrix $T(i,j)$, where distances are estimated only within some tolerance, demonstrate that the reconstructed tree could produce an arbitrarily bad approximation of the real tree. To determine the quality of tree reconstruction, we compute $T'(i,j)$ given the reconstructed tree and compute $\sum_{ij} T'(i,j)/T(i,j)$. Provide a way to be able to tolerate some moderate amounts of noise.

(d) (6 points) In an average case, your algorithm might run in time $O(n \log n)$. In the worst case it may be $O(n^2)$. Provide an example to prove that you cannot do better than $O(n^2)$ in the worst case. Assume no restrictions on tree topology.