Sequence clustering

Introduction

Data clustering is one of the key tools used in various incarnations of data-mining - trying to make sense of large datasets. It is, thus, natural to ask whether clustering approaches can enhance our understanding of biological sequences. A common application of clustering is the analysis of 16S rRNA sequences extracted from an environment. In theory, each distinct rRNA sequence should represent a single organism, however, due to sequencing errors and recent evolutionary changes a single organism is represented by many closely related RNA sequences. Thus, by clustering the set of sequences we might infer how many organisms are present in our sample.

Clustering basics

Clustering approaches can be classified into three main categories:

- Agglomerative - algorithm starts with each sequence as an independent cluster, then iteratively joins together sequences to create larger clusters
- Divisive - algorithm starts with all the sequences in one cluster then iteratively breaks up the cluster into a collection of smaller clusters.
- Partitioning/partition methods - e.g. k-means - algorithm starts with a pre-defined set of clusters then refines the cluster membership to improve cluster quality.

Various approaches use different criteria for when the clustering should stop, usually based on some concept of cluster quality. The basic goal is to create clusters such that all sequences within a cluster are similar to each other(clusters are compact), and sequences contained in different clusters are dissimilar to each other (clusters are well separated). There are a variety of mathematical measures that capture one or both of these criteria (see http://machaon.karanagai.com/validation_algorithms.html).

It is important to note that the quality of a cluster can be assessed either in an unsupervised way (clustering algorithm and validation algorithm have access to the same information) and in a supervised way (validation algorithm has access to additional information that can be used to evaluate cluster quality).

Hierarchical clustering

Hierarchical clustering is a variant of agglomerative clustering that continues the agglomeration process until only one cluster remains. Essentially, through hierarchical clustering you construct a tree, having the sequences as leaves, that captures the relationship between the sequences at various levels. Clusters can be constructed by cutting the tree at particular points (imagine slicing a head of broccoli with one slice - each floret will be an individual cluster) - i.e. for a

given node, all its descendents are part of the same cluster.

The basic hierarchical clustering algorithm proceeds as follows, in a greedy manner:

1. Compute all pair-wise distances between the sequences
2. Merge together the sequences that are closest (most similar) to each other
3. Compute the distance between the newly created cluster and all other sequences/clusters
4. Repeat from 2.

Different hierarchical clustering methods differ in the way they define the distance between already computed clusters, or between clusters and individual sequences. Thus we have:

- Nearest neighbor (single linkage) - distance between clusters i and j is equal to the **smallest** distance among all pairs of sequences, the first from cluster i and the second from cluster j.
- Furthest neighbor (complete linkage) - distance between clusters i and j is equal to the **largest** distance among all pairs of sequences, the first from cluster i and the second from cluster j.
- Average neighbor (average linkage, UPGMA) - distance between clusters i and j is equal to the **average** distance among all pairs of sequences, the first from cluster i and the second from cluster j.
- Ward's clustering - the clusters being merged are the ones that will construct a cluster with the lowest variance in within-cluster distances.

Semi-supervised hierarchical clustering (VI-cut)

In general the goal of clustering is to uncover certain relationships between a set of previously unknown sequences. In many cases, however, these relationships are known, or can be inferred. In the 16S rRNA example above, we might already know the organisms associated with some of the sequences. This information could be used to validate the clusters constructed through one of the clustering methods. A good metric for comparing the "true" clustering to the computed clustering is the Variation of Information metric: VI(C1,C2) = H(C1) + H(C2) - 2 I(C1,C2), where H is the entropy, and I is the mutual information between these clusterings. VI(C1,C2) = 0 iff C1 and C2 are the same clustering.

Navlakha et al.(paper on syllabus page) showed that the VI distance has a nice decomposition property (it is easy to compute the VI distance between two clusterings given the distances between subsets of these clusterings) which allows the construction of a dynamic programming algorithm for finding the "perfect" cut-set in a hierarchical clustering tree - i.e. the set of cuts in the tree that maximizes the concordance between the resulting clusters and previously known clustering of a subset of the sequences. I.e. this results in a semi-supervised clustering approach.

Phylogenetic trees

The hierarchical clustering approaches described above can apply to any type of data, as long as a distance can be defined between the objects being clustered. One biological application of clustering attempts to infer the evolutionary history that gave rise to a set of sequences observed today. In this version, the distance between sequences attempts to capture the evolutionary relatedness of the sequences, or the time (usually measured as number of mutations) that separates the sequences from each other.

Thus, the goal is to reconstruct a tree that best captures the "heritage" of a set of sequences - the root of the tree being the most recent common ancestor of all the sequences. In the context of protein sequences, the BLOSUM and PAM matrices discussed during the sequence alignment lectures, provide a reasonable approximation of the evolutionary distance. For nucleotide sequences, several models of evolution apply that attempt to estimate the likelihood a certain base will mutate into another base during a given evolutionary time. The evolutionary time separating two sequences can be estimated from the number of identities in the alignment between the sequences.

Neighbor-joining

Simple hierarchical clustering approaches have been applied in the context of phylogenetic tree reconstruction, specifically, the UPGMA (average linkage) clustering algorithm. One limitation of this algorithm is that it does not allow for uneven mutation rates along different branches of a tree. A solution is proposed by the neighbor-joining algorithm of Saitou and Nei. The basic idea is to correct the distance between two sequences/clusters by a factor that depends on the distance between each sequence and the rest of the sequences. Part of the intuition is that two distant sequences should be clustered together if they are "isolated" (far from) the rest of the sequences.

$$NJD(i,j) = d(i,j) - \frac{1}{n-2}\left(\sum_{k \neq i,j} d(i,k) + d(j,k)\right)$$

where d(i,j) is the distance between sequences i and j (out of n sequences).

The two sequences with the lowest neighbor-joining distance are clustered together. Whenever two sequences are clustered together, a new node (cluster) m is created and it's distance to all other sequences/clusters is computed as follows:

$$d(m,k) = \frac{1}{2}(d(i,k) + d(j,k) - d(i,j)) \quad \forall\, k \neq i,j$$

$$d(i,m) = \frac{1}{2}\left(d(i,j) + \frac{1}{n-2}\left(\sum_{k \neq i,j} d(i,k) - \sum_{k \neq i,j} d(j,k)\right)\right)$$

$$d(j,m) = \frac{1}{2}\left(d(i,j) + \frac{1}{n-2}\left(\sum_{k \neq i,j} d(j,k) - \sum_{k \neq i,j} d(i,k)\right)\right)$$

Then the process is repeated until all sequences belong to a single cluster (the root of the tree).

The neighbor-joining algorithm can be shown to produce the optimal phylogenetic tree - in terms of reconstructing the true evolutionary history of a set of sequences - if the pairwise distances between sequences are additive - d(i,j) = d(i,k) + d(k,j) for any sequences i,j,k. This property should hold for true evolutionary distances however is unlikely to hold for distances computed heuristically, e.g. through inexact sequence alignment.

Computing parsimony/likelihood scores

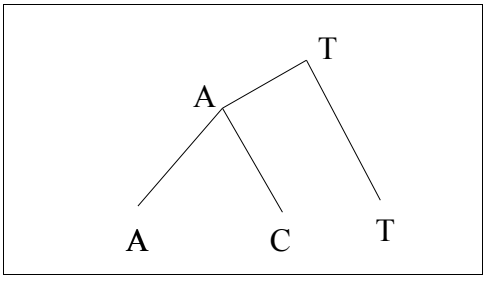
Complementary to distance based methods, which assume that we can accurately estimate the evolutionary distance between sequences without using any information about their heritage, are approaches that define global measures of the quality of a phylogenetic tree.

The two major classes of such methods are *parsimony* (best tree is the one that minimizes the number of mutations during evolution) and *likelihood* (best tree is the one that optimizes the "fit" with a predefined stochastic model of evolution). A key component of these methods is an algorithm for computing the "goodness" of a tree, given a set of sequences stored at its leaves. For simplicity, I will only focus on the parsimony approach, however the algorithm can be easily adapted to compute the likelihood of a tree.

Also, I will focus on a special case where each sequence is comprised of a single nucleotide - for longer sequences the parsimony score (to be defined) is simply the sum of the parsimony scores of each individual character. Note that in phylogeny we generally try to work with "traits" (in our case nucleotides) that can be inferred to be related to each other. In the context of sequence analysis a trait is represented by one column in the multiple alignment of the sequences being aligned, as we can infer that all the letters in that column have been derived, through evolution, from a same ancestral nucleotide. If this assumption is incorrect, so will the conclusions of the analysis - hence the "obsession" in the field for manually curating multiple alignments.

Problem definition: Given a phylogenetic tree (assume binary for now but not an essential restriction) having the leaves labeled with letters from a given alphabet (ACGT for nucleotides): determine a labeling of the internal nodes of the tree (i.e. ancestral sequences) that minimizes the sum, over all the edges in the tree, of the number of differences between adjacent letters.

In other words, once you label all the nodes of a tree, you can label all the edges with a 1 if the ends of the edge have different labels, or 0 if they have the same label. We are looking for the assignment of labels in the tree that minimizes the number of 1s (in the picture below the parsimony score is 2).



The computation of the parsimony score, and the assignment of ancestral sequences, can be easily done with dynamic programming. Specifically, we'll define V[n, c] for all nodes n in the tree, and all characters c in the alphabet, to be the minimum parsimony score for the subtree rooted at n, given the root is labeled c.

V[n,c] can clearly be easily computed for the leaves by setting V[leaf, leaf_label] = 0, and V[leaf, not_leaf_label] = -infty.

Also, once the V values have been computed for the children of a node n , the V values for n can be easily computed with the following recurrence:

$$V[n,c] = min_{i \in \Sigma}(V[l,i] + \{1, if\ i \neq c\}) + min_{i \in \Sigma}(V[r,i] + \{1, if\ i \neq c\})$$

where l and r are the left and right children of n, respectively.

Using this recurrence, the tree can be filled in using a post-order traversal.

The parsimony score is the minimum value in V[root,]. The ancestral states can be easily assigned through backtracking.

The algorithm described in this section is due to David Sankoff and is commonly known as Sankoff's algorithm.

Finding the best tree

While Sankoff's algorithm provides a way to evaluate (in polynomial time) the quality of a given tree, finding the tree that maximizes the quality is NP-hard. Most algorithms for constructing phylogenetic trees rely on heuristic search through the space of possible trees. A discussion of these algorithms is beyond the scope of this class.

Greedy clustering

Most of the clustering approaches described above require the computation of all pairwise distances between a set of sequences, i.e. the running time is proportional to n^2, where n is the number of sequences. Given the large size of datasets commonly seen in current research (millions to billions of sequences), such algorithms are impractical.

An alternative is provided by greedy clustering algorithms, exemplified by CD-hit (Li et al.) The basic algorithm is as follows:

1. sort the sequences in decreasing order of their lengths
2. pick a sequence not assigned to a cluster - this sequence becomes the center of a new cluster
3. compare all unasigned sequences to already computed cluster centers, first using a quick k-mer distance approach, then checking the promising alignments with a full smith-waterman algorithm.
4. repeat from 2.

The reason sequences are first sorted by length is to enable the comparison of short sequences

that match different parts of a longer cluster center.

Note that only sequences within a prescribed radius (in terms of similarity) from the cluster center are added to a cluster - the goal of the algorithm is to create clusters that are fairly tight.

This basic algorithm can be implemented in various flavors:

- the sequences could be indexed making it easy to find all the sequences that match a given cluster center (approach used in DNAclust - Mohammad Ghodsi's code)
- the clusters could be indexed, making it easy to find the appropriate cluster center for each sequence
- the algorithm can be sped up by relaxing the criteria that define the distance from the center of a cluster (e.g. the clustering tool UClust does not implement the full dynamic programming algorithm).