Clustering II

With application to gene-expression profiling technology

Arjun Krishnan

Thanks to Kevin Wayne, Matt Hibbs, & SMD for a few of the slides
Hierarchical Clustering

Thanks to Kevin Wayne for many of the slides
Hierarchical clustering
Tumors in similar tissues cluster together

Reference: Botstein & Brown group

Gene 1

Gene n

Skin  Liver  Lung  Breast Tumors Luminal  Breast Tumors Basal  Normal  Kidney  Prostate  Brain  APL  Ovary

\(\text{gene over expressed}\)

\(\text{gene under expressed}\)
Hierarchical clustering

Start with each gene in its own cluster

Merge the closest pair of clusters into a single cluster

Until all genes are merged into a single cluster

Compute distance b/w new cluster and each of the old clusters

Merges are “greedy”

Complexity is at least $O(n^2)$; Naïve $O(n^3)$

S. C. Johnson (1967)
"Hierarchical Clustering Schemes"
Psychometrika, 2:241-254
Hierarchical clustering

• Distance metric

• Linkage criteria
  • Single/Minimum linkage (nearest neighbors)
  • Complete/Maximum linkage (farthest neighbors)
  • Average linkage (average of all pairs)
Single-Link Hierarchical Clustering

**Input.** Pre-computed matrix of distances between all pairs of genes \((i, j)\).

**Scheme.** Erase rows and columns in the distance matrix as old clusters are merged into new ones.

**Begin.**
Each gene in its own cluster.
For each cluster \(i\), create/maintain index \(d_{\text{min}}[i]\) of closest cluster.

```
<table>
<thead>
<tr>
<th>d_{\text{min}}</th>
<th>gene0</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>5.5</td>
<td>7.3</td>
<td>8.9</td>
<td>5.8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5.5</td>
<td>-</td>
<td>6.1</td>
<td>2.14</td>
<td>5.6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7.3</td>
<td>6.1</td>
<td>-</td>
<td>7.8</td>
<td>5.6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8.9</td>
<td>2.14</td>
<td>7.8</td>
<td>-</td>
<td>5.5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5.8</td>
<td>5.6</td>
<td>5.6</td>
<td>5.5</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2.14</td>
<td>7.8</td>
<td>-</td>
<td>5.5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>5.5</td>
<td>5.5</td>
<td>5.5</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>d_{\text{min}}</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>5.5</td>
<td>7.3</td>
<td>8.9</td>
<td>5.8</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2.14</td>
<td>5.6</td>
<td>5.6</td>
<td>5.5</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5.6</td>
<td>5.6</td>
<td>5.5</td>
<td>5.5</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>5.5</td>
<td>5.6</td>
<td>5.6</td>
<td>5.5</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>5.5</td>
<td>5.6</td>
<td>5.6</td>
<td>5.5</td>
</tr>
</tbody>
</table>
```
Single-Link Hierarchical Clustering

**Iteration.**
- Find closest pair of clusters \((i_1, i_2)\).
- Replace row \(i_1\) by min of row \(i_1\) and row \(i_2\).
- Infinity out row \(i_2\) and column \(i_2\).
- Update \(d_{min}[i]\) and change \(d_{min}[i']\) to \(i_1\) if previously \(d_{min}[i'] = i_2\).

<table>
<thead>
<tr>
<th>(d_{min})</th>
<th>dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>gene0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

Closest pair

New min distance

gene1 closest to gene3, dist = 2.14
Single-Link Clustering: Main Loop

for (int s = 0; s < N - 1; s++) {

Find closest pair of clusters (i1, i2).

Replace row i1 by min of row i1 and row i2.

Infinity out row i2 and column i2.

Update dmin[i]
Change dmin[i'] to i1 if previously dmin[i'] = i2.

}
Single-Link Clustering: Main Loop

```cpp
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;
    }
}
```
Dendrogram

- Leaves = genes.
- Internal nodes = hypothetical ancestors.

Dendrogram of Human tumors

Tumors in similar tissues cluster together.

Reference: Botstein & Brown group

gene over expressed

gene under expressed
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.

```
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: Implementation

**Node.** Left pointer, right pointer, parent pointer. **Consequence.** Can go up or down the tree.

How would you represent a node in Java to be able to find LCAs or Children? Hint: think pointers!

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;
    }
}
Ancestor Tree: Helper Functions

```java
// 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;
}
```
// 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;
}
public String toString() { return name; }

public void showLeaves() {
    if (left == null && right == null) System.out.println(this);
    else {
        left.showLeaves();
        right.showLeaves();
    }
}

public void show() {
    if (left == null && right == null) return;
    System.out.println(name + " " + left.name + " " + right.name);
    left.show();
    right.show();
}
Hierarchical clustering implementation
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;
    }
}
```

<table>
<thead>
<tr>
<th>dmin</th>
<th>dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>gene0</td>
<td>-</td>
<td>5.5</td>
<td>7.3</td>
<td>8.9</td>
<td>5.8</td>
</tr>
<tr>
<td>1</td>
<td>5.5</td>
<td>-</td>
<td>6.1</td>
<td>2.14</td>
<td>5.6</td>
</tr>
<tr>
<td>2</td>
<td>7.3</td>
<td>6.1</td>
<td>-</td>
<td>7.8</td>
<td>5.6</td>
</tr>
<tr>
<td>3</td>
<td>8.9</td>
<td>2.14</td>
<td>7.8</td>
<td>-</td>
<td>5.5</td>
</tr>
<tr>
<td>4</td>
<td>5.8</td>
<td>5.6</td>
<td>5.6</td>
<td>5.5</td>
<td>-</td>
</tr>
</tbody>
</table>
Single-Link Clustering: Main Loop

\[
\text{for} \ (\text{int} \ s = 0; \ s < N - 1; \ s++) \ \{ \\
\]

Find closest pair of clusters (i1, i2).

Replace row i1 by min of row i1 and row i2.

Infinity out row i2 and column i2.

Update dmin[i]
Change dmin[i'] to i1 if previously dmin[i'] = i2.

\[
\}
\]
Single-Link Clustering: Main Loop

```c
for (int s = 0; s < N-1; s++) {
    // find 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.
    
    for (int i = 0; i < N; i++) {
        if (d[i] [dmin[i]] < d[i1] [dmin[i1]]) i1 = i;
    }
    int i2 = dmin[i1];
    
    for (int j = 0; j < N; j++) {
        if (d[i2] [j] < d[i1] [j]) d[i1] [j] = d[j] [i1];
    }
    d[i1] [i1] = INFINITY;
    for (int i = 0; i < N; i++) d[i2] [i] = d[i] [i2] = INFINITY;
    
    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;
    }
}
```

<table>
<thead>
<tr>
<th>dmin</th>
<th>dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>gene0</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>5.5</td>
<td>7.3</td>
<td>8.9</td>
<td>5.8</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>5.5</td>
<td>-</td>
<td>6.1</td>
<td>2.14</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>7.3</td>
<td>6.1</td>
<td>-</td>
<td>7.8</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8.9</td>
<td>2.14</td>
<td>7.8</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5.8</td>
<td>5.6</td>
<td>5.6</td>
<td>5.5</td>
</tr>
</tbody>
</table>

closest pair: (1, 3), dist = 2.14

<table>
<thead>
<tr>
<th>dmin</th>
<th>dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>node1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>5.5</td>
<td>7.3</td>
<td>-</td>
<td>5.8</td>
</tr>
<tr>
<td>5.5</td>
<td>-</td>
<td>6.1</td>
<td>-</td>
<td>5.5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7.3</td>
<td>6.1</td>
<td>-</td>
<td>-</td>
<td>5.6</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>5.6</td>
</tr>
<tr>
<td>4</td>
<td>5.8</td>
<td>5.5</td>
<td>5.6</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

new min dist: 2.14
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?
Hierarchical clustering: problems

- Hard to define distinct clusters
- Genes assigned to clusters on the basis of all experiments
- Optimizing node ordering hard (finding the optimal solution is NP-hard)
- Can be influenced by one strong cluster – a problem for gene expression b/c data in row space is often highly correlated
Distance Metrics

- Choice of distance measure is important for most clustering techniques
- Linear measures: Euclidean distance, Pearson correlation
- Non-parametric: Spearman correlation, Kendall’s tau

\[
d = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - y_i)^2}
\]

\[
r = \frac{1}{n} \sum_{i=1}^{n} \left( \frac{x_i - \bar{x}}{\sigma_x} \right) \left( \frac{y_i - \bar{y}}{\sigma_y} \right)
\]

\[
\rho = 1 - \frac{6 \sum_{i=1}^{n} [rank(x_i) - rank(y_i)]}{n(n^2 - 1)}
\]
Distance Metrics

Consider the following plot of 3 pairs of genes

- **Y** (No correlation)
- **X**
- **V** (Positive correlation)
- **U**
- **Z** (Negative correlation)
- **W**
Pearson correlation ($r$) is a measure of the linear correlation (dependence) between two variables $X$ and $Y$.

$$r = \frac{1}{n-1} \sum_{i=1}^{n} \left( \frac{X_i - \bar{X}}{s_X} \right) \left( \frac{Y_i - \bar{Y}}{s_Y} \right)$$

$+1 \leq r \leq -1$

+1 is total positive correlation

0 is no correlation

−1 is total negative correlation.
Distance Metrics

11 datapoints

Mean (x) = 9
Var (x) = 11

Mean (y) = 7.50
Var (y) ~ 4.12

Cor (x, y) = 0.816

Linear regression line:
y = 3.00 + 0.500x

Distance Metrics

- Choice of distance measure is important for most clustering techniques
- Linear measures: Euclidean distance, Pearson correlation
- Non-parametric: Spearman correlation, Kendall’s tau

\[ d = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - y_i)^2} \]

\[ r = \frac{1}{n} \sum_{i=1}^{n} \left( \frac{x_i - \bar{x}}{\sigma_x} \right) \left( \frac{y_i - \bar{y}}{\sigma_y} \right) \]

\[ \rho = 1 - \frac{6 \sum_{i=1}^{n} [\text{rank}(x_i) - \text{rank}(y_i)]}{n(n^2 - 1)} \]
Distance Metrics

- Choose your distance measure carefully.

- In general, before you begin analysis:
  - Explore your data by:
    - Doing simple **sanity-checks**: “Is the mean, variance, and range of values as expected?”, “Are there too many missing values?”
    - **Looking** at your data [There is no substitute for this]: Plot small portions of the data in different ways and visualize trends, shapes, relationships, etc.
The End