医保欺诈行为的主动发现

摘　　要

随着我国医保制度推广范围的不断扩大，以及管理上存在的一定程度的疏忽，医保欺诈现象越发严重，造成了医疗资源的分配不公和公众利益的损失。

本文着眼于分析医保欺诈的**三种最常见实施方式**，探讨这些行为所具有的不同特征，对这些手段，分别设计了检测方法，建立了三种不同的数学模型。

模型一针对一张卡在一定时间内反复多**次拿药。**对此我们可以统计出每张卡在医院开药单的时间频率，同时还必须探究开药频率和病人自身属性的关联性，通过比较数据在不同分类水平下的频率直方图，进一步通过**列联表法**检验了不同病人属性下就诊频次之间的独立性是否存在，通过　绘**制盒式图**，通过分位数界定了一部分离群点，作为高度怀疑的对象。

模型二**对于单张处方药费过高进行**判**别。**对于特定类型的病人，医生往往会有对应的开药模式，若某些外在因素相似的病人，在开药模式上呈现出很大的差异性，则有理由怀疑为行为异常者。对此，我们采用了**在线无监督机器学习**，建立了**智**能过滤器模型**，对于数据库中的离散分类数据和连续变量分别采用 **SDLE，SDEM**方法，引入**高斯混合模型**刻画其概率密度，基于新数据点对原有数据分布的影响大小，计算了 **Hellinger Distance 或 Logarithmic Loss**作为得分，得分越高代表该数据点在与其类似的数据点中行为越异常，理论上结合经验阈值可以转化为**有监督机器学习模型**来优化原有模型。

模型三探测一人持多卡配药的情况。因为不同的病人去医院开药的行为通常是相互独立的，如果若干张医保卡在开药行为上存在高度的一致性，则很可能这些医保卡为同一人在使用。为探测这种关联性，采用了 **Eclat 关联规则算法**，对庞大的数据库按时序进行数据挖掘，**提取高度频繁项集**，作为怀疑的对象。

我们队以上模型都基于现有数据库进行了模型仿真，对结果进行了评价，得到了合理的结果，并对模型的进一步完善提出了展望。

关键词 无监督机器学习, 列联表, Hellinger 距离, 对数损失, Eclat, 频繁项集

1 问题重述

随着经济社会的发展，各种医疗保险的覆盖范围越来越广泛，同时医保诈骗也越来越常见。骗保人常用的手段包括：拿着别人的医保卡配药，在不同的医院和医生处重复配药等。题目要求我们根据附件中的数据，建立模型求解下列问题：

1. 单张处方药费过高的判定
2. 一张卡一定时间反复多次拿药的识别和判定
3. 一人持多卡同时重复配药的识别和判定

2 问题分析

问题一：一张卡在一定时间内反复多次拿药的识别

病人的拿药次数可能受多种因素影响，如何在多种可能因素的影响下，制定合理的可以量化的标准，识别出这些拿药次数异常的点，涉及到离群点的检测，在这之前我们还需要对可能的影响因素进行比较分析。

问题二：单张处方药费过高的判别

注意到医生开药一般是有一个既定模式的，这个模式可能会受到病人本身、药物特性的影响，寻找单张处方药费过高的例子，就是要寻找离群点。然而本题中并未标识出已经确定的欺诈个例，因此我们需要采用无监督的机器学习方式判别离群点。

问题三：一人持多卡实施欺诈

在现实生活中，一人持多卡实施诈骗是一种常见的手段。本文中，我们利用数据挖掘中的关联规则，找寻到经常同时出现的卡号，以此作为检测的依据。

3 模型基本假设

1. 一个人只能合法持有一张医保卡；
2. 骗保人只是所有参保人群中很小的一部分。

4 模型的建立与求解

4.1 模型一：一张卡在一定时间内反复多次拿药的识别

根据题意，在一定时间内反复多次拿药的同一张卡，有较大的医保欺诈嫌疑。然而，对于不同的个体，“多次”的具体标准并不相同。例如：婴儿一般免疫力较差，普遍就诊拿药的次数会多一些，然而对于正值青壮年的男士来说，拿药次数普遍会比较低。也就是说，我们应根据病人的个体特点，制定一个拿药次数“过高”的判断标准。自然地，我们会联想到: 病人的经济状况、健康状况、医生的习惯等

因素会对拿药频率产生一定影响。然而由于附件中所给信息，尤其是病人经济状况、病史等方面信息的匮乏，在本例中，我们能够获取的对病人拿药频率可能会产生影响的因素具体地包括：年龄、性别。附件中给出的记录时长为一个月，我们首先衡量年龄和性别对一个月内拿药次数的影响，再据此判断拿药频次过高的离群点，这些点就是骗保的重点怀疑对象。

附件“病人资料”中的每一行为一个医保卡号对应的病人信息，医保卡号对应为 rowid，合并“费用明细表”中相同时间的订单，统计出每张卡的就诊频次。

符号定义

Q_3: 就诊频次的上四分位数
Q_1: 就诊频次的下四分位数
ΔQ: 就诊频次上下四分位数的间距

4.1.1 年龄对于一个月内就诊频次的影响

首先，我们将人按年龄分为 8 个离散类：0 到 70 岁，每 10 岁分为一类；70 岁以上为一类。对不同离散类人群的就诊次数进行统计，我们可以得到不同年龄段的人的就诊频次分布直方图如图 1。

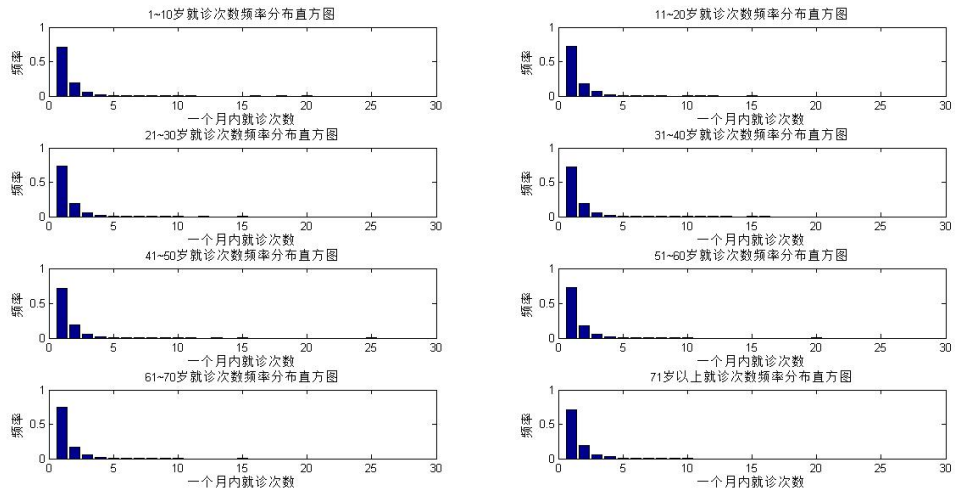

图 1: 年龄频率图

从上述图表可以看出，不同年龄段的人就诊频次有很相似的分布。为了验证年龄与就诊频次独立与否，列联表独立检验法是数理统计中对两个特性是否独立的一种常用检验方法。这里我们就利用列联表法对年龄和就诊频次的独立性进行检验。

首先根据已有所给数据可得表 1

令原假设为：年龄与就诊频次独立。取检验统计量为

$$\chi^2 = \sum_{i=1}^{r}\sum_{j=1}^{s} \frac{(n_{ij} - n\hat{p}_{i\cdot}\hat{p}_{\cdot j})^2}{n\hat{p}_{i\cdot}\hat{p}_{\cdot j}}$$

其中 $\hat{p}_{i\cdot} = \frac{n_{i\cdot}}{n}$,$\hat{p}_{\cdot j} = \frac{n_{\cdot j}}{n}$

表 1: 各年龄段实际频数表

年龄段	1 次	2 次	3 次	4 次	5 次	6 次	7 次及以上
0-10 岁	7607	2227	687	301	136	81	101
11-20 岁	2218	507	164	69	30	13	10
21-30 岁	8440	1955	557	152	64	22	14
31-40 岁	10402	2696	795	311	98	36	25
41-50 岁	6353	1531	468	175	62	25	39
51-60 岁	3374	886	249	120	42	12	20
61-70 岁	2065	544	182	61	23	11	10
70 岁以上	1339	413	147	86	23	19	16

拒绝域为 $\{\chi^2 > \chi^2_\alpha((r-1)(s-1))\}$，现将年龄分为 8 类，就诊频次分为 7 类。故此处 $r = 8, s = 7$，则拒绝域为 $\{\chi^2 > \chi^2_\alpha(42)\}$

根据 $n_{ij} = n\hat{p}_{i\cdot}\hat{p}_{\cdot j}$ 构造另一个列联表如表 2

表 2: 各年龄段理论频数表

年龄段	1 次	2 次	3 次	4 次	5 次	6 次	7 次及以上
0-10 岁	8026.3	2066.007	623.8922	244.8331	91.78839	42.05368	45.12609
10-20 岁	2169.406	558.4153	168.6301	66.17525	24.80923	11.36657	12.19701
21-30 岁	8072.411	2077.876	627.4765	246.2396	92.31572	42.29528	45.38534
31-40 岁	10348.45	2663.739	804.3953	315.6676	118.3444	54.22055	58.18187
41-50 岁	6234.432	1604.772	484.6086	190.1742	71.29667	32.66521	35.05171
51-60 岁	3388.482	872.211	263.3901	103.3617	38.75052	17.7539	19.05099
61-70 岁	2086.55	537.0876	162.1896	63.6478	23.86169	10.93245	11.73116
70 以上	1471.969	378.8916	114.4176	44.90071	16.83336	7.712358	8.275817

由此可得表 3

表 3: 各年龄段 χ^2 值表

年龄段	1 次	2 次	3 次	4 次	5 次	6 次	7 次及以上
0-10 岁	21.90451	12.54534	6.383459	12.88521	21.29535	36.06857	69.18156
11-20 岁	1.088468	4.733998	0.12713	0.120577	1.08605	0.234731	0.39574
21-30 岁	16.73868	7.266351	7.915741	36.06694	8.685196	9.738636	21.70392
31-40 岁	0.277097	0.390707	0.109738	0.069017	3.497376	6.122929	18.92405
41-50 岁	2.254962	3.391311	0.569211	1.210763	1.212233	1.798717	0.444742
51-60 岁	0.061893	0.217994	0.786186	2.678278	0.27249	1.864794	0.047274
61-70 岁	0.222563	0.088963	2.419717	0.110151	0.031117	0.000417	0.255467
70 岁以上	12.01157	3.070495	9.278418	37.6197	2.259049	16.52035	7.209317

计算可得 $\chi^2 = 433.4652$，故该问题的 p-value 为 $1 - \chi^2(433.4652, 42) = 0$，其中 $\chi^2(\cdot, 42)$ 表示自由度为 42 的卡方分布。

因此几乎一定可以认为，原假设不成立，即年龄与就诊频次是非独立的，就诊频次受到年龄的影响。

4.1.2 性别对于一个月内就诊频次的影响

对于性别因素，分析方法与年龄因素完全相同。男性和女性在一个月内的就诊次数分布直方图如 2。

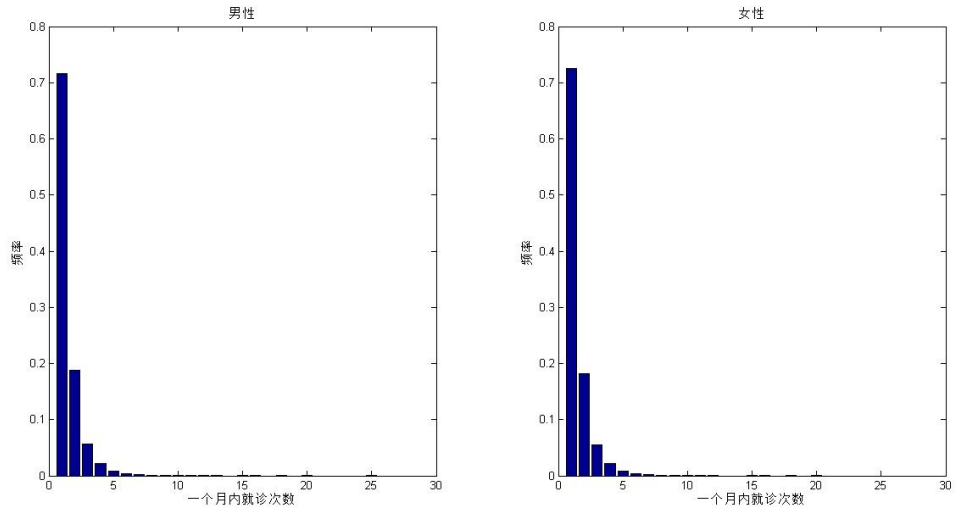

图 2: 性别直方图

可见不同性别的单月就诊频次的分布大致相似。同样地，我们猜想：性别与就诊频次是独立的。与上面考虑年龄与就诊频次的影响一样，我们采用列联表独立性检验法对原假设进行检验。

首先根据题中所给数据可得表 4

表 4: 不同性别实际频数表

性别	1 次	2 次	3 次	4 次	5 次	6 次	7 次及以上
男	23692	6161	1861	719	269	107	104
女	18095	4598	1388	556	209	112	131

原假设：年龄与就诊频次独立。

检验统计量与上一节相同，现将性别分为 2 类（在题中所给数据中，实际上性别被分成了 4 类，其中类别为 3 和 4 的各有 4 人和 7 人，这里考虑到可能存在的数据录入错误或病人性别不明等情况，在此处将这几个人的数据去除），就诊频次分为 7 类。故此处 $r = 2, s = 7$，则拒绝域为 $\{\chi^2 > \chi^2_\alpha(6)\}$

构造理论频数的列联表如表 5

表 5: 不同性别理论频数表

性别	1 次	2 次	3 次	4 次	5 次	6 次	7 次及以上
男	23711.86	6105.151	1843.632	723.4936	271.2392	124.2707	133.3498
女	18075.14	4653.849	1405.368	551.5064	206.7608	94.72934	101.6502

χ^2 值如表 6

表 6: 不同性别 χ^2 值表

性别	1 次	2 次	3 次	4 次	5 次	6 次	7 次及以上
男	0.01664	0.510899	0.163618	0.027909	0.018485	2.400211	6.459782
女	0.02183	0.670222	0.214642	0.036613	0.02425	3.148716	8.474264

计算可得 $\chi^2 = 22.1881$，故该问题的 p-value 为 $1 - \chi^2(22.1881, 6) = 0.0011$，可见即使在显著水平为 0.005 的假设检验上，仍然可以拒绝原假设，即性别与就诊频次存在一定的关联性。

4.1.3 识别频率过高的离群点

由于盒式图可以较好地显示数据分散的情况，并给出全局离群点，因此在该模型中可以使用盒式图来确定就诊频次异常的点。根据上面的分析，就诊频次与年龄、性别都存在一定的关联性，因此下面对不同的年龄、性别组成的离散类分别进行离群点的判定。将盒式图上、下四分位间距设为 $Q_3 - Q_1 = \Delta Q$，则可以定义最大最小值区间为 $[Q_3 + 1.5\Delta Q, Q_1 - 1.5\Delta Q]$, 区间之外的点则认为是就诊频次过高的点，即骗保的重点怀疑点，在图 3中用十字标出，此处仅绘制了部分年龄段和性别的盒式图作为示意。

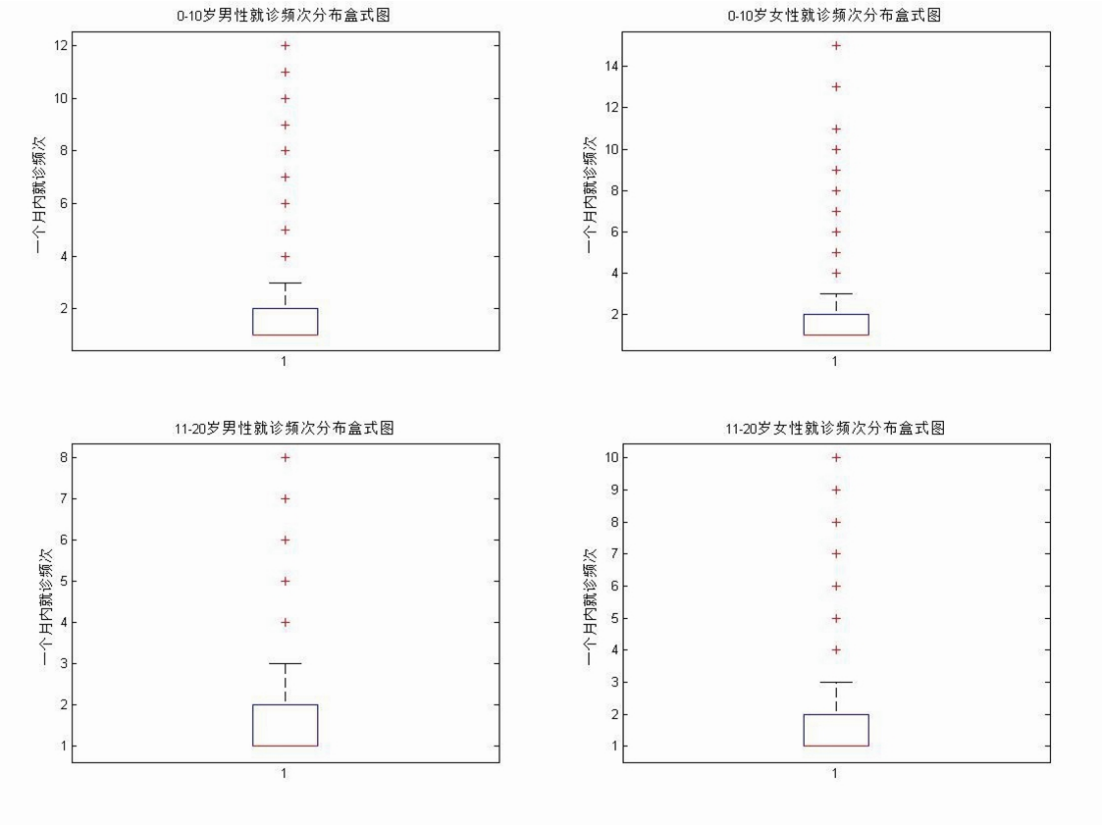

图 3: 就诊频率盒式图

表 7给出了综合各年龄段和性别中单月就诊频次最高的一部分人的重点怀疑对

象名单，实际应用时，可以累积每个病人就诊的次数，当该次数超过其所属群体的阈值后引起怀疑。

表 7: 单月高就诊频次怀疑名单

医保卡号	就诊频率
163667	25
452048	20
230472	20
309748	18
523605	18
160429	18
423611	16
210035	16
338362	16
178495	15
242020	15
363949	15
...	...

4.1.4 模型的优缺点及展望

优点：

- 模型简单，可以快速找到具有较大骗保嫌疑的医保卡
- 分析思路明确，考虑年龄、性别的影响因素
- 具有拓展性，对于不同影响因素的分析都可按照模型中提出的方式进行

缺点：

- 考虑的因素较少，没有挖掘数据之间隐藏的联系，比如就诊科室对于就诊频率可能的影响
- 在考虑的影响因素较多时，按照上述方法进行处理步骤较为繁琐

4.2 模型二：单张处方药费过高的识别

同样地，在识别单张处方药费过高的个例时，我们首先要解决的问题是如何将“过高”这一抽象概念转化为可以量化的标准，而同时这一标准又可能受到多种因素影响。自然地，我们会认为，在某些相同的条件下，医生开药是服从一个特定的模式的。而如何在大量数据中发现这一模式，仅通过人工的判断和分析是困难的。识别单张处方药费过高的个例，实际就是要判断离群点。

离群点的判断方法已有很多种，如 K-means，离差分析，或比较成熟的机器学习算法。然而本问题的特殊性在于：数据库不存在已知的标记（客户是否曾经骗

保），不适用于目前流行的有监督的机器学习算法；且数据中存在不可忽略的非数值的分量，一般的数值聚类解法往往难以解决。

基于以上特点，我们选择建立一种无监督的离群点探测的机器学习模型——智能过滤机模型，并定义了基于 Hellinger Distance 或 Logarithmic Loss 的判定离群点的指标，而且可以同时接受数值化和非数值化的参数。事实上，这个模型完全可以应用在更复杂的欺诈预测模型中。

4.2.1 基本思路

医院的总账单中每一行可以代表一个数据点，对于这样一个数据点，记它的属性集为 $(\vec{x}, \vec{y})$, 其中 $\vec{x}$ 表示离散变量类型的属性所构成向量，$\vec{y}$ 表示连续变量类型的属性所构成的向量，它们可以具有任意的长度和取值范围。对于这两种向量，我们分别建立它们的概率模型，再进行整合。

为了表示离散类别的数据概率分布，针对离散的参数而定义的“细胞”这个概念如下：

定义 1 对于 k 个交集为空的离散类别的数据 $A^1, A^2, ..., A^k$, 假设 A^i 中的元素可以有 v_i 个不同的取值（分类值）, 即 $A^i = \bigcup_{j=1}^{v_i} A_i^j$ 且 $\forall i, \forall j, k \in \{1, 2, ..., v_i\}$, 我们称 $A_{j1}^{(1)} \times A_{j2}^{(2)} \times ... \times A_{jn}^{(n)}$ 为“细胞 $(j_1, j_2, ..., j_n)$”。

每输入一个新的数据点，我们都先判断它属于哪个细胞，然后利用随序列衰减的拉普拉斯估计（SDLE，Sequentially Discounting Laplace Estimation）算法更新各离散类的概率，记为 $p(x)$。

然后对各个连续数据类，因为影响因素非常复杂，不易于用特定的函数模型来刻画，不妨记其条件概率密度为 $p(y|x)$，假设上面的数据点服从有限维的高斯混合分布，引入有限维高斯混合模型，利用随序列衰减的期望最大化（SDEM，Sequentially Discounting Expectation and Maximizing）算法，对有限维的高斯混合分布进行更新以逼近 $p(y|x)$，从而可以得到 $p(x, y) = p(x)p(y|x)$，至于具体的混合分布个数，可以根据训练过程中的收敛效果来进行增减。下面两节给出具体的算法推导及过程。

4.2.2 SDLE

定义折扣参数为 r_h，每次新输入一个数据之后，之前的 1 衰减为 $1 - r_h$

初始化：对于任意的细胞 $(j_1, j_2, ..., j_n)$, 令 $T(j_1, j_2, ..., j_n) = 0$

记输入第 t 个样本后，细胞 $(j_1, j_2, ..., j_n)$ 对应的 T 为 $T_t(j_1, j_2, ..., j_n)$

每输入一个新的数据 $x_t = (x_1, x_2, ..., x_n)$, 对每一个细胞 $(j_1, j_2, ..., j_n)$ 进行更新, 具体过程如下: 若 $x_t \in A_{j1}^{(1)} \times A_{j2}^{(2)} \times ... \times A_{jn}^{(n)}$, 则 $T_t(j_1, j_2, ..., j_n) = (1 - r_h)T_{t-1}(j_1, j_2, ..., j_n) + 1$

否则，$T_t(j_1, j_2, ..., j_n) = (1 - r_h)T_{t-1}(j_1, j_2, ..., j_n)$

对应的, 细胞 $(j_1, j_2, ..., j_n)$ 对应的概率也应更新为

$$p_t(j_1, j_2, ..., j_n) = \frac{T_t(j_1, j_2, ..., j_n)}{\frac{1-(1-r_h)^t}{r_h}}$$

其中分母部分的 $\frac{1-(1-r_h)^t}{r_h}$ 可以通过如下计算得到：由于每经过一次更新，之前出现的 1 都衰减为 $1 - r_h$，因此，在第 t 次更新之后，第一次输入的数据衰减为

$(1-r_h)^{t-1}$，第二次输入的数据衰减为 $(1-r_h)^{t-2}$，.....，第 (t-1) 次输入的数据衰减为 $1-r_h$，第 t 次输入的数据不衰减，因此实际包含的总次数为 $1+(1-r_h)+...+(1-r_h)^{t-1}=\frac{1-(1-r_h)^t}{r_h}$

在每个细胞中, 各个元素出现的概率相等，即

$$\vec{x}\in A_{j1}^{(1)}\times A_{j2}^{(2)}\times...\times A_{jn}^{(n)}, p_t(\vec{x})=\frac{p_t(j_1,j_2,...,j_n)}{\left|A_{j1}^{(1)}\right|\times\left|A_{j2}^{(2)}\right|\times...\times\left|A_{jn}^{(n)}\right|}$$

其中 $\left|A_{j_i}^{(i)}\right|$ 表示集合 $A_{j_i}^{(i)}$ 所含元素个数

以上步骤完成了一次更新，只要令 $t=t+1$，重复上述步骤，即可完成下一次更新

4.2.3 SDEM

假设数据服从混合高斯分布，即：数据可以看作是从数个高斯分布中生成出来的。这里我们假设数据分布的密度函数满足以下关系式：

$$p(y|x)=\sum_{i=1}^{k}\pi_i p(y|\mu_i,\Sigma_i)$$

其中

- π_i 为任意一个数据点落在第 i 个高斯分布中的概率
- μ_i 为第 i 个高斯分布的均值向量
- Σ_i 为第 i 个高斯分布的协方差矩阵
- k 为生成数据的高斯分布的个数

定义 $S_i^{(s)}$ 如下：(其中 s 为已经录入的数据点的个数)

定义 2

$$\begin{aligned}S_i^{(s)}&:=(\pi_i^{(s)},\mu_i^{(s)},\Sigma_i^{(s)})\\&=(p(\mu_i^{(s)},\Sigma_i^{(s)}),E(y|\mu_i^{(s)},\Sigma_i^{(s)})p(\mu_i^{(s)},\Sigma_i^{(s)}),E(yy^T|\mu_i^{(s)},\Sigma_i^{(s)})p(\mu_i^{(s)},\Sigma_i^{(s)})),i\in\{1,2,...,k\}\end{aligned}$$

上述参量可以做如下直观理解：根据上述定义，每输入一个新的数据，就可以对 $S_i^{(s)}$ 进行一次更新。为了使参数在训练过程中具有比较可靠的收敛性，我们随机给定 $\mu_i^{(0)}$ 和 $\Sigma_i^{(0)}$，并假设初始时 $\pi_i^{(0)}=\frac{1}{k}$（其中 k 为高斯分布的个数），即任意给定一个数据点，它由各个高斯分布生成的概率相同，即 $S_i^{(0)}=(\frac{1}{k},\mu_i^{(0)},\Sigma_i^{(0)})$

每输入一个新的数据点，就将 $S_i^{(s-1)}$ 更新到 $S_i^{(s)}$

更新的具体步骤如下：

(注意我们用 $p(\mu_i,\Sigma_i|y_u)$ 表示对于给定的 y_u，它落在第 i 个高斯分布的概率)

(1) 将 $\pi_i^{(s-1)}$ 更新到 $\pi_i^{(s)}, i\in\{1,2,...,k\}$

$$\begin{aligned}\pi_i^{(s)} = p(\mu_i^{(s)}, \Sigma_i^{(s)}) &= \frac{1}{s}\sum_{u=1}^{s} p(\mu_i^{(s-1)}, \Sigma_i^{(s-1)}|y_u) \\ &= \frac{1}{s}\sum_{u=1}^{s-1} p(\mu_i^{(s-1)}, \Sigma_i^{(s-1)}|y_u) + \frac{1}{s}p(\mu_i^{(s-1)}, \Sigma_i^{(s-1)}|y_s) \\ &= \frac{s-1}{s}p(\mu_i^{(s-1)}, \Sigma_i^{(s-1)}) + \frac{1}{s}p(\mu_i^{(s-1)}, \Sigma_i^{(s-1)}|y_s)\end{aligned}$$

这里，同样地，我们利用折扣参数 r 来确定更新前后的数据所占的权重。具体地，我们规定：

$$\pi_i^{(s)} = p(\mu_i^{(s)}, \Sigma_i^{(s)}) = (1-r)p(\mu_i^{(s-1)}, \Sigma_i^{(s-1)}) + rp(\mu_i^{(s-1)}, \Sigma_i^{(s-1)}|y_s)(*)$$

注意到，在之前的数学推导中，$\frac{1}{s}$ 是一个随着输入数据量而变化的数值，然而在数据挖掘的过程中，我们固定 r 的值，这是因为，在处理大量数据时，新输入的数据往往更能体现当前的情况，而以前的数据，则会因为一些其他的因素而不适用当前环境，但是在之前的数学推导中，每个数据点，无论录入时间先后，都会对当前的概率分布产生相同的影响，这与我们之前所述是矛盾的。因此这里我们固定 r 的值，并将它命名为折扣参数，它的值越小，表明以前的数据对当前的影响越大。

$(*)$ 式中的 $p(\mu_i^{(s-1)}, \Sigma_i^{(s-1)}|y_s)$ 可利用以下公式得到：

$$p(\mu_i^{(s-1)}, \Sigma_i^{(s-1)}|y_s) = \frac{p(y_s|\mu_i^{(s-1)}, \Sigma_i^{(s-1)})p(\mu_i^{(s-1)}, \Sigma_i^{(s-1)})}{\sum_{i=1}^{k} p(y_s|\mu_i^{(s-1)}, \Sigma_i^{(s-1)})p(\mu_i^{(s-1)}, \Sigma_i^{(s-1)})}$$

（其中 k 为高斯分布个数）这里，为了提高 π_i 的估计的稳定性，我们引进一个新的参数 α，于是

$$p(\mu_i^{(s-1)}, \Sigma_i^{(s-1)}|y_s) = (1-\alpha r)\frac{p(y_s|\mu_i^{(s-1)}, \Sigma_i^{(s-1)})p(\mu_i^{(s-1)}, \Sigma_i^{(s-1)})}{\sum_{i=1}^{k} p(y_s|\mu_i^{(s-1)}, \Sigma_i^{(s-1)})p(\mu_i^{(s-1)}, \Sigma_i^{(s-1)})} + \alpha r\frac{1}{k}$$

将得到的 $p(\mu_i^{(s-1)}, \Sigma_i^{(s-1)}|y_s)$ 代入 $(*)$ 式，即可得到 $\pi_i^{(s)} = p(\mu_i^{(s)}, \Sigma_i^{(s)})$

(2) 对 $\bar{\mu}_i^{(s-1)}, \bar{\Sigma}_i^{(s-1)}$ 进行更新

同样的，这里我们利用折扣参数 r 来衡量更新前后的数据所占的权重，具体公式如下：

$$\begin{aligned}\bar{\mu}_i^{(s)} &= (1-r)\bar{\mu}_i^{(s-1)} + rp(\mu_i^{(s-1)}, \Sigma_i^{(s-1)}|y_s)y_s \\ \bar{\Sigma}_i^{(s)} &= (1-r)\bar{\Sigma}_i^{(s-1)} + rp(\mu_i^{(s-1)}, \Sigma_i^{(s-1)}|y_s)y_s y_s^T\end{aligned}$$

(3) 对 $\mu_i^{(s-1)}, \Sigma_i^{(s-1)}$ 进行更新

这里我们利用（1）中得到的 $p(\mu_i^{(s)}, \Sigma_i^{(s)})$ 和下列公式：

$$\begin{aligned}
\mu_i^{(s)} &= E(y|\mu_i^{(s)}, \Sigma_i^{(s)}) \\
&= \frac{E(y|\mu_i^{(s)}, \Sigma_i^{(s)})p(\mu_i^{(s)}, \Sigma_i^{(s)})}{p(\mu_i^{(s)}, \Sigma_i^{(s)})} \\
&= \frac{\bar{\mu}_i^{(s)}}{\pi_i^{(s)}} \\
\Sigma_i^{(s)} &= E(yy^T|\mu_i^{(s)}, \Sigma_i^{(s)}) - E(y|\mu_i^{(s)}, \Sigma_i^{(s)})E^T(y|\mu_i^{(s)}, \Sigma_i^{(s)}) \\
&= \frac{E(yy^T|\mu_i^{(s)}, \Sigma_i^{(s)})p(\mu_i^{(s)}, \Sigma_i^{(s)})}{p(\mu_i^{(s)}, \Sigma_i^{(s)})} - \mu_i^{(s)}(\mu_i^{(s)})^T \\
&= \frac{\bar{\Sigma}_i^{(s)}}{\pi_i^{(s)}} - \mu_i^{(s)}(\mu_i^{(s)})^T
\end{aligned}$$

即可得到更新后的均值向量 $\mu_i^{(s)}$ 和协方差矩阵 $\Sigma_i^{(s)}$

重复上述步骤 $(1)(2)(3)$，即可实现对所有数据点的录入和 $S_i^{(s)}$ 的更新

4.2.4 离群点的判断——基于概率分布的衡量

我们可以使用 Hellinger Distance 来度量输入一个新的数据点前后两个概率分布的相似度，并以此作为判定离群点的标准。具体表达式如下：

$$Score_H(x_s, y_s) = \frac{1}{r^2}\sum_x \int (\sqrt{p^s(x,y)} - \sqrt{p^{(s-1)}(x,y)})^2 dy$$

这里 $p^s(x,y)$ 为在录入 t 个数据后 (x,y) 的联合分布函数，即：

$$p^s(x,y) = p(x)p(y|x) = p(x)p(y|\mu_i^{(s)}, \Sigma_i^{(s)})$$

r 为之前所定义的折扣参数。显然，$Score_H(x_s, y_s)$ 越大，该数据点的离群程度越高。

然而，上述公式虽然合理，在低维情形下也能推出解析形式的结果，但计算过程复杂，不利于推广，若使用数值方法来计算，会有较高的计算代价，因此不妨转到信息论的角度，定义 Logarithmic Loss 为：

$$Score_L(x_s, y_s) = -\log\, p^{s-1}(x_s) - \log\, p^{s-1}(y_s|x_s)$$

可以理解为假设当前的数据是由之前的概率密度函数所生成的情况下，编码新的数据所需要的码长（对数底数取为 2）。显然，仍然是当 $Score_L(x_s, y_s)$ 越大时，离群程度越高，而且，计算的复杂度得到了显著降低，而且可以轻松推广到高维情形。

在本题中，医院的药单明细具有非常丰富的信息，但在此问题中我们仅需要其中一部分。显然，账单编号，病人编号等编号类信息与问题的求解无关，可以去除。而医嘱子类和开药科室具有较强的关联性，也可以只保留其一。因为模型是对时序具有适应性，我们选取 3 个离散参量，分别为：年龄段、医嘱子类、核算分

表 8: Logarithmic Loss 高低分表

卡号	总价	Logarithmic Loss	训练数量
682554	1163.77	295.9876	4623
405032	1884.3	282.0778	2965
477945	1674.81	176.0137	3671
463011	1884.3	160.4927	2966
608684	1860.9	150.6019	3672
628287	1302.63	146.4362	3670
367970	243.04	124.9444	9316
642467	487.2	113.5794	3407
628287	1860.9	110.058	3673
220040	393.75	108.6868	9455
524683	706.2	107.5304	1130
173602	262.5	98.7646	3399
217527	1884.3	98.6519	3713
650134	3645.2	95.9136	4551
397488	1884.3	89.5151	2606
...	...	...	...
679227	0.16	0.2198	1014
677213	0.16	0.2192	1013
680128	0.16	0.2192	1015
690797	0.16	0.219	1020
691320	0.16	0.2182	1021
695317	0.16	0.2178	1022
695424	0.16	0.2168	1023
695424	0.16	0.2158	1024

类，1 个连续参量，为药品总价。也就是说，我们认为根据年龄段、医嘱子类、核算分类就可确定医生开药的模式。我们依据上述原理和算法，对附件中所给的药单明细进行了录入。并对每一个药单根据 Logarithmic Loss 定义其得分值。表 8仅列出得分值最高和最低的部分数据。

实际上某些得分值较高的点，可能是由于较先录入，训练样本不足，其得分并不能代表其真实的离群程度。因此往往需要等待模型经过一段时间的训练才能认为算得的得分是有代表性的，为提高准确度可以随机抽取样本先行训练，训练样本的数量应该远大于总细胞个数，此处暂取 1000 为阈值，训练次数不足的数据不加以显示。

4.2.5 算法的时空复杂度估计

智能过滤的算法主要的时间用于更新参数和计算得分，对于单行数据，更新参数的时间复杂度为 $O(KM)$，计算得分为 $O(Kd)$，其中 K 为混合高斯分布的个数，M 为离散域的细胞数量，d 为数据的维度。容易得出，总的时间复杂度为 $O(TK(M+d))$，其中 T 为数据的行数，在我们的求解中，T 为 289000; 而经过反

复对比，K 的值取为 5 就已能使结果具有很好的收敛性; 根据数据的特性，M 值为 528，d 为 1。

空间上因为医疗机构必然具有自己的数据库系统，此处仅考虑算法本身的空间消耗，若不需要记录所有训练参数，则只需记录当前最新的离散和连续域参数，空间复杂度为 $O(K(M+d))$，通常不构成负担。

因此总的算法效率可以得到保证，除开数据库的存储开销，对于大多数的计算机，都能在数秒之内得到全部结果，若使用于在线情形，输入一行新的药单数据后，无需等待就能看到结果，并及时采取措施。

4.2.6 模型的优缺点

优点：

本模型是一个非监督的监测模型：无需对真实诈骗案例的先验信息，就可以检测出存在诈骗嫌疑的卡号。同时我们的模型是实时在线的：任意输入一个数据，就可以立即给出这个数据的得分值，从而判定其离群程度，推断其诈骗嫌疑，相比于必须得到完整数据才能运行的离线有监督算法，具有更高的实用性。

缺点：

正因为本模型是非监督的，其结果的可靠性尚未得到验证。对于一些极端的情形，例如若某地出现大面积的骗保行为，则此种方法不一定能很好地检测出涉嫌骗保的卡号。此时若能够结合有监督的机器学习方式，对一部分结果进行标记来划分得分值的阈值，则监测效果会更佳。

4.3 模型三：单人使用多张医保卡骗保行为的识别

骗保人的常用手段之一是一是拿着别人的医保卡配药，即多张医保卡经常在同一时间同时就诊。该模型旨在识别这种骗保行为，即找出频繁出现的医保卡组合。为此提出基于 Eclat 算法的多卡骗保模式识别。

4.3.1 Eclat 算法

Eclat 算法是关联化规则中的一个重要算法，用于从大量数据中挖掘出有价值的数据项之间的相关关系，一个经典实例是购物篮分析，用以发掘客户的购买习惯。而在本例中，类似地，一个人可能同时使用多张医保卡进行诈骗，具体地，这就体现为某些卡经常会同时出现。本模型使用的就是基于 Eclat 的算法，用以挖掘骗保人使用多张医保卡骗保的行为模式。

为了方便讨论，首先给出如下几个定义：

频繁项集 (Frequent Pattern, FP)K：医保卡集合 $K=\{K_1,K_2,...,K_p\}$

频繁项集项数 (模式长度):$\|K\|$

模式空间 D：就诊日期的集合 $D=d_1,d_2,...,d_q$

支持度：support=$\frac{\|D\|}{31}$ (其中 31 为一个月的天数)

多卡骗保模式：$\forall K_i\in K,\forall d_j\in D, Value(K_i,d_j)=1$，且 $\|K\|\geq 2$，μ 大于给定阈值的模式

注：以上范数定义为向量的维数

Eclat 的算法思想是由频繁 k 项集求交集，生成候选 $k+1$ 项集。对候选 $k+1$ 项集做裁剪，去掉小于给定支持度阈值的项集，生成频繁 $k+1$ 项集，再求交集生成候选 $k+2$ 项集。如此迭代，直到项集归一。

为方便理解 Eclat 算法思想，我们做出如下证明。

定理 1 每个 $k+1$ 项的频繁项集可以由两个前 $k-1$ 项都相同的 k 项频繁项集经过或运算生成。

证明 1 假设所生成的 $k+1$ 项的频繁项集的支持度为 a，阈值限制为 b，则有 $a \geq b$。该 $k+1$ 项中的前 k 项的集合 $C1$ 的支持度显然大于等于 a，同样前 $k-1$ 项和最后一项组成的集合 $C2$ 支持度显然也大于等于 a。易得，$C1, C2$ 均为频繁项集，且这两个集合经过或运算可以得到上述的 $k+1$ 项的频繁项集。

因此可以通过已知所有的 k 项频繁项集找到所有的 $k+1$ 项频繁项集。

由于需要对模式空间求交，因此将数据 Eclat 算法采用如表 9的垂直数据格式。该算法的思想类似于广度优先搜索，如图 4所示，以 a,b,c,d 作为模式对象的模式

表 9: Eclat 数据格式

对象 < 医保卡 >	模式空间 < 就诊日期 >
K_1	$\{d_1, ...\}$
K_2	$\{d_1, d_2, ...\}$
...	...
K_p	$\{d_3, d_5, ...\}$

识别。

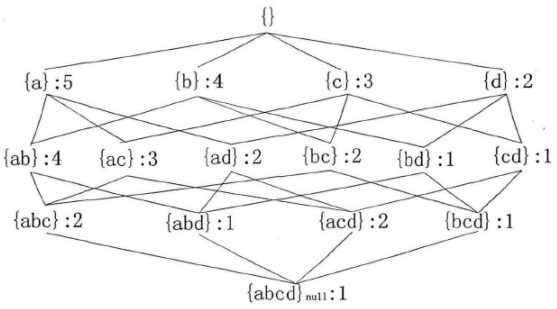

图 4: 算法过程

算法的伪代码如下：

```
ECLAT(FP, SUPPORT:s)
 1  for P_i ∈ FP
 2       do
 3          FP_i = {}
 4          for P_j ∈ FP, j > i
 5               do
 6                  P_ij = P_i ∪ P_j
 7                  tidset(P_ij)=tidset(P_i)∩ tidset(P_j)
 8                  support(P_ij)=|tidset(P_ij)|
 9                  if (support(P_ij)≥ s)
10                     then Add P_ij to FP_i
11          Eclat(FP_i,s)
```

4.3.2 多卡骗保模式识别

在多卡骗保的模式识别中，首先得到支持度大于给定阈值的所有频繁项集（即同时出现频繁的医保卡组合），其次对于得到的组合进行合并子集处理，找到最大长度骗保模式。

所谓“同时出现”，实际上是指在较短时间内均出现，考虑到现实中使用多张卡进行骗保的人一般不会在同一天过于频繁的刷卡（如果刷卡频率过大则会被模型一检测出），因此我们将较短时间定义为一天。

在 Eclat 算法中，影响算法复杂度的另一项重要因素是支持度阈值，阈值越大则剪裁时就能够排除更多的频繁项集从而加快算法的运行速度。根据模型一的结果，单月就诊频率为 4 以上的人就有理由被怀疑为骗保人，由于使用多张卡骗保，相同收益的情况下可以减少就诊频次，同时考虑到算法复杂度，在检查骗保力度较大时推荐将支持度阈值设为 0.1，即每月有三天都同时出现的医保卡组合可以被列入多卡骗保的怀疑对象，而检查骗保力度较小时推荐将支持度阈值设为 0.15，这样可以大大缩减 Eclat 算法输出的并未进行子集合并的医保卡号组合（从 199502 组到 1563 组）。表 10是我们按照支持度阈值 0.15 得出的结果中取按支持度由高到低排列的部分结果。对于找到的频繁项集进行可视化处理如图 5所示。其中圆圈的大小表示支持度的高低。

观察表 10不难发现，第 2、4、5 组医保卡组合是第 3 组医保卡组合的子集，其中第 2 组的支持度要高于第 3、4、5 组，很可能是一个人可以比较固定地使用编号为 [223085,523612] 的医保卡，而编号为 163696 的医保卡使用频率稍微低一些。然而这三张卡都是这个人有能力（但并不一定）使用的医保卡。因此在上述 16 组数据的范围内，第 3 组卡号组合为一种最大长度骗保模式，不论是它本身还是它的子集出现，都很有可能是一个骗保者在操作，即单人使用多张卡骗保，因此我们只需要知道第 3 组的卡号组合而不必保留第 2、4、5 组的卡号组合。

由于 $k+1$ 项频繁模式是由 k 项频繁模式空间求交得到，因此一个频繁项集的子集的支持度要小于等于该频繁项集的支持度，即使是上述第 3 组卡号组合也很有可能是某个频繁项集的子集。因此我们需要对以上结果进行子集合并，即找出最大长度骗保模式，当卡号组合取最大长度骗保模式的任意子集时则认为这是一种多卡骗保行为。

进行子集合并方法是对每组数据添加一标志位，初始值为 1，对第 i 组数据向下扫描，若与第 i 组数据存在包含或者被包含关系，则将该组数的标志位置 0，若

表 10: Eclat 支持度排序

排名	医保卡号组合 (未合并)	支持度
1	[242023,452114]	0.40625
2	[223085,523612]	0.40625
3	[163696,223085,523612]	0.375
4	[163696,223085]	0.375
5	[163696,523612]	0.375
6	[223085,338370,523612]	0.34375
7	[223085,338370]	0.34375
8	[338370,523612]	0.34375
9	[163696,223085,338370,523612]	0.3125
10	[163696,223085,338370]	0.3125
11	[163696,338370,523612]	0.3125
12	[163696,338370]	0.3125
13	[242023,435116,452114]	0.28125
14	[435116,452114]	0.28125
15	[242023,435116]	0.28125
16	[363950,543311]	0.28125
...	...	...

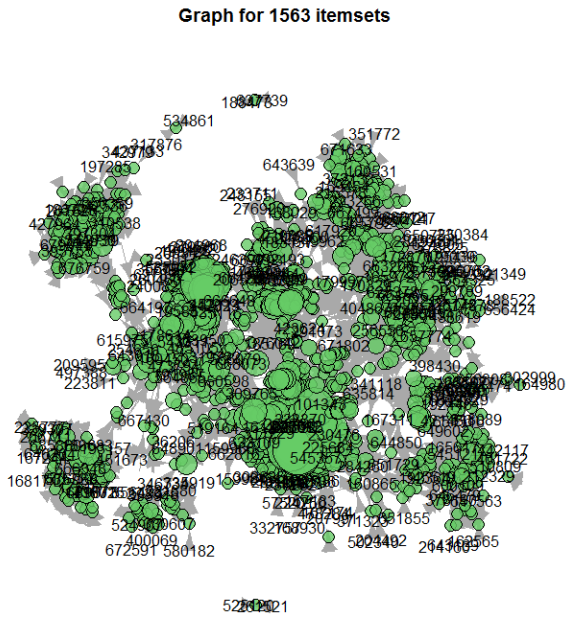

图 5: 频繁项集可视化

该组卡号为目前的最长模式序列，则记住它的编号，扫描结束后将第 i 组数据替换为最长模式序列。

伪代码如下:

MergeSubsets

```
1  for i = 1 : total_combination
2       do
3          num = i;
4          for j = i + 1 : total_combination − i
5               do
6                  if (flag(j) == 1)
7                    then
8                         if (i_combination ⊆ j_combination or j_combination ⊆ i_combination)
9                           then
10                               flag(j) = 0;
11                               if (length(i) > num)
12                                 then num = j;
13         i_combination = j_combination;
```

对于上步得到的 1563 组卡号组合进行子集合并得到的最大子集支持度大于 0.21875 的结果如表 11所示。其中最大子集支持度是指该最长模式序列所有具有最大支持度的子集的支持度。

表 11: 模式序列合并结果

最长模式序列	最大子集支持度
[195852,242023,363950,435116,452114,543311]	0.40625
[163696,223085,309765,311229,338370,523612]	0.40625
[309765,363950,473794,543311,564865]	0.28125
[178378,376042,461304,479399]	0.25
[309765,338370,376042]	0.25
[291073,338370,376042]	0.25
[309765,376042,566079]	0.25
[344930,395755,397304,563739,676759,679044]	0.21875
[175167,193157,576968,635727,666348]	0.21875
[221880,519164]	0.21875
[423624,448707,452293]	0.21875
[170329,193785,223266,649962,665792,667499]	0.21875
[183137,423624,452293]	0.21875
[165617,192364,545121,600109,646479]	0.21875
[256550,279041,404867,625378,655834,659876]	0.21875
[256550,279041,404867,625378,659876]	0.21875
[170329,223266,649962,665792,667499]	0.21875
[170329,202212,256550,404867,655834]	0.21875
[165617,341118,649602]	0.21875

4.3.3 模型的优缺点及展望

优点：

- 算法效率高，可有效识别大数据集中的多卡骗保行为
- 输出结果为最长模式序列，避免了重复结果，提高识别效率

缺点及展望：

- 支持度阈值的选取对算法复杂度的影响较大，需要比较丰富的经验才能确定合适的阈值，在此我们只是进行了试探性的工作
- 算法效率可以进一步提高，寻找最长模式序列可以合并到 Eclat 算法中，将广度优先搜索的思想转换为深度优先搜索，因为最长模式序列一般处于搜索空间的较底层

参考文献

[1] Phua, Clifton, et al. "A comprehensive survey of data mining-based fraud detection research." *arXiv preprint arXiv*:1009.6119 (2010).

[2] Yamanishi, Kenji, et al. "On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms." *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2000.

[3] Li, Jing, et al. "A survey on statistical methods for health care fraud detection." *Health care management science* 11.3 (2008): 275-287.

[4] 何俊华. 数据挖掘技术在医保领域中的研究与应用. MS thesis. 复旦大学, 2011.

[5] Ortega, Pedro A., Cristián J. Figueroa, and Gonzalo A. Ruz. "A Medical Claim Fraud/Abuse Detection System based on Data Mining: A Case Study in Chile." *DMIN* 6 (2006): 26-29.

[6] Nikulin, Mikhail S. "Hellinger distance." *Encyclopedia of Mathematics* (2001).

附录

注：本题中所有数据预处理使用 R 语言完成，不附代码。下面附实现智能过滤机的 C++ 实现

```
#include <string.h>
#include <math.h>
#include <Windows.h>
#include <ctime>

#define NORMDIM 5
#define CID1 22      //itemcat
#define CID2 3       //TAREC
#define CID3 8       //age
#define rh 0.0003
```

```
#define R 0.0005
#define alpha 2.0
#define trainThresh 1000

using namespace std;

const double PI = atan(1.0) * 4;
int hash[215];
const int totalcell = CID1 * CID2 * CID3;

struct cellparams
{
    double frequency;
    int train_rec;
    double prob;
    double num[NORMDIM][5]; //0 for c, 1 for miu, 2 for
       sigma^2, 3 for es_miu, 4 for es_sigma^2
}cellold[CID1][CID2][CID3], cellnew[CID1][CID2][CID3];

double quick_calc();
void inherit(int t);
void renew();
double log_calc(int itemcat, int tarec, int age, double
   price);
double calc_norm(double val, double u, double sq);

int main()
{
    FILE *fin, *fout;
    fin = fopen("handle_small.csv", "r");
    fout = fopen("scores2.csv", "w");

    hash[1] = 1; hash[3] = 2; hash[4] = 3; hash[5] = 4;
       hash[6] = 5; hash[8] = 6; hash[10] = 7;
    hash[11] =8 ; hash[12] =9 ; hash[13] = 10; hash[14]
       = 11; hash[15] = 12; hash[18] = 13; hash[19] =
       14;
    hash[20] = 15; hash[21] = 16; hash[23] = 17; hash
       [24] = 18; hash[25] = 19; hash[26] = 20; hash
       [209] = 21;
    hash[214] = 22;

    // initialization
    memset(cellold, 0, sizeof(cellold));
    memset(cellnew, 0, sizeof(cellnew));
```

```
for (int c1 = 0; c1 < CID1; c1++){
        for (int c2 = 0; c2 < CID2; c2++){
                for (int c3 = 0; c3 < CID3; c3++){
                        for (int l = 0; l < NORMDIM;
                            l++){
                                cellold[c1][c2][c3].
                                   num[l][0] = 1.0 /
                                    NORMDIM;
                                cellold[c1][c2][c3].
                                   num[l][1] = (
                                   double)(rand() %
                                   100) / 100;
                                cellold[c1][c2][c3].
                                   num[l][2] = (
                                   double)(rand() %
                                   100) / 10;
                                cellold[c1][c2][c3].
                                   num[l][3] = (
                                   double)(rand() %
                                   100) / 100;
                                cellold[c1][c2][c3].
                                   num[l][4] = (
                                   double)(rand() %
                                   100) / 10;
                        }
                }
        }
}

int cnt = 0;
fscanf(fin, "%*s\n");
fprintf(fout, "WORKLOAD_ID,WORKLOAD_ITEMCAT_DR,
   WORKLOAD_TAREC_DR,AGE_GROUP,WORKLOAD_TOTALPRICE,
   SCORE,TRAIN_REC\n");
printf("excuting...\n");
__int64 t = GetTickCount();

while (!feof(fin)){
        int itemcat, tarec, age, id;
        double price;
        if (cnt % 1000 == 0) printf("%d datum
           handled...\n", cnt);
        // read in a new datum and convert to
           subscript
```

```
fscanf(fin, "%d,%d,%d,%d,%lf\n", &id, &
   itemcat, &tarec, &age, &price);
//printf("datum %d :%d,%d,%d,%d,%.2lf read\n
   ", cnt, id, itemcat, tarec, age, price);
itemcat = hash[itemcat] - 1;
tarec--;
cnt++;
// inherit the invariant values
inherit(cnt);

// update the discrete part
cellnew[itemcat][tarec][age].frequency += 1;
cellnew[itemcat][tarec][age].prob = cellnew[
   itemcat][tarec][age].frequency / ((1 -
   pow(1 - rh, cnt)) / rh);
cellnew[itemcat][tarec][age].train_rec += 1;

double gamma, var, mean, temp, sum = 0;
temp = 0;
for (int i = 0; i < NORMDIM; i++){
        temp += cellold[itemcat][tarec][age
           ].num[i][0] * calc_norm(price,
           cellold[itemcat][tarec][age].num[
           i][1], cellold[itemcat][tarec][
           age].num[i][2]);
}
for (int i = 0; i<NORMDIM; i++){
        mean = cellold[itemcat][tarec][age].
           num[i][1];
        var = cellold[itemcat][tarec][age].
           num[i][2];
        gamma = (1 - alpha * R) * cellold[
           itemcat][tarec][age].num[i][0] *
           calc_norm(price, mean, var) /
           temp + alpha * R / NORMDIM;
        if (_isnanf(gamma)) gamma = 1.0;
        // update c_i
        cellnew[itemcat][tarec][age].num[i
           ][0] = (1 - R) * cellold[itemcat
           ][tarec][age].num[i][0] + R *
           gamma;
        //printf("%.4lf %.4lf\n", gamma,
           cellnew[itemcat][tarec][age].num[
           i][0]);
        // update mu_i
```

```
                cellnew[itemcat][tarec][age].num[i
                    ][3] = (1 - R) * cellold[itemcat
                    ][tarec][age].num[i][3] + R *
                    gamma * price;
                cellnew[itemcat][tarec][age].num[i
                    ][1] = cellnew[itemcat][tarec][
                    age].num[i][3] / cellnew[itemcat
                    ][tarec][age].num[i][0];
                // update sigma^2_i
                cellnew[itemcat][tarec][age].num[i
                    ][4] = (1 - R) * cellold[itemcat
                    ][tarec][age].num[i][4] + R *
                    gamma * price * price;
                cellnew[itemcat][tarec][age].num[i
                    ][2] = cellnew[itemcat][tarec][
                    age].num[i][4] / cellnew[itemcat
                    ][tarec][age].num[i][0];
                cellnew[itemcat][tarec][age].num[i
                    ][2] -= cellnew[itemcat][tarec][
                    age].num[i][1] * cellnew[itemcat
                    ][tarec][age].num[i][1];
            }
            /*
            for (int i = 0; i < NORMDIM; i++){
                printf("D%d:p%.4lf,u%.4lf,s%.4lf\n",
                    i, cellnew[itemcat][tarec][age].
                    num[i][0], cellnew[itemcat][tarec
                    ][age].num[i][1], cellnew[itemcat
                    ][tarec][age].num[i][2]);
            }
            */

            // calculate score if trained enough
            double score = 0;
            score = log_calc(itemcat, tarec, age, price)
                ;

            renew();
            if (cellnew[itemcat][tarec][age].train_rec >
                trainThresh) fprintf(fout, "%d,%d,%d,%d
                ,%.2lf,%.4lf,%d\n", id, itemcat, tarec +
                1, age, price, score, cellnew[itemcat][
                tarec][age].train_rec);
        }
```

```
    t = GetTickCount() - t;
    printf("\nPrecalculation completed. Time elapsed:
       %.2lfs\n", (double)t / 1000);
    fclose(fin);
    return 0;
}

void renew(){
    int c1, c2, c3;
    for (c1 = 0; c1 < CID1; c1++){
            for (c2 = 0; c2 < CID2; c2++){
                    for (c3 = 0; c3 < CID3; c3++){
                            cellold[c1][c2][c3] =
                               cellnew[c1][c2][c3];
                    }
            }
    }
}

void inherit(int t){
    int c1, c2, c3;
    for (c1 = 0; c1 < CID1; c1++){
            for (c2 = 0; c2 < CID2; c2++){
                    for (c3 = 0; c3 < CID3; c3++){
                            cellnew[c1][c2][c3].
                               frequency = cellold[c1][
                               c2][c3].frequency * (1 -
                               rh);
                            cellnew[c1][c2][c3].
                               train_rec = cellold[c1][
                               c2][c3].train_rec;
                            cellnew[c1][c2][c3].prob =
                               cellnew[c1][c2][c3].
                               frequency / ((1 - pow(1 -
                                rh, t)) / rh);
                            for (int k = 0; k < NORMDIM;
                                k++){
                                    cellnew[c1][c2][c3].
                                       num[k][0] =
                                        cellold[c1][c2][
                                       c3].num[k][0];
                                    cellnew[c1][c2][c3].
                                       num[k][1] =
                                        cellold[c1][c2][
```

```
                                                    c3].num[k][1];
                                                cellnew[c1][c2][c3].
                                                    num[k][2] =
                                                    cellold[c1][c2][
                                                    c3].num[k][2];
                                        }
                                }
                        }
                }
        }
}

double log_calc(int itemcat, int tarec, int age, double
    price){
        double p1, p2, u, s, c;
        p1 = cellold[itemcat][tarec][age].prob;
        p2 = 0;
        for (int i = 0; i < NORMDIM; i++){
                c = cellold[itemcat][tarec][age].num[i][0];
                u = cellold[itemcat][tarec][age].num[i][1];
                s = cellold[itemcat][tarec][age].num[i][2];
                p2 += c * calc_norm(price, u, s);
        }
        return (-log10(p1)-log10(p2));
}

double calc_norm(double val, double u, double sq){
        return exp(-(val - u)*(val - u) / (2 * sq)) / (sqrt
            (2 * PI * sq));
}
```