

基因微阵列数据中的聚类技术研究

马 煜, 陈 莉, 方鹤鹤

(西北大学 计算机科学系, 陕西 西安 710069)

摘 要:微阵列技术是后基因组时代功能基因组研究的主要工具。由于采用了高效的并行杂交技术, 每次实验可以得到大量丰富的数据, 因此其结果分析成为一项很有挑战性而且具有重要意义的工作。聚类分析是微阵列数据分析中使用最为广泛的一类方法。微阵列实验得到的大量数据通过聚类分析, 可以得到很多有用的信息, 其成功应用已广泛涉及到基因功能研究和生物医学研究中的各个领域。文中介绍了基因微阵列数据的聚类分析方法及其重要应用。

关键词:微阵列; 基因表达谱; 聚类分析

中图分类号: TP391

文献标识码: A

文章编号: 1005-3751(2006)02-0117-03

Clustering Analysis of Microarray Gene Expression Data

MA Yu, CHEN Li, FANG He-he

(Department of Computer Science, Northwest University, Xi'an 710069, China)

Abstract: Microarray technology is the chief tool for functional genome research. As adopting the high efficient and parallel DNA hybridization technology, can achieve abundant data from each experiment, so the data analysis of microarrays data becomes a more challenging and meaningful task. Clustering is the useful and most widely used method of microarray data analysis. Abundant useful information can be obtained through the microarray clustering. This paper presents a system of clustering analysis for DNA microarray data.

Key words: microarray; gene expression profiles; clustering analysis

0 引 言

近年来生物信息学持续迅速发展。随着果蝇基因组测试和人类基因组工程的第一个草图的完成, 基因组测序研究蓬勃发展。高密度 cDNA 微阵列与寡核苷酸微阵列技术已经可以同时测定多个基因(甚至整个基因组)在某一条条件下的转录水平。大规模基因表达谱数据为研究基因功能、基因之间的调控机制及医药研究提供了新的思路。怎样从如此浩如烟海的数据集中找到研究者所感兴趣的信息, 不仅是生物信息学研究者的一个重要课题, 也对计算机研究者提出了新的课题。近几年, 高性能生物序列聚类算法^[1-4]有了很大提高, 这些算法都能自动把数量非常庞大的基因数据库进行聚类, 它是微阵列研究中的一个重要的工具。文中系统评述了微阵列数据分组基因问题中所用到的各种聚类方法。由于聚类问题的多样性和“开放性”, 评价一个聚类问题的优劣不仅仅是要看其数学上表现, 而且要在具体的生物学的环境下进行评价, 聚类问题和聚类算法中, 特别是在基因表达条件下, 有很多

重要问题需要考虑。因此在这里并不对所有的聚类算法给定一个统一的评价标准。

1 微阵列数据简介

在过去短短几年里, 基于微阵列的新技术大量涌现并且迅速发展。这一类技术包括 DNA 杂交阵列(hybridization array)[基因表达阵列以及用于测序和多态性研究的寡核苷酸(oligonucleotide)阵列]、蛋白质阵列、组织阵列等。由于这些高能量方法使大量分子与一个大型文库之间的组合成为可能。

DNA 基因表达微阵列便利生物学家能够在基因组层次上研究任何种类细胞的任何时间、任何给定条件下的基因表达模式^[5]。利用这些微阵列, 人们正产生出大量的数据, 它们可以帮助人们深入地认识诸多生物过程的本质, 如基因功能、发育、癌症、衰老和药理等。即使是对现有信息的部分理解也能够提供很有价值的线索。例如, 新基因的共表达(co-expression)就有助于推断许多缺乏相关信息的基因的功能。然而, 基因微阵列数据分析方法的发展现在才刚刚起步。

2 基因聚类分析

聚类就是将物理或抽象对象分组成为多个类或簇(cluster), 在同一个簇中的对象之间具有较高的相似度。

收稿日期: 2005-05-27

基金项目: 陕西省自然科学基金(98X11); 陕西省教育厅重点科研项目(00JK015)

作者简介: 马 煜(1975—), 女, 陕西子州人, 硕士研究生, 研究方向为数据挖掘、生物信息学; 陈 莉, 副教授, 硕士研究生导师, 研究方向为人工智能、数据挖掘。

而不同簇中的对象差别较大。基因聚类就是将基因在属性的基础上分组,这些属性往往是基因在一些特定的情况下的表达水平或其子集。

2.1 分级聚类

聚类可以通过分级的分支过程得到。因此有一些方法,可以根据两两相似度从数据中自动建立一棵树。对于基因表达的情况,这就是文献[6]所用的方法。这种方法的输出是一棵树而非一组类别。特别地,如何从树中定义类别往往不明显。因为类别是通过在树的某些点剪枝得到,而这一过程或多或少带有主观性。

分级聚类算法的步骤:

Step1:建立 Gene-experiment 矩阵。

建立 $m \times n$ 矩阵,其中每一列是不同的组织,或者在不同的条件下的样本,每一行是基因的编号,每个基因的表达量用标准化后的 $\log_2 R/G$ 表示。

Step2:计算所有基因之间的相关系数(correlation coefficient)。

基因的相似分值(similarity score)可以由 Pearsons correlation 公式计算:

$$s(X, Y) = \frac{1}{N} \sum_{i=1}^N \left(\frac{X_i - X_{\text{offset}}}{\phi_X} \right) \left(\frac{Y_i - Y_{\text{offset}}}{\phi_Y} \right)$$

G_{offset} 一般取值为标准化后的中位数 $\phi_G =$

$\sqrt{\frac{\sum_{i=1}^N (G_i - G_{\text{offset}})^2}{N}}$, 或平均值。它等于 0, 即 $\log_2 R/G = 0$, 表示表达无差异。

Step3:建立 Gene-Gene 的距离矩阵(见表 1)。

表 1 Gene-Gene 距离矩阵

	G1	G2	G3	G4	G5
G1	0				
G2	2	0			
G3	6	5	0		
G4	10	9	4	0	
G5	9	8	5	3	0

Step4:建立系统发育树(dendrogram)。

根据 Gene-Gene 矩阵的分值,首先找到矩阵中两个最相似的元素(具有最大相关性,距离最近的),生成一个结点将它们结合在一起,例如对表 1,有 5 个基因的矩阵,得到的系统发育树如图 1 所示。

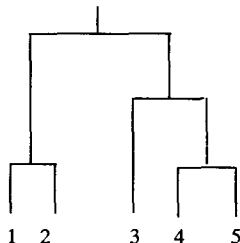


图 1 系统发育树图

Step5:建立表达图谱,通过求两个元素表达谱(expression profile)(或向量)的平均(缺失的数据可以忽略,求

平均时可以按照向量中元素的个数进行加权)生成新节点的表达谱(或向量)。

Step6:用新节点取代两个结合的元素,按照新计算的表达谱(或向量)计算新的相关矩阵。这个相关矩阵比原来的矩阵要小一些。

Step7:从 N 个点开始,这个过程将最多重复 $N-1$ 次,直至只剩下 1 个单节点。

正如已经指出的,在建立了这样一个标准树图以后,如何显示结果以及如何选取类别仍然是个问题。这一步往往是由人工来完成的,既浪费时间又带着强烈的主观性。文献[6]用了一种启发式近似算法,它用平均表达水平、染色体位置和最大诱导时间(time of maximal induction)对基因加权。通过对一组基因表达数据聚类得到的主要类别确实显示出了生物学上的相关性。

2.2 K 均值聚类法

在所有的聚类算法中,K 均值聚类法^[7]可能具有最清晰的概率表述。K 均值聚类法与分层聚类有本质的区别,首先类别数被固定为一个值 K ,然后将全部的基因按照相似性的距离,归入这几类中。一开始就给各类选择代表点或类中心,这样 K 个代表点或类中心的选择或多或少带有随意性。它们也被称为质心(centroid)或原型(proto-type)。

K 均值聚类算法:

Step1:将 gene-experiments 矩阵转化为 gene-gene distance 矩阵,但与分层聚类计算相关系数的方法不同,用欧氏距离(Euclidean distance)公式计算:

$$d(X, Y) = \sqrt{\sum_{i=1}^m (X_i - Y_i)^2}$$

式中, X, Y 为两个基因。

然后将所有的基因随机分配到 K 类中。

Step2:计算出每个类中的基因的均值,把每个点分到离它最近的代表点所代表的类内;分类诸新的代表点,比如取每一个新类的平均或重心。

Step3:重复上面两个步骤,直到系统收敛或涨落很小。

这里要注意:K 均值聚类法要求选择类别数,要求可以计算点与点之间的距离或相似度,并且对于每一类在给定其成员时可以计算代表点。

2.3 基于图论的聚类算法

该算法基于图论中最小生成树方法的聚类分析。其作法是利用最小生成树算法(MST)将多维基因数据建树,这个表示的关键性是表达数据的每个簇对应 MST 的一个子树,这样将一个多维数据的聚类问题严格地转换为一个树的划分问题,然后删去最大边产生聚类^[8]。

$D = \{d_i\}$ 是一个表达数据的集合,其中每个 $d_i = \{e_i^1, e_i^2, \dots, e_i^t\}$ 表示从时间 1 到时间 t 基因 i 的表达水平。在这里定义一个带权图 $G(D) = (V, E)$ 。向量集 $V = \{d_i \mid d_i \in D\}$, 边集 $E \in \{(d_i, d_j) \mid d_i, d_j \in D \text{ 且 } i \neq j\}$ 。因

此 $G(D)$ 是一个完全图, 每个边 $(u, v) \in E$ 都有一个权重用来代表两个结点之间的距离或(相异性)。 $\rho(u, v)$ 在 u 和 v 之间的距离可以被定义为欧氏距离、相关性系数或是其它一些距离测试方法。

连接带权图 $G(D)$ 造成树 T 是 $G(D)$ 一个连通子图。树的属性为: (1) T 包含 $G(D)$ 中每个向量; (2) T 不包含任何的环。MST 是距离总和最小的生成树。

最小生成树算法:

Step1: 用最小生成树(MST)算法数据建树。

Step2: 找到最小生成树中两个叶结点之间距离最长的剪枝, 生成一个新簇。

Step3: 若没有执行到 $N-1$ 步, 则执行 Step2; 否则, 输出聚类结果。

基于图论理论的聚类算法的优点:

(1) 树形结构有利于高效地实现严格的聚类算法;

(2) 基于图论的聚类不依赖于簇的几何形状, 它可以克服其它划分聚类算法的问题即严重依赖簇的几何形状, 一般在面对包含无重叠的向量集中的簇时并不能很好地发挥性能。

缺点: 因为过渡区域的点, 这个算法所面对的是一套强连通的基因, 所以基于图论的聚类算法的计算复杂性高。

2.4 自组织映射

自组织映射(self-organized map, SOMs)分析^[2]是人工神经网络应用于聚类分析中的例子。它采用的是结构简单的单层竞争性神经网络。模式在输入端引入并与输出结点关联, 其间的权重通过学习反复变更, 直到达到终止标准。结果是相似的模式被分入同组, 并为同一个单位(神经元)所代表。SOMs 法有着和 K-means 相同的不足, 在未知分块数目时其初始权重选择很可能不合适而导致产生次优解。另外收敛受到多种参数影响, 结果可能不稳定。Mavroudi 等^[1]提出了改进的 SOMs 算法, 称为 sNet-SOM(supervised network self-organized map), 它通过一个动态扩展过程可以自适应地确定分组数目, 同时有效地降低了计算代价。

2.5 模拟退火算法聚类

模拟退火算法^[9]是一种受统计力学启发的通用的优化算法。在模拟退火聚类算法中假设 N 是基因表达水平的数量, 每一个表达水平包含 M 个时间点上的数据。起初每个基因表达水平用一个 M 维向量 $\{e'_1, e'_2, \dots, e'_M\}$ 表示, 每个维的值 e'_{im} 被标准化为 $[0, 1]$ 之间的值。两个向量 i 和 j 之间的距离用欧氏距离表示:

$$d_{ij} = \left[\sum_{m=1}^M (e'_{im} - e'_{jm})^2 \right]^{1/2}$$

对于给定簇的数量 K 时, 使用最小化簇中所有点之间距离 d_{ij} 之和来得到簇的最优分布。簇内所有点之间距离之和用下式表示:

$$E(K) = \frac{1}{K} \sum_{k=1}^K \left[\sum_{i \in \alpha} \sum_{j \in \alpha} d_{ij} \right]$$

这里计算最小化簇中所有点之间距离之和时使用的是模拟退火算法。

模拟退火算法步骤:

Step1: 将集体中的所有向量任意分到 K 个簇内。

Step2: 选择一个簇中的任意一个向量, 将它分配至另一簇中。计算一个新的 E^{new} 和原来的值 E^{old} 进行比较。如果 E^{old} 大于 E^{new} 则向量就被无条件地分配到新的簇中, E^{new} 做为下一次迭代的开始。

Step3: 计算新的分配被接受的可能性 $\exp[-(E^{\text{new}} - E^{\text{old}})/T]$ 。

Step4: 如果 T 没有接近 0, 则执行 Step2, else 输出所得到的簇。

这里可能性表达式中如果 E 的值可以看作系统的能量时, T 可以被理解为“温度”。这个算法保证了经过有限步的迭代后系统在给定的温度下服从波耳兹曼—吉布斯分布。因此, 如果温度 T 接近 0, 那么系统中 E 函数也就接近了全局的最小值。

3 展望

基因微阵列数据的聚类分析方法已经在生命科学的各个领域内得到了许多成功的应用, 如基因表达谱与生物个体行为关系的研究、肿瘤分类等等。由于其应用的广泛性, 出现了大量可用的聚类分析软件, 更加方便了其推广和应用。

聚类算法在目前生物信息的分析中应用极其广泛, 但大多数是基于统计理论, 而生物领域的知识很少被涉及。而一个聚类结果的质量的好坏不仅仅要看其在数学形态上的表现, 生物领域的知识是要起非常大的作用的。生物信息的聚类分析进行应该充分考虑将基因的生物学意义和聚类算法很好地结合起来。

参考文献:

- [1] Mavroudi S, Papadimitriou S, Bezerianos A. Gene expression data analysis with a dynamically extended self-organized map that exploits class information[J]. Bioinformatics, 2002, 18: 1446-1453.
- [2] Goulb T R, Slonim D K, Tamayo P, et al. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring[J]. Science, 1999, 286(18): 1194-1206.
- [3] Cooper G F. Computational complexity of probability inference using Bayesian Belief Networks[J]. Artificial Intelligence, 1993, 15: 246-255.
- [4] Lukashin A V, Fuchs R. Analysis of temporal gene expression profiles: clustering by simulated annealing and determining the optimal number of clusters[J]. Bioinformatics, 2001, 17: 405-414.
- [5] Holstege F C P, Jennings E G, Wyrick J J, et al. Dissecting the regulatory circuitry of a eukaryotic genome[J]. Cell, 1998, 95:

(下转第 122 页)

```

synchronized void LeftEnter() throws InterruptedException {
a) -- while (nRight > 0) wait ();
    nLeft ++ ; //获取资源, nLeft 加 1
}
synchronized void LeftExit() {
    nLeft -- ; //释放资源线处, nLeft 减 1
b) -- if (nLeft == 0)
    notifyAll();
}

```

a) 判断另一类线程是否占用资源: 如果 nRight 等于 0 说明没有占用该资源, 如果 nRight 大于 0, 说明另一类线程正在占用该资源, 这样就挂起本线程。

b) 释放资源并唤醒线程: 如果 nLeft(nRight) 等于零, 就说明该类线程已经全释放了资源, 重新唤醒所有线程继续竞争资源, 这样就保证了多线程的延续性。

(3) synchronized 的作用^[5]。

Java 提供了专门机制以解决这种冲突, 即 synchronized 关键字, 它有效避免了同一个数据对象被多个线程同时访问。synchronized 方法使每个类实例对应一把锁, 每个 synchronized 方法都必须获得调用该方法的类实例的锁方能执行, 否则所属线程阻塞本程序就是使用第一种方法来实现对公共资源的互斥访问。

3.3.3 关于 FairTunnel.java 的控制分析

FairTunnel 类是对 SafeTunnel 类的改进, 在实现多线程并发执行中的安全性的同时, 也保证了多个线程对资源访问的公平性。

```

private int nLeft = 0, nRight = 0, waitRight = 0, waitLeft = 0;
private boolean Rightturn = true;
synchronized void LeftEnter() throws InterruptedException {
    ++ waitLeft; //等待的 LeftTrain 线程的个数加 1
a) -- while (nRight > 0 || (waitRight > 0 && Rightturn)) wait();
    -- waitLeft; //等待的 LeftTrain 线程个数减 1
    ++ nLeft; //占用该资源的线程个数随之加 1
}
synchronized void LeftExit() {
    -- nLeft; //释放资源, nLeft(nRight) 减 1
b) -- Rightturn = true;
c) -- if (nLeft == 0)
    notifyAll();
}

```

(上接第 119 页)

717-728.

- [6] Eisen M B, Spellman P T, Brown P O, et al. Cluster analysis and display of genome-wide expression patterns[A]. Proc. Natl. Acad. Sci USA, 95 [C]. USA: [s. n.], 1998. 14863-14868.
- [7] Duda R O, Hart P E. Pattern Classification and Scene Analysis

在这个类中引用了以下几个公共变量:

* nLeft, nRight: 代表占用资源的同类线程数;

* waitLeft, waitRight: 代表申请资源的同类线程数;

* Rightturn: 是一个时间片开关, 它实现了两边的两类线程公平地访问资源, 两类线程申请资源获得允许使用权力的时间片就是一列 Train 通过单行隧道的的时间。

a) 挂起线程: 当 nRight 大于 0 时说明 RightTrain 线程正在占用资源, 当 waitRight 大于 0 时, 说明有新的 RightTrain 线程将准备使用资源, Rightturn 等于 true 时, 说明 RightTrain 线程有权继续使用资源。在上述情况下, 挂起 LeftTrain 线程。

b) 当第一个使用资源的 LeftTrain 线程到达资源释放处时, 使得 Rightturn 改为 true, 使得 LeftTrain 线程失去申请资源获得允许使用权力。

c) 释放资源并唤醒线程: 如果 nLeft(nRight) 等于零, 就说明该类线程已经全释放了资源, 重新唤醒所有线程继续竞争资源, 这样就保证了多线程的延续性。

4 结束语

文中简单模拟火车行驶单行隧道, 通过这个实验模拟了多线程并发系统中的安全性与公平性问题, 分别演示了在不同情况下产生何种后果: 只考虑安全性, 将会有一类 Train(LeftTrain 或者 RightTrain) 长时间等待, 申请不到资源; 只考虑公平性, 会出现两边火车相撞的情况。由此, 可充分认识打到并发系统中的安全性与公平性二者的辩证关系, 通过两者的结合, 设计解决实际问题的最优方案。

参考文献:

- [1] Magee J, Kramer J. Concurrency: State Models & Java Programs[M]. [s. l.]: Wiley publishing company, 1999.
- [2] 任爱华, 王雷. 操作系统实用教程(第 2 版)[M]. 北京: 清华大学出版社, 2004.
- [3] Stallings W. 操作系统精髓与设计原理(第 3 版)[M]. 北京: 清华大学出版社, 1998.
- [4] 陶冶. Java 多线程学习笔记[EB/OL]. http://www.cs-dn.com.cn/program/index.htm, 2003-06-18.
- [5] Deitel H M, Deitel P J. Java 程序设计教程[M]. 施平安, 施惠琼译. 北京: 清华大学出版社, 2004.

[M]. [s. l.]: John Wiley and Sons, 1973.

- [8] Xu Ying, Olman V, Xu Dong. Clustering gene expression data using a graph-theoretic approach: an application of minimum spanning trees[J]. Bioinformatics, 2002, 18: 536-545.
- [9] Kirkpatrick S, Gelatt C D, Vecchi M P. Optimization by simulated annealing[J]. Science, 1983, 220: 671-680.