 

**DNA序列分类**

|  |  |
| --- | --- |
| 设计题目 | DNA序列分类 |
| 学号 | 201211442125 201311621219 |
| 学生姓名 | 姚帆 王君怡 |
| 所在专业 | 计算机科学与技术 |
| 所在班级 | 1132班 |
| 指导教师 | 邹阿金 |
| 成绩 |  |
| 教师签字 |  |

评语:

课程论文时间：2016年 10 月 23日

DNA 序列分类

分工：1，姚帆 （建模及编程），2（寻找参考文献及编程）3（建模及撰写论文）

摘要：

2000年6月，人类基因组计划中DNA全序列草图完成，预计2001年可以完成精确的全序列图，此后人类将拥有一本记录着自身生老病死及遗传进化的全部信息的“天书”。这本大自然写成的“天书”是由4个字符A，T，C，G按一定顺序排成的长约30亿的序列，其中没有“断句”也没有标点符号，除了这4个字符表示4种碱基以外，人们对它包含的“内容”知之甚少，难以读懂。破译这部世界上最巨量信息的“天书”是二十一世纪最重要的任务之一。充分发掘序列的结构对理解DNA全序列是十分有意义的。目前在这项研究中最普通的思想是省略序列的某些细节，突出特征，然后将其表示成适当的数学对象。这种被称为粗粒化和模型化的方法往往有助于研究规律性和结构。

1. 问题简述

有20个已知类别的人工制造的序列， 其中序列标号1-10为A类， 11-20为B类。从中提取特征， 构造分类方法， 并用这些已知类别的序列， 衡量你的方法是否足够好， 然后用你认为满意的方法， 对另外20个未标明类别的人工序列（标号21-40）进行分类，把结果用序号（按从小到大的顺序）表明它们的类别（无法分类的不写入）

详细描述方法，并给出计算程序。

1. 假设

经过观察， A类DNA序列中G碱基含量比较多，T碱基含量少，而B类恰好G碱基少，T碱基较多，所以假设分类方法与TG碱基的数量有相关关系。经过统计得出结论：确实有一定的相关关系，对于碱基这类意义不明的特征，我们决定采用模糊聚类的做法，将20个DNA序列片段分成二类，找到较为准确的λ值，提高DNA分类准确度，建立DNA分类的数学模型。

1. 建模
2. 提取原始数据，统计各类碱基在自身序列中所在频率。
3. 将碱基频率作标准化处理。
4. 标准化后进一步进行极差变换，得到关于碱基频率的模糊矩阵。
5. 利用相关系数法对碱基频率矩阵进行求模糊相似矩阵
6. 利用自乘法对模糊相似矩阵对传递闭包
7. 构造λ截矩阵，对关系进行划分，测量出最佳λ值。
8. 模型求解

先提取各碱基的频率值，构造初始数据矩阵MO ;

a c t g

[1,] 0.2909091 0.17272727 0.13636364 0.40000000

[2,] 0.2727273 0.15454545 0.15454545 0.41818182

[3,] 0.2727273 0.21818182 0.06363636 0.44545455

[4,] 0.4181818 0.10909091 0.29090909 0.18181818

[5,] 0.2363636 0.22727273 0.10909091 0.42727273

[6,] 0.3454545 0.12727273 0.12727273 0.40000000

[7,] 0.3454545 0.10000000 0.19090909 0.36363636

[8,] 0.2727273 0.16363636 0.19090909 0.37272727

[9,] 0.2000000 0.20909091 0.15454545 0.43636364

[10,] 0.1818182 0.27272727 0.13636364 0.40909091

[11,] 0.3545455 0.04545455 0.50000000 0.10000000

[12,] 0.3272727 0.02727273 0.50000000 0.14545455

[13,] 0.2545455 0.10000000 0.51818182 0.12727273

[14,] 0.3000000 0.08181818 0.50000000 0.11818182

[15,] 0.2909091 0.00000000 0.64545455 0.06363636

[16,] 0.3636364 0.08181818 0.46363636 0.09090909

[17,] 0.3545455 0.24545455 0.26363636 0.13636364

[18,] 0.2909091 0.11818182 0.50000000 0.09090909

[19,] 0.2181818 0.14545455 0.56363636 0.07272727

[20,] 0.2000000 0.17272727 0.56363636 0.06363636

利用以上矩阵进行平移标准差转换求出标准化矩阵Mstd；

a c t g

[1,] 0.0215733 0.4631190 -0.9880163 1.0024799

[2,] -0.2660707 0.2161222 -0.8945869 1.1187095

[3,] -0.2660707 1.0806111 -1.3617341 1.2930538

[4,] 2.0350817 -0.4013698 -0.1938661 -0.3922748

[5,] -0.8413588 1.2041095 -1.1281605 1.1768243

[6,] 0.8845055 -0.1543730 -1.0347311 1.0024799

[7,] 0.8845055 -0.5248682 -0.7077280 0.7700208

[8,] -0.2660707 0.3396206 -0.7077280 0.8281356

[9,] -1.4166469 0.9571126 -0.8945869 1.2349390

[10,] -1.7042910 1.8216015 -0.9880163 1.0605947

[11,] 1.0283275 -1.2658587 0.8805725 -0.9153078

[12,] 0.5968614 -1.5128555 0.8805725 -0.6247339

[13,] -0.5537148 -0.5248682 0.9740019 -0.7409634

[14,] 0.1653953 -0.7718650 0.8805725 -0.7990782

[15,] 0.0215733 -1.8833507 1.6280080 -1.1477669

[16,] 1.1721495 -0.7718650 0.6937136 -0.9734225

[17,] 1.0283275 1.4511063 -0.3340102 -0.6828486

[18,] 0.0215733 -0.2778714 0.8805725 -0.9734225

[19,] -1.1290029 0.0926238 1.2075755 -1.0896521

[20,] -1.4166469 0.4631190 1.2075755 -1.1477669

为了把数据维持在0到1之间，对矩阵进行极差标准变换，得到M；

a c t g

[1,] 0.46153846 0.6333333 0.125000 0.88095238

[2,] 0.38461538 0.5666667 0.156250 0.92857143

[3,] 0.38461538 0.8000000 0.000000 1.00000000

[4,] 1.00000000 0.4000000 0.390625 0.30952381

[5,] 0.23076923 0.8333333 0.078125 0.95238095

[6,] 0.69230769 0.4666667 0.109375 0.88095238

[7,] 0.69230769 0.3666667 0.218750 0.78571429

[8,] 0.38461538 0.6000000 0.218750 0.80952381

[9,] 0.07692308 0.7666667 0.156250 0.97619048

[10,] 0.00000000 1.0000000 0.125000 0.90476190

[11,] 0.73076923 0.1666667 0.750000 0.09523810

[12,] 0.61538462 0.1000000 0.750000 0.21428571

[13,] 0.30769231 0.3666667 0.781250 0.16666667

[14,] 0.50000000 0.3000000 0.750000 0.14285714

[15,] 0.46153846 0.0000000 1.000000 0.00000000

[16,] 0.76923077 0.3000000 0.687500 0.07142857

[17,] 0.73076923 0.9000000 0.343750 0.19047619

[18,] 0.46153846 0.4333333 0.750000 0.07142857

[19,] 0.15384615 0.5333333 0.859375 0.02380952

[20,] 0.07692308 0.6333333 0.859375 0.00000000

这时数据的预处理就已经完毕。

相关系数法用于测量各向量间的相关性，相关系数绝对值愈大，向量的线性关系愈密切，为0则线性无关。利用相关系数法对M进行处理。得到模糊相似矩阵R20x20

（由于数据比较大，这里就不放图了。）

接着对模糊相似矩阵进行自乘求传递闭包。

当求得R.R ==R.R.R = R 时，R即为需要的模糊等价矩阵。

（数据依然比较大。）

从 [0,1]之间取λ值，确定相应的截矩阵 R(λ)。

观测结果集是否符合预期，即可确定较准确的λ值，至此模糊聚类分析就结束了。

1. 结果分析

利用上述模型对前20条DNA序列进行分类得出结果：

A类： {1，2，3，5，6，7，8，9，10}

B类： {11，12，13，14，15，16，18，19，20}

无法识别： {4，17}

发现该模型分类结果准确率为100%，识别率为90%，没有误判的情况出现。

对剩下待分类的DNA进行分类得出结果：

A类： {21，22，23，25，26，27，28，29，30}

B类： {31， 32， 33， 34， 35， 36， 38， 39， 40}

无法识别： {24， 37}

1. 模型改进

无论怎么调整λ值也无法使结果更优化，因此考虑更多元组的频率矩阵，探测三元组频率值，获得频率矩阵如下：

|  |
| --- |
| acg cag tag atg atc acc cac  [1,] 0.063636364 0.000000000 0.000000000 0.009090909 0.000000000 0.009090909 0.027272727  [2,] 0.027272727 0.009090909 0.000000000 0.009090909 0.000000000 0.009090909 0.000000000  [3,] 0.090909091 0.000000000 0.000000000 0.000000000 0.009090909 0.000000000 0.027272727  [4,] 0.009090909 0.018181818 0.018181818 0.027272727 0.009090909 0.009090909 0.000000000  [5,] 0.036363636 0.009090909 0.000000000 0.000000000 0.000000000 0.018181818 0.018181818  [6,] 0.018181818 0.018181818 0.009090909 0.000000000 0.000000000 0.000000000 0.009090909  [7,] 0.000000000 0.009090909 0.009090909 0.036363636 0.018181818 0.000000000 0.000000000  [8,] 0.009090909 0.000000000 0.036363636 0.009090909 0.009090909 0.009090909 0.000000000  [9,] 0.000000000 0.009090909 0.000000000 0.018181818 0.018181818 0.000000000 0.000000000  [10,] 0.027272727 0.018181818 0.000000000 0.000000000 0.009090909 0.000000000 0.018181818  [11,] 0.018181818 0.000000000 0.018181818 0.018181818 0.009090909 0.000000000 0.000000000  [12,] 0.000000000 0.000000000 0.027272727 0.009090909 0.018181818 0.000000000 0.000000000  [13,] 0.009090909 0.018181818 0.009090909 0.018181818 0.000000000 0.018181818 0.000000000  [14,] 0.000000000 0.009090909 0.036363636 0.009090909 0.018181818 0.000000000 0.000000000  [15,] 0.000000000 0.000000000 0.009090909 0.009090909 0.000000000 0.000000000 0.000000000  [16,] 0.000000000 0.000000000 0.009090909 0.009090909 0.018181818 0.000000000 0.000000000  [17,] 0.018181818 0.009090909 0.000000000 0.036363636 0.036363636 0.027272727 0.027272727  [18,] 0.027272727 0.000000000 0.009090909 0.000000000 0.000000000 0.000000000 0.000000000  [19,] 0.000000000 0.000000000 0.009090909 0.000000000 0.036363636 0.009090909 0.000000000  [20,] 0.000000000 0.000000000 0.000000000 0.000000000 0.018181818 0.000000000 0.009090909 |
|  |
| |  | | --- | |  | |

并未获取到较有特征的值，因此大概猜测分类准确率的提升空间已经几乎没有了，

1. 比较结论

因为DNA序列分类还包括了空间及环境的因素， 仅靠碱基分类误差是难以避免的。

附录：R语言程序代码

# fetch data

tempDNA <- read.table('data.txt', header=TRUE);

DNA <- mutate(tempDNA, DNA = substr(DNA, 4, 1000));

rm(tempDNA);

# the list for modeling

originalDNA <- DNA$DNA[1:20];

# the list wait for clustering

futureDNA <- DNA$DNA[21:40];

library(stringr);

library(dplyr);

#take the frequency matrix

takefreqMatrix <- function(DNA){

length = nchar(DNA[1]);

a <- str\_count(originalDNA, pattern = "a")/length;

c <- str\_count(originalDNA, pattern = "c")/length;

t <- str\_count(originalDNA, pattern = "t")/length;

g <- str\_count(originalDNA, pattern = "g")/length;

return(cbind(a,c,t,g));

}

# STD transfrom

#afterSTD <- scale(takefreqMatrix(originalDNA));

#take the max-min standard matrix

takeMMS <- function (col) {

return ((col-min(col)) / (max(col)-min(col)))

}

#max-min transform

#fuzzyMatrix <- apply(afterSTD, takeMMS, MARGIN = 2);

#get fuzzy similar martix

#FSM <- cor(t(fuzzyMatrix));

takeFSMfinal <- function(m) {

for(i in 1:20) {

for(j in 1:20) {

if(m[i,j] < 0) {

m[i,j] = (m[i,j] + 1) /2;

}

}

}

return (m)

}

# now I have got a fuzzy similar matrix

#FSMfinal <- takeFSMfinal(FSM)

# to generate a fuzzy equivalent matrix

multiSelf <- function(m, n=m) {

new <- matrix(0, 20, 20)

for(i in 1:20) {

for(j in 1:20){

new[i,j] = max(pmin(m[i, ], n[,j]));

}

}

return(new)

}

takeFEM <- function (m) {

mm = multiSelf(m);

if(all(mm ==m)) {

return (mm)

} else {

takeFEM(mm)

}

}

takeLamdaMatrix <- function(m) {

for(i in 1:20) {

for(j in 1:20) {

if(m[i,j] >= 0.8) {

m[i,j] =1;

} else {

m[i,j] =0;

}

}

}

return(m)

}

takeData <- function(m){

s <- matrix(NA, 20, 1);

count = 0;

for(i in 1:20) {

for(j in 1:20) {

if ((m[i,j] == 1) && (i != j)) {

if((is.na(s[j, 1])) && (is.na(s[i, 1]))){

s[i, 1] = count;

count = count+1

} else if(is.na(s[i, 1])) {

s[i, 1] = s[j, 1];

}

}

}

}

return(s)

}

analysis <- function(m) {

afterSTD <- scale(takefreqMatrix(m));

fuzzyMatrix <- apply(afterSTD, takeMMS, MARGIN = 2);

FSM <- cor(t(fuzzyMatrix));

FSMfinal <- takeFSMfinal(FSM);

FEM <- takeFEM(FSMfinal);

lamdaM <- takeLamdaMatrix(FEM);

return(takeData(lamdaM))

}

takefreqTestMatrix <- function(DNA){

length = nchar(DNA[1]);

acg <- str\_count(originalDNA, pattern = "acg")/length;

cag <- str\_count(originalDNA, pattern = "cag")/length;

tag <- str\_count(originalDNA, pattern = "tag")/length;

atg <- str\_count(originalDNA, pattern = "atg")/length;

atc <- str\_count(originalDNA, pattern = "atc")/length;

acc <- str\_count(originalDNA, pattern = "acc")/length;

cac <- str\_count(originalDNA, pattern = "cac")/length;

#seqA <- (a-mean(a))/sd(a);

#seqC <- (c-mean(c))/sd(c);

#seqT <- (t-mean(t))/sd(t);

#seqG <- (g-mean(g))/sd(g);

#afterScale <- scale(cbind(a,c,t,g));

#return(cbind(seqA,seqC, seqT, seqG));

return(cbind(acg,cag,tag,atg, atc, acc, cac));

}