1 Clustering

1.1 Introduction

The goal of clustering a microarray is to find a partition of the genes (or arrays) into groups, according to a common expression pattern. The problem is that it is hard to define what is a good result for clustering and it depends on the specific case. This is why there are many different clustering methods and we won’t go through them all. These are the main clustering categories:

1. **Model based**: the main example is K-means - Iteratively find centroids (means), and then divide the genes into groups according to the closest centroid.

2. **Hierarchical clustering**: The main advantage of this method is that we can divide the genes to groups in different resolutions, but it is not always clear which resolution to choose. There are two ways of implementing the hierarchical clustering:
   
   (a) **Top-down**: Ratio cuts and Normalized cuts: We start with all the nodes in one cluster and then we part it into two clusters and so on, we would want the distance between the two clusters to be maximized.

   (b) **Bottom-up (Agglomerative Clustering)**: NJ and UPGMA, the output we get from these algorithms is a tree, if we “cut” the tree at a certain height we will get a group of separate subtrees which can be treated as clusters.

1.2 Agglomerative Clustering

We will use a variation of UPGMA - in UPGMA we have a distance matrix, in each iteration we take the pair \(i, j\) with the minimal distance: \(\forall k, l : d_{i, j} < d_{k, l}\) and create a new node which is their parent. In the UPGMA algorithm the new distances for this node is the average of the two distances of it’s children: \(\forall l : D_{i+j,l} = \frac{1}{2}(d_{i,l} + d_{j,l})\).

We will talk about a very similar idea for clustering genes which is known as **Eisen clustering**: if we measure the expression of the genes in \(m\) different conditions, each gene is represented by a vector in \(\mathbb{R}^m\). if we are joining two genes \(i, j\) then we create a new “gene” \((i + j)\) with a new vector which is defined: \(\forall a : X_{i+j,a} = \frac{1}{2}(X_{i,a} + X_{j,a})\).

If we use the Euclidean distance then this is the same as normal UPGMA, but instead, in this method we use **correlation** - a measure to how related are two vectors. An intuition for correlation is this: if we have two vectors \(X, Y \in \mathbb{R}^n\) and we want to represent \(Y\) as a linear combination of \(X\) (i.e.: \(Y_i = aX_i + b\)) then \(a\) is the correlation between \(X\) and \(Y\) and the larger \(a\) is means \(X\) and \(Y\) are more correlated. Un-Normalized correlation:

\[
\arg \min_a \sum_{i=1}^n (Y_i - aX_i + b)^2
\]
In normalized correlation each vector $X$ has $E[X] = 0$ and $Var(X) = 1$ this can be done this way:

\[ \forall i : \hat{X}_i = \frac{X_i - E[X]}{std(X)}, \hat{Y}_i = \frac{Y_i - E[Y]}{std(Y)} \]

and we get normalized correlation (Pearson’s Correlation):

\[
\arg \min_{a} \sum_{i=1}^{n} (\hat{Y}_i - a\hat{X}_i)^2
\]

For illustration if we’ve got two vectors that behave like in Figure 1 then their un-normalized and normalized correlation will look like in Figure 1.2:

Figure 1: we see the values of gene X and Y as a function of the samples(arrays)

(a) the un-normalized version of the linear approximation  (b) the normalized version of the linear approximation

Why did Eisen want to cluster genes according to the correlation and not the regular(Euclidean) distance? This is because we are more interested in genes with the same pattern - that are over-expressed at the same situations and
under-expressed at the same situations and not necessary the ones which are the closest which can mean that in some places they are very close and in some places they do totally different things. The second reason is that our reference sample (our zero) is not completely reliable, so we wouldn’t want to give too much weight to the zero line - this is why we would want to use the normalized correlation, which ignores the zero line.

A problem with the normalized correlation may occur when we have a gene that doesn’t change its expression, in this case the standard deviation which is normalized to 1 is actually noise. This is usually not so bad because usually this noise won’t be correlative to other genes.

This is the general algorithm:
repeat until there is only one gene:
  Step 1: calculate the distance function (Euclidean, correlation, normalized correlation etc’)
  step 2: choose the closest pair according to this function (the minimal distance or maximal correlation)
  step 3: build the new distances in the tree according to the given function.

1.3 cutting the tree

So now that we’ve got a tree we need to use it for the clustering, but how do we decide where to “cut” it? how do we decide which cut will give us the best clustering?

One way to show the tree is to give the same height to each level of the tree as shown in Figure 2 and number each inner node as the number of iteration it was created, if we cut the tree at a certain height it is as if we stopped after a certain number of iterations.

![Figure 2](image)

Figure 2: a UPGMA tree with uniform height, the numbers are the number of iteration the junction was created

One option to decide which ”cut” is a good one is to use a statistical test in order to estimate what is the chance to get a certain set of clusters in random. The question is what statistical test to use.

![Figure 3](image)

Figure 3: a UPGMA tree with heights representing distance, we would cut the tree where the red line is
A second option is to show the tree in a way that will express the similarity of the nodes we connected. The height of the new node will be directly correlated to the distance between his sons. The question of determining the height of the node is not trivial - for example correlation isn’t even monotonic, but if we can find a way to do this then the height is informative and we can use this to cut the tree in a good location. We can cut it where the difference in heights starts to be bigger, this is shown in Figure 3.

The third option is to choose according to \( \frac{\text{in}}{\text{out}} \). We would like to compare between cuts according to the ratio between the distances within the cluster and the distances between clusters.

A forth option is to check the robustness of the clustering. We can build many trees and each time add “noise” to the tree, i.e. each time we don’t use some of the arrays in the calculations. If we see that a cluster(sub-tree) appears in most of the trees we can assume that it is a real cluster.

There is no one way to do the clustering and each of the given strategies can be used. Even after we choose one strategy we still have many options to cut the tree. On the other hand in many cases the clustering will be clear from the tree without using any clustering method.

1.4 optimal leaf ordering

One of the most important uses of making trees and clustering the microarray results is to order the genes or arrays according to the tree so it is easy to visualize the microarray data, this is shown in Figure 4.

Given a specific tree there are approximately \( 2^{N-1} \) ways to order the leaves such that the order will be consistent with the tree (because for each inner node we can swap between the right and left sons but there is symmetry between some of the options). This is much less than the possible options without a tree (\( N! \)) but it is still quite a lot of options. We would like to get a matrix which looks good - that we can learn the most from looking at it.

A computational question that we can define for this concept is:

Given \( T \) with leaves \( 1 \ldots N \) and we look at permutations defined as \( \sigma : \{1 \ldots N\} \mapsto \{1 \ldots N\} \) then:

\[
\sigma_{\text{opt}} = \arg \min_{\sigma} \sum_{i} d_{\sigma(i)\sigma(i+1)}
\]

s.t.

\( \sigma \) is consistent with \( T \)

We will use a dynamic programing algorithm, if we look at two adjacent sub trees as in Figure 5, we can see that if we fix the extreme left node of the right sub-tree and the extreme right node of the left sub-tree then we can address each sub-tree as a separate problem.
We define a cost function $V[t,i,j] = \text{cost of best internal order in the sub-tree rooted by } t$ if $i, j$ are extremes, this cost doesn’t depend on the order of the other sub-trees. If we denote the right and left sons of $t$ as $t_r$ and $t_l$ then the recursive step is:

$$V[t,i,j] = \min_{k \in t_l, l \in t_r} [V[t_l,i,k] + d_{k,l} + V[t_r,l,j]]$$

The base case is trees of 2 nodes where the cost is the distance between the two nodes, after we fill the $V$ table the best order for the tree $T$ is given by:

$$\min_{i,j} V[T,i,j]$$

What is the size of $V$? if every combination of $t, i, j$ is possible we would get $(N - 1) * N * N = O(N^3)$ but we can observe that every pair $i, j$ can appear only with their last common ancestor - $LCA(i, j)$ because we need that they will be in the same tree but not in the same sub-tree, so each $i, j$ can appear only once (or twice if we count both $(i, j)$ and $(j, i)$) in the $V$ matrix so the real size of $V$ is $O(N^2)$.

What is the time complexity? for each cell in $V$ we make $O(N^2)$ calculations so to calculate all $V$ we need $O(N^4)$ calculations. We would want to reduce the complexity, we can do this with another dynamic programing:

We define a new matrix $V_t$ which will reduce the calculations of each cell in $V$:

$$V_t[k,j] = \min_{l} d_{k,l} + V[t_r,l,j]$$

and then we can use it to calculate $V$:

$$V[t,i,j] = \min_{k} [V[t_l,i,k] + V_t[k,j]]$$

calculating all the $V_t$’s takes $O(N^3)$ and so does calculating $V$ so we reduced the total complexity to $O(N^3)$. 

Figure 5: if we fix the nodes k and l then each subtree is a separate problem