Hierarchical Clustering

Thanks to Olga Troyanskaya & Kevin Wayne for many of the slides
Microarrays: Genome-scale measurement of gene expression levels

1. Spot slide with known sequences
2. Add mRNA to slide for Hybridization
3. Hybridize
4. Add green dye
5. Add red dye
6. Scan hybridized array

Reference mRNA

Test mRNA

A: 1.5
B: 0.8
C: -1.2
D: 0.1
Each gene is represented as a vector of expression values across experiments.

<table>
<thead>
<tr>
<th>Gene</th>
<th>Exp. 1</th>
<th>Exp. 2</th>
<th>Exp. 3</th>
<th>Exp. 4</th>
<th>Exp. 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gene 1</td>
<td>0.78</td>
<td>3.22</td>
<td>-0.32</td>
<td>1.87</td>
<td>0.50</td>
</tr>
<tr>
<td>Gene 2</td>
<td>1.73</td>
<td>-0.92</td>
<td>2.06</td>
<td>2.67</td>
<td>-0.09</td>
</tr>
<tr>
<td>Gene 3</td>
<td>1.99</td>
<td>3.60</td>
<td>-2.48</td>
<td>0.10</td>
<td>1.56</td>
</tr>
<tr>
<td>Gene 4</td>
<td>0.60</td>
<td>0.26</td>
<td>0.11</td>
<td>0.21</td>
<td>0.41</td>
</tr>
<tr>
<td>Gene 5</td>
<td>0.44</td>
<td>0.84</td>
<td>0.42</td>
<td>0.84</td>
<td>0.86</td>
</tr>
<tr>
<td>Gene 6</td>
<td>0.07</td>
<td>0.18</td>
<td>1.49</td>
<td>0.30</td>
<td>0.05</td>
</tr>
<tr>
<td>Gene 7</td>
<td>0.28</td>
<td>0.49</td>
<td>0.47</td>
<td>0.93</td>
<td>0.89</td>
</tr>
<tr>
<td>Gene 8</td>
<td>0.91</td>
<td>0.95</td>
<td>0.76</td>
<td>0.22</td>
<td>0.00</td>
</tr>
<tr>
<td>Gene 9</td>
<td>0.72</td>
<td>1.51</td>
<td>0.35</td>
<td>0.54</td>
<td>0.81</td>
</tr>
<tr>
<td>Gene 10</td>
<td>-3.34</td>
<td>0.83</td>
<td>0.62</td>
<td>3.15</td>
<td>0.71</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Microarray Data Visualization and Analysis

• The order of genes in data matrix can be reorganised for better recognition of expression patterns
Clustering genes in microarray experiments

- Function prediction
- Pathway discovery
- Gene regulation studies
- Comparative genomics
- Disease pathways
How does Hierarchical Clustering work?

1. Compare all expression patterns to each other.

2. Join patterns that are the most similar out of all patterns.

3. Readjust similarity of newest pattern to all other patterns.

4. Go to step 2, and repeat until all patterns are joined.
Hierarchical Clustering: Visually
Dendrogram
AKA "Clustering tree"
Dendrogram. Scientific visualization of hierarchies.

- Leaves = genes.
- Internal nodes = clusters.

Dendrogram of Human tumors

Tumors in similar tissues cluster together.

Reference: Botstein & Brown group
Ancestor Tree

Root. Node with no parent.
Leaf. Node with no children.
Depth. Length of path from node to root.
Least common ancestor. Common ancestor with largest depth.

```java
public static void main(String[] args) {
    TreeNode a = new TreeNode("GENE1");
    TreeNode b = new TreeNode("GENE2");
    TreeNode c = new TreeNode("GENE3");
    TreeNode d = new TreeNode("GENE4");

    TreeNode x = new TreeNode("NODE1", b, c);
    TreeNode y = new TreeNode("NODE2", a, x);
    TreeNode z = new TreeNode("NODE3", d, y);

    System.out.println(a.lca(b));
    a.lca(b).showLeaves();
}
```

lca(a, b) = y, leaves of y = { a, b, c }
Ancestor Tree: Java Implementation

Node. Left pointer, right pointer, parent pointer.
Consequence. Can go up or down the tree.
public class TreeNode {
    private TreeNode parent;       // parent
    private TreeNode left, right;  // two children
    private String name;           // name of node

    // create a leaf node
    public TreeNode(String name) {
        this.name = name;
    }

    // create an internal node that is the parent of x and y
    public TreeNode(String name, TreeNode x, TreeNode y) {
        this.name = name;
        this.left = x;
        this.right = y;
        x.parent = this;
        y.parent = this;
    }
}
// return depth of this node in the tree  
// depth of root = 0
public int depth() {
    int depth = 0;
    for (TreeNode x = this; x.parent != null; x = x.parent)
        depth++;
    return depth;
}

// return root
public TreeNode root() {
    TreeNode x = this;
    while (x.parent != null)
        x = x.parent;
    return x;
}
Ancestor Tree: Least Common Ancestor

// return the lca of node x and y
public TreeNode lca(TreeNode y) {
    TreeNode x = this;
    int dx = x.depth();
    int dy = y.depth();
    if (dx < dy) {
        for (int i = 0; i < dy-dx; i++) y = y.parent;
    }
    else {
        for (int i = 0; i < dx-dy; i++) x = x.parent;
    }
    while (x != y) {
        x = x.parent;
        y = y.parent;
    }
    return x;
}
// return string representation
public String toString() { return name; }

// print all leaves in tree rooted at this node
public void showLeaves() {
    if (left == null && right == null) System.out.println(this);
    else {
        left.showLeaves();
        right.showLeaves();
    }
}

// print the tree rooted at this node
public void show() {
    if (left == null && right == null) return;
    System.out.println(name + " " + left.name + " " + right.name);
    left.show();
    right.show();
}
Hierarchical clustering implementation
Vector Data Type

Vector data type.
• Set of values: sequence of N real numbers.
• Set of operations: distanceTo, scale, plus.

Ex.  \( p = (1, 2, 3, 4), q = (5, 2, 4, 1) \).
• \( \text{dist}(p, q) = \sqrt{4^2 + 0^2 + 1^2 + 3^2} = 5.099. \)
• \( t = \frac{1}{4} p + \frac{3}{4} q = (4, 2, 3.75, 1.75). \)

```java
public static void main(String[] args) {
    double[] pdata = { 1.0, 2.0, 3.0, 4.0 };
    double[] qdata = { 5.0, 2.0, 4.0, 1.0 }
    Vector p = new Vector(pdata);
    Vector q = new Vector(qdata);
    double dist = p.distanceTo(q);
    Vector r = p.scale(1.0/4.0);
    Vector s = q.scale(3.0/4.0);
    Vector t = r.plus(s);
    System.out.println(t);
}
```
public class Vector {
    private int N;       // dimension
    private double[] data; // components

    // create a vector from the array d
    public Vector(double[] d) {
        N = d.length;
        data = d;
    }

    // return Euclidean distance from this vector a to b
    public double distanceTo(Vector b) {
        Vector a = this;
        double sum = 0.0;
        for (int i = 0; i < N; i++)
            sum += (a.data[i] - b.data[i])*(a.data[i] - b.data[i]);
        return Math.sqrt(sum);
    }
}
public String toString() {
    String s = "";
    for (int i = 0; i < N; i++)
        s = s + data[i] + " ";
    return s;
}

public Vector plus(Vector b) {
    Vector a = this;
    double[] d = new double[N];
    for (int i = 0; i < N; i++)
        d[i] = a.data[i] + b.data[i];
    return new Vector(d);
}
Single-Link Hierarchical Clustering

Iteration.
- Closest pair of clusters \((i, j)\) is one with the smallest dist value.
- Replace row \(i\) by min of row \(i\) and row \(j\).
- Infinity out row \(j\) and column \(j\).
- Update dmin[\(i\)] and change dmin[\(i'\)] to \(i\) if previously dmin[\(i'\)] = \(j\).

![Matrix and Table]

- Closest pair
- New min dist
- Gene1 closest to gene3, dist=2.14
Single-Link Clustering: Java Implementation

Single-link clustering.
• Read in the data.

```java
public static void main(String[] args) {
    int M = StdIn.readInt();
    int N = StdIn.readInt();

    // read in N vectors of dimension M
    Vector[] vectors = new Vector[N];
    String[] names    = new String[N];
    for (int i = 0; i < N; i++) {
        names[i] = StdIn.readString();
        double[] d = new double[M];
        for (int j = 0; j < M; j++)
            d[j] = StdIn.readDouble();
        vectors[i] = new Vector(d);
    }
}
```
Single-Link Clustering: Java Implementation

Single-link clustering.

• Read in the data.
• Precompute \( d[i][j] = \text{distance between cluster } i \text{ and } j \).
• For each cluster \( i \), maintain index \( d_{\text{min}}[i] \) of closest cluster.

```java
double INFINITY = Double.POSITIVE_INFINITY;
double[][] d = new double[N][N];
int[] dmin = new int[N];
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        if (i == j) d[i][j] = INFINITY;
        else d[i][j] = vectors[i].distanceTo(vectors[j]);
        if (d[i][j] < d[i][dmin[i]]) dmin[i] = j;
    }
}
```
for (int s = 0; s < N-1; s++) {
    // find closest pair of clusters (i1, i2)
    int i1 = 0;
    for (int i = 0; i < N; i++)
        if (d[i][dmin[i]] < d[i1][dmin[i1]]) i1 = i;
    int i2 = dmin[i1];

    // overwrite row i1 with minimum of entries in row i1 and i2
    for (int j = 0; j < N; j++)
        if (d[i2][j] < d[i1][j]) d[i1][j] = d[j][i1] = d[i2][j];
    d[i1][i1] = INFINITY;

    // infinity-out old row i2 and column i2
    for (int i = 0; i < N; i++)
        d[i2][i] = d[i][i2] = INFINITY;

    // update dmin and replace ones that previous pointed to
    // i2 to point to i1
    for (int j = 0; j < N; j++) {
        if (dmin[j] == i2) dmin[j] = i1;
        if (d[i1][j] < d[i1][dmin[i1]]) dmin[i1] = j;
    }
}
Store Centroids in Each Internal Node

Cluster analysis.
Centroids distance / similarity.

Easy modification to TreeNode data structure.

- Store Vector in each node.
  - leaf nodes: directly corresponds to a gene
  - internal nodes: centroid = average of all leaf nodes beneath it
- Maintain count field in each TreeNode, which equals the number of leaf nodes beneath it.
- When setting z to be parent of x and y,
  - set z.count = x.count + y.count
  - set z.vector = $\alpha p + (1-\alpha)q$, where $p = x$.vector and $q = y$.vector, and $\alpha = x$.count / z.count
Analysis and Micro-Optimizations

Running time. Proportional to $MN^2$ ($N$ genes, $M$ arrays)
Memory. Proportional to $N^2$.

Ex. $[M = 50, N = 6,000]$ Takes 280MB, 48 sec on fast PC.

Some optimizations.

• Use float instead of double
• Store only lower triangular part of distance matrix
• Use squares of distances instead of distances.

How much do you think would this help?
The End