基于贝叶斯推断模型的全基因组关联性分析

摘要

随着人类所获取的海量生物 DNA 信息量的不断增长，针对各类复杂的遗传性疾病的遗传机理及性状与相关的关联基因以及关联的位点的研究成为遗传学领域的研究热点之一，如何中在大规模 DNA 序列信息中找到关联位点及对应基因带来了很大挑战。现有的一些计算方法,能在小规模研究中,有效地识别基因的相互作用。而题目所需处理数据的样本规模大、维度高，使用这些方法比较难处理。通过实验探索发现贝叶斯推断模型能较好的适应高纬度数据，进行全基因组关联性分析，针对题目提出的要求以及数据样本，进行了实验探索发现使用马尔科夫链蒙特卡洛（Markov Chain Monte Carlo, MCMC）方法的贝叶斯推断模型比较有利于找到致病位点与疾病的关联关系。

对于问题一，由于一个基因位点只存在三种不同的编码方式，并且呈离散状态，因此用 0,1,2 对三种编码方式进行重新编码，此外，使用-1 表示不符合要求的碱基对（II、ID）。这方便使用程序进行统计分析，比较有利于贝叶斯推断模型的实现。

针对问题二，考虑到样本基数大，每个样本的位点数量多，这意味着数据规模大，特征维度高，我们不能使用一种简单的方法来确定致病位点是一个还是多个，也就是说不能简单地确定是单个位点的影响还是多个位点共同作用。因此我们需要这样的一种方法，它既能高效地处理大规模数据，又能在特征维度高的情况下，找出单个结点的边缘效应和多个位点的关联效应。我们建立了使用马尔科夫链蒙特卡洛方法（Markov Chain Monte Carlo, MCMC）方法的贝叶斯推断模型，它既能够很好的检测出全基因组数据中具有互关联作用的位点同时也能检测出具有主边缘效应作用的位点，并且它能很好的处理大规模高维数据。在模型中，我们将位点分成三组：组 0 包含那些与所研究疾病无关联作用的位点集合；组 1 包含那些只对所研究疾病呈现边际效应作用的位点集合；而组 2 则包含那些相互关联起来以对疾病产生影响的位点集合。然后运用 MCMC 方法可以求得分组的后验概率。

针对问题三，通过贝叶斯推断模型，得到基因位点与疾病 A 的后验概率分布，取概率较大的一些位点组成集合 S。假设基因 $1 \sim N$，包含位点集合为$G_1 \sim G_N$，S 与$G_i(\mathrm{i} = 1 \ldots \mathrm{N})$的交集包含的位点后验概率之和越大，则该基因与疾病 A 的关联性更强。

针对问题四，对十种性状，建立贝叶斯推断模型，分别计算出在该性状下，样本数据中基因位点的后验概率分布$S_i(i = 1 \ldots 10)$。设在S_i中单边影响或关联影

响较大的位点子集s_i，则这些位点子集合$(s_1, s_2, \dots, s_{10})$的交集$s$是与 10 个性状有关联的位点集合。

关键字： 马尔科夫链蒙特卡洛方法 贝叶斯推断模型 全基因组关联性分析 位点

一、 问题重述

针对遗传性疾病的全基因组关联性问题的研究，本文依次提出如下问题：

1、为了更方便对基因数据进行处理分析，选取合适的编码方式将数据集中每个位点的碱基（A,T,C,G）编码方式转化为数值编码方式。

2、对于给定的样本数据以及可能致病的染色体片段上的位点上的编码信息，找到某种疾病与一个或者多个致病位点的关联关系。

3、由给出基因与位点关系的数据集，可将对应基因看作若干位点组成的集合，由问题二可知某种疾病可能与一个或多个位点存在关联关系，我们可以找到遗传疾病与基因的关联性，可以由基因中包含的所有位点或者部分位点表现出来，找到与疾病关联的一个或多个基因。

4、通常，实际研究过程中，相关科研人员会将有关联的性状或疾病看作一个整体，以便找到与它们相关的位点或者基因。根据所给出的样本与性状相关性的信息以及所有位点的编码信息，找到给出 10 个性状有关联的位点。

二、 模型假设

假设三组中基因型对应的病例数服从多项式分布，其中参数服从狄利克雷分布。

三、 问题分析

题目要求给出便于数据分析的碱基数值编码方式，然而从材料中可知各个位点虽然碱基的组合不同，但也都分别只有三种不同编码。由于每个位点相当于样本的一个离散性特征,所以我们可以使用离散值来表示位点的编码,比如 0,1,2,并且所有的位点的编码都可以使用一套离散值来表示。

材料给出 1000 个样本在某条有可能致病的染色体片段上的 9445 个位点的编码信息和样本患有遗传疾病 A 的信息，要求找出某种疾病最有可能的一个或几

个致病位点。这是一个相关性分析问题，需要分别找出单独对疾病关联的位点和交互作用与疾病关联的位点，即需要对染色体中的致病位点或其组合进行定位。我们发现样本的维度相当高，如果找出所有单独对疾病关联的位点和交互作用与疾病关联的位点，则会面临计算复杂度高，难于对全部可能致病位点或其组合进行计算等复杂问题。对于高维度的问题，通常可以先使用降维的方法减少维度，然后在使用合适方法找出位点与疾病之间的关系。其中一个特别有名的方法为多因素维数约减方法（Multifactor Dimensionality Reduction, MDR）。MDR 方法试图识别出那些通过互关联作用而非单个位点所具有的边缘效应作用对疾病等模型输出变量产生影响的变异位点集合。MDR 通过将高维的多位点模型约减为一维模型从而减少了需要进行二分类的预测变量的维数。MDR 面临一个重要的问题即它不可被用于分析、学习预测变量数目过多的数据集。MDR 方法只适合应用在最多含有几百个变异位点的基因组数据集上。然而，材料提供的是一个含有 9445 个位点的样本集，所以 MDR 方法不适用。递归分区方法（Recursive Partitioning Approaches, RPA）生成树的过程中通过将其上的每个路径与某些预测变量的

特定值组合相对应进而考虑这些预测变量之间所可能具有的互关联作用及这种互关联作用所可能具有的对输出变量的影响。递归分区方法的一个缺陷即在于它在第一阶段条件依赖于各个变量的边缘效应并在其后的各阶段条件依赖于先前已经被选中的变量的主效应。贝叶斯模型选取方法提供了一种新的可选择、学习出那些对所研究的如疾病等生理型特征起某种预测作用的遗传变异位点及它们间的相互关联作用的方法。在贝叶斯方法中对未知回归参数的先验分布及模型中的维度参数进行先验指定。基于已经观测到的数据，这些参数的一个后验概率分布可以通过一个马尔可夫链蒙特卡罗（Markov Chain Monte Carlo, MCMC）仿真方法计算得出。本题我们采用贝叶斯模型选取方法，并用 MCMC 计算后验概率。

四、 模型建立与解决

4.1 问题一的解决

设定出现频率最高的基因为 T,通常称为主要等位基因，则 C 可以表示出现频率较低的另外一个等位基因，可知，对于一个某个染色体的碱基对信息，某个特定的碱基对位置，可能会出现 TT,TC,CC 三种基因型状态，同理，对于染色体其它位置上面的碱基的组合可能不同，但是也只有三种状态，据此，对应的数值编码我们设定为 0,1 与 2 这三个离散数值分别表示这三种状态。

对于样本中所给出疾病特征，为离散型数值特征，设定 1 表示患病状态，0

表示健康状态，据此，对于给出表 1 中基因信息，比如位点 rs100015,我们分别用 0,1,2 表示三个不同的编码 TT、TC，CC，其他位点的编码使用相似的方式表示，可以将给定样本的基因组信息处理为如下表 2 所示。

表 1.　在对每个样本采集完全基因组信息后，一般有以下的数据信息

（以 6 个样本为例，其中 3 个病人，3 个健康者）：

样本编号	样本健康状况	染色体片段位点名称和位点等位基因信息			
		rs100015	rs56341	...	rs21132
1	1	TT	CA	...	GT
2	0	TT	CC	...	GG
3	1	TC	CC	...	GG
4	1	TC	CA	...	GG
5	0	CC	CC	...	GG
6	0	TT	CC	...	GG

注：位点名称通常以rs开头。

表2.　全基因组信息中数值编码状态信息

样本编号	样本健康状况	染色体片段位点名称和位点等位基因信息			
		rs100015	rs56341	...	rs21132
1	1	0	1	...	1
2	0	0	0	...	0
3	1	1	0	...	0
4	1	1	1	...	0
5	0	2	0	...	0
6	0	0	0	...	0

本文使用程序 1 对位点的碱基对进行编码。编码效果如图 1 所示。图中 ID 为位点标记，Chr 表示位点位于同一染色体。

```
ID         Chr       Pos       1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
rs3094315  chr1      4924223       0 1 0 0 1 1 1 0 0 0 1 2 0 0 1 0 0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 1 0 1 1 0 0 0 0 0 1 0 0 1 2 0 0 1 2 0
rs3131972  chr1      4924223       1 1 0 2 1 2 1 0 2 1 2 0 0 1 1 0 2 1 1 0 2 1 1 2 1 1 1 1 1 0 1 1 2 1 0 1 2 1 0 2 1 1 2 2 1 1 1 1 0 1 1 1
rs3131969  chr1      4924223       0 1 0 0 1 0 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0
rs1048488  chr1      4924223       1 1 0 1 2 1 1 1 1 1 2 1 0 1 1 2 0 0 1 1 2 0 1 2 1 1 2 2 1 1 1 2 0 2 1 1 1 2 1 1 0 1 2 1 1 1 1 1 1 1 0 0
rs12562034 chr1      4924223       0 2 2 1 1 0 1 1 2 2 1 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 0 2 2 1 2 2 1 2 1 2 2 2 1 2 1 2 0 0 2 1 1 2 2 2 2 2
rs12124819 chr1      4924223       0 1 1 0 0 0 1 0 0 1 1 0 1 2 0 0 0 0 0 1 0 0 0 1 0 0 1 0 1 0 0 1 0 1 0 2 0 0 1 0 0 0 1 0 0 1 0 1 0 1 0 0
rs4040617  chr1      4924223       0 0 2 1 1 0 1 1 0 0 1 0 0 2 0 0 2 0 1 0 0 1 0 1 1 1 1 0 0 0 0 0 0 1 0 1 1 0 0 0 1 2 1 0 1 0 0 1 0 1 0
rs2980300  chr1      4924223       0 0 0 1 0 0 0 1 2 1 0 0 0 0 1 0 0 1 1 0 1 0 1 1 0 0 1 2 1 1 0 2 0 0 1 0 0 1 0 1 0 0 2 0 1 0 1 0 0 0 0 0
rs4970383  chr1      4924223       0 0 0 1 2 2 1 1 2 0 0 1 2 1 1 1 1 0 0 1 1 1 0 1 0 0 0 1 1 1 0 2 0 2 1 0 1 1 1 0 1 0 2 1 2 1 1 0 1 0 0 1
rs4475691  chr1      4924223       0 0 0 1 0 2 1 0 2 0 1 1 0 1 1 1 0 1 1 0 0 1 1 2 0 1 1 1 0 2 1 1 0 1 1 1 0 0 0 1 0 1 2 0 1 1 1 1 1 1 0 1
rs1806509  chr1      4924223       0 0 0 1 1 0 0 1 1 1 1 1 1 0 1 1 0 2 1 0 0 0 1 1 0 1 2 0 0 2 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 1 2 0 0
rs7537756  chr1      4924223       0 0 0 0 0 1 0 1 1 0 0 1 1 0 0 0 0 1 1 1 2 0 0 1 0 1 1 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 2 0 1 0 1 0 0 1 0 1
```

图 1 位点的碱基对进行编码

4.2 问题二的模型建立和解决

我们的算法通过马尔科夫链蒙特卡洛方法进行贝叶斯上位性推断。算法首先

通过贝叶斯推断模型将位点空间分为三组：组 0 包含那些与所研究疾病无关联作用的位点集合；组 1 包含那些只对所研究疾病呈现边际效应作用的位点集合；而组 2 则包含那些相互关联起来以对疾病产生影响的位点集合。然后假设三组中基因型对应的病例数服从多项式分布，其中参数服从狄利克雷分布，根据贝叶斯推断可以求得三种分类的后验概率表达式，再运用马尔科夫链蒙特卡洛方法可以求得三种分类的后验概率。

定义：

假设有N_d患者和N_u个正常人被标记了L个位点。令病人基因为$D = (d_1, \ldots, d_{N_d})$，其中$d_i = (d_{i1}, \ldots., d_{iL})$表示病人i令常人基因为$U = (u_1, \ldots, u_{N_d})$, 其中$u_i = (u_{i1}, \ldots., u_{iL})$表示正常人i。这个L个位点被划分成三组：组 0 包含那些与所研究疾病无关联作用的位点集合；组 1 包含那些只对所研究疾病呈现边际效应作用的位点集合；而组 2 则包含那些相互关联起来以对疾病产生影响的位点集合。令$I = (I_1, \ldots, I_L)$分别用I_j =0, 1 和 2 来标识位点所属组别。我们的目标是推理出与疾病有关联的位点的集合，即集合$\{j: I_j > 0\}$。令l_1, l_2, l_3分别表示每组中位点的数量（$l_1 + l_2 + l_3 = L$），并且令D_1, D_2, D_3分别表示在组 0，1, 2 中的病人基因.

贝叶斯位点划分模型：

当与正常人基因相比时，在相关联位点上的患病基因应该表现出不同的分布。具体来说，我们描述似然模型假设位点独立在对照人群。

令$\Theta_1 = \{(\theta_{j1}, \theta_{j2}, \theta_{j3}): I_j = 1\}$是组 1 中每个等位基因位点的基因型频率，我们把$D_1$的似然写为：

$$P(D_1|\Theta_1) = \prod_{j:I=1} \prod_{k=1}^{3} \theta_{jk}^{n_{jk}},$$

其中$\{n_{j1}, n_{j2}, n_{j3}\}$是组 1 中每个位点j的基因计数。假设$\{\theta_{j1}, \theta_{j2}, \theta_{j3}\}$的先验概率分布为Dirichlet(α)，其中$\alpha = (\alpha_1, \alpha_2, \alpha_3)$，我们整合出$\Theta_1$，获得边缘概率：

$$P(D_1|I) = \prod_{j:I=1} \left(\left(\prod_{k=1}^{3} \frac{\Gamma(n_{jk}+\alpha_k)}{\Gamma(\alpha_k)} \right) \frac{\Gamma(|\alpha|)}{\Gamma(N_d+|\alpha|)} \right) \quad (1)$$

这里符号$|\alpha|$表示在α中所有元素之和。

组 2 中位点通过相互作用影响疾病风险。因此，在这个组的l_2个位点的每个基因型组合都是一个候选关联。共存在3^{l_2}个频率为$\Theta_2 = (\rho_1, \ldots, \rho_{3^{l_2}})$可能的基因型组合。令$n_k$是基因组合k在$D_2$中的数量。假设$\Theta_2$的先验概率分布是Dirichlet(β)，其中$\beta = (\beta_1, \ldots, \beta_{l_2})$，我们整合出$\Theta_2$，因此

$$P(D_2|I) = \left(\prod_{k=1}^{3^{l_2}} \frac{\Gamma(n_k+\beta_k)}{\Gamma(\beta_k)}\right) \frac{\Gamma(|\beta|)}{\Gamma(N_d+|\beta|)} \quad (2)$$

剩下的数据D_0包含的位点服从和正常人相同的分布。令$\Theta = \{\theta_1, \ldots, \theta_L\}$表示L个位点在正常人的基因频率，令$n_{ik}$和$m_{jk}$是个体中分别在D和U中位点j基因型k的数量。假设$\theta_j (j = 1, .., L)$的Dirichlet先验概率$\gamma = (\gamma_1, \gamma_2, \gamma_3)$，我们整合出$\Theta$，获得

$$P(D_0, U|I) = \prod_{j=1}^{L} \left(\left(\prod_{k=1}^{3} \frac{\Gamma(n_{jk}+m_{jk}+\gamma_k)}{\Gamma(\gamma_k)}\right) \frac{\Gamma(|\gamma|)}{\Gamma\left(\sum_{k=1}^{3} (n_{jk}+m_{jk})+|\gamma|\right)} \right) \quad (3)$$

结合公式（1)、(2）和（3)，我们获得后验分布为：

$$P(I|D, U) \propto P(D_1|I)P(D_2|I)P(D_0, U|I)P(I) \quad (4)$$

注意，I决定了D_i的结构，我们令$P(I) \propto p_1^{l_1} p_2^{l_1} (1 - p_1 - p_2)^{L-l_1-l_2}$，这个可以修改来反映出关于每个与疾病关联的位点的先验知识。默认的，我们设置$p_1 = p_2 = 0.01$。

MCMC 采样：

我们的目标是从分布（4）中找到划分器I。我们按照先验概率P(I)初始化I，使用 Metropolis-Hastings (MH)算法来更新I。使用两类方法：（i）随机改变一个位点群成员（ii）随机交换组 0，1,2 中的两个位点。按照 MH 率接受移动，这个比率是一个γ函数。输出是位点和相互作用的后验分布。为了提高采样效率，我们首先给组 2 中位点数量设置一个较低的边界值，然后不断减少这个边界值直到 0。这强迫算法去搜索高阶关联空间。

使用程序 2，求得各位点分别属于三组的后验概率，但是求解出的属于组 2 的位点集合为空，即不存在多个位点的关联效应。图 2 为各位点属于组 1 的后验

概率分布。图中后验概率最大（0.371625）的位点 2250358（即 rs2250358）。

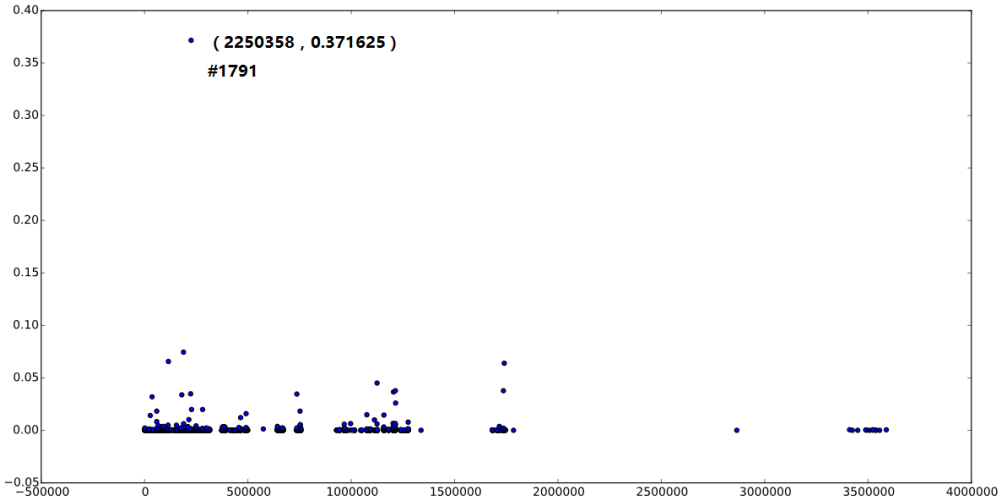

图 2. 各位点属于组 1 的后验概率分布

4.3 问题三的模型建立和解决

通过问题二模型计算出属于组 1 集合位点的后验概率（即单个位点具有边缘效应的后验概率），进行降序排序之后（如图 3），取后验概率大于 0.019905（位点 rs2250358）的位点组成集合 S=（'rs2273298','rs12145450','rs364642','rs1802353','rs7368252','rs2229579','rs12042240','rs12136961"rs17361679','rs11249209','rs17401924','rs1152984"rs1883567','rs2250358'）

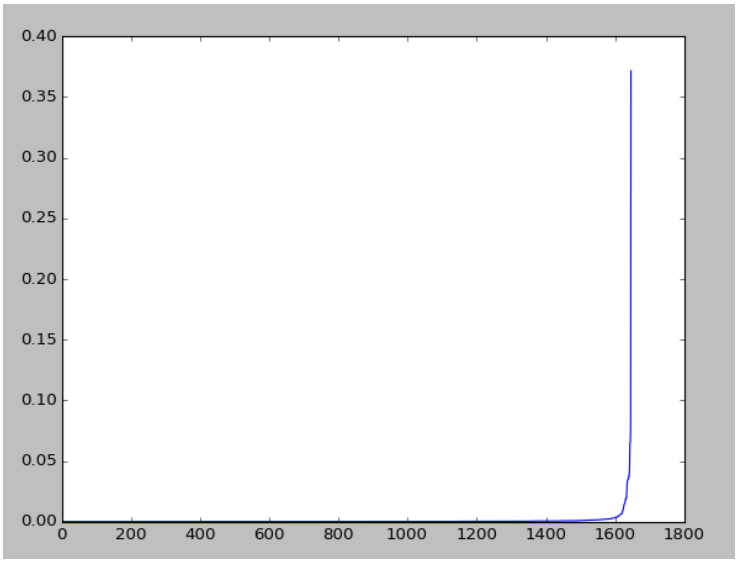

图 3　　单个位点边缘效应后验概率排序

将 300 个基因位点集合G_i与 S 求交集，后验概率和为p_i，p_i较大的基因与疾病 A 的相关性强。通过程序 3 计算出基因编号为[44，61，101，102，188，230，240，297]相关性较强。

4.4 问题四的解决

分别对十种性状，建立贝叶斯推断模型，计算出在该性状下，样本数据中基因位点的后验概率$S_i(i = 1 \ldots 10)$。使用程序 4 选取设在S_i中单边影响或关联影响较大的位点 100 个组成子集s_i，对这些子集取交集得到 s=set(['rs7513908', 'rs10909989', 'rs16824089', 'rs10157835', 'rs6663452', 'rs951805', 'rs17376524', 'rs10927473', 'rs12746773', 'rs4949516'])为与十种性状都具有关联性的位点集合。

五、 模型评价及改进

5.1 模型评价

本文中，我们使用三个离散数值来表示每个位点的不同编码，满足位点的编码只有三个并且是离散的性质。

本文建立了使用 MCMC 方法的贝叶斯推断模型，在数据规模大特征维度高的情况下，有效地找出了对疾病有影响作用的单个位点的边缘效应和多个位点的关联效应。

本文通过查找每个基因中包含的位点是否对疾病有影响以及影响的程度（即后验概率）来确定哪些基因对疾病有影响。因为基因是位点的一个子集，所以位点对疾病的影响以及影响程度与基因对疾病的影响是正相关的。

本文先查找出对每个性状有影响的作用的单个位点的边缘效应和多个位点的关联效应，然后在从中找出对 10 个形状都有影响作用的位点。

5.2 模型改进

本文在解决问题三，只是对基因中哪些对疾病有影响的后验概率进行简单的叠加，虽然能反映出不同基因对疾病的影响以及程度，但是方法过于粗糙，并且没能反映出不同基因对疾病的关联效应。在解决问题四时，没有进行性状对疾病影响的分析，即没有将性状与疾病关联起来，而只是找出了对 10 个性状都有影响的位点。

参考文献

[1] 权晟, 张学军, 全基因组关联研究的深度分析策略 [J]. 遗传, 2011. 33(2): 100-8.

[2] STADLER Z. K., THOM P., ROBSON M. E., et al., Genome-wide association studies of cancer [J]. Journal of Clinical Oncology, 2010. 28(27): 4255-67.

[3] JOSTINS L., BARRETT J. C., Genetic risk prediction in complex disease [J]. Human molecular genetics, 2011. 20(2): 182-8.

[4] EVANS D. M., VISSCHER P. M., WRAY N. R., Harnessing the information contained within genome-wide association studies to improve individual prediction of complex disease risk [J]. Human molecular genetics, 2009. 18(18): 3525-31.

[5] CHORK N J., FALLIN D., LANCHBURY J. S., Single nucleotide polymorphisms and the future of genetic epidemiology [J]. Clinical genetics, 2001. 58(4): 250-64.

[6] MITRA S., ACHARYA T., Data Mining: multimedia, soft computing, and bioinformatics [M]. Wiley-Interscience, 2003. 44(2): 637-89.

[7] FREITAS A. A., Understanding the crucial role of attribute interaction in data mining [J]. Artificial Intelligence Review, 2001. 16(3): 177-99.

[8] MOORE J. H., Computational analysis of gene-gene interactions using multifactor dimensionality reduction [J]. Expert review of molecular diagnostics, 2004. 4(6): 795-803.

[9] CHO Y., RITCHIE M., MOORE J., et al., Multifactor-dimensionality reduction shows a two-locus interaction associated with Type 2 diabetes mellitus [J]. Diabetologia, 2004. 47(3): 549-54.

[10] 尚军亮. 全基因组单核苷酸多态性交互作用研究[D]. 西安电子科技大学, 2013.

[11] 胡晓菡. CUDA 平台下的复杂疾病全基因组基因-基因相互作用研究[D]. 上海交通大学, 2010.

[12] 袁敏. 关联分析中的统计方法研究[D]. 中国科学技术大学, 2009.

[13] 韩建文,张学军, 全基因组关联研究现状[J]. 遗传, 2010, 33(1): 25-35.

源程序引索

程序 1

```
#include <iostream>
#include <fstream>
#include <string>
#include <string.h>
#include <sstream>


using namespace std;

void takeSubstring(string &str, string takens[]){
   stringstream ss;
   string subStr;
   ss.str(str);
   int length = 0;
   while(getline(ss, subStr, ' ')){
      takens[length++]= subStr;
   }
}


/*
TT:0 TC:1 CC:2
*/
bool isSame(string s){
if(s[0] == s[1]){
//cout << "isSame"  << endl;
return true;
}else return false;
}

void encoder(string* genoType[1001], string *rs){//genoType 1000*9445
bool first = true;
string firstSame = "";
int* genoCode[9445];//一共 9445 个位点，每个位点有 1000 个编码
for(int i = 0; i < 9445; i++){
genoCode[i] = new int[1000];
}

for(int i = 0; i < 9445; i++){
```

```
first = true;
for(int j = 0; j < 1000; j++){//跳过 rs 头部
if(genoType[j][i] == "ID" || genoType[j][i] == "II") {//脏数据
genoCode[i][j] = -1;
}
if(isSame(genoType[j][i])){
if(first){//如果第一次读到重复标记为 0
genoCode[i][j] = 0;
firstSame = genoType[j][i];
first = false;
}else if(genoType[j][i] == firstSame){ //和第一次重复的一样的重复对标记为 0
genoCode[i][j] = 0;
}else{//再出现重复对标记为 2
genoCode[i][j] = 2;
}
}else{//如果不是重复对标记为 1
genoCode[i][j] = 1;
}
}
}

//写入文件
freopen("data.txt", "w+", stdout);

//写入头信息
cout << "ID Chr   Pos   ";
for(int i = 0; i < 1000; i++){
if(i < 500){
cout << 1 << " ";
}else if(i < 999) cout << 0 <<" ";
else cout << 0 << endl;
}

//写入 code
for(int i = 0; i < 9445; i++) {
cout << rs[i] << " chr1   4924223   ";
for(int j = 0; j < 999; j++){
cout << genoCode[i][j] << " ";
}
cout << genoCode[i][999] << endl;
}

}
```

```
int fun(){
    string **genoType;
    genoType = new string* [1001];
    for(int i=0; i<1000; i++){
        genoType[i] = new string[9445];
    }

    ifstream in;
    string str;
    string takens[9445];
    string rs;
    string header[9445];

    in.open("genotype.dat");
    if(!(in.is_open())){
        cout<<"The file can't be open!"<<endl;
        return -1;
    }
    getline(in, rs);//位点信息读入
    takeSubstring(rs, header);

    for(int i=0; i<1000; i++){
        getline(in, str);
        takeSubstring(str, takens);
        for(int j=0; j<9445;j++){
            genoType[i][j]=takens[j];
        }
    }
    in.close();
    encoder(genoType, header);

}

int main()
{

fun();

    return 0;
}
```

程序 2

```
// main.cpp : Defines the entry point for the console application.
//

#include "datastructure.h"
#include "BiCluster.h"

void outputResult(const char *filename, vector<double> const &para, vector<OUTPUT> const &output, vector<int> const &totalcounts, vector<double> const &group1, vector<double> const &group2, vector<string> const &rs, vector<vector<int> > const &positions, int samplesize);
int loadCaseControl(const char *filename, vector<vector<char> > &dataC, vector<vector<char> > &dataU, vector<string> &rs, vector<vector<int> > &positions);
void loadParameters(const char *filename);

bool SNPid = true;
bool SNPpos = true;
int burnin = 100000;
int mcmc = 100000;
int thin = 1000;
int indRuns = 3;
bool multiTry = false;
bool singleOnly = false;
double prior1 = 0.01;
double prior2 = 0.01;
int mindistance = 1000000;
char outputfile[200];
char inputfile[200];
int startTries = 20;
int tryLen = 20;
double pThreshold = 0.1;
extern double autoRestart;

typedef struct MySortTypeP {
    int chr;
    int pos;
    int index;
} MySortTypeP;

bool operator<(const MySortTypeP &a, const MySortTypeP &b)
{   if(a.chr != b.chr) return a.chr < b.chr;
    else return a.pos < b.pos;
}
```

```
int main(int argc, char* argv[])
{
	sprintf(inputfile, "%s", "data.txt");
	sprintf(outputfile, "%s", "result.txt");
	loadParameters("parameters.txt");
	if(argc > 1)
	{	int i;
		sprintf(inputfile, "%s", argv[1]);
		sprintf(outputfile, "%s", argv[2]);
	}

	vector<vector<char> > dataC, dataU;
	vector<string> rs;
	vector<vector<int> > positions;
	printf("Loading data...\n");
	int alleleCn = loadCaseControl(inputfile, dataC, dataU, rs, positions);
	int NC = (int)dataC.size();
	int NU = (int)dataU.size();
	int L = 0;
	if(NC > 0) L = (int)dataC[0].size();
	if(burnin <= 0) burnin = L * 10;
	if(mcmc <= 0) mcmc = L * max(100, L);
	if(thin <= 0) thin = L;
	if(startTries <= 0) burnin = L * 100;
	if(tryLen <= 0) tryLen = L * 20;

	printf("_____\n");
	printf("Input file: \"%s\"\n", inputfile);
	printf("Maximum number of distinct alleles per marker: %d\n", alleleCn);
	printf("\n[PARAMETERS]\n");
	printf("# of Cases = %d\n# of Controls = %d\n# of Markers = %d\n", NC, NU, L);
	printf("# of chains = %d\nBurnin in each chain = %d\nIteration after burnin
= %d\nSample at every %d updates\n", indRuns, burnin, mcmc, thin);
	printf("Priors: p1 = %f, p2 = %f\n", prior1, prior2);
	printf("Threhold for Bonferroni corrected p-values: %f\n", pThreshold);
	if(singleOnly) printf("Test for marginal associations only.\n");
	else printf("Test for both marginal and epistatic associations.\n");
	printf("_____\n");
	if(L <= 0) return 0;

	BiCluster bcluster(alleleCn);
	vector<OUTPUT> output;
	vector<int> totalcounts;
	vector<double> group1, group2;
```

```
        bcluster.mBurnin = burnin;
        bcluster.mMCMC = mcmc;
        bcluster.mThin = thin;
        bcluster.minDistance = mindistance;
        bcluster.mPrior1 = prior1;
        bcluster.mPrior2 = prior2;

        time_t st, ed;
        time(&st);

        int samplesize = bcluster.runChains(dataC, dataU, pThreshold, output, totalcounts, group1, group2, indRuns, positions, multiTry, singleOnly, 0, startTries, tryLen);
        vector<double> para;
        para.push_back((int)dataC.size());
        para.push_back((int)dataU.size());
        para.push_back((int)dataC[0].size());
        para.push_back(indRuns);
        para.push_back(bcluster.mBurnin);
        para.push_back(bcluster.mMCMC);
        para.push_back(bcluster.mThin);
        para.push_back(bcluster.mPrior1);
        para.push_back(bcluster.mPrior2);
        outputResult(outputfile, para, output, totalcounts, group1, group2, rs, positions, samplesize);

        time(&ed);
        printf("\nDone!\nTotal Time = %d sec.\nResults are in the file \"", (int)ed - st);
        printf("%s\".\n", outputfile);

        return 0;
    }

    void outputResult(const char *filename, vector<double> const &para, vector<OUTPUT> const &output, vector<int> const &totalcounts, vector<double> const &group1, vector<double> const &group2,
                      vector<string> const &rs, vector<vector<int> > const &positions, int samplesize)
    {   int i, j, k;

        FILE *f = fopen(filename, "w");
        fprintf(f, "[INPUT_FILE]\n\"%s\"\n", inputfile);

        fprintf(f, "\n[PARAMETERS]\n");
```

```
	fprintf(f, "# of Cases = %d\n# of Controls = %d\n# of Markers = %d\n", (int)para[0], (int)para[1], (int)para[2]);
	fprintf(f, "# of chains = %d\nBurnin in each chain = %d\nIteration after burnin = %d\nSample at every %d updates\n", (int)para[3], (int)para[4], (int)para[5], (int)para[6]);
	fprintf(f, "Priors: p1 = %f, p2 = %f\n", para[7], para[8]);
	fprintf(f, "Threhold for Bonferroni corrected p-values: %f\n", pThreshold);
	if(singleOnly) printf("Test for marginal associations only.\n");
	else printf("Test for both marginal and epistatic associations.\n");

	fprintf(f, "\n[POSTERIOR_SIZES]\n");
	fprintf(f, "# of Marginal Associations:\n");
	for(k = 0; k < (int)group1.size(); k++)
		fprintf(f, "%d: %1.3f, ", k, group1[k]);

	fprintf(f, "\nInteraction Sizes:\n");
	for(k = 0; k < (int)group2.size(); k++)
	{	if(k == 0) fprintf(f, "0: %1.3f, ", group2[k]);
		else fprintf(f, "%d: %1.3f, ", k + 1, group2[k]);
	}

	fprintf(f, "\n\n[DETECTED_ASSOCIATIONS]\nMarker_Group\tPvalue\n");
	for(i = 0; i < (int)output.size(); i++)
	{	fprintf(f, "( ");
		for(j = 0; j < (int)output[i].markers.size(); j++)
			fprintf(f, "%d ", output[i].markers[j]);
		fprintf(f, " )\t%f\n", output[i].pvalue);
	}
	fprintf(f, "\n[POSTERIOR_DISTRIBUTION]\nMarginal + Interaction = Total\n");
	int L = (int)totalcounts.size() / 2;
	for(i = 0; i < L; i++)
	{	fprintf(f, "%d:", i);
		if((int)rs.size() > 0)
			fprintf(f, "%s\t", rs[i].c_str());
		if((int)positions.size() > 0)
		{	if(positions[i][0] < 23)
				fprintf(f, "Chr%d:%d\t", positions[i][0], positions[i][1]);
			else if(positions[i][0] == 23)
				fprintf(f, "ChrX:%d\t", positions[i][1]);
			else fprintf(f, "ChrY:%d\t", positions[i][1]);
		}
		fprintf(f, "%f\t+\t%f\t=\t%f\n", (double)totalcounts[i] / samplesize, (double)totalcounts[i + L] / samplesize, (double)(totalcounts[i] + totalcounts[i + L]) / samplesize);
	}
	fclose(f);
```

```
}

//disease status are represented by integers, take mean, and larger values will be taken as cases
int loadCaseControl(const char *filename, vector<vector<char> > &dataC, vector<vector<char> > &dataU, vector<string> &rs, vector<vector<int> > &positions)
{
	FILE *f = fopen(filename, "r");
	if(f == NULL)
	{	printf("Cannot open the data file \"%s\"\n", filename);
		return 0;
	}
	dataC.clear();
	dataU.clear();
	rs.clear();
	positions.clear();

	int i, j;
	char tmp[100000];
	vector<int> status;
	int alleleCn = 2;

	double mean = 0;
	fgets(tmp, 100000, f);
	for(i = 0; i < (int)strlen(tmp); i++)
		if((tmp[i] >= 48 && tmp[i] < 58) || tmp[i] == '-') //integers
		{	j = atoi(&tmp[i]);
			status.push_back(j);
			mean += (double)j;
		}
	mean /= (double)status.size();

	vector<vector<char> > tmpgenotype;
	while(fgets(tmp, 100000, f) != NULL)
	{	if((int)strlen(tmp) < 4) continue;
		i = 0;
		if(SNPid)
		{	char str[100];
			j = 0;
			for(i = i; i < (int)strlen(tmp); i++)
			{	str[j++] = tmp[i];
				if(tmp[i] == ' ' || tmp[i] == '\t')
					break;
			}
			str[j] = 0;
```

```
        rs.push_back(str);
}
if(SNPpos)
{       vector<int> p(2);
        for(i = i; i < (int)strlen(tmp); i++)
                if(tmp[i] == 'C' || tmp[i] == 'c')
                        break;
        if(tmp[i + 3] == 'X' || tmp[i + 3] == 'x') p[0] = 23;
        else if(tmp[i + 3] == 'Y' || tmp[i + 3] == 'y') p[0] = 24;
        else p[0] = atoi(&tmp[i + 3]);
        for(i = i + 3; i < (int)strlen(tmp); i++)
                if(tmp[i] == ' ' || tmp[i] == '\t')
                        break;
        for(i = i; i < (int)strlen(tmp); i++)
                if(tmp[i] >= 48 && tmp[i] < 58)
                        break;
        p[1] = atoi(&tmp[i]);
        for(i = i; i < (int)strlen(tmp); i++)
                if(tmp[i] == ' ' || tmp[i] == '\t')
                        break;
        positions.push_back(p);
}
vector<char> row;
do
{       for(i = i; i < (int)strlen(tmp); i++)
                if((tmp[i] >= 48 && tmp[i] < 58) || tmp[i] == '-')
                        break;
        if(i < (int)strlen(tmp))
        {       j = atoi(&tmp[i]);
                row.push_back(j);
                if(j >= 0 && j >= alleleCn) alleleCn = j + 1;
        }
        i++;
} while(i < (int)strlen(tmp));

//makeup missing alleles
vector<int> cn(alleleCn, 0);
for(i = 0; i < (int)row.size(); i++)
        if(row[i] >= 0) cn[row[i]]++;
for(i = 1; i < (int)cn.size(); i++)
        cn[i] += cn[i - 1];
for(i = 0; i < (int)row.size(); i++)
        if(row[i] < 0)
        {       int k = (int)((double)rand() / RAND_MAX * (double)cn[alleleCn - 1]);
```

```
                    for(j = 0; j < alleleCn - 1; j++)
                        if(k < cn[j])
                            break;
                    row[i] = j;
                }
            tmpgenotype.push_back(row);
        }
        fclose(f);

        int L = (int)tmpgenotype.size();
        //if positions are provided, sort SNPs
        vector<MySortTypeP> posmap;
        if((int)positions.size() == (int)tmpgenotype.size())
        {   posmap.resize(L);
            for(i = 0; i < L; i++)
            {   posmap[i].chr = positions[i][0];
                posmap[i].pos = positions[i][1];
                posmap[i].index = i;
            }
            sort(posmap.begin(), posmap.end());
            for(i = 0; i < L; i++)
            {   positions[i][0] = posmap[i].chr;
                positions[i][1] = posmap[i].pos;
            }
            if((int)rs.size() > 0)
            {   vector<string> rsbak = rs;
                for(i = 0; i < L; i++)
                    rs[i] = rsbak[posmap[i].index];
            }
        }

        for(i = 0; i < (int)status.size(); i++)
        {   vector<char> row(L);
            for(j = 0; j < (int)tmpgenotype.size(); j++)
            {   if((int)posmap.size() > 0)
                    row[j] = tmpgenotype[posmap[j].index][i];
                else row[j] = tmpgenotype[j][i];
            }
            if((double)status[i] <= mean)
                dataU.push_back(row);
            else dataC.push_back(row);
        }
        return alleleCn;
    }
```

```
void loadParameters(const char *filename)
{
	FILE *f = fopen(filename, "r");
	if(f == NULL)
	{	printf("Cannot open the parameter file \"%s\"\n", filename);
		return;
	}
	int i;
	char tmp[1000];
	while(fgets(tmp, 1000, f) != NULL)
	{	char str[1000];
		sprintf(str, "%s", tmp);
		str[11] = 0;
		if(strcmp(str, "INC_SNP_POS") == 0)
		{	for(i = 11; i < (int)strlen(tmp); i++)
				if(tmp[i] >= 48 && tmp[i] < 58)
					break;
			if(tmp[i] == 48) SNPpos = false;
			else SNPpos = true;
		}
		else if(strcmp(str, "SINGLE_ONLY") == 0)
		{	for(i = 11; i < (int)strlen(tmp); i++)
				if(tmp[i] >= 48 && tmp[i] < 58)
					break;
			if(tmp[i] == 48) singleOnly = false;
			else singleOnly = true;
		}
		else if(strcmp(str, "MINDISTANCE") == 0)
		{	for(i = 11; i < (int)strlen(tmp); i++)
				if(tmp[i] >= 48 && tmp[i] < 58)
					break;
			mindistance = atoi(&tmp[i]);
		}
		else if(strcmp(str, "INITIALTRYS") == 0)
		{	for(i = 11; i < (int)strlen(tmp); i++)
				if(tmp[i] >= 48 && tmp[i] < 58)
					break;
			startTries = atoi(&tmp[i]);
		}
		else if(strcmp(str, "AUTORESTART") == 0)
		{	for(i = 11; i < (int)strlen(tmp); i++)
				if(tmp[i] >= 48 && tmp[i] < 58)
					break;
```

```
        autoRestart = (double)atoi(&tmp[i]);
}
else if(strcmp(str, "P_THRESHOLD") == 0)
{       for(i = 11; i < (int)strlen(tmp); i++)
                if(tmp[i] >= 48 && tmp[i] < 58)
                        break;
        pThreshold = atof(&tmp[i]);
}

str[10] = 0;
if(strcmp(str, "INC_SNP_ID") == 0)
{       for(i = 10; i < (int)strlen(tmp); i++)
                if(tmp[i] >= 48 && tmp[i] < 58)
                        break;
        if(tmp[i] == 48) SNPid = false;
        else SNPid = true;
}
else if(strcmp(str, "TRY_LENGTH") == 0)
{       for(i = 10; i < (int)strlen(tmp); i++)
                if(tmp[i] >= 48 && tmp[i] < 58)
                        break;
        tryLen = atoi(&tmp[i]);
}

str[7] = 0;
if(strcmp(str, "INPFILE") == 0)
{       for(i = 7; i < (int)strlen(tmp); i++)
                if(tmp[i] != ' ' && tmp[i] != '\t')
                        break;
        int j;
        for(j = i + 1; j < (int)strlen(tmp); j++)
                if(tmp[j] == ' ' || tmp[j] == '\t' || tmp[j] == "")
                        break;
        tmp[j] = 0;
        if(tmp[i] == "") sprintf(inputfile, "%s", &tmp[i + 1]);
        else sprintf(inputfile, "%s", &tmp[i]);
}
else if(strcmp(str, "OUTFILE") == 0)
{       for(i = 7; i < (int)strlen(tmp); i++)
                if(tmp[i] != ' ' && tmp[i] != '\t')
                        break;
        int j;
        for(j = i + 1; j < (int)strlen(tmp); j++)
                if(tmp[j] == ' ' || tmp[j] == '\t' || tmp[j] == "")
```

```
                    break;
        tmp[j] = 0;
        if(tmp[i] == "") sprintf(outputfile, "%s", &tmp[i + 1]);
        else sprintf(outputfile, "%s", &tmp[i]);
}
str[6] = 0;
if(strcmp(str, "PRIOR1") == 0)
{       for(i = 6; i < (int)strlen(tmp); i++)
                if(tmp[i] != ' ' && tmp[i] != '\t')
                        break;
        prior1 = atof(&tmp[i]);
}
else if(strcmp(str, "PRIOR2") == 0)
{       for(i = 6; i < (int)strlen(tmp); i++)
                if(tmp[i] != ' ' && tmp[i] != '\t')
                        break;
        prior2 = atof(&tmp[i]);
}
else if(strcmp(str, "BURNIN") == 0)
{       for(i = 6; i < (int)strlen(tmp); i++)
                if(tmp[i] != ' ' && tmp[i] != '\t')
                        break;
        burnin = atoi(&tmp[i]);
}

str[5] = 0;
if(strcmp(str, "CHAIN") == 0)
{       for(i = 5; i < (int)strlen(tmp); i++)
                if(tmp[i] != ' ' && tmp[i] != '\t')
                        break;
        indRuns = atoi(&tmp[i]);
}
str[4] = 0;
if(strcmp(str, "THIN") == 0)
{       for(i = 4; i < (int)strlen(tmp); i++)
                if(tmp[i] != ' ' && tmp[i] != '\t')
                        break;
        thin = atoi(&tmp[i]);
}
else if(strcmp(str, "MCMC") == 0)
{       for(i = 4; i < (int)strlen(tmp); i++)
                if(tmp[i] != ' ' && tmp[i] != '\t')
                        break;
        mcmc = atoi(&tmp[i]);
```

```
            }
        }

        fclose(f);
}
```

程序 3

```
#-------------------------------------------------------------------------------
# Name:          findGene
# Purpose:
#
# Author:
#
# Created:       10/09/2016
# Copyright:     (c) 2016
# Licence:       <your licence>
#-------------------------------------------------------------------------------
#-*-coding: utf-8 -*-

import os

'''
从rs.txt中读取位点名称，标号为0-9445
'''
def getRs():#返回位点名，与位点标号
    f = open("rs.txt", 'r')
    rs_dic = {}
    rs = f.readline().split()
    f.close()
    for i in range(len(rs)):
        rs_dic[rs[i]] = i;

    return rs_dic

'''
返回一个300*len的矩阵gene_rs
len表示第i个基因中，位点个数
gene_rs[i]表示第i+1基因中所有位点
'''
def getGene():
    files = os.listdir('./gene_info/')
    gene_rs = [[]]*300
    gene_num = 0
```

```
    for i in range(300):
        filePath = './gene_info/' + files[i]
        gene_num = (files[i].split('.')[0]).split('_')[1]#获取基因标号
        f = open(filePath, 'r')
        l = lambda rs:rs.strip()
        gene_rs[int(gene_num)-1]= map(l, f.readlines())
        f.close()
    return gene_rs

'''
high_prob_rs表示高概率致病位点
返回1-300个基因中含有高致病位点个数
'''
def getGeneWithHighProbRS(high_prob_rs):
    gene_rs = getGene()
    gene_with_high_rs = [0]*300

    #print gene_rs[0]
    for i in range(300):
        for j in range(len(gene_rs[i])):
            gene_rs_set = set(gene_rs[i])
            interaction_len = len(high_prob_rs & gene_rs_set)
            if interaction_len is not 0:#如果两个set交集不为空
                gene_with_high_rs[i] = interaction_len

    return gene_with_high_rs


def main():
    high_prob_rs = ['rs2273298','rs12145450','rs364642','rs1802353'\
                    'rs7368252','rs2229579','rs12042240','rs12136961'\
                    'rs17361679','rs11249209','rs17401924','rs1152984'\
                    'rs1883567','rs2250358'
    ]
    gene_with_high_rs = getGeneWithHighProbRS(set(high_prob_rs))
    print gene_with_high_rs
    for i in range(300):
        if gene_with_high_rs[i] is not 0:
            print i, ' ',


if __name__ == '__main__':
    main()
```

程序 4

```
#-------------------------------------------------------------------------------
# Name:          findRs
# Purpose:
#
# Author:
#
# Created:       20/09/2016
# Copyright:     (c) 2016
# Licence:       <your licence>
#-------------------------------------------------------------------------------

#-*-coding: utf-8 -*-

import matplotlib.pyplot as plt
import os

def getMaxProb(filePath, HP):
    f = open(filePath, 'r')
    prob = {}
    for i in range(9445):#9445
        #print i
        line = f.readline().split()
        temp = line[0].split(':')[1]
        prob[temp] = float(line[2])

        #print line,prob
    sorted_prob = sorted(prob.iteritems(), key = lambda a:a[1],reverse = False)
    f.close()
    f = open('max_prob'+HP+'.txt', 'w+')
    lst_prob = []
    ax = []
    for i in range(len(sorted_prob)):
        if sorted_prob[i][1] != 0:
```

```
            lst_prob.append(sorted_prob[i][1])
            ax.append(float(sorted_prob[i][0][2:-1]))
            f.write(sorted_prob[i][0] + ' ' + str(sorted_prob[i][1]))
            f.write('\n')
    f.close()
    plt.scatter(ax,lst_prob)
    plt.annotate('local max', xy = (ax[-1], lst_prob[-1]), xytext = (3, 1.5), \
        arrowprops = dict(facecolor = 'black', shrink = 0.1))
    #plt.show()
    return sorted_prob



def main():
    files = os.listdir('results/')
    filePath = ''
    lst_prob = []*10
    for i in range(len(files)):
        filePath = 'results/'
        filePath += files[i]
        #print filePath
        sorted_prob = getMaxProb(filePath,str(i))
        lst_prob.append(sorted_prob[-100:])

    lst_rs = []*10
    rs = []
    print len(lst_prob)
    #print lst_prob
    for i in range(10):
        rs = []
        for j in range(100):
            rs.append(lst_prob[i][j][0])
        lst_rs.append(rs)

    set_a = set(lst_rs[3])
```

```
        #print set(lst_rs[0])
        for i in range(3):
            set_b = set(lst_rs[i])
            set_a = set_a & set_b
        print len(set_a), set_a
        #getMaxProb(file)


if __name__ == '__main__':
    main()
```