Bioinformatics Practice Final Exam Solutions

1 A Recipe for Inference

From the beginning of the course we have emphasized a consistent, systematic approach to solving inference problems. Based on Bayes Law, list each of the basic questions that we must answer to solve any inference problem, and briefly explain its meaning.

Solution

From Bayes' Law:

$$p(\theta|obs) = \frac{p(obs|\theta)p(\theta)}{\sum_\theta p(obs|\theta)p(\theta)}$$

we can enumerate the different ingredients for inference:

- The observed variable(s) obs. These represent variables whose values are known with zero uncertainy.
- The hidden variable(s) θ. This represents unknown parameters in our model, which we will seek to infer from the observations.
- The likelihood model $p(obs|\theta)$. This represents our model for how the probability distribution of the observable(s) depends on our model parameters.
- The prior probability $p(\theta)$: this represents the probability of the hidden variable before taking into account any observations.
- The posterior probability of the hidden variable given the observations. This represents our probability distribution after taking into account the observations. Typically this requires summing over all possible values of the hidden variable(s) θ in the denominator of Bayes' Law either analytically, exhaustively, or by sampling.

2 A Simple Protein Coding Model

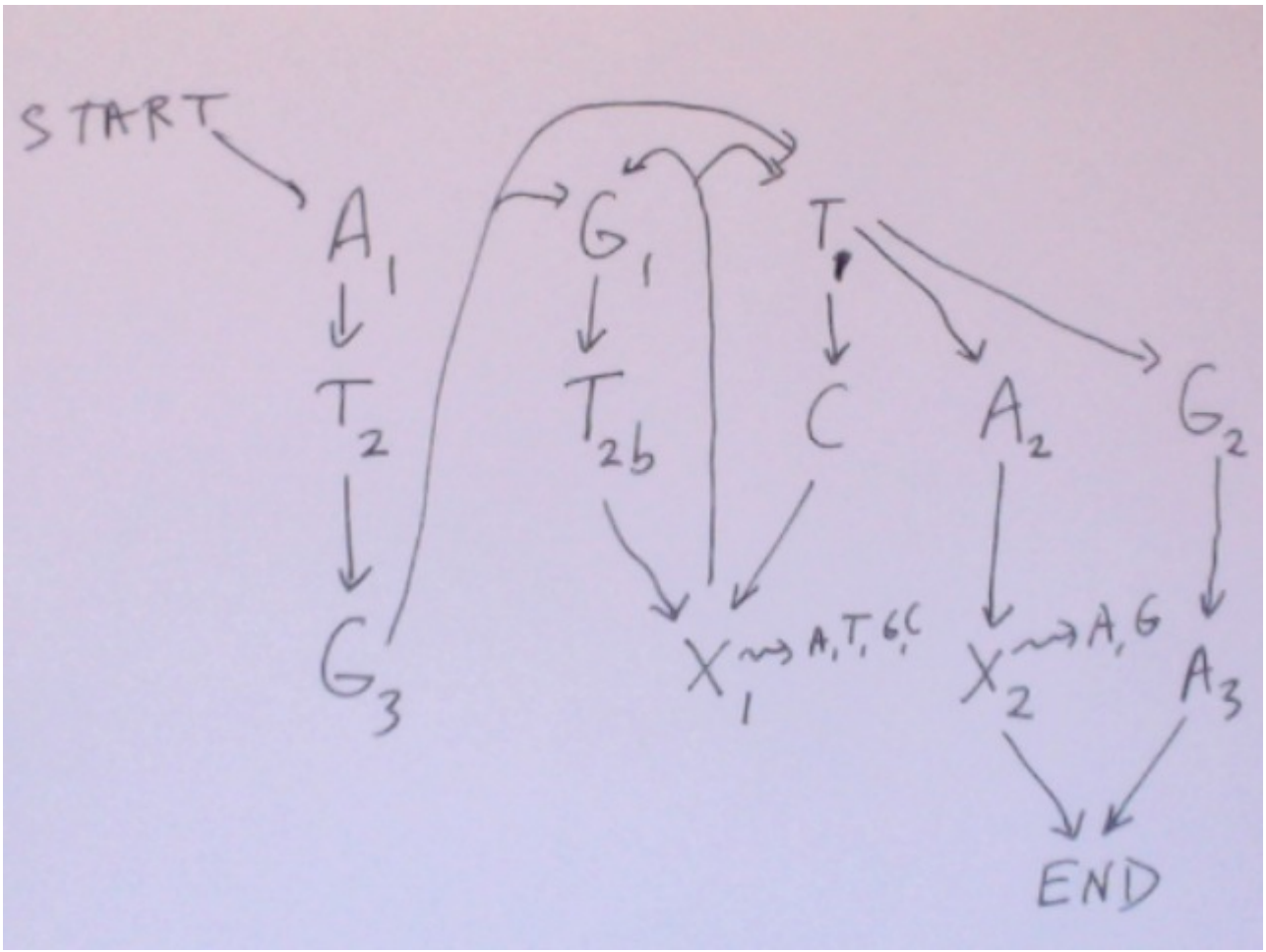
A gene sequence (a string of A, T, G, C nucleotide letters) encodes a *protein* sequence in the form of a series of *codons*, each three nucleotide letters long. Each three-letter codon encodes one letter in the protein sequence (which has a twenty letter "alphabet").

Let's design a simple Markov model of a codon sequence, based on the following rules:

- A codon sequence *must* begin with the codon ATG.
- The codons TAA, TGA and TAG are *STOP codons* that terminate the protein.
- Between the start and stop codons we can have any number of regular codons.
- For simplicity, let's assume only two kinds of regular codons are allowed: GTx (where x means any of A, T, G or C), or TCx.

Draw a Markov chain representing this model, assuming:

- Each node emits one nucleotide letter at a time;
- Label each node with the nucleotide letter(s) it can emit;
- Show the direction of connectivity of nodes by drawing arrows;
- Include explicit "start" and "end" nodes indicating where we can enter or exit the model.
- No need to indicate any transition probabilities (edge weights) on this model.



3 Hidden Markov Models

1. Define a Markov chain.
2. Define a hidden markov model, explaining each of its basic elements.
3. Derive an expression for the posterior probability of a specific state s_i given the observations, i.e. $p(\theta_t = s_i|\vec{O}_1^n)$.
4. Explain how this can be calculated efficiently using the forward-backward algorithm (include the specific recurrence relations for performing this calculation).
5. Analyze the computational complexity of the forward-backward algorithm.

Solution

1. A Markov chain is a sequence of variables $X_1, X_2, ...X_n$ that obeys the joint probability rule

$$p(X_1, X_2, ...X_n) = p(X_1)p(X_2|X_1)...p(X_n|X_{n-1})$$

   i.e. each variable depends only on the previous variable in the sequence.

2. An HMM is a set of hidden variables $\theta_1, \theta_2, ...\theta_n$ that obey the Markov property, and which in turn emit a set of observable variables, where typically each observable depends on only one hidden variable i.e. $p(X_t|\theta_t)$.

3.

$$p(\theta_t = s_i|\vec{X}_1^n) = \frac{p(\theta_t = s_i, \vec{X}_1^n)}{p(\vec{X}_1^n)} = \frac{p(\theta_t = s_i, \vec{X}_1^n)}{\sum_i p(\theta_t = s_i, \vec{X}_1^n)}$$

4. We divide the numerator into two parts, the forward and backward probabilities:

$$p(\theta_t = s_i|\vec{X}) = \frac{p(\theta_t = s_i, \vec{X}^t)p(\vec{X}_{t+1}^n|\theta_t = s_i)}{p(\vec{X})}$$

The forward probability is just

$$p(\theta_t = s_i, \vec{X}^t) = \sum_j p(\theta_{t-1} = s_j, \theta_t = s_i, \vec{X}^t) = \sum_j p(\theta_{t-1} = s_j, \vec{X}^{t-1})p(\theta_t = s_i|\theta_{t-1} = s_j)p(X_t|\theta_t = s_i)$$

The backward probability is just

$$p(\vec{X}_{t+1}^n|\theta_t = s_i) = \sum_j p(\theta_{t+1} = s_j, \vec{X}_{t+1}^n|\theta_t = s_i) = \sum_j p(\vec{X}_{t+2}^n|\theta_{t+1} = s_j)p(X_{t+1}|\theta_{t+1} = s_j)p(\theta_{t+1} = s_j|\theta_t = s_i)$$

5. If each variable θ_t has m states, the computational complexities for computing all the forward and backward probabilities are both $O(nm^2)$. Note this enables us to compute all nm posterior probabilities $p(\theta_t = s_i|\vec{X})$.

4 HMM Profiles

1. Assuming that you are given an alignment of a set of related sequences representing a protein homology family, briefly outline a procedure for aligning a new sequence $\vec{X}$ to this family, taking into account the details of its position-specific letter probabilities (e.g. at a crucial "catalytic" site, only one specific letter might be observed).

   (4)

2. Draw a fragment of the HMM structure needed to allow both insertion and deletion during the emission of the new sequence $\vec{X}$ relative to the family model.

   (4)

3. Assuming that you are given a series of such HMMs $F_1, F_2, ...F_N$ representing different protein homology families, and the prior probability $p(F_k)$ that a randomly chosen protein sequence will be a member of each family, derive an expression for the posterior probability that sequence $\vec{X}$ is a member of family F_k, i.e. $p(F_k|\vec{X})$. Explain how you would actually compute this.

   (4)

4. Assuming that $\vec{X}$ is from a specific family HMM model, use Bayes Law to derive an expression for the posterior probability that sequence letter X_t was actually emitted from HMM state $\Theta_t = s_i$ from this model, i.e. $p(\Theta_t = s_i|\vec{X})$. Use an information graph structure diagram to justify your choice of how to split up the total observation likelihood.

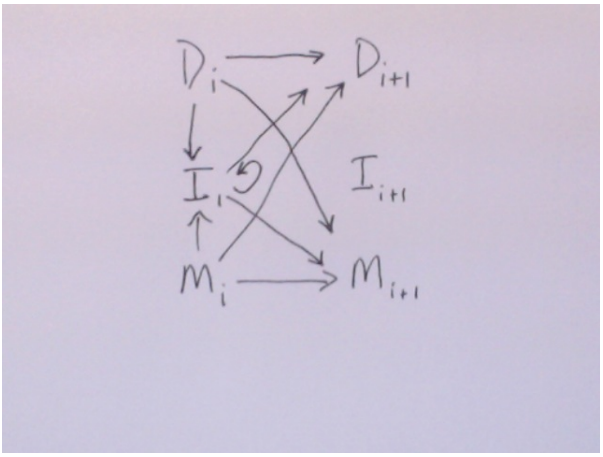
   (4)

5. Derive recursion rules for actually calculating the probability component(s) necessary to compute the posterior probability $p(\Theta_t = s_i|\vec{X})$.

   (4)

Solutions

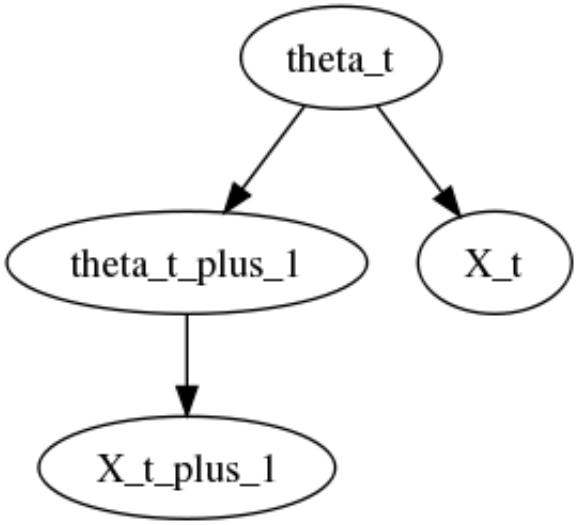
1. For each column of the alignment we derive emission probabilities for the M (match) state by taking the expectation value of the likelihood of each amino acid, which is equivalent to calculating it including a pseudocount of +1 for each amino acid, i.e. $p(s_i|n_i, N) = \frac{n_i+1}{N+20}$. The transition probabilities to the D (deletion) state for each column can either be estimated directly from the alignment (if there are enough sequences), or simply use a standard "deletion probability". Similarly, we can also use a standard value for the insertion state transition probability, unless we have enough data to estimate it specifically for one column.

2. 

3. For each protein family profile F_k compute the likelihood of the observed sequence $\vec{X}$ by the forward probability calculation to obtain $p(\vec{X}|F_k)$. Finally compute the posterior for the protein family via Bayes' Law:

$$p(F_k|\vec{X}) = \frac{p(\vec{X}|F_k)p(F_k)}{\sum_j p(\vec{X}|F_j)p(F_j)}$$

4. The posterior probability that letter X_t was emitted by state s_i (which must be either a match or insertion state) is $p(\theta_t = s_i|\vec{X})$. Given the information graph



it is convenient to split it into two parts referred to as the forward and backward probabilities:

$$p(\theta_t = s_i, \vec{X}) = p(\theta_t = s_i, \vec{X}^t)p(\vec{X}^n_{t+1}|\theta_t = s_i)$$

Finally, we compute the posterior via:

$$p(\theta_t = s_i|\vec{X}) = \frac{p(\theta_t = s_i, \vec{X}^t)p(\vec{X}_{t+1}^n|\theta_t = s_i)}{p(\vec{X})}$$

where $p(\vec{X})$ is computed either from the forward probability at the END state, or the backward probability at the START state.

5. The forward probability is just

$$p(\theta_t = s_i, \vec{X}^t) = \sum_j p(\theta_{t-1} = s_j, \theta_t = s_i, \vec{X}^t) = \sum_j p(\theta_{t-1} = s_j, \vec{X}^{t-1})p(\theta_t = s_i|\theta_{t-1} = s_j)p(X_t|\theta_t = s_i)$$

The backward probability is just

$$p(\vec{X}_{t+1}^n|\theta_t = s_i) = \sum_j p(\theta_{t+1} = s_j, \vec{X}_{t+1}^n|\theta_t = s_i) = \sum_j p(\vec{X}_{t+2}^n|\theta_{t+1} = s_j)p(X_{t+1}|\theta_{t+1} = s_j)p(\theta_{t+1} = s_j|\theta_t = s_i)$$

5 Poly-pyrimidine tracts

Consider the following 2-state model for detecting "poly-pyrimidine tracts" (regions with a high density of C or T):

- State α: emits one of the four nucleotides (A, T, G, C) with equal probability (1/4 each).
- State β: preferentially emits C or T (with probability 1/3 each), or A or G with probability 1/6 each.
- The transition probabilities are $p(\beta|\alpha) = 1/5$, $p(\alpha|\beta) = 1/10$.

1. Calculate the likelihood $p(CTA|\alpha\alpha\alpha)$
2. Calculate the joint probability $p(CTA, \beta\beta\beta)$
3. Calculate the odds ratio $\frac{p(\beta\beta\beta|CTA)}{p(\alpha\alpha\alpha|CTA)}$
4. Say you are given a sequence O and want to calculate the posterior probability that the HMM was in state β for both letter i and $i+1$, i.e. $p(S_i = \beta, S_{i+1} = \beta|\vec{O}^n)$. Derive an expression for calculating this posterior probability; express your answer in terms of the standard forward-backward values.

Solution

1.

$$p(CTA|\alpha\alpha\alpha) = \left(\frac{1}{4}\right)^3$$

2. Assuming that the priors are equal to the stationary distribution $p(\alpha) = 1/3, p(\beta) = 2/3$,

$$p(CTA, \beta\beta\beta) = p(CTA|\beta\beta\beta)p(\beta)p(\beta|\beta)^2 = \left(\frac{1}{3}\right)^2 \frac{1}{6}\frac{2}{3}(0.9)^2$$

3.

$$\frac{p(\beta\beta\beta|CTA)}{p(\alpha\alpha\alpha|CTA)} = \frac{p(CTA, \beta\beta\beta)}{p(CTA, \alpha\alpha\alpha)} = \frac{\left(\frac{1}{3}\right)^2 \frac{1}{6}\frac{2}{3}(0.9)^2}{\left(\frac{1}{4}\right)^3 \frac{1}{3}(0.8)^2}$$

4.

$$p(\theta_t = \beta, \theta_{t+1} = \beta|\vec{O}^n) = \frac{p(\theta_t = \beta, \theta_{t+1} = \beta, \vec{O}^n)}{p(\vec{O}^n)}$$

$$= \frac{p(\theta_t = \beta, \vec{O}^t)p(\theta_{t+1} = \beta|\theta_t = \beta)p(O_{t+1}|\theta_{t+1})p(\vec{O}_{t+2}^n|\theta_{t+1} = \beta)}{p(\vec{O}^n)}$$

where the first factor in the numerator is the forward probability for θ_t, and the last factor in the numerator is the backward probability for θ_{t+1}.

6 Affine Gap Alignment

1. Briefly state the simple gap scoring model and affine gap scoring model.
2. Describe exactly how to change the dynamic programming alignment algorithm from the simple gap model to the affine gap model.
3. Describe a basic HMM model of pairwise sequence alignment, and relate each of its components to the specific details of your answer in (b) for how to implement affine gap alignment by dynamic programming.

Solution

1. In the simple gap model, a gap is assigned the same penalty regardless of what the previous move was. In the affine gap model, a different penalty is used for a *gap opening* move (i.e. if the previous move as a diagonal) vs. a *gap extension* move (i.e. if the previous move was a gap move).
2. The simple gap model requires keeping only a single matrix for scores and backtracking. By contrast, the affine gap model requires keeping three separate matrices, one representing the M state (diagonal move), one representing the D state (deletion, vertical move), the other representing the I state (insertion, horizontal move). Moves from the M state to D or I matrices represent gap opening; moves within the D or I matrices (or between them) represent gap extension.
3. The HMM alignment model's M, D, I states correspond exactly to the three matrices required for affine gap models. Moves from the M to D or I matrices represent $M \rightarrow D$ or $M \rightarrow I$ state transitions, respectively. Moves within each matrix represent "self-transitions" for the corresponding state to itself.

7 Simple Sequence Assembly

Genome sequencing produces many short sequences that represent fragments of the complete genome sequence. These short sequences must be *assembled* into a continuous sequence representing the complete genome sequence, by aligning sequences that have matching, overlapping ends.

Outline an algorithm to solve this problem for a pair of sequences, i.e. to find whether the head of one sequence matches the tail of the other sequence (with possible mismatches due to sequencing error).

Solution

The assembly problem requires aligning the overlap between two sequences completely from the beginning of one sequence to the end of whichever sequence ends first. Concretely, this means the best-scoring path that begins on either the $(x, 0)$ edge of the matrix or $(0, y)$ edge, and ends on either the (x, M) edge or the (L, y) edge (where L, M are the lengths of the x and y sequences respectively). That means we assign zero penalties along both $(x, 0)$ and $(0, y)$ edge cells; and we find the maximum score in the (x, M) and (L, y) edge cells to begin our traceback.

8 Local vs. Global Alignment

1. Briefly define the *local* vs. *global* alignment problems.
2. Briefly describe the *local* vs. *global* alignment algorithms.
3. Is one of these alignment problems more general than the other? i.e. are the path constraints of one a subcase of the other?
4. Based on your answer to (c), is there any possible way to make one of these alignment algorithms actually perform the other kind of alignment? If so, briefly explain how. If not, explain why not.

Solution

	A	B	C	D	E	F
B	3					
C	7.5	6.5				
D	6.5	5.5	3			
E	4	3	4.5	3.5		
F	3.5	2.5	5	4	1.5	

1. Global alignment means alignment of the entire length of both sequences. Concretely, this means the best-scoring path from (0,0) to (L,M) on the alignment matrix (using the same length definitions as above).

   Local alignment means alignment of any subsegment of one sequence vs. any subsegment of the other sequences. Concretely, this means the best-scoring path from any point (x_0, y_0) to any point (x_1, y_1), where $x_0 \leq x_1, y_0 \leq y_1$.

2. The two algorithms use the same basic Viterbi recursion to choose the maximum scoring move to each cell. Where they differ is their handling of the endpoints of the path. Global alignment enforces gap penalties in the $(x, 0)$ and $(0, y)$ edge cells, and only permits traceback from the (L, M) corner of the matrix. Local alignment assigns zero scores in the $(x, 0)$ and $(0, y)$ edge cells; adds an extra "path-start" move option (with incoming score zero) in every cell; and allows traceback from the maximum scoring cell in the entire matrix.

3. Since local alignment can begin and end at any matrix cell, the set of global alignment paths is a *subset* of possible local alignment paths.

4. Yes, we can make a local alignment program perform global alignment as follows. We define unique "START" and "END" symbols, e.g. \$ and #. We add them to our scoring matrix as follows: matching START to itself is assigned a huge positive score (e.g. $1000LM$), and the same for END to itself; matching START or END to anything but itself is assigned a negative score. Now we prefix both sequences with the START symbol, and add the END symbol at their ends. The huge positive scores force local alignment to align START to START and END to END, so that the best scoring path is guaranteed to go from the beginnings of both sequences to the ends of both sequences, exactly like global alignment.

9 Ultrametric Test?

You have been given a set of pairwise distance metrics for a set of sequences A - F (above). Do they appear suitable for re-constructing the phylogenetic tree using the UPGMA (hierarchical clustering) algorithm? Explain.

Solution

UPGMA is only appropriate for ultrametric distances, i.e. where the distance between any pair of modern sequences is just proportional to their time of divergence from their MRCA. Ultrametric distances obey the following simple test: for any three modern sequences, of the three pairwise distances, the two largest distances should be equal.

This set of distances does not fit this test well, e.g. $D(C, E) = 3 < D(B, E) = 4.5 \neq D(B, C) = 6.5$.

10 Tree Construction

1. Explain the neighbor joining principle and how it can be used to reconstruct a phylogenetic tree.

2. For a set of additive distances, how can we identify a pair of nodes that are neighbors in the tree?

3. Outline the neighbor joining tree construction algorithm in pseudocode.

4. Analyze the computational complexity of the algorithm.

5. How would you change the algorithm to build a tree from a set of ultrametric distances?

Solution

1. Neighbor joining is a recursive algorithm for reconstructing an unrooted tree from a set of additive distances. The basic recursion consists of 1. finding a pair of sequences that must be neighbors in the tree; 2. replacing them with a "cluster" node located at the junction of their external edges in the tree; 3. computing the distance of the new cluster node from the other remaining sequences. The recursion is repeated until the distance matrix is reduced to the trivial case (three sequences, a trivial unrooted tree).

2. We compute a "corrected distance" function

$$Q(X,Y) = D_{XY} - \frac{1}{n-2}\sum_Z D_{XZ} - \frac{1}{n-2}\sum_Z D_{YZ}$$

   The pair (X,Y) that minimizes Q(X,Y) is guaranteed to be neighbors.

3. This Python code is more detailed than you need to provide as pseudocode; e.g. it provides the exact calculations of all the tree reduction distances, and all the necessary initializations; this code would actually run. It assumes an input D that stores distances between any two sequences x,y as D[x][y]. It creates internal nodes (clusters) as (x,y) tuples indicating the pair of nodes clustered together as neighbors. It returns trees as nested tuples, e.g. a 4 sequence tree would be ((0, 1),(2, 3)), where the sequence IDs are specified as integers.

```
def njoin(D):
  minQ, minX, minY = 0, None, None
  for x in D: # find neighbors
    for y in D:
      if x != y:
        q = calc_q(D, x, y)
        if q < minQ:
          minQ, minX, minY = q, x, y
  dX = D[minX]
  dY = D[minY]
  del D[minX] # tree reduction
  del D[minY]
  newC = (minX, minY) # ID of the new cluster node
  dC = {}
  dCX = dCY = 0.
  for z in D: # compute distances to new cluster node
    dC[z] = (dX[z] + dY[z] - dX[minY]) / 2.
    dCX += (dX[z] + dX[minY] - dY[z]) / 2.
    dCY += (dY[z] + dY[minX] - dX[z]) / 2.
  dC[minX] = dCX / len(D) # save external edge lengths
  dC[minY] = dCY / len(D)
  if len(D) == 2: # reduced to trivial case, done!
    D[newC] = dC
    return (newC, tuple(D))
  D[newC] = dC
  return (newC, njoin(D)) # recurse
```

4. This requires $O(n^2)$ distance computations as input. Consider the basic recursion step for i clusters remaining to be joined. Computing $\sum_Z D(X,Z)$ for every possible X will take $O(i^2)$ time. If those values are stored in an array, then computing every possible $Q(X,Y)$ to find the minimum value will also take $O(i^2)$ time. Building the whole tree will take n recursion steps, so the overall complexity is $O(\sum_{i=1}^{n} i^2) = O(n^3)$.

5. For ultrametric distances, we can use the pairwise distances directly in the minimization step (instead of having to compute and minimize Q(X,Y)). The resulting tree can be interpreted as a rooted tree.

11 Tree Construction

1. Explain the difference between a rooted vs. unrooted tree, and derive the number of rooted trees associated with a given unrooted tree with n leaf nodes.

   (4)

2. Explain the assumptions of an "ultrametric distance" metric, and outline a test for evaluating whether the distances D_{ij} between a set of modern sequences i, j obey these assumptions.

   (4)

3. Derive an inductive rule for constructing the correct phylogenetic tree given a set of D_{ij} ultrametric distances. Provide a brief proof that this rule guarantees correctly connecting pairs of nodes that are adjacent in the tree.

   (4)

4. Explain how an additive distance metric differs from an ultrametric distance metric, and give an example of when it is more appropriate.

   (3)

5. Explain how the tree construction algorithm in (c) must be modified to work with an additive distance metric.

   (4)

Solutions

1. An unrooted tree is a binary tree of undirected edges. A rooted tree is a binary tree of directed edges, which all point from a single root node towards the leaf nodes. For a given unrooted tree, the set of possible distinct rooted trees that can be derived from it is simply generated by placing the root node on one of its undirected edges (splitting that edge into two edges), and setting the direction of all edges relative to this root location. Therefore the number of possible rooted trees is just equal to the number of edges on the unrooted tree. For an unrooted tree with n leaf nodes, that is just $2n - 3$.

2. An ultrametric distance metric assumes that the distance between any two modern sequences is just proportional to the time of divergence since their most recent common ancestor. Therefore for any three modern sequences A, B, C, if the closest pair is A, B, then the distances to the third sequence C must be equal, i.e. $D_{AC} = D_{BC} > D_{AB}$.

3. The true structure of the tree should reflect the branching order in evolutionary time. If the distances are just directly proportional to time, then the pair with the shortest distance is guaranteed to be the most recent branch in evolutionary time, i.e. neighbors in the tree. Proof: Assume that the minimum distance pair A, B are *not* neighbors. That means there must be some sequence C that branches below their MRCA, say on the edge going to A. By definition, that implies $D_{AC} < D_{AB}$ (since the time of divergence to the MRCA of A, C is less than that to the MRCA of A, B), contrary to our assumption. RAA; QED.

4. An additive distance metric assumes only that the total distance between any pair is equal to the sum of the distances of the edges between them. In effect this means assuming that the rate matrix is the same on all edges, but does not require the ultrametric assumption that the rate parameter λ is also the same on all edges.

   The additive assumption is needed in cases where the ultrametric assumption breaks down due to variations in mutation rate on different edges, e.g. due to paralog divergence.

5. We can no longer assume that the minimum distance pair are neighbors, due to possible distortions caused by variations in mutation rate. Concretely, the Neighbor Joining algorithm adds a "correction" to the pairwise distance that renders it insensitive to these distortions, such that the pair X, Y that minimizes the corrected score

$$Q(X, Y) = D_{XY} - \frac{1}{n-2} \sum_Z D_{XZ} - \frac{1}{n-2} \sum_Z D_{YZ}$$

   is guaranteed to be neighbors in the correct tree.

12 Ancestral State Inference

1. Consider the following tree of three species A, B, C. Assuming that you know the states of a given character (e.g. nucleotide sequence) for each of these three species, explain the intuitive principle(s) for infering the state of the most recent common ancestor of A, B.

   (4)

2. Derive the recursion rule for infering the most likely ancestral states using the Viterbi algorithm, given a tree topology, and the observed characters X_u of leaf nodes u of the tree.

   (4)

3. Explain the detailed traceback rule for infering the most likely ancestral states using the Viterbi algorithm.

   (2)

4. Explain briefly how your Viterbi algorithm satisfies the intuitive principle(s) you stated in part (a).

   (2)

Solutions

1. Typically we assume that each site evolves independently, so each can be inferred separately. For a given ancestor, if both child states agree (i.e. the same nucleotide), we infer that must have been present in the ancestor as well. If the child states disagree, then the nearest outgroup may "break the tie"; that is, if the outgroup matches one of the child states, we infer that is most likely to have been the ancestral state (by parsimony).

2. Based on the Markov property, we know that the global optimum state of the tree must also be locally optimal (that is, at any given step we must choose the maximum; if we did not, it cannot be the global maximum, because we could get an even better overall score by choosing the maximum at this step, since that will not affect the scores elsewhere in the tree). This gives us a simple recursion. Say β, γ are the child nodes of node α, and we know the maximum probability solution for the subtrees below β (given by $p^*(D_\beta|\beta)$) and below γ (given by $p^*(D_\gamma|\gamma)$), where D_β means all descendants of β. Then the maximum probability subtree given a specific state of α is just

$$p^*(D_\alpha|\alpha) = \text{Max}\,[p^*(D_\beta|\beta)p(\beta|\alpha)p^*(D_\gamma|\gamma)p(\gamma|\alpha)]$$

3. Associated with each value of α we must store both $p^*(D_\alpha|\alpha)$ and the values of β, γ that maximized it. When we reach the root of the tree ρ, we choose the value of ρ that maximizes $p(\rho)p^*(D_\rho|\rho)$. From that value of ρ we backtrack to the stored values of its children β, γ that maximized $p^*(D_\rho|\rho)$ for that value of ρ. We then backtrack recursively by this rule.

4. Consider a node α with child nodes β, γ. Say the values of β, γ that maximize $p^*(D_\beta|\beta)$ and $p^*(D_\gamma|\gamma)$ respectively are β^*, γ^*. If $\beta^* = \gamma^*$, then for a typical mutation matrix (e.g. Jukes-Cantor), it is guaranteed that the value α^* that maximizes $p^*(D_\alpha|\alpha)$ will just be $\alpha^* = \beta^* = \gamma^*$.

   If $\beta^* \neq \gamma^*$ but the nearest outgroup ϵ has $\epsilon^* = \beta^*$, then for a typical mutation matrix Viterbi will choose the value of the MRCA ρ to be $\rho^* = \beta^* = \epsilon^*$, and the backtracking path from that value ρ^* will backtrack at node alpha to set $\alpha = \rho^*$.