参赛密码 _____

（由组委会填写）

第十三届“华为杯”

全国研究生数学建模竞赛

学　　校	湖南师范大学
参赛队号	10542011
队员姓名	1.林益贤
	2.陈绍鹏
	3.刘婷

参赛密码 _____

（由组委会填写）

第十三届“华为杯”全国研究生数学建模竞赛

具有遗传性疾病和性状的遗传位点分析

摘 要

人类复杂疾病往往是由于多基因突变引发的，而这些发生突变的基因则通过相互作用影响疾病发展、调控生理活动。大量研究表明，人体的许多表型性状差异以及药物和疾病的易感性等都可能与某些位点相关联，或和包含有多个位点的基因相关联。因此，定位与性状或疾病相关联的位点在染色体或基因中的位置，能帮助研究人员了解性状和一些疾病的遗传机理，也能使人们对致病位点加以干预，防止一些遗传病的发生。

问题一针对不同的性状位点的碱基对类别的差异，采用一种新方法对碱基对进行编码，这种 0,1,2 编码方式不仅可以减少不同碱基对冗余信息的干扰，便于后续数据的处理。

问题二每个个体对应一个样本，每个个体的分类属性为是否患病，样本的每个特征代表一个 SNP 位点，其取值表示对个体 SNP 测试的结果。运用统计学以及机器学习的相关知识，对此数据进行分析，可以找到一个或多个 SNP 位点的子集，子集内的 SNP 位点或某些 SNP 位点的组合与所研究疾病有着统计意义上的强关联关系。针对具有遗传性疾病和性状的遗传位点进行相关性分析，通过降维处理筛选出 111 个的位点信息，然后 ReliefF 算法各自的特点及使用条件，提出改进的 ReliefF 和 PCA 算法结合起来进行 SNP 关联分析的算法。本文基于改进的 ReliefF 算法和 PCA 算法的结合来分析 SNP 关联分析，得出位点 rs2273298、 rs7368252、rs2807345、rs1009806 最有可能与疾病相关。

问题三对于问题二的求解，获得了每个位点对于疾病影响程度的权值。由于一个基因有多个位点，分析将一个基因理解为若干个位点组成的集合。将一个基因中的所有位点的权值累加起来，得到该基因的权值，该权值即表示该基因对疾病的影响程度。这样我们就得到了遗传疾病与该基因的关联性。选取权值大于设定的阈值（本文阈值设置为 2）的基因，就找出了与疾病最有可能相关的 19 个基因（gene_55、gene_ 217、gene_ 293 等）。

问题四中找出跟性状有关联的位点，文中给出的是 10 个性状，然后每个性状有 1000 样本，考虑到 10 个性状和位点之间的关系，可以转化先考虑单个性状和位点之间的关系，而且性状的属性只有 0 和 1 两种，这样就类似于疾病和位点之间的关系，从而可以用问题二的 ReliefF 算法找出每个性状的关联位点。最后用统计分析的方法确定与 10 个性状都存在相关联的位点为：rs10909903,rs6686587,rs311469,rs3128326,rs608203,rs10799145,rs10492987,rs7545115,rs2501401,rs4314833,rs351619,rs770706 ,rs6678618,rs11247946 。

关键词：致病基因，**PCA**，**ReliefF**，全基因组关联性分析, **SNP**

目 录

一、 问题的重述

1.1 问题的背景

随着信息技术的发展，现代生物学越来越多地将这些技术用于大规模生物数据的收集、分析、挖掘等过程。大量计算机技术，特别是机器学习方法，被用来进行复杂疾病的分析。常见复杂疾病通常是在遗传因素和环境因素的共同作用下产生的，往往是受到多个基因的联合作用。

单核酸多态（SNP），是一种主要的遗传差异，已经成功被用来检测单基因疾病。然而大多数疾病是复杂疾病，这些疾病由多个基因以及基因与环境的交互作用导致。目前，已经有提出了不少研究 SNP 相互作用的算法。但是这些算法对于寻找 3 个及 3 个以上的有交互作用的 SNP 的组合并不是特别有效。本文借鉴了一种算法，可以同时分析所有的 SNP。该方法将每个个体的 SNP 序列对应到高维空间中的一个点，这些点都携带相同大小的能量。然后寻找一个新的坐标系，使得疾病人群与健康人群的能量分布差异达到最大。根据这个新坐标系的信息，可以找到有交互作用的多位点的 SNP 的组合。在模拟数据上的实验显示本方法是有效的。

生物信息学是一门利用计算机科学、信息学、统计学和应用数学的方法研究生物问题的学科。现有的这一学科的研究基本上只是利用了信息技术与分子生物学。生物学技术能够生产大量的数据，其中包含很多嘈杂数据。生物学信息学就是利用计算工具从这些数据中提取相关的生物学习信息。GWAS 是指在人类全基因组范围内找到与所有的序列变异，即单核苷酸多态性(SNP),从中筛选出于疾病相关的 SNPS，利用大量的病例/对照组样本进行全基因遗传标记分型，找出所有变异的等位基因频率，用统计分析来找到以前未发现的基因，从而找与复杂疾病相关的遗传因素，得到一些复杂疾病致病机理的新的线索。

1.2 问题的要求

问题一：用适当的方法，把 genotype.dat 中每位点的碱基(A、T、C、G)编码方式转化为数值编码方式，便于进行数据分析。

问题二：根据附录中 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(见 phenotype.txt 文件)。设计或采用一个方法，找出某种疾病最有可能的一个或几个致病位点，并给出相关的理论依据。

问题三：同上题中的样本患有遗传疾病 A 的信息（phenotype.txt 文件）。现有 300 个基因，每个基因所包含的位点名称见文件夹 gene_info 中的 300 个 dat 文件，每个 dat 文件列出了对应基因所包含的位点(位点信息见文件 genotype.dat)。由于可以把基因理解为若干

个位点组成的集合，遗传疾病与基因的关联性可以由基因中包含的位点的全集或其子集合表现出来请找出与疾病最有可能相关的一个或几个基因，并说明理由。

问题四：在问题二中，已知 9445 个位点，其编码信息见 genotype.dat 文件。在实际的研究中，科研人员往往把相关的性状或疾病看成一个整体，然后来探寻与它们相关的位点或基因。试根据 multi_phenos.txt 文件给出的 1000 个样本的 10 个相关联性状的信息及其 9445 个位点的编码信息（见 genotype.dat），找出与 multi_phenos.txt 中 10 个性状有关联的位点。

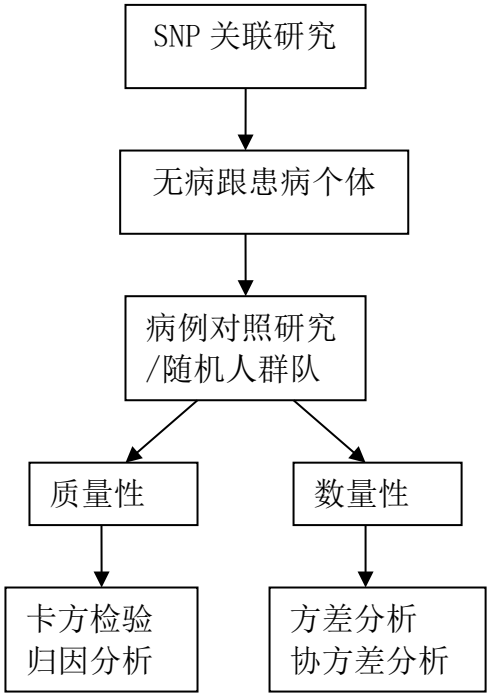
1.3 问题的分析

1.3.1 对问题 1 的分析

采用碱基(A、T、C、G)的编码方式来获取每个位点信息，因为染色体具有双螺旋结构，所以用两个碱基的组合表示一个位点的信息。对每个 SNP 位点进行编码，编码成最小等位基因的个数：如果最小等位基因为 a,那么基因型 AA,Aa,aa 一个由(1,0,0),(0,1,0),(0,0,1)组成的向量。

1.3.2 对问题 2 的分析

为 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，找出某种疾病最有可能的一个或几个致病位点，本文使用各种降维算法，数据挖掘算法与机器学习等算法寻找高阶 SNP 模式，这些模式在患病人群与健康人群中出现的频率应该存在非常大的差异，那么这些 SNP 模式就很有可能是致病的遗传标记。



1.3.3 对问题 3 的分析

由于问题二的求解，获得了每个位点对于疾病影响程度的权值。由于一个基因有多个位点，所以我们有如下分析，将一个基因理解为若干个位点组成的集合。将一个基因中的所有位点的权值累加起来，得到该基因的权值，该权值即表示该基因对疾病的影响程度。这样我们就得到了遗传疾病与该基因的关联性。选取权值大的一些基因，就找出了与疾病最有可能相关的 19 个基因。

1.3.4 对问题 4 的分析

问题四中讨论 10 个相关性状的信息与其相关联的位点，给出的是 10 个性状，然后每个性状有 1000 样本，考虑到 10 个性状和位点之间的关系，可以转化先考虑单个性状和位点之间的关系，也就是转化为了当初的疾病和位点之间的关系，这里的不同之处在于当初疾病前 500 个是 0，后 500 个是 1，这里性状 1000 个样本的 0 和 1 是随机出现的，没有规律，先单个用算法计算出与每个性状相关的位点集合，最后统计存在这 10 个位点集合中共同位点作为整体相关联的性状位点。

二、 基本假设

(1).假设题目所给的数据真实可靠。

(2).假设间隔定义为一个样本的最近邻相同类别样本与最近邻不同类别样本的距离，对于 K 近邻分类器，假设间隔小于样本间隔。

三、 符号说明和名词解释

3.1 符号说明

符号	符号说明
W[i]	位点 i 的特征值权重值；
K	为最邻近的数目；
M	重复次数；
R	随机选择的实例样本数；
X_p	P 个随机变量；
$X_{n\times p}$	表示 n 个样本的位点为 p 个；
N	训练样本 N 个特征；

3.2 名词解释

（1）MAF：等位基因最小频率；

（2）SNP：位点；

（3）ReliefF：机器学习中监督学习的算法；

四、 模型的建立与求解

4.1 问题 1 的分析与求解

4.1.1 问题的分析与求解

人的染色体都是成对出现的。如果得到每条染色体上对应 SNP 的信息，即单体型信息，可以推断出很多信息。但是现有的微矩阵技术单独得到每条染色体上对应 SNP 的信息很困难，而得到一对染色体上的 SNP 位点的混合信息相对容易，即基因型信息。目前各种研究机构已经积累了很多基因型数据，如何从这些数据中挖掘到有用的信息是非常重要的研究课题。本文中我们研究的重点也是基因以及位点型数据。当前我们主要研究二对等位基因的 SNP，即给定一个 SNP 位点，它的值只有两种可能，主等位基因与次等位基因。s_i 处的 SNP 的值 g_i 有 3 种可能。g_i=0，代表 s_i 处的两条染色体上都是主等位基因。g_i=2，代表 s_i 处的两条染色体上都是次等位基因。g_i=1 代表 s_i 处的两条染色体上是一个主等位基因和一个次等位基因。如果一个人的 DNA 序列有 n 个 SNP 位点，那么其 SNP 数据可以表示为一个由 0，1，2 组成的向量。在我们的算法中在此基础上进行了一个变换，得到新的向量 $G=(v_{1,1},v_{1,2},v_{1,3}\ldots,v_{n,1},v_{n,2},v_{n,3})$。 其变换如公式所示：

$$\begin{cases} v_{i,1}=1,\ v_{i,2}=0,\ v_{i,3}=0, & \text{if } g_i=0; \\ v_{i,1}=0,\ v_{i,2}=1,\ v_{i,3}=0, & \text{if } g_i=1; \\ v_{i,1}=0,\ v_{i,2}=0,\ v_{i,3}=1, & \text{if } g_i=2; \end{cases}$$

这种编码方式的是每个人的 SNP 序列对应一个点，这些点位于高维空间中的不同位置，它们携带相同的能量。如果使用 0,1,2 的编码方式，虽然它们表示的点的位置不一样，但是同时它们携带的能量也不一样，因而不符合我们算法的要求。我们使用 hi 表示第 i 个健康人的 SNP 序列，dj 表示第 j 个患者的 SNP 序列。则健康人群的 SNP 数据可以表示为 3n×t 的矩阵 H=(h1,h2,……,ht)，患病人群的 SNP 表示 3n×r 矩阵 D=（d1,d2,……,dr）,其中 t 和 r 分别为两种人群人数。

4.1.2 数据的采集与处理

采用题目中给定的 genotype.dat 与 phenotype.txt 文件数据，genotype.dat 文件中一个位点为碱基对构成，碱基对有 3 中可能，如 AA、AT、TT，SNP 位点数据可以表示为由 0,1,2 组成的向量。

MATLAB 核心代码实现如下：

```
for j=1:1:9445
    letter(1)='A';letter(2)='T';letter(3)='C';letter(4)='G';
    judge(:,1)=strcmp(genotype_cell(:,j),[letter(1),letter(1)]);
```

```
        sum1(1)=sum(judge(:,1));
        judge(:,2)=strcmp(genotype_cell(:,j),[letter(2),letter(2)]);
        sum1(2)=sum(judge(:,2));
        judge(:,3)=strcmp(genotype_cell(:,j),[letter(3),letter(3)]);
        sum1(3)=sum(judge(:,3));
        judge(:,4)=strcmp(genotype_cell(:,j),[letter(4),letter(4)]);
        sum1(4)=sum(judge(:,4));
        [sum1_sorted,sum1_idx]=sort(sum1,'descend');
        goodLetter(1)=letter(sum1_idx(1));
        goodLetter(2)=letter(sum1_idx(2));
        judge5=strcmp(genotype_cell(:,j),[goodLetter(1),goodLetter(2)]);
        sum2=sum(judge5);
        judge6=strcmp(genotype_cell(:,j),[goodLetter(2),goodLetter(1)]);
        sum3=sum(judge6);
        bp{1}=[goodLetter(1),goodLetter(1)];
        bp{3}=[goodLetter(2),goodLetter(2)];
        if sum2>=sum3
            bp{2}=[goodLetter(1),goodLetter(2)];
        else
            bp{2}=[goodLetter(2),goodLetter(1)];
        end
        genotype_cell(:,j)=strrep(genotype_cell(:,j),bp(2),'100');
        genotype_cell(:,j)=strrep(genotype_cell(:,j),bp(1),'010');
        genotype_cell(:,j)=strrep(genotype_cell(:,j),bp(3),'001');
    end
```

4.1.3 结论

SNP 位点由碱基对构成，对碱基对进行编码，本文采用 0,1,2 的编码方式，以便后面数据的分析处理。

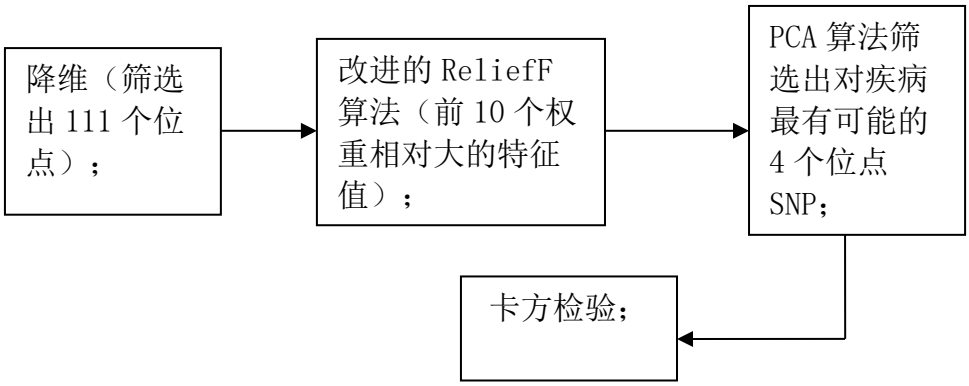
4.2 问题 2 的分析及求解

4.2.1 问题的分析

鉴于复杂疾病通常是由基因网络上多个位点的变异引起，每个 SNP 位点的变异可能单独对疾病产生致病作用，也有可能每个位点的变异只有比较微弱的致病效应，但是通过和其他 SNP 位点的统计交互作用就能对疾病产生很强的致病作用。每个个体对应一个样本，每个个体的分类属性为是否患病，样本的每个特征代表一个 SNP 位点，其取值表示对个体 SNP 测试的结果。运用统计学以及机器学习的相关知识，对此数据进行分析，可以找到一个或多个 SNP 位点的子集，子集

内的 SNP 位点或某些 SNP 位点的组合与所研究疾病有着统计意义上的强关联关系。本文提出的方法同时分析单个位点和多个位点的交互作用，识别对疾病有致病作用的 SNP 位点。

本文全基因组致病 SNP 位点识别主要包括下面几项工作：



（1） 先进行降维处理，对 9445 个位点进行筛选；

（2） 深入分析各种识别致病 SNP 位点方法的性能；

本文通过模拟数据集，对各种识别致病位点的方法进行比较，分析了卡方检验、ReliefF 和 PCA 等方法在各种模型下识别致病位点的能力，分析这些方法在哪些模型下表现突出，或者在哪些模型下表现不足，从而为本文提出的方法提供依据。

（3） 提出结合 ReliefF 和 PCA 的识别方法；

本文提出了两种结合 ReliefF 和 PCA 的识别方法，这两种方法都是基于后向缩减的迭代策略，都能最终产生对所有 SNP 位点的特征值排名，可以利用这个排名去筛选得到致病位点。

（4） 验证方法的有效性；

问了验证提出方法的有效性，同时在大量模拟数据集和真实数据集进行验证，通过设置不同模拟数据集的模型，分析提出的方法在各种模型下的表现。

4.2.2 数据的采集与处理

采用题目中给定的 genotype.dat 与 phenotype.txt 文件数据，从 phenotype.txt 文件中 1000 个样本中前 500 样本是无病的，后 500 是患病的，分析此两文件数据的关联关系，从而分析出疾病最有可能是哪些位点造成的。

4.2.3 模型的建立与求解

本文先用最小等位基因频率 MAF 对 SNP 位点进行筛选，把筛选过后的数据进行 ReliefF 改进后的算法分析，采用 ReliefF 改进后的算法与 PCA 算法结合，最终分析出最有可能的位点对疾病的关联。

1）由题目给出的数据，500 个样本是患病的，500 个是无病的数据，本文统计碱基在在患病与无病中的数量，然后把患病和无病中的碱基数量进行相减，如果相差过大，则说明此碱基对患病的影响的概率可能会比较大，如在位点 rs1 处的碱基对为 TC、TT、CC，我们统计等位基因 T 在患病的样本中出现的次数和统计 T 的在无病的样本中出现的次数，用患病中 T 的次数减去无病中 T 的次数得到的结果，如果此结果相关过大，说明此碱基在患病中的影响可能会比较大，因此可以将它留下进行分析，去除影响小的碱基作为分析，然后进行筛选出一定的位点，再用剩余的位点进行 MAF 算法处理，进一步进行筛选位点工作；最小等位基因频率（Minor Allel Frequencey,MAF）是指某个位点少见等位基因的观察频率，GWAS 中较小的 MAF 会使得统计效能降低，因此导致出现假阳性的结果。当 MAF 小于 0.05 时不作为全基因关联分析的研究对象，国际人类基因组单体型计划建议将 MAF 设置在 0.05 到 0.5 之间，根据一个 SNP 位点的 MAF，可以筛选出一定数量的 SNP 位点，把未筛选的 SNP 进行关联分析处理。本文最终筛选后剩下 111 个 SNP 进行数据分析处理。

2）Relief 算法是一种监督学习方法，通过对每个样本的寻找最近邻的样本来更新特征权重，是一种特征选择方法。Relief 在更新每个特征权重时，考虑整个特征空间，具体做法是如果考虑某一个样本 X 的重要性特性时，先找到同一个类别最近邻的样式 Y，然后找到一个不同类别的最近邻样本 Z。如果某个特征在 X 和 Y 比较一致，但是在 X 和 Z 之间非常不同，那么这样的特征被认为是重要特征，对样本具有很强的区分度，会给这些特征赋一个较高的权重。Relief 方法流程如下：

输入：M 个训练样本 X，每个训练样本 N 个特征，迭代次数为 T(通常设为 M）；

输出：每个特征 F_i 的权重 W[i] $\in$ [-1,1]；

初始化：对每个特征的权重初始化为 0，W[i]=0；

for t=1 to T do

随机从数据集中选择一个样本 X；

找到 X 的相同类别的最近邻样本 NH_x 和不同类别最近邻样本 NM_x；

for i=1 to N do

$$W[i] = W[i] + diff(X^{(i)}, NM_x^{(i)})/(N \times T) - diff(X^{(i)}, NH_x^{(i)})/N \times T$$

end for

end for

return(W).

对于 SNP 数据，基因型只有 0,1,2 三种取值，所以对每个特征位点 i，样本 X 和 Y 的在特征 i 处的距离函数 diff 可以通过下面公式计算：

$$diff(X^{(i)},Y^{(i)})=\begin{cases}0, X^{(1)}=Y^{(1)}\\ 1, X^{(1)}\neq Y^{(1)}\end{cases}$$

每一次迭代，每个特征的权重都得到更新，并且归一化为-1 到 1 之间，输出是每个特征的权值，原始论文中，作者提出一个关联阈值 τ，可以通过选择阈值大于等于 τ 的权重对应的特征来过滤特征，得到重要特征。Relief 可以用来对特征进行排名。Relief 算法同时进行全局和局部搜索，不像其他启发式搜索会陷入局最优。

3）Relief 算法改进之后的算法为 ReliefF 算法，ReliefF 算法 Relief 方法的改进版本，ReliefF 算法的主要思想是根据属性在区分相互靠近的样本实例的能力上对属性的质量进行评价，首先随机地从训练数据中选择一个样本实例 R，然后再训练数据中找出和样本实例 R 在同一类中的 K 个最近邻的样本实例，把这 K 个样本实例称为 Nhits，然后再训练数据中分别找出和样本实例 R 不在同一个类中的 K 个最近邻的样本实例，把这些 K 个样本实例称作 Nmisses;每个属性 A 的权重 W[A]的更新依赖于随机选择的样本实例 R、和 R 在同一类中的 K 个近邻 NHits 以及和 R 不在同一类中的若干 K 个近邻 Nmisses,在属性权重的更新公式中所有 NHits 和 NMisses 的贡献是经过平均处理的。

基于对于 Relief 算法改进的主特征提取，基于前面介绍的原理，对改进 Relief 算法的过程如下：

(1) 去除无关特征变量。利用改进 ReliefF 算法删除原始特征中那些与分类不相关的特征。即当修改参数 M（重复次数）和 K（为最近邻的数目）时，有些特征的权重是始终保持不变的，这些特征就是与分类不相关的特征，可以把这些特征删除。经过处理后得到数据矩阵为

$$X_{n\times p}=\begin{bmatrix}x_{11} & x_{12} & \cdots & x_{1p}\\ x_{21} & x_{22} & \cdots & x_{2p}\\ \vdots & \vdots & & \vdots\\ x_{n1} & x_{n2} & \cdots & x_{np}\end{bmatrix}_{n\times p}=(X_1,\ X_2,\cdots,\ X_p)=\begin{pmatrix}I_1\\ I_2\\ \vdots\\ I_n\end{pmatrix}$$

其中，$X_{n\times p}$ 表示一个 n 行 p 列的一个矩阵，代表 n 个样本实例，并且每个样本实例是 p 维的，可表示 p 个 SNP 位点；x_{ij} 表示第 j 人特征变量在第 i 个样本实例上的观测值；X_j 代表第 j 个特征变量的观测向量；I_i 代表第 i 个样本实例的观测向量。

(2) 数据归一化处理。将矩阵 $X_{n\times p}$ 转化为矩阵 $Z_{n\times p}$，变换公式为

$$Z_{ij}=\frac{x_{ij}-\bar{x}_j}{\sqrt{D(x_j)}}, i=1,2,...,n;\ j=1,2,...,p$$

其中，$\bar{x}_j$代表$X_{n\times p}$中第 j 列（第 j 列特征变量）的均值；$\sqrt{D(x_j)}$代表矩阵$X_{n\times p}$中第 j 个特征的标准差。

(3)进行主成分变换，先计算矩阵$Z_{n\times p}$协方差矩阵$\sum$，然后计算协方差矩阵$\sum$的特征值和特征向量，特征值从大到小依次为λ_1，λ_2，…，λ_p。

在近邻距离 K=1 时，各位点属性权重归一化后的值如下：

位点	权重值	位点	权重值
rs2273298	0.841	rs556596	0.793
rs7368252	0.828	rs4391636	0.775
rs2807345	0.821	rs3795438	0.757
rs1009806	0.821	rs10864301	0.748
rs10915035	0.812	rs6429695	0.739
rs4391636	0.811	rs11580218	0.730
rs2982376	0.802	rs2801178	0.712

在近邻距离 K=3 时进行测试，前 14 位点属性权重归一化如下：

位点	权重值	位点	权重值
rs2273298	0.791	rs11584631	0.67
rs7368252	0.761	rs4391636	0.651
rs2807345	0.751	rs3795438	0.649
rs1009806	0.736	rs10864301	0.642
rs11588669	0.691	rs1009806	0.637
rs4391636	0.682	rs11580218	0.632
rs4646092	0.682	rs2801178	0.627

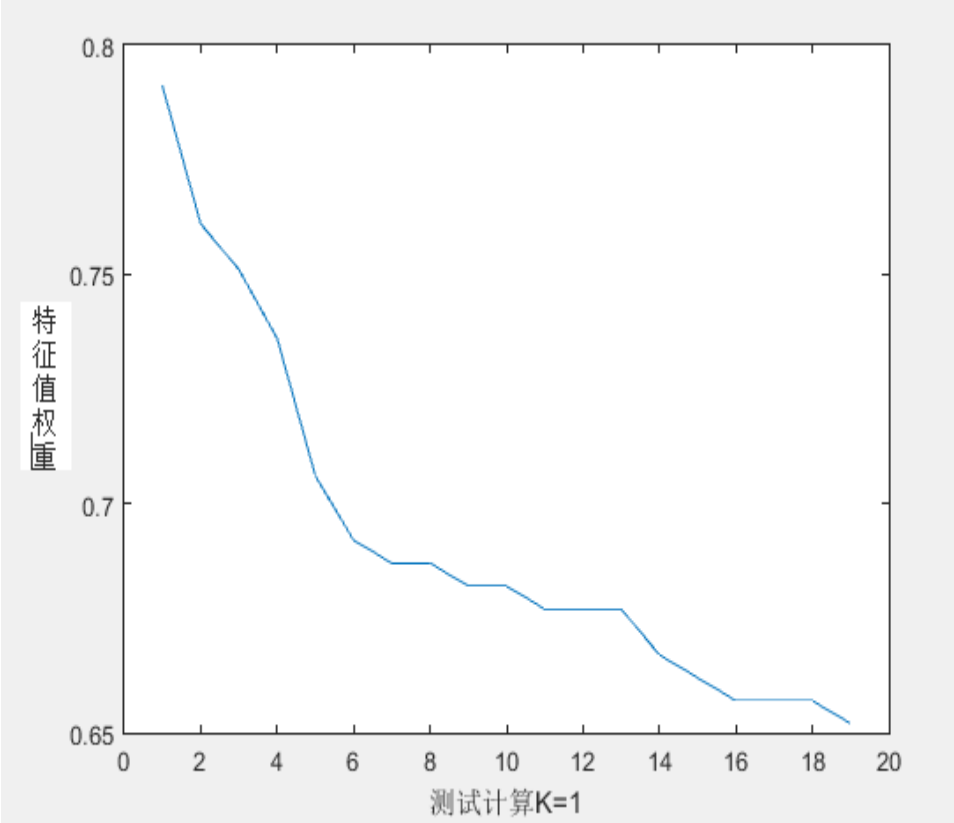
在近邻距离 K=5 时进行测试，前 14 位点属性权重归一化如下：

位点	权重值	位点	权重值
rs2273298	0.584	rs556596	0.508
rs7368252	0.563	rs4391636	0.504
rs2807345	0.556	rs3795438	0.504
rs1009806	0.546	rs10864301	0.497
rs10915035	0.525	rs6429695	0.491
rs4391636	0.515	rs2801178	0.487
rs11580218	0.511	rs2473253	0.484

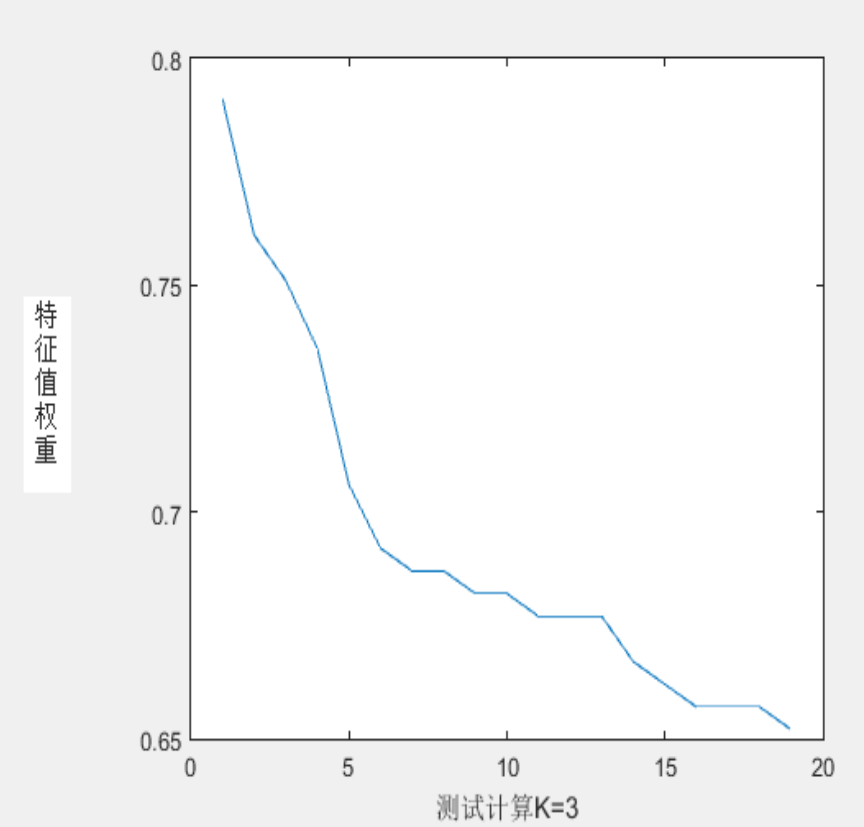
从以上图中可以得出在近邻距离 K=1、K=3、K=5 时，各位点的大小顺序趋势基本都是一致的。在 K=1 时从按照从小到大顺序排列，可知各个位点的权重关系如下：

rs2273298 < rs7368252 < rs2807345 < rs1009806 < rs10915035 < rs4391636 < rs4391636 < rs2982376 < rs556596，我们选定权重阀值为 0.9，则其位点权重小于 0.9 的剔除。

从上面的特征权重可以看出，rs2273298 大小是最主要的影响因素，说明患者的症状最先表现了位点 rs2273298 大小上，将直接导致患病的变化，其次是 rs7368252、rs2807345 以及 rs1009806 等，后几个位点权重大小接近，但是从多次计算规律来看，还是能够说明其中不同的重要程度，下面是着重对几个重要的位点属性进行分析。下面是测试近邻 K=1 时，位点的特征值权重图：



从上图中可以看到该属性权重大部分在 0.75-0.8 左右，是权重最大的一个属性。下面由近邻 K=3 时的权重分布：



从以上实验中，可以得知位点 rs2273298、 rs7368252、rs2807345、rs1009806、 rs10915035、 rs4391636、rs4391636、rs2982376、 rs556596 最有可能对疾病造成影响。

4）主成分分析是特征提取中很常见的一种变换方法，该方法通过使用变换后的几个主要的成分来代替原始的特征信息，变换后的主成分之间是相互独立的，并且每个主成分都是变换钱所有特征的一个线性组合，如果用变换后的所有主成分来代替原始的特征信息，一般没有信息的损失，因此，被广泛应用到模式匹配的相关领域中。但在很多情况下，通常选择少数的几个包含大部分原始特征信息，但可以使用更少的特征变量来代替原始较多的特征变量，这样可以大大减轻后续分类器的计算量，同时对提高分类器的性能也是有帮助的。

设 X_1，X_2，……，X_p 为 p 个随机变量，记 $X=(X_1,X_2,.....,X_p)^T$，令 $\sum$ 为 X 的协方差矩阵，进行线性变换：

$$\begin{cases} Y_1 = a_{11}X_1 + a_{12}X_2 + ... + a_{1p}X_p = a_1{}^t X \\ Y_2 = a_{21}X_1 + a_{22}X_2 + ... + a_{2p}X_p = a_2{}^t X \\ \vdots \\ Y_p = a_{p1}X_1 + a_{p2}X_2 + ... + a_{pp}X_p = a_p{}^t X \end{cases}$$

可以得到：$\mathrm{var}(Y_i) = \mathrm{var}(a_i^T X) = a_i^T \sum a_i, i = 1,2,..., p$

$\mathrm{cov}(Y_i, Y_j) = \mathrm{cov}(a_i^T X, a_j^T X) = a_i^T \sum a_j, i, j = 1,2,..., p$

显然我们希望Y_1是X_1，X_2，…，X_p的所有线性函数中方差最大的，此处限制a_1为单位向量，即有$a_i^T a_i = 1$，这样可使得$\text{var}(Y_i) = a_i^T \sum a_i$达到最大，此时就称$Y_i$为第一主成分。如果第一主成分所包含的信息还不够多，不足以代表原始的 p 个变量，就要考虑使用Y_2、Y_3、Y_4等，一般来说，X 的第 i 主成分$Y_i - a_i^T X$指：$a_i^T a_i = 1$和$\text{cov}(Y_i, Y_j)$=0，k=1, 2…, i-1 下寻找a_i, 使得$\text{var}(Y_i) = a_i^T \sum a_i$达到最大。

主成分分析结果：

成分	初始特征值			提取平方和载入		
	合计	方差的%	累积%	合计	方差的%	累积%
1	5.024	50.235	50.23	5.024	50.235	50.235
2	2.080	20.35	70.5	2.080	20.799	71.034
3	0.735	7.385	77.9	0.735	7.386	78.3
4	0.686	6.857	86.7	0.686	6.56	84.83
5	0.376	3.767	91.4			
6	0.303	3.024	94.4			
7	0.286	2.78	96.6			
8	0.224	2.56	98.1			
9	0.211	1.047	99.1			
10	0.083	0.830	100			

这就是主成分分析的结果，表中第一列为 10 个成分；第二列为对应的“特征值”，表示所解释的方差的大小；第三列为对应的成分所包含的方差占总方差的百分比；第四列为累计的百分比。选择“特征值”大于 1 的成分作为主成分，这也是 SPSS 默认的选择。

在本题中成分 1、2、3、4 的特征值相对比较大，他们合计能解释大于 90%的方差，所以我们可以提取 1、2、3、4 作为主成分，抓住了主要矛盾，其余成分包含的信息较少，故弃去。

综上所述，通过对位点和疾病的关联分析，我们去选最有可能引起疾病的位点为 rs2273298、 rs7368252、rs2807345、rs1009806。

4.2.4 结论

由以上分析得，疾病发生的最有可能出现的位点为 rs2273298、rs7368252、rs2807345、rs1009806，在这些位点处出现的权重以及与疾病的关联性是最高的。

4.2.5 模型的进一步验证

卡方统计量是基于卡方分布的一种检验方法，根据数值频数值来构造统计量，是一种非参数检验方法。卡方值越大，越不符合；卡方值越小，偏差越小，越趋于符合，若两个值完全相等，卡方值为 0，表明理论值完成符合。检验 rs2273298、 rs7368252、rs2807345、rs1009806 位点与疾病的关联关系。

(1).提出原假设：

由于总体分布为离散型，则假设具体为

H_0:.总体 X 的分布律为 P{X=x_i}=p_i，i=1,2,…;

皮尔逊卡方检验统计量 $\chi^2=\sum_{i=1}^{k}\frac{(f_i-np_i)^2}{np_i}$；

针对位点 rs2273298，通过皮尔逊卡方检验，卡方值是 28.835，自由度 2，渐进显著水平为 0.000，远小于 5%。所以在是否患病的情况下，这三种碱基的数量有明显的差别。说明该位点为致病位点。

表：位点 rs2273298 与是否患病交叉表

			是否患病		总计
			患病	不患病	
rs2273298	AA	计数	222	305	527
		预期计数	263.5	263.5	527.0
	AG	计数	218	161	379
		预期计数	189.5	189.5	379.0
	GG	计数	60	34	94
		预期计数	47.0	47.0	94.0
总计		计数	500	500	1000
		预期计数	500.0	500.0	1000.0

表：卡方检验			
	值	自由度	渐近显著性 （双向）
皮尔逊卡方	28.836[a]	2	.000
似然比(L)	29.018	2	.000
线性关联	27.379	1	.000
有效个案数	1000		

a.0 个单元格 (0.0%) 具有的预期计数少于 5。最小预期计数为 47.00。

针对位点 rs2807345，通过皮尔逊卡方检验，卡方值是 13.198，自由度 2，渐进显著水平为 0.001，远小于 5%。所以在是否患病的情况下，这三种碱基的数量有明显的差别。说明该位点为致病位点。

rs2807345 * 是否患病 交叉表

			是否患病		
			患病	不患病	总计
rs2807345	CC	计数	281	333	614
		预期计数	307.0	307.0	614.0
	CT	计数	179	145	324
		预期计数	162.0	162.0	324.0
	TT	计数	40	22	62
		预期计数	31.0	31.0	62.0
总计		计数	500	500	1000
		预期计数	500.0	500.0	1000.0

卡方检验

	值	自由度	渐近显著性（双向）
皮尔逊卡方	13.198[a]	2	.001
似然比(L)	13.285	2	.001
线性关联	13.184	1	.000
有效个案数	1000		

a. 0 个单元格 (0.0%) 具有的预期计数少于 5。最小预期计数为 31.00。

同理可以得出 rs7368252、rs2807345、rs1009806 这三个位点的卡方检验分析，渐进显著水平为 0.000，远小于 5%。所以在是否患病的情况下，这三种碱基的数量有明显的差别。说明该位点为致病位点。

4.3 问题 3 的分析求解

4.3.1 问题的分析

就目前所知，不同人的DNA序列99.9%是相同的，然而剩下的0.1%的不同导致了个体的多样性，它们导致了人们不同的属性与性格等差异。DNA 上的差异有很大一部分是 SNP。

对于问题二的求解，获得了每个位点对于疾病影响程度的权值。由于一个基因有多个位点，所以我们有如下分析，将一个基因理解为若干个位点组成的集合。将一个基因中的所有位点的权值累加起来，得到该基因的权值，该权值即表示该基因对疾病的影响程度。这样我们就得到了遗传疾病与该基因的关联性。选取权值大的一些基因，就找出了与疾病最有可能相关的基因。最终可以算得疾病最有可能的基因是 gene_ 55、gene_ 217、gene_ 293、gene_ 30、gene_ 83、gene_ 113、gene_ 144、gene_ 162、gene_ 169、gene_ 22 等 19 个基因。

4.3.2 数据的采集与处理

导入附件中的 gene_info 文件夹里的 gene_0.dat 到 gene_300.dat 的基因序列文件。从而我们可以得到 9445 个位点所在 300 个基因的位置。

MATLAB 代码如下：

```
temp=importdata('.\OrigionData\WeiDianIdByMAF.txt');
temp=temp';
numLocus_MAF=size(temp,2);
IndexOfEachGenotype_MAF(1,:)=1:1:numLocus_MAF;
IndexOfEachGenotype_MAF(2,:)=temp(:);
```

4.3.3 模型的建立与求解

问题 3 中，分析遗传疾病 A 与基因的关联关系，基因为若干个位点组成集合，遗传病与基因的关联性可以由包含的位点的全集或子集合表现出来。检查与疾病在统计学上的关联的因素，并以此作为潜在病因。从三百个基因文件中做统计，发现最多一个基因文件中有 60 个位点；利用 geneotype.dat 文件的第一行的位点名称对 9445 个位点进行编号，按照 1,2,3,4……形式，因此每个基因文件中的位点可以用编号代替，将三百基因文件形成一个 300*60 的矩阵的形式，300 表示基因个数中 60 表示最大位点个数，矩阵里面数字为基因中位点的编号，对于很多基因没有 60 个位点的用 0 来填充。

在问题二中，我们算得出了所有位点的权值。定义起方程为

$$W_j = \sum_{n=1}^{9445} \alpha_{jn} W_n$$

其中W_j 表示第 j 个基因的权值；α_{jn} 表示第 n 个位点是否在第 j 个基因中。

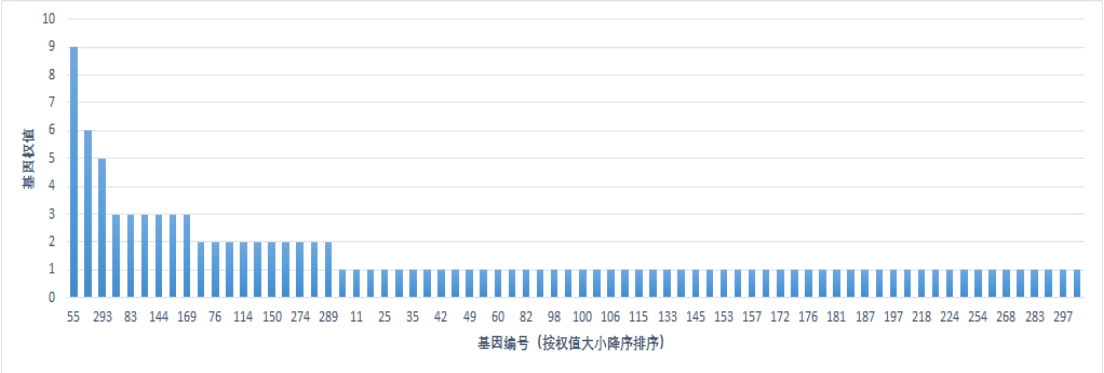
$$\alpha_{jn} = \begin{cases} 1 & \text{第n个位点在第j个基因中} \\ 0 & \text{第n个位点不在第j个基因中} \end{cases}$$

MATLAB 代码如下：

```
weightOfEachGene_MAF=zeros(2,numGene);
weightOfEachGene_MAF(1,:)=1:1:numGene;
for j=1:numLocus_MAF

   weightOfEachGene_MAF(2,mapOfGenotype_cell{3,IndexOfEachG
   enotype_MAF(2,j)})=weightOfEachGene_MAF(2,mapOfGenotype
   _cell{3,IndexOfEachGenotype_MAF(2,j)})+1;
end
[result , idx]=sort(weightOfEachGene_MAF(2,:),'descend');
numGene_MAF=numGene-
   sum(weightOfEachGene_MAF_sorted_result==0);
weightOfEachGene_MAF_sorted_result=result(1:numGene_MAF);
weightOfEachGene_MAF_sorted_index=idx(1:numGene_MAF);
```

将所有基因以基因权值为依据，按照从大到小的顺序排序，据此计算出 300 个基因中，与 111 个致病位点有关的 72 个候选致病基因，并计算出 72 个候选致病基因的权值，将这些基因以基因权值为依据，按照从大到小的顺序排序。我们发现，即使是权值最大的基因，对疾病的贡献率也仅有 0.087%。因此，我们需要计算累计贡献率，并将累计贡献率在 50%前的所有基因列入与疾病有关的基因集合中。绘制出基因权值排序柱状图如下：



从图中我们可以发现前 19 个基因的权重相对来说是比较大的，后

面的基因都是处于一种趋势发展；所以我们认为前 19 个基因可能是造成疾病的原因。

基因的权重值如下图：

基因	权重值	基因	权重值
gene_ 55	9	gene_ 22	2
gene_ 217	6	gene_ 76	2
gene_ 293	5	gene_ 102	2
gene_ 30	3	gene_ 114	2
gene_ 83	3	gene_ 149	2
gene_ 113	3	gene_ 150	2
gene_ 144	3	gene_ 241	2
gene_ 162	3	gene_ 274	2
gene_ 169	3	gene_ 280	2

4.3.4 结论

通过计算，我们得到的与疾病有关的基因集合有 19 个。

4.3.5 模型的进一步验证

问题三进行卡方检验，针对基因 gene_ 55，通过皮尔逊卡方检验，卡方值是 28.835，自由度 2，渐进显著水平为 0.015，远小于 5%。所以在是否患病的情况下，该基因里面的位点数量有明显的差别。说明该基因为致病基因。同理可得其它基因的卡方检验，得到它们的渐进显著水平都为小于 5%，因此在是否引起疾病的情况下，该基因里面的位点数量有明显的差别，所检验的基因为致病的基因。

Gene_55* 是否患病 交叉表					
			是否患病		总计
			患病	不患病	
gene_55	rs7368252	计数	158	160	298
		预期计数	148.0	150.0	298.0

	rs7366321	计数	251	261	513
		预期计数	246.5	256.5	513.0
	rs11801490	计数	67	122	189
		预期计数	84.5	104.5	189.0
总计		计数	500	500	1000
		预期计数	500.0	500.0	1000.0

卡方检验			
	值	自由度	渐近显著性 （双向）
皮尔逊卡方	18.435[a]	2	.015
似然比(L)	8.474	2	.014
线性关联	6.831	1	.009
有效个案数	1000		
a. 0 个单元格 (0.0%) 具有的预期计数少于 5。最小预期计数为 94.50。			

4.4 问题 4 的分析与求解

4.4.1 问题的分析与求解

问题四中讨论 10 个相关性状的信息与其相关联的位点，给出的是 10 个性状，然后每个性状有 1000 样本，考虑到 10 个性状和位点之间的关系，可以转化先考虑单个性状和位点之间的关系，也就是转化为了当初的疾病和位点之间的关系，这里的不同之处在于当初疾病前 500 个是 0，后 500 个是 1，这里性状 1000 个样本的 0 和 1 是随机出现的，没有规律，先单个用算法计算出与每个性状相关的位点集合，最后统计存在这 10 个位点集合中共同位点作为整体相关联的性状位点。

4.4.2 数据的采集与处理

采用题目中给定的genotype.dat与multi_phenos.txt文件数据，从multi_phenos.txt文件中1000个样本中10个性状与位点的关系，分析此两文件数据的关联关系，从而分析出疾病最有可能是哪些位点造成的。

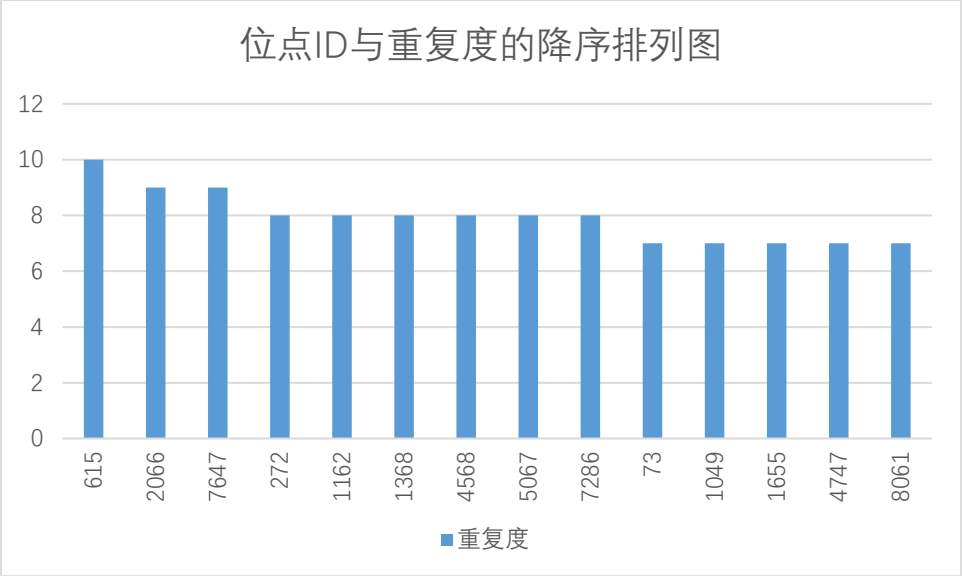
MATLAB核心代码实现如下：

```
%% 筛选出十个权值里的位点的重复项
mark_Q4=zeros(numSamples,10);
repeatted=zeros(1,numLocus);
for i=1:1:numSamples
    for j=1:1:10
        if IndexOfEachGenotype_Q4(i,j)==0
            break;
        end
        repeatted(1,IndexOfEachGenotype_Q4(i,j))=
        repeatted(1,IndexOfEachGenotype_Q4(i,j))+1;
    end
end

%% 计算重复项的权值
[repeatted_result,repeatted_index]=sort(repeatted,'descend');
```

4.4.3 结论

通过求解得出与10个性状相关的位点为：
rs10909903,rs6686587,rs311469,rs3128326,rs608203,
rs10799145,rs10492987,rs7545115,rs2501401,rs4314833,
rs351619,rs770706,rs6678618,rs11247946 。



五、 模型评价与推广

本文建议的疾病与位点以及基因的理论基本模型合理，基本符合实际情况，可广泛应用到疾病的分析求解问题中，在分析位点、基因和疾病的问题时，有独特的用途。但文章中的一些参数是通过理论计算得到的，对于不同的疾病和位点要做不同的研究。

六、 参考文献

[1] 黎成. 基于随机森林和 ReliefF 的致病 SNP 识别方法[D]. 西安：西安电子科技大学，2014.
[2]. 吴水秀，曾庆鹏等. 基于改进 ReliefF 算法的主成分特征提取. 计算机工程，2008.
[3] Md.Mamun Monir,Zhu Jun 等.数量性状位点定位和全基因组关联分析的方法与软件综述.浙江大学学报，2014.
[4] 陈海林. 网络医学中若干关联问题的计算分析[M]. 中南大学信息科学与工程学院. 2013.
[5] 彭倩倩. 群体病例对照研究设计的整体基因关联分析统计推断方法研究[D]. 山东大学. 2009.
[6] 胡艳玲. 复杂性状与基因组多位点的关联分析方法研究[M]. 上海交通大学. 2009.
[7] 杨小红. 关联分析的一般思路-存在问题及发展趋势. 中国农业大学国家玉米改良中心. 2011.
[8] 丁小军. 大规模生物数据信息挖掘算法设计[M]. 中南大学信息科学与工程学院. 2014.
[9] 葛鹏程. 生物学数据挖掘算法的设计与实现[D]. 哈尔滨工业大学, 2005.
[10] 张平. 生物数据多层关联规则挖掘算法的优化设计[D]. 华中科技大学, 2007.

七、 附录

核心代码如下：

(1)降维

```
public  void CalMAF(float yuzhi,String path,String pathid)
{
    init(path);
    Sample head=al.get(0);
    String [] headName=head.getGenetype().split("\\s++");
    Sample tail=al.get(500);
    String [] tailName=head.getGenetype().split("\\s++");
    float MAFa;
    for(int i=0;i<N;i++)//位点
    {

        int rate1=0;
        int rate2=0;
        GeneMAF gm=new GeneMAF();

        //识别标识
        String gene=headName[i];
        char tag1=gene.charAt(0);
         rate1++;
         if(gene.charAt(1)==tag1)
             rate1++;
        for(int j=1;j<500;j++)
        {
            Sample sample=al.get(j);
             gene=sample.getGenetype().split("\\s++")[i];
             if(gene.charAt(0)==tag1)
                 rate1++;
             if(gene.charAt(1)==tag1)
                 rate1++;
        }
        String tai=tailName[i];
        char tag2=tai.charAt(0);
           rate2++;
        if(gene.charAt(1)==tag2)
             rate2++;
        for(int k=501;k<1000;k++)
        {

             Sample sample=al.get(k);
```

```
                gene=sample.getGenetype().split("\\s++")[i];
                if(gene.charAt(0)==tag2)
                    rate2++;
                if(gene.charAt(1)==tag2)
                    rate2++;

            }

            //筛选数据
            if(tag1==tag2) //tag1和tag2表示位点相同
            {

              MAFa=Math.abs((rate1-rate2)/(float)1000);
             if(MAFa>yuzhi)
             {
                 WeiDianId.add(i);
                 System.out.println(MAFa);
             }

            }
             //表示位点不相同
            else
            {
                 MAFa=Math.abs((1000-rate1-rate2)/(float)1000);
                 if(MAFa>yuzhi)
                  {
                      WeiDianId.add(i);
                      System.out.println(MAFa);
                  }
                        }
                   }
                   writeWeidianId(pathid);

        System.out.println("剩余位点:"+ WeiDianId.size());
    }
```

(2) 获取同一类别的近邻 id

```
    public ArrayList<Neighbor> getSameNearstId(Sample training)
    {

        ArrayList<Neighbor> Neighbors=new ArrayList<Neighbor>();
         int id=training.getId()-1;
        // System.out.println("id"+id);
```

```
        if(training.isTag())//表示训练集是有病的状态
        {
            Neighbors = lastCase(training, Neighbors, id);
            //排序按照 weight相关性大小排序
            Collections.sort(Neighbors,comparator);
           // paixu(Neighbors);
        }
        else
        {
            Neighbors = forecase(training, Neighbors, id);
                       //排序按照 weight相关性大小排序
            Collections.sort(Neighbors,comparator);
           // paixu(Neighbors);
          // System.out.println("Neighbors:"+Neighbors.size());

        }
       return Neighbors;
    }
```

(3)获取在不同类别中的近邻idsss

```
    public ArrayList<Neighbor> getDistinctNearstId(Sample training)
    {
        ArrayList<Neighbor> Neighbors=new ArrayList<Neighbor>();
         int id=training.getId()-1;
        // int sameGene=0;//相同的基因组
         if(!training.isTag())//表示训练集是无病的状态
         {
             Neighbors = lastCase(training, Neighbors, id);
             //排序按照 weight相关性大小排序
             Collections.sort(Neighbors,comparator);
            // paixu(Neighbors);
         }
         //有病的状态
         else
         {
            Neighbors = forecase(training, Neighbors, id);
             //排序按照 weight相关性大小排序
             Collections.sort(Neighbors,comparator);
            // paixu(Neighbors);
                    }
        return Neighbors;
    }
```

（4）ReliefF算法

```
    //releifF算法主程序,path是路径，k是控制测试的近邻个数
    public void mainProcedure(String path,int k,String outpath,String
pathimportid)
    {
        init(path);
        readImportweidian(pathimportid);
            //初始化一个位点的数组
        double weight[]=new double[WeiDianId.size()];
        float diffsame;
        float diffdistinct;
        for(int i=0;i<WeiDianId.size();i++)
        {
            weight[i]=0;
                    }
         int e=10;
         int s=0;
         int id;
         int importantid;
        for(int j=0;j<1000;j++)//迭代次数
        {
            //int  id=(int) Math.round(Math.random()* 999);
              //获取样本
              Sample test=al.get(j);
              ArrayList<Neighbor> sameNeighbors=getSameNearstId(test);
              ArrayList<Neighbor>
distinctNeighbors=getDistinctNearstId(test);

              for(int k1=0;k1<WeiDianId.size();k1++)
              {
                  importantid=WeiDianId.get(k1);
                  for(int f=0;f<k;f++)
                  {
              diffsame=diffValue(j,
  sameNeighbors.get(f).getId(),importantid);

     diffdistinct=diffValue(j,distinctNeighbors.get(f).getId(),importantid);
                  weight[k1]=weight[k1]+(diffdistinct-
diffsame)/(double)(N*T);
                  }
              }
              System.out.println(j);
        }
        s       writeTofile(outpath, weight);
```

```
        for(int t=0;t<10;t++)
        {
            System.out.println(SNPname[WeiDianId.get(t)]+":"+weight[t]);
        }

    }
```

（5）在相同类型的样本

```
        private ArrayList<Neighbor> SameCase(Sample training,
ArrayList<Neighbor> Neighbors,
                int id)
         {
            int tag=training.isTag();
            int sameGene=0;//相同的基因组
            String []trainingSplit=training.getGenetype().split("\\s+");
            for(int i=0;i<1000;i++)
             {
               if(i!=id&&biaoshi.get(i)==tag)
               {
                 sameGene=0;
                 Neighbor neighbor=new Neighbor();
                 //获取样本
                 Sample sample=al.get(i);
                 String []sampleSplit=sample.getGenetype().split("\\s+");
            //   System.out.println("samplesplit:"+sampleSplit.length+"id"+i);
                 for(int j=0;j<sampleSplit.length;j++)
                 {
                     if(trainingSplit[j].equals(sampleSplit[j]))
                     {
                         sameGene++;
                     }

                 }
                 //将id和权值加入到邻居中
                 neighbor.setId(i);
                 neighbor.setWeight(sameGene/(float)N);
                 Neighbors.add(neighbor);
               }
             }
            return Neighbors;
        }
```

```
        //在不同类型的样本
                private ArrayList<Neighbor> DifferentCase(Sample training,
ArrayList<Neighbor> Neighbors,
                        int id)
                {
                   int tag=training.isTag();
                   int sameGene=0;//相同的基因组
                   String
[]trainingSplit=training.getGenetype().split("\\s+");

    //System.out.println("trainingSplit:"+trainingSplit.length);
                   for(int i=0;i<1000;i++)
                    {
                      if(i!=id&&biaoshi.get(i)!=tag)
                      {
                        sameGene=0;
                        Neighbor neighbor=new Neighbor();
                        //获取样本
                        Sample sample=al.get(i);
                        String
[]sampleSplit=sample.getGenetype().split("\\s+");
                   //
    System.out.println("samplesplit:"+sampleSplit.length+"id"+i);
                        for(int j=0;j<sampleSplit.length;j++)
                        {
                            if(trainingSplit[j].equals(sampleSplit[j]))
                            {
                                sameGene++;
                            }

                        }
                        //将id和权值加入到邻居中
                        neighbor.setId(i);
                        neighbor.setWeight(sameGene/(float)N);
                        Neighbors.add(neighbor);
                      }
                    }
                   return Neighbors;
                }
```