Lecture Thu 25.11.

Gene Prediction Continued
And
Completing the Pipelines
Part I

STATISTICAL APPROACHES TO GENE PREDICTION:
HMMS, VITERBI, FORWARD, BACKWARD
Consider using GC content to segment a DNA sequence into coding and non-coding parts.

A natural way to model this is to study "two"-state finite automaton reading the DNA, where one state corresponds to being in coding part and the other to being in non-coding part:
Now, the above automaton just recognizes any DNA sequence and the path taken through the automaton labels each position as coding or non-coding.

- Plug in the state-dependent probabilities of emitting a symbol and small probabilities to change the state.
- The most probable path through the automaton corresponds to the most probable segmentation given the parameters (transition and emission probabilities).
Hidden Markov Models could be defined as the probabilistic extension of finite automaton like the previous example suggests, but the convention is to separate emissions and transitions:

- Nodes emit symbols and all transitions are epsilon-transitions.
- Equivalent notion in terms of expressibility.
Hidden Markov Models: Definition

- **Hidden Markov Model (HMM)** is a tuple \((H, \Sigma, T, E, p)\), where \(H=\{1,\ldots,|H|\}\) is the set of states, \(\Sigma\) the set of symbols, \(T\) the set of transitions, \(E\) the set of emissions, and \(p\) the probability distribution for elements of \(T\) and \(E\), with the following conditions:
  - Let \(p(t)\) and \(p(e)\) denote the probability for transition \(t:=(h,h')\) in \(T\) and emission \(e:=(h,c)\) in \(E\), respectively, where \(h,h'\) in \(H\) and \(c\) in \(\Sigma\):
    - \(\sum_{h'} \in H p(h,h')=1\), \(\sum_c \in \Sigma p(h,c)=1\) for all \(h\) in \(H\).
  - There is a single **start state** \(h_{\text{start}}\) in \(H\) such that there is no transition \((h, h_{\text{start}})\) in \(T\).
  - There is a single **end state** \(h_{\text{end}}\) in \(H\) such that there is no transition \((h_{\text{end}}, h)\) in \(T\).
Hidden Markov Models: Definition

- A path through HMM is a sequence $\mathbf{h}$ of hidden states $\mathbf{h} = h_0, h_1, h_2, \ldots, h_n, h_{n+1}$, where $h_0 = h_{\text{start}}$, $h_{n+1} = h_{\text{end}}$, and $h_i$ in $\mathcal{H}\setminus\{h_{\text{start}}, h_{\text{end}}\}$. The probability of path $\mathbf{h}$ given a sequence $\mathbf{c} = c_1, c_2, \ldots, c_n$, $c_i$ in $\Sigma$, is

$$ P(\mathbf{h} \mid \mathbf{c}) = \prod_{i=0}^{n} p(h_i, h_{i+1}) \prod_{i=1}^{n} p(h_i, c_i). $$

- Let us denote
  - by $\mathcal{H}(n)$ the set of all paths through HMM of length $n + 2$ including start ($h_0$) and end ($h_{n+1}$) states,
  - by $\mathcal{H}^p(n)$ the set of all (prefix) paths of length $n+1$ inside HMM excluding end state ($h_{n+1}$),
  - by $\mathcal{H}^s(n)$ the set of all (suffix) paths of length $n+1$ inside HMM excluding start state ($h_0$),
  - and by $\mathcal{H}^*(n)$ the set of all (local) paths of length $n$ inside HMM excluding start ($h_0$) and end ($h_{n+1}$) states.
Three common problems studied on HMM

- **Most probable path:**
  - Given sequence $c=c_1,c_2,...,c_n$, $c_i$ in $\Sigma$, find the path $h^*$ having the highest probability:
    $$h^* = \arg \max_{h \in H(n)} P(h \mid c) = \arg \max_{h \in H(n)} \prod_{i=0}^{n} p(h_i, h_{i+1}) \prod_{i=1}^{n} p(h_i, c_i).$$

- **Probability of sequence being generated by HMM:**
  - Given sequence $c=c_1,c_2,...,c_n$, $c_i$ in $\Sigma$, compute the probability
    $$\sum_{h \in H(n)} P(h \mid c) = \sum_{h \in H(n)} \prod_{i=0}^{n} p(h_i, h_{i+1}) \prod_{i=1}^{n} p(h_i, c_i).$$

- **Probability of $c_j$ matching state $k$:**
  - Given sequence $c=c_1,c_2,...,c_n$, $c_i$ in $\Sigma$, compute the probability
    $$\sum_{h \in H(n)} P(h \mid c, h_j = k) = \sum_{h \in H(n), h_j = k} \prod_{i=0}^{n} p(h_i, h_{i+1}) \prod_{i=1}^{n} p(h_i, c_i).$$
Three common problems studied on HMM

- All three problems can be solved using dynamic programming:
  - First one with an algorithm called viterbi.
  - Second one with an algorithm called forward.
  - Third one with a combination of executing forward algorithm on \( c \) and its reverse (called backward algorithm then).
  - The difference in viterbi and forward is simply to replace max with sum.
Viterbi algorithm

- Compute a matrix $V[0...n+1,1...|H|]$ such that $V[j,k]$ equals

$$
\max_{h \in H^p(j), h_j = k} P(h | c_1 \ldots c_j) = \max_{h \in H^p(j-1)} \left( P(h | c_1 \ldots c_{j-1}) \max_{(h_{j-1}, k) \in T} p(h_{j-1}, k) p(k, c_j) \right).
$$

- That is, $V[j,k] = \max_{(k',k) \in T} V[j-1,k'] p(k',k) p(k,c_j)$.
- Let $k=1$ denote the start state and $k=|H|$ the end state.
  - Initialization: $V[0,1] = 1$.
  - Finalization: $V[n+1,|H|] = \max_{(k',|H|) \in T} V[n,k'] p(k',|H|)$.
- Probability of the most probable path is $V[n+1,|H|]$.
- The most probable path can be traced back checking which $V[n,k'] p(k',|H|)$ equals $V[n+1,|H|]$, and so on.
- Running time $O(n |T|)$. 


Forward algorithm

• Compute a matrix $F[0...n+1,1...|H|]$ such that $F[j,k]$ equals

$$
\sum_{h \in H \cap (j), h_j = k} P(h | c_1 \ldots c_j) = \sum_{h \in H \cap (j-1)} P(h | c_1 \ldots c_{j-1}) \sum_{(h_{j-1}, k) \in T} p(h_{j-1}, k) p(k, c_j).
$$

• That is, $F[j,k] = \sum_{(k', k) \in T} F[j-1, k'] p(k', k) p(k, c_j)$.

• Let $k=1$ denote the start state and $k=|H|$ the end state.
  
  ○ Initialization: $F[0,1] = 1$.
  ○ Finalization: $F[n+1,|H|] = \sum_{(k', |H|) \in T} F[n, k'] p(k', |H|)$.

• Probability of the sequence is $F[n+1,|H|]$. 
Backward algorithm

- Compute a matrix $B[0...n+1,1...|H|]$ such that $B[j,k]$ equals

$$\sum_{h \in H^s(j)} P(h | c_{n-j+1} \ldots c_n) = \sum_{h \in H^s(j-1)} \left( P(h | c_{n-j+2} \ldots c_n) \sum_{(k, h_1) \in T} p(k, h_1) p(k, c_{n-j+1}) \right).$$

- That is, $B[j,k] = \sum_{(k,k') \in T} B[j-1,k'] p(k,k') p(k, c_{n-j+1})$.

- Let $k=1$ denote the start state and $k=|H|$ the end state.
  - Initialization: $B[0,|H|] = 1$.
  - Finalization: $B[n+1,1] = \sum_{(1,k') \in T} B[n,k'] p(1,k')$.

- Probability of the sequence is $B[n+1,1]$. 
Probability of $c_j$ matching state $k$ 

- Can be computed through $F$ and $B$: 

$$\sum_{h \in H(n)} P(h \mid c_1 \ldots c_n, h_j = k)$$

$$= \sum_{h \in H^p(j), h_j = k} \left( P(h \mid c_1 \ldots c_j) \sum_{(k,k') \in T} p(k,k') \sum_{h' \in H^s(n-j), h'_1 = k'} P(h' \mid c_{j+1} \ldots c_n) \right)$$

$$= F[j, k] \sum_{(k,k') \in T} p(k,k') B[n - j + 1, k']$$
Log transform

- Multiplication is the source of numerical problems in HMM algorithms.
- However, easy to go over by transforming series of multiplications into summation of logarithms:
  \[
  \log \prod_i p_i = \sum_i \log p_i
  \]
- For example, in viterbi we can fill in table VL[] with rule
  \[
  VL[j,k] = \max_{(k',k) \in T} VL[j-1,k'] + \log p(k',k) + \log p(k,c_j).
  \]
- Then
  \[
  V[j,k] = 2^{VL[j,k]}
  \]
  with initialization \( VL[0,1]=1 \).
Where the probabilities?

- Learn from the data (Introduction to Machine Learning course).

- Given a set of valid annotated hidden paths (like DNA sequences with exon/intron annotation), it is easy to compute the frequencies of different emissions and transitions taken:
  - Add pseudocounts to cope with too sparse training data.

- Without the annotation, the task of optimizing the parameters is hard:
  - General local optimization routines such as EM algorithm / simulated annealing are used.
Part II

COMPLETING THE PIPELINES: CLUSTERING, UPGMA, NEIGHBOR JOINING
Recall: Phylogeny by distance method pipeline

Element 1: For all pairs of species, find the homologous genes.

Element 2: Compute the rearrangement distance for all pairs of species.

Element 3: Build the phylogenetic tree from the distances.

- Genome sequences of the species
- Permutations representing the homologs
- D(A,B) for all species A and B
Recall: Progressive multiple alignment example
Recall: Time series expression profiling

- It is possible to make a series of microarray experiments to obtain a time series expression profile for each gene.

- *Cluster* similarly behaving genes.
Clustering

- **Hierarchical clustering**
  - Iteratively join two closest clusters until forming a tree hierarchy (agglomerative... also divisive version exists)
  - Distance between clusters can be e.g. max pair-wise distance (complete linkage), min (single-linkage), UPGMA (average linkage), neighbor joining

- **Partitional clustering**
  - k-means, etc.
Distances in a phylogenetic tree

- Distance matrix $D = (d_{ij})$ gives pairwise distances for leaves of the phylogenetic tree.
- In addition, the phylogenetic tree will now specify distances between leaves and internal nodes.
  - Denote these with $d_{ij}$ as well.

Distance $d_{ij}$ states how far apart species $i$ and $j$ are evolutionary.
Distances in evolutionary context

- Distances $d_{ij}$ in evolutionary context satisfy the following conditions
  - Positivity: $d_{ij} \geq 0$
  - Identity: $d_{ij} = 0$ if and only if $i = j$
  - Symmetry: $d_{ij} = d_{ji}$ for each $i, j$
  - Triangle inequality: $d_{ij} \leq d_{ik} + d_{kj}$ for each $i, j, k$

- Distances satisfying these conditions are called **metric**

- In addition, evolutionary mechanisms may impose additional constraints on the distances
  - *additive* and *ultrametric* distances
Additive trees

- A tree is called *additive*, if the distance between any pair of leaves \((i, j)\) is the sum of the distances between the leaves and a node \(k\) on the shortest path from \(i\) to \(j\) in the tree

\[ d_{ij} = d_{ik} + d_{jk} \]
Additive trees: example

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

![Additive tree diagram](diagram.png)
A rooted additive tree is called an ultrametric tree, if the distances between any two leaves $i$ and $j$, and their common ancestor $k$ are equal

$$d_{ik} = d_{jk}$$

Edge length $d_{ij}$ corresponds to the time elapsed since divergence of $i$ and $j$ from the common parent

In other words, edge lengths are measured by a molecular clock with a constant rate.
Identifying ultrametric data

- We can identify distances to be ultrametric by the three-point condition:

  \[ D \text{ corresponds to an ultrametric tree if and only if for any three species } i, j \text{ and } k, \text{ the distances satisfy } d_{ij} \leq \max(d_{ik}, d_{kj}) \]

- If we find out that the data is ultrametric, we can utilise a simple algorithm to find the corresponding tree
Ultrametric trees

Observation time

Time
Ultrametric trees

Only vertical segments of the tree have correspondence to some distance $d_{ij}$:

Horizontal segments act as connectors.

Observation time
Ultrametric trees

\[ d_{ik} = d_{jk} \] for any two leaves \( i, j \) and any ancestor \( k \) of \( i \) and \( j \)
Observation time

Three-point condition: there are no leaves $i, j$ for which $d_{ij} > \max(d_{ik}, d_{jk})$ for some leaf $k$. 
UPGMA algorithm

- UPGMA (unweighted pair group method using arithmetic averages) constructs a phylogenetic tree via clustering
- The algorithm works by at the same time
  - Merging two clusters
  - Creating a new node on the tree
- The tree is built from leaves towards the root
- UPGMA produces a ultrametric tree
Let distance $d_{ij}$ between clusters $C_i$ and $C_j$ be

$$d_{ij} = \frac{1}{|C_i \cap C_j|} \sum_{p \in C_i, q \in C_j} d_{pq},$$

that is, the average distance between points (species) in the cluster.
UPGMA algorithm

- **Initialisation**
  - Assign each point $i$ to its own cluster $C_i$
  - Define one leaf for each sequence, and place it at height zero
- **Iteration**
  - Find clusters $i$ and $j$ for which $d_{ij}$ is minimal
  - Define new cluster $k$ by $C_k = C_i \cup C_j$, and define $d_{kl}$ for all $l$
  - Define a node $k$ with children $i$ and $j$. Place $k$ at height $d_{ij}/2$
  - Remove clusters $i$ and $j$
- **Termination:**
  - When only two clusters $i$ and $j$ remain, place root at height $d_{ij}/2$
In naive implementation, each iteration takes $O(n^2)$ time with $n$ sequences => algorithm takes $O(n^3)$ time.

The algorithm can be implemented to take only $O(n^2)$ time (see Gronau & Moran, 2006, for a survey).
Problem solved?

- We now have a simple algorithm which finds a ultrametric tree
  - If the data is ultrametric, then there is exactly one ultrametric tree corresponding to the data (proof left as an exercise)
  - The tree found is then the "correct" solution to the phylogeny problem, if the assumptions hold
- Unfortunately, the data is not ultrametric in practice
  - Measurement errors distort distances
  - *Basic assumption of a molecular clock does not hold usually very well*
Incorrect reconstruction of non-ultrametric data by UPGMA

Tree which corresponds to non-ultrametric distances

Incorrect ultrametric reconstruction by UPGMA algorithm