商品2/26

1.      结合实训项目或实际应用，构想一套数据挖掘方案包括：数据获取、数据清理、数据挖掘算法的选取、评价，希望最终挖掘的结论是什么？能应用到什么领域？

答：

1）主题：根据用浏览过的商品来推荐用户可能喜欢的商品

2）数据收集：从淘宝、京东等电商平台爬取大量的商品数据，收集的数据是商品的属性有图片URL、商品标题、折扣价、原价、发货地址、商品销量

3)数据预处理：数据有冗余、删减数据提高数据质量，选取完整的数据，有些数据是无效的

4）数据挖掘：建立模型基于协同过滤，一般来说，推荐引擎都是在协同过滤算法（Collaborative Filter）的基础上，搭建一套符合自身特点的规则库。即该算法会同时考虑其他顾客的选择和行为，在此基础上搭建产品相似性矩阵和用户相似性矩阵。基于此，找出最相似的顾客或最关联的产品，从而完成产品的推荐。

5)模式评估：

缺点：用户的爱好在不同的时期可能会发生变化，导致在一个较长的时间段里没有使用可能会出现一些偏差，推荐存在延迟性

优点：智能推荐，用户可以方便的找到自己可能喜欢的物品，更加快捷，节省时间，

6)知识表示：在页面上根据用户的选择和行为，和其他用户的选择和行为，推荐显示用户可能感兴趣的物品列表，展示在“猜你喜欢”区域；网站管理者也可根据用户购买数据集来得知热销产品的走向趋势，预测下一季新品，方便库存的管理；

7）能应用到的领域：电商网站的猜你喜欢，推荐引擎

2.      OLTP，OLAP，概念查询、区别及应用，最好能通过例子说明。

答：

（1）概念

OLTP是联机事务处理过程(On-Line Transaction Processing)，也称为面向交易的处理过程，其基本特征是前台接收的用户数据可以立即传送到计算中心进行处理，并在很短的时间内给出处理结果，是对用户操作快速响应的方式之一。

OLAP是联机分析处理过程(On-Line Analytical Processing)，使分析人员、管理人员或执行人员能够从多种角度对从原始数据中转化出来的、能够真正为用户所理解的、并真实反映企业维特性的信息进行快速、一致、交互的存取，从而获得对数据更深入了解的一类软件技术。

（2）区别

①OLTP是传统的关系型数据库的主要应用，主要是基本的、日常的事务处理，记录即时的增、删、改、查，比如在银行存取一笔款，就是一个事务交易。强调数据库内存效率，强调内存各种指标的命令率，强调绑定变量，强调并发操作；

OLAP是数据仓库系统的主要应用，支持复杂的分析操作，侧重决策支持，并且提供直观易懂的查询结果。

②交易一般是确定的，所以OLTP是对确定性的数据进行存取(比如存取款都有一个特定的金额)。支持大量并发用户定期添加和修改数据。并发性要求高并且严格的要求事务的完整、安全性 (比如这种情况：有可能你和你的家人同时在不同银行取同一个帐号的款)；

而对于OLAP，因为重点在于决策支持，所以查询一般是动态的，也就是说允许用户随时提出查询的要求。于是在OLAP中通过一个重要概念“维”来搭建一个动态查询的平台（或技术），供用户自己去决定需要知道什么信息。

③OLTP实时性要求高；OLAP实时性要求不是很高，很多应用顶多是每天更新一下数据。

④OLTP数据量不是很大；OLAP数据量大，因为OLAP支持的是动态查询，所以用户也许要通过将很多数据的统计后才能得到想要知道的信息，例如时间序列分析等等，所以处理的数据量很大；

⑤OLTP是面向顾客的，用于事务和查询处理；OLAP是面向市场的，用于数据分析。

⑥OLTP系统管理当前数据；OLAP系统管理大量历史数据,提供汇总和聚集机制。

⑦OLTP系统的访问主要由短的原子事务组成，这种系统需要并行和回滚恢复机制；OLAP系统的访问大部分是只读操作。

⑧OLTP采用实体—关系ER模型和面向应用的数据库设计；OLAP采用星型或雪花模型和面向主题的数据库设计。



（3）应用

OLTP主要应用于数据库，是细节的。

OLAP主要应用于数据仓库，是综合或者提炼的，支持决策需求。

（4）例子

OLTP：客户订单、库存水平和银行账户查询

OLAP：销售分析、市场推广分析、客户利润率分析。

3.搜索常见的爬虫第三方应用，了解第三方软件的使用--------八爪鱼

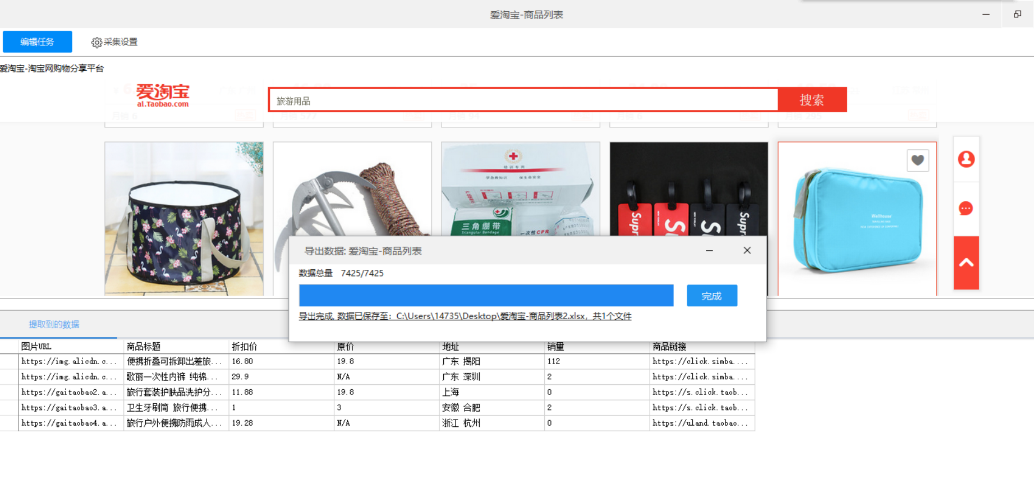
了解第三方库：webcollector

**1）下载八爪鱼软件，注册账号之后登录，自定义网址爬取想要的数据**

**爬取数据图如下**



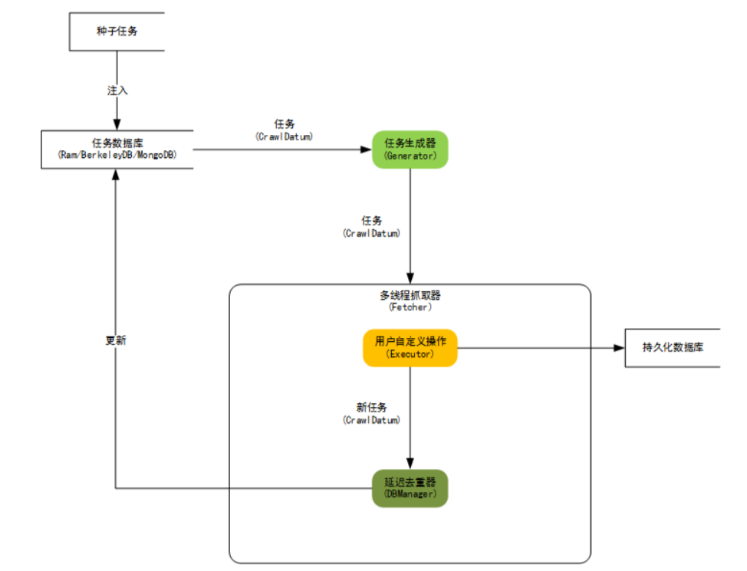
导出Excel表格，得到数据集



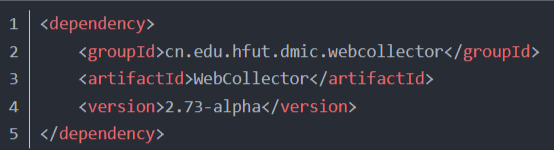
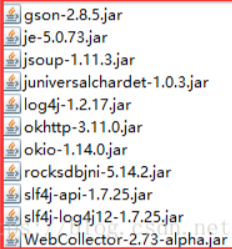
2）webcollector第三方库



webcollector的内核构架图



一开始导入特定的jar包和设置selenium的maven依赖



代码实现思路：

（1）利用Selenium获取登陆新浪微博weibo.cn的cookie

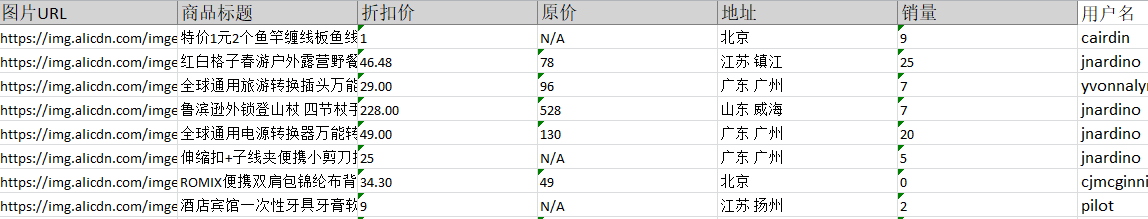
（2）利用WebCollector和获取的cookie爬取新浪微博并抽取数据

4.根据题1的要求，找到相关的离线数据并下载了解数据集结构，生成文档，写Java的程序去读大文档

数据集结构：

商品图片URL、商品标题、折扣价、原价、地址、销量、用户名

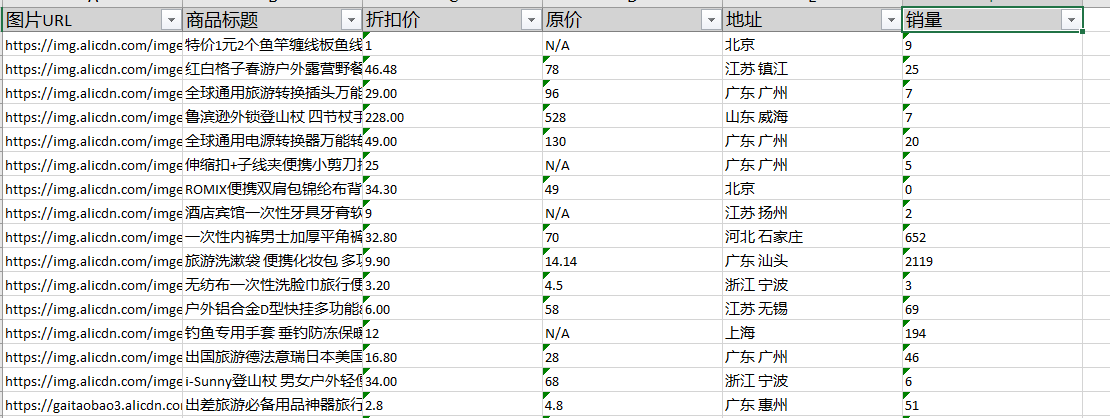
经过整理后的excel部分截图：



2/27

作业三：

1.      根据下载的数据集，详细列出有多少列，每个列的含义、类型（名词性、数值型、多元性）



共6列

图片url 含义是旅游商品展示图片的地址 类型为名词性

商品标题 含义是旅游商品全称 类型为名词性

折扣价 含义是旅游商品打折之后的商品价格 类型为数值型

原价 含义是旅游商品原本的价格 类型为数值型

地址 含义是旅游商品卖家的发货地址 类型为名词性

销量 含义是旅游商品的月销售数量 类型为数值型

2.      针对某一、二个列。去计算sum、avg、count（分类）、中位数、众数，分析并得到有用的结论；

答：

月销量列：

sum：883050

其中 电器类43020 洗漱用品467100 生活起居品276900 衣服类96030

avg： 368

count： 电器类 270 洗漱用品类 1020 生活起居类900 衣服类210

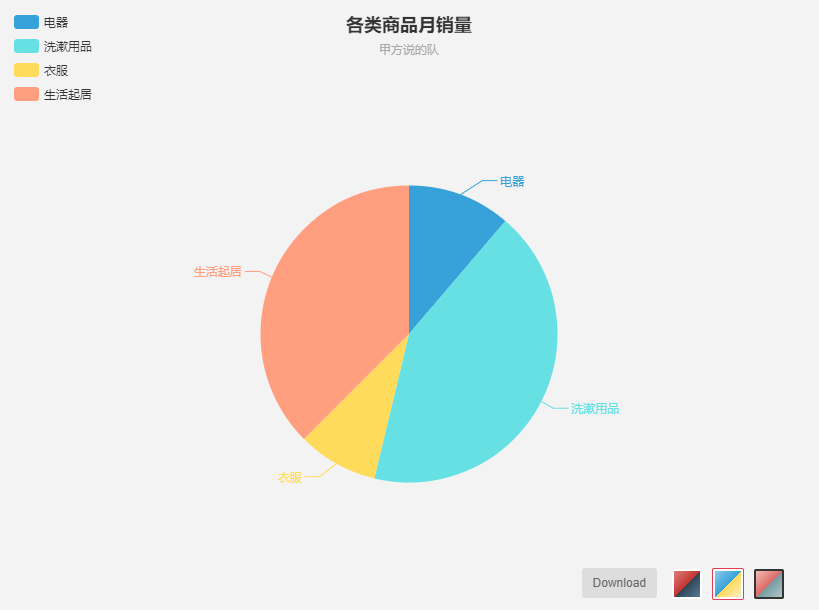
中位数：630

众数： 0

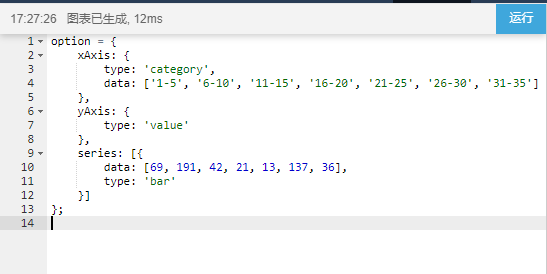
分析：现有的很多旅行相关商品上架未被购买过或者销售量很低，说明对于旅行用品，用户的购买种类样式比较集中，同时也比较热门，其中较多的是旅行换洗用的一次性用品、隔脏睡袋、旅行化妆小样分装瓶、户外折凳、便携插座转换器等用品

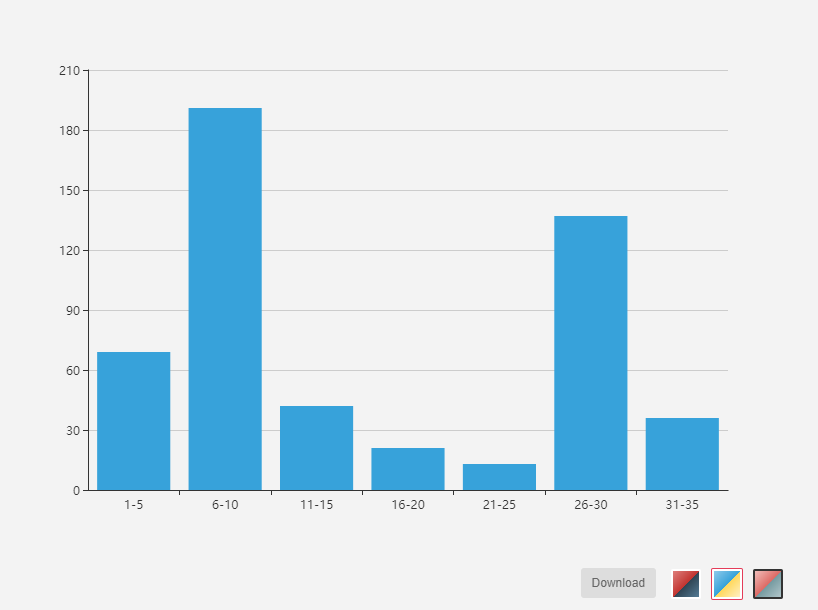
3.使用可视化工具展示作业2中你选定的某些有价值的列，直观的说明数据隐含的知识。可选工具：Excel、highcharts、echarts等





针对于洗漱用品这一类，我们绘图分析了它其中一些商品的价格区间与销量的分布





同一种商品，不同的店铺价格不同，质量也不同，针对洗漱用品类的商品，经过数据处理之后，我们发现价格最便宜的在1,2块，价格稍贵的在30左右，可以看出6-10范围和26-30范围用户购买的更多。分析出用户购买商品上不同的用户群体针对个人的消费情况会选择不同的价位的商品。

4.欧氏距离的定义和公式是什么？请根据欧氏距离公式计算出如下矩阵。

欧氏距离的定义

欧氏距离（也称欧几里得度量（euclidean metric））是一个通常采用的距离定义，指在m维空间中两个点之间的真实距离，或者向量的自然长度（即该点到原点的距离）。在二维和三维空间中的欧氏距离就是两点之间的实际距离。

欧氏距离的公式（n维空间）

https://gss2.bdstatic.com/9fo3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D469/sign=2bee509a5582b2b7a39f38c208accb0a/ae51f3deb48f8c541e440db136292df5e1fe7f9d.jpg

|  |  |  |  |
| --- | --- | --- | --- |
|  | A1 | A2 | A3 |
| a | 30 | 103 | 31 |
| b | 27 | 271 | 25 |
| c | 18 | 156 | 17 |
| d | 27 | 193 | 50 |

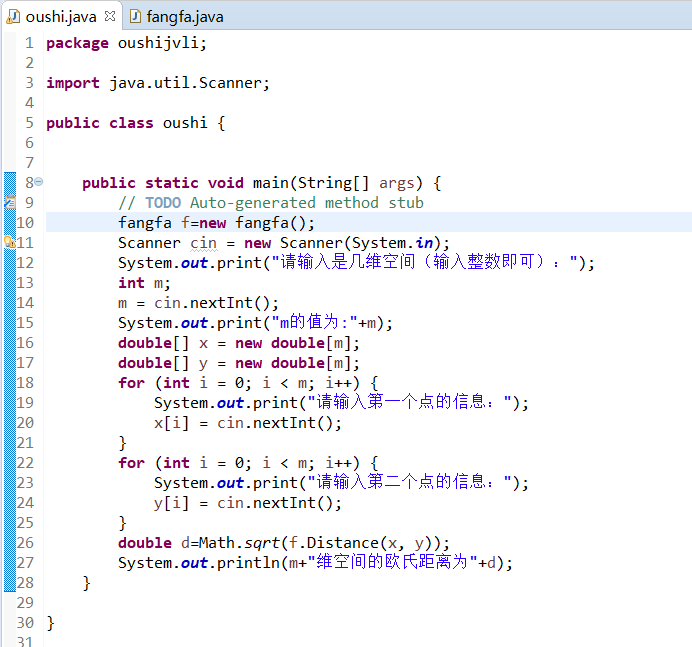
两属性的距离矩阵：

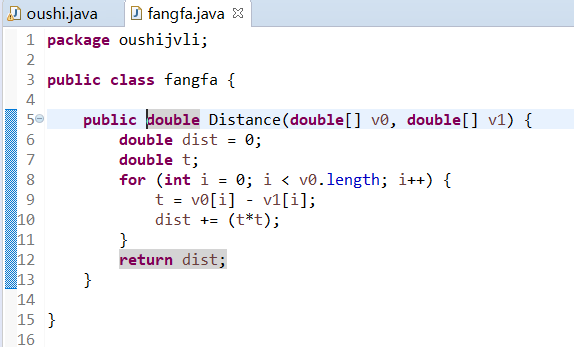
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | a | b | c | d |
| a | 0 | 168.03 | 54.34 | 90.05 |
| b | 168.03 | 0 | 115.35 | 78.00 |
| c | 54.34 | 115.35 | 0 | 38.08 |
| d | 90.05 | 78.00 | 38.08 | 0 |

三属性的距离矩阵：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | a | b | c | d |
| a | 0 | 168.13 | 56.12 | 92.03 |
| b | 168.13 | 0 | 115.63 | 81.91 |
| c | 56.12 | 115.63 | 0 | 50.39 |
| d | 92.03 | 81.91 | 50.39 | 0 |

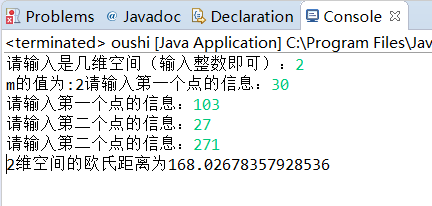
Java实现代码为：



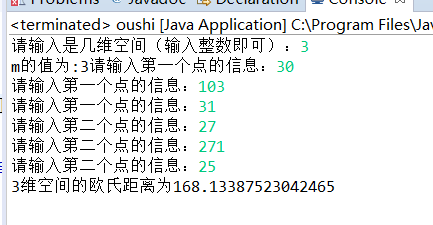


运行过程及结果：

二维的：



三维的：



2/28

作业四

1.根据你们选择的数据集，结合你要分析和挖掘的目标，去筛选出数据的列，可能只要1列，或者两列，或者三列。考虑有效数据的比例（空值率，数值有效率）

答：

分析和挖掘的目标：根据旅行相关商品的淘宝销量情况，挖掘最适宜在旅游相关网站发售的商品，以及最适价格区间。

选择的数据集数据结构为如下：

商品图片URL、商品名称、商品折扣后售价、原价、发货地址、月销量

筛选的目标列为：

商品名称、商品折扣后售价、发货地址、商品月销量

空值率：

以原价列N/A为空值，去重之前总数为8310 去重之前空值为3936 空值率为47.36%

数值有效率：

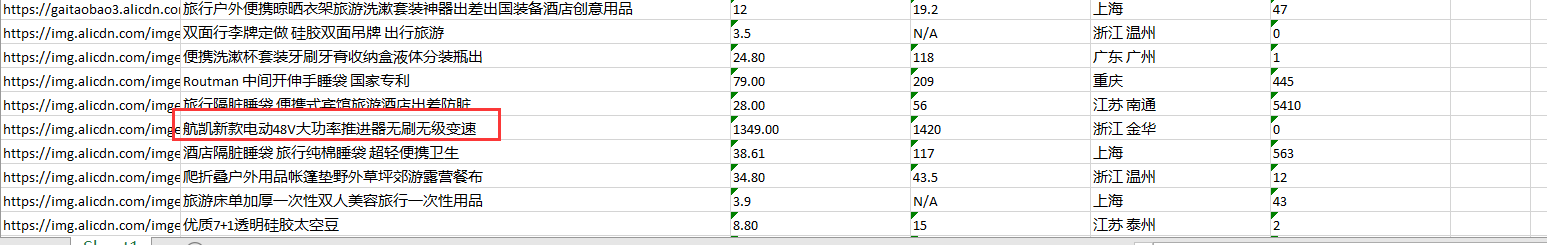
去重之后的有效数据总数为2516 数值有效率为30.81%

2.根据自己的数据集找到数据道中“脏”的地方，数据的问题。

答：商品原价列存在部分不适用的情况（N/A），部分样品数据存在重复（商品完全名称相同项），可能与淘宝网的重复店家推荐有关

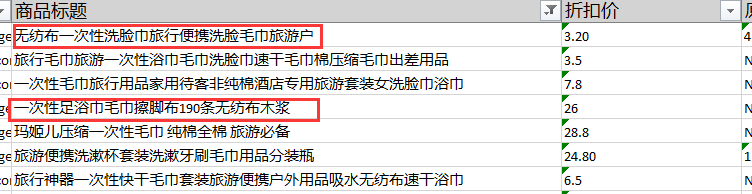
3.找到数据中的异常点（离群点）并做分析和处理。

答：



1）针对商品折扣后价格分析可以发现有些商品价格在1000多，有的商品将在几十，价格在1000以上的都是属于一些大型的电器之类，其他的那些属于旅行所需要的小型物品

针对所有的商品进行大致分类，我们分为电器、洗漱用品、生活家居、衣服四大类



2）还可以发现相同类型的商品，由于店铺的销售的基本单位不同，导致相同类型的商品的价格之间相差好几倍

4.搜索归一化的概念和公式。计算归一化之后的结果。

|  |  |  |  |
| --- | --- | --- | --- |
|  | A1 | A2 | A3 |
| a | 30 | 103 | 0.1 |
| b | 27 | 271 | 3.87 |
| c | 18 | 156 | 2.7 |
| d | 27 | 193 | 1.9 |

答：

**归一化概念：**

归一化是一种简化计算的方式，即将有量纲（物理量的基本属性）的表达式，经过变换，化为量纲的表达式，成为标量。

**归一化的公式：**

一、min-max标准化（Min-Max Normalization）

也称为离差标准化，是对原始数据的线性变换，使结果值映射到[0 – 1]之间。转换函数如下：x∗=(x−min)/(max−min)

其中max为样本数据的最大值，min为样本数据的最小值。这种方法有个缺陷就是当有新数据加入时，可能导致max和min的变化，需要重新定义。

二、Z-score标准化方法

这种方法给予原始数据的均值（mean）和标准差（standard deviation）进行数据的标准化。经过处理的数据符合标准正态分布，即均值为0，标准差为1，转化函数为：

x∗=(x−μ)/σ

其中 μ为所有样本数据的均值，σ为所有样本数据的标准差。

三 、Z-scores 简单化

模型如下：

x∗=1/(1+x)

x越大证明x∗ 越小，这样就可以把很大的数规范在[0-1]之间了。

**计算之后的结果：**

|  |  |  |  |
| --- | --- | --- | --- |
|  | A1 | A2 | A3 |
| a | 1 | 0 | 0 |
| b | 0.75 | 1 | 1 |
| c | 0 | 0.32 | 0.69 |
| d | 0.75 | 0.54 | 0.48 |

3/4

作业五

1.分析上次作业中你们数据集中的空值的原因，如何做处理？

答：

上次作业中数据集中空值部分为：商品原价列存在部分不适用的情况（N/A）

**空值原因：** 在使用八爪鱼工具时，自动爬虫未能获取到淘宝网部分商品原价数据

**处理方法：**自动填充

与目标元组同一类的所有样本的属性均值，因为同类商品之间原价不会有太大的变动，可以参考同款商品的原价，但是该原价一定是比这个样品数据的折扣价要贵

2.分析上次作业中你们数据集中，是否有以下这种数据问题：

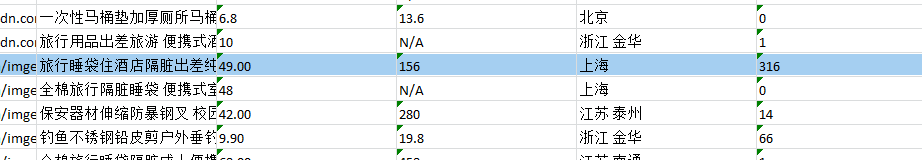
重复记录

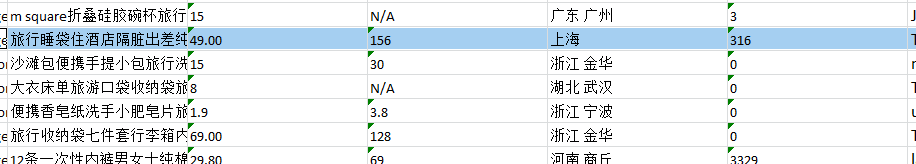
数据不完整记录

不一致的数据

答：

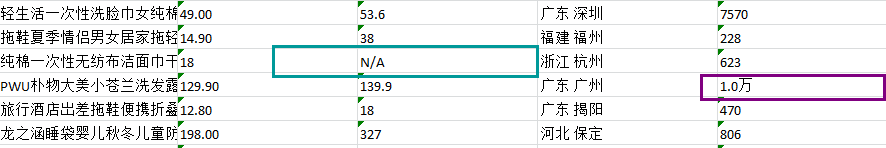
**重复记录**：可能是由于我们的数据集是通过淘宝网抓取获得的，使得记录存在了重复记录（商品全称完全一致项为重复项）





**数据不完整记录情况**：8332条商品数据中有2835项的原价列为N/A

**不一致数据情况**：有96条商品数据的月销量数据以万作为单位



3.针对于作业2的问题，设计解决方法（参考缺失值的处理办法）

答：

（1）针对有重复记录的情况：选择删除多余重复记录，只留下一条数据；

（2）针对商品原价数据不完整的情况：根据该商品的类别，找到该类别商品原价的平均值，填补上去，因为同类商品之间原价不会有太大的变动，可以参考同款商品的原价

（3）针对商品数据销量单位不一致的情况：利用Excel工具转换格式，设置自定义单元格格式，消除万单位。

4.查询ETL的概念、ETL作用、常见ETL工具

答：

**ETL概念：**

是英文Extract-Transform-Load的缩写，用来描述将数据从来源端经过萃取（extract）、转置（transform）、加载（load）至目的端的过程。

**ETL作用：**

是构建数据仓库的重要一环，用户从数据源抽取出所需的数据，经过数据清洗和加工，最终按照预先定义好的模型，将数据加载到目的数据仓库中去。在ETL整个过程中主要完成数据抽取、数据转换、数据清洗、数据加载的任务

**ETL工具：**

**1、Apache Camel**

Apache Camel 是一个非常强大的基于规则的路由以及媒介引擎，该引擎提供了一个基于 POJO 的企业应用模式(Enterprise Integration Patterns)的实现，你可以采用其异常强大且十分易用的 API (可以说是一种 Java 的领域定义语言 Domain Specific Language)来配置其路由或者中介的规则。 通过这种领域定义语言，你可以在你的 IDE 中用简单的 Java Code 就可以写出一个类型安全并具有一定智能的规则描述文件。

**2、Apache Kafka**

Apache Kafka 是一个开源的消息系统，用 Scala 和 Java 写成。该项目为处理实时数据提供了一个统一、高通量、低延时的平台。有如下特性：

* 通过 O(1) 的磁盘数据结构提供消息的持久化，这种结构对于即使数以TB的消息存储也能够保持长时间的稳定性能。
* 高吞吐量：即使是非常普通的硬件 kafka 也可以支持每秒数十万的消息。
* 支持通过 kafka 服务器和消费机集群来分区消息。
* 支持 Hadoop 并行数据加载。

**3、Apatar**

Apatar 用 Java 编写，是一个开源的数据抽取、转换、 装载(ETL)项目。模块化的架构。提供可视化的 Job 设计器与映射工具，支持所有主流数据源，提供灵活的基于 GUI、服务器和嵌入式的部署选项。它具有符合 Unicode 的功能，可用于跨团队集成数据，填充数据仓库与数据市场，在连接到其他系统时在代码少量或没有代码的情况下进行维护。

**4、Heka**

来自 Mozilla 的 Heka 是一个用来收集和整理来自多个不同源的数据的工具，通过对数据进行收集和整理后发送结果报告到不同的目标用于进一步分析。

**5、Logstash**

Logstash 是一个应用程序日志、事件的传输、处理、管理和搜索的平台。你可以用它来统一对应用程序日志进行收集管理，提供 Web 接口用于查询和统计。Logstash 现在是 ElasticSearch 家族成员之一。

**6、Scriptella**

Scriptella 是一个开源的 ETL (抽取-转换-加载)工具和一个脚本执行工具，采用 Java 开发。Scriptella 支持跨数据库的 ETL 脚本，并且可以在单个的 ETL 文件中与多个数据源运行。Scriptella 可与任何 JDBC / ODBC 兼容的驱动程序集成，并提供与非 JDBC 数据源和脚本语言的互操作性的接口。它还可以与 Java EE，Spring，JMX，JNDI 和 JavaMail 集成。

**7、Talend**

Talend (踏蓝) 是第一家针对的数据集成工具市场的 ETL(数据的提取 Extract、传输 Transform、载入Load)开源软件供应商。Talend 以它的技术和商业双重模式为 ETL 服务提供了一个全新的远景。它打破了传统的独有封闭服务，提供了一个针对所有规模的公司的公开的，创新的，强大的灵活的软件解决方案。最终，由于 Talend 的出现，数据整合方案不再被大公司所独享。

**8、Kettle**

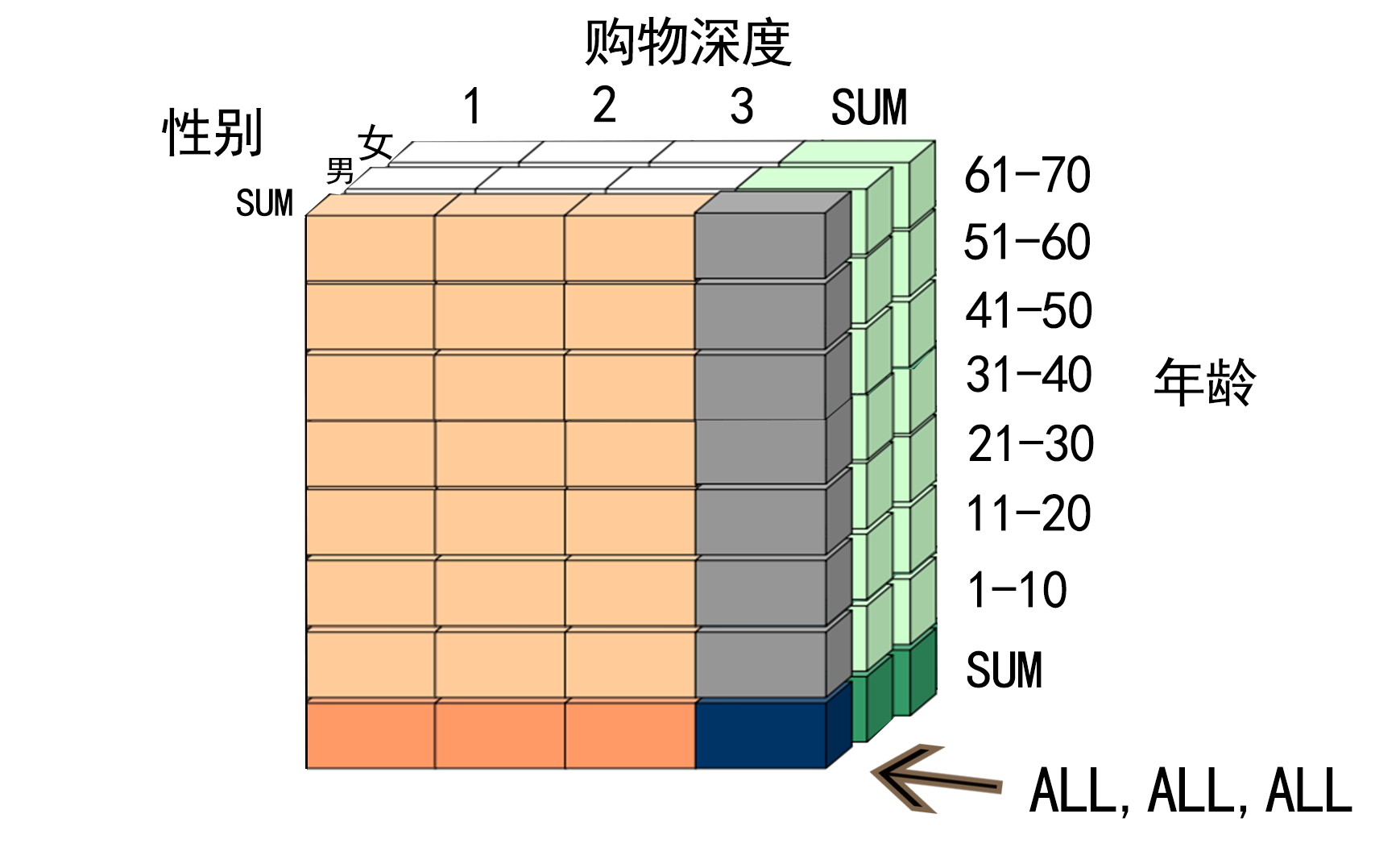
Kettle 是一款国外开源的 etl 工具，纯 java 编写，绿色无需安装，数据抽取高效稳定(数据迁移工具)。Kettle 中有两种脚本文件，transformation 和 job，transformation 完成针对数据的基础转换，job 则完成整个工作流的控制。

3/5

第六次作业

1.对自己的数据集进行数据立方体设计，并展示。

答：



2.什么叫做启发式方法？贪心算法？

答：

**启发式方法**指人在解决问题时所采取的一种根据经验规则进行发现的方法。其特点是在解决问题时,利用过去的经验,选择已经行之有效的方法，而不是系统地、以确定的步骤去寻求答案。

**启发式算法**（heuristic algorithm)是相对于最优化算法提出的。一个问题的最优算法求得该问题每个实例的最优解。启发式算法可以这样定义：一个基于直观或经验构造的算法，在可接受的花费（指计算时间和空间）下给出待解决组合优化问题每一个实例的一个可行解，该可行解与最优解的偏离程度一般不能被预计。现阶段，启发式算法以仿自然体算法为主，主要有蚁群算法、模拟退火法、神经网络等

**贪心算法**（又称贪婪算法）是指，在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，他所做出的是在某种意义上的局部最优解。贪心算法不是对所有问题都能得到整体最优解，关键是贪心策略的选择，选择的贪心策略必须具备无后效性，即某个状态以前的过程不会影响以后的状态，只与当前状态有关。

**基本思路**：

从问题的某一个初始解出发一步一步地进行，根据某个优化测度，每一步都要确保能获得局部最优解。每一步只考虑一个数据，他的选取应该满足局部优化的条件。若下一个数据和部分最优解连在一起不再是可行解时，就不把该数据添加到部分解中，直到把所有数据枚举完，或者不能再添加算法停止

**基本过程**：

1. 建立数学模型来描述问题；
2. 把求解的问题分成若干个子问题；
3. 对每一子问题求解，得到子问题的局部最优解；
4. 把子问题的局部最优解合成原来解问题的一个解。

**应用算法：**

贪心算法的应用算法有迪杰斯特拉的单源点最短路径

3.查询信息熵的概念，计算公式？

答：

信息论之父Shannon 借鉴了热力学的概念，把信息中排除了冗余后的平均信息量称为“信息熵”，并给出了计算信息熵的数学表达式。

**现代定义：**

信息是**物质、**[**能量**](https://baike.baidu.com/item/%E8%83%BD%E9%87%8F)**、信息及其属性的标示**。【**逆维纳**信息定义】

信息是**确定性的增加**。【**逆香农**信息定义】

信息是**事物现象及其属性标识**的集合。【2002年】

**最初定义：**

信息理论的鼻祖之一Claude E. Shannon把信息（熵）定义为[离散](https://baike.baidu.com/item/%E7%A6%BB%E6%95%A3" \t "_blank)[随机事件](https://baike.baidu.com/item/%E9%9A%8F%E6%9C%BA%E4%BA%8B%E4%BB%B6)的出现概率。

所谓信息熵，是一个[数学](https://baike.baidu.com/item/%E6%95%B0%E5%AD%A6" \t "_blank)上颇为抽象的概念，在这里不妨把信息熵理解成某种特定信息的出现概率。而信息熵和热力学熵是紧密相关的。根据Charles H. Bennett对Maxwell's Demon的重新解释，对信息的销毁是一个不可逆过程，所以销毁信息是符合[热力学第二定律](https://baike.baidu.com/item/%E7%83%AD%E5%8A%9B%E5%AD%A6%E7%AC%AC%E4%BA%8C%E5%AE%9A%E5%BE%8B" \t "_blank)的。而产生信息，则是为系统引入负（[热力学](https://baike.baidu.com/item/%E7%83%AD%E5%8A%9B%E5%AD%A6)）熵的过程。所以信息熵的符号与热力学熵应该是相反的。

一般而言，当一种信息出现概率更高的时候，表明它被传播得更广泛，或者说，被引用的程度更高。我们可以认为，从信息传播的角度来看，信息熵可以表示信息的价值。这样子我们就有一个衡量信息价值高低的标准，可以做出关于知识流通问题的更多推论。

**计算公式**：

H(x) = E[I(xi)] = E[ log(2,1/p(xi)) ] = -∑p(xi)log(2,p(xi)) (i=1,2,..n)

在信源中，考虑的不是某一单个符号发生的不确定性，而是要考虑这个信源所有可能发生情况的平均不确定性。若信源符号有n种取值：U 1 …U i …U n ，对应概率为：P 1 …Pi…P n ，且各种符号的出现彼此独立。这时，信源的平均不确定性应当为单个符号不确定性-logP i 的统计平均值（E），可称为**信息熵**，即   https://gss0.bdstatic.com/94o3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D244/sign=49cc170a35d3d539c53d08c70e86e927/2e2eb9389b504fc2554487f1e6dde71190ef6d2e.jpg  ，式中对数一般取2为底，单位为**比特**。但是，也可以取其它对数底，采用其它相应的单位，它们间可用换底公式换算。

4.对自己的数据集进行分析，根据自己的分析目标，列出自己数据集中可离散化的列。注意：离散化的本质是把连续单数字列转为标记列，或概念层次：国家-州-城市-街道

答：

性别：男（1）；

女（2）。

年龄：1-10岁（0）；

11-20岁（1）；

21-30岁（2）；

31-40岁（3）；

41-50岁（4）；

51-60岁（5）；

61-70岁（6）。

是否大学生：是（1）；

否（2）。

购物深度 ：浅层用户，购物频率较低，一个月购物次数在0~5之间（1），

中层用户，购物次数在5~10次/月（2）；

深度用户，10次以上/月（3）。

消费档次：低档（1）；

中档（2）；

高档（3）。

城市层级：一线城市（1）；

二线城市（2）；

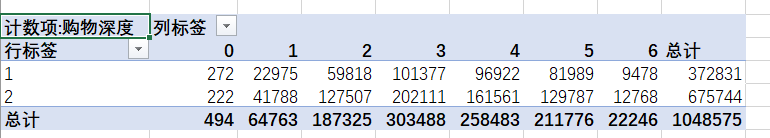
三线城市（3）；

四线城市（4）。

5.根据自己的数据集，结合自己的挖掘目标，构造数据立方，挖掘有用的结果。（Excel中的数据透视表）

答：

表中行标签为性别（男性为1，女性为2），列标签为年龄层次；



可以看出女性在购物总数中占据比例大于男性所占比，在年龄层次1-4随着年龄的增大，男女差距越来越大，年龄层次在4-6男女差距又逐渐减小。

6.结合作业5，尝试在自己选择的数据集中使用上卷、下钻、切片、切块，并分析有用的结果。

答：

上卷： 所有数据总和为1048575，其中男性总和372831 ，女性总和 675744

下钻：所有数据总和1048575，分成的不同年龄段的总和依次是494，64796，187325，303488 ，258483 ，211776 ，22246

切片：选择男性维度进行切片，男性在不同年龄段的购物深度的分析

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 浅层用户 | 45 | 4168 | 4114 | 7635 | 8329 | 9235 | 1330 |
| 中度用户 | 60 | 6821 | 10722 | 17833 | 17181 | 16105 | 1858 |
| 深度用户 | 167 | 11986 | 44982 | 75909 | 71412 | 56649 | 6290 |
| sum | 272 | 22975 | 59818 | 101377 | 96922 | 81989 | 9478 |

女性：年龄层次\*购物深度

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 浅 | 36 | 4329 | 4295 | 8221 | 8975 | 11318 | 1669 |
| 中 | 30 | 7614 | 11610 | 19907 | 18585 | 19149 | 2094 |
| 深 | 156 | 29845 | 111602 | 173983 | 134001 | 99320 | 99320 |
| sum | 222 | 41788 | 127507 | 202111 | 161561 | 129787 | 12768 |

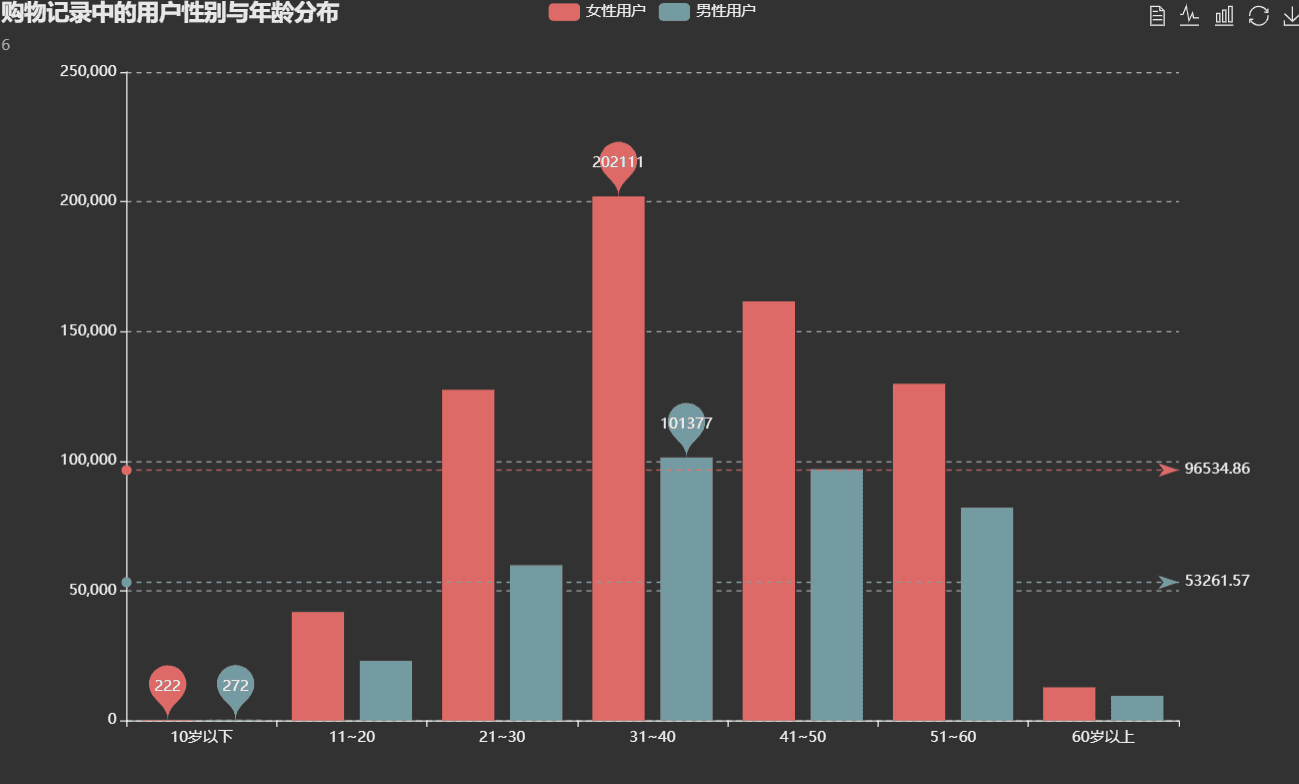
切块：男性购物深度的计数总和372831，男性在不同年龄段购物深度计数项依次为272 ，22975， 59818， 101377， 96922， 81989 ，9478；

结论：女性用户占约70%，而男性用户只占到了约30%，消费市场发展空间较大，其中30~40岁的用户为购物的主要人群，针对男性用户购物市场，由他们的购物深度分布可以看出：同等年龄段，购物深度越高，用户越多，30岁左右的用户偏向于频繁购物，消费的价格也偏高，所以对于这类用户，网站可以多推荐他们一些价格偏高的商品，或发放购物优惠等方式来促进商品消费。

3/6

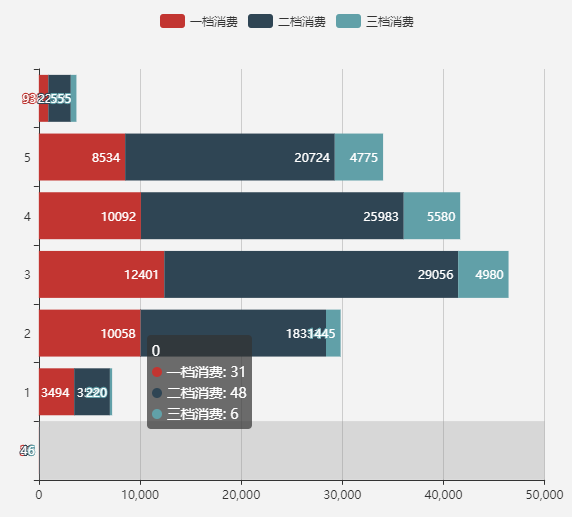
1、结合上次作业你们完成的数据立方体，形成可视化成果。

答：



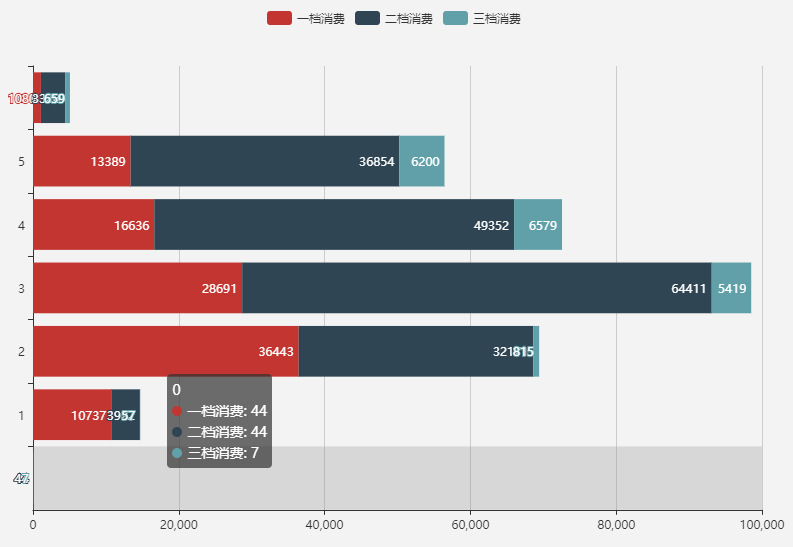
关于男性用户各年龄层次的购物层次的比例分布

y轴为年龄层次（0~6层），X为人数



关于女性用户各年龄层次的购物层次的比例分布

y轴为年龄层次（0~6层），X为人数



2、把作业1中的可视化结果结合你们的实训项目。

答：

根据前面的分析结果，可以观测到这个样本中各个年龄层次的男女用户的购买力水平分布，由此在自己的网站商城推荐时，我们可以参考这个普遍用户的样本，根据自身网站用户的资料信息来推荐价位与其购买力相适的热门商品，更好地实现商品智能推荐功能

例如：20岁以下用户以低等价位的商品推荐为主，20岁以上的用户以中等价位的商品推荐为主；可重点关注30~59岁人群，此类人群通常财力自由而且有一定积蓄，这也使得他们对于高价商品的接受能力较高，可多推荐给这类用户一些高档商品

同时，随用户年龄的增长，消费档次在提升，推荐的商品价格可随之有所上涨。

3、搜索在线数据挖掘工具，并试用。

图说在线工具

ensensoft在线工具

图表秀在线工具

4.数据挖掘算法（机器学习算法）监督式学习，非监督式学习的概念理解

**监督学习**： 可以由训练资料中学到或建立一个模式（函数），并依此模式推测新的实例。

对具有概念标记（分类）的训练样本进行学习，以尽可能对训练样本集外的数据进行标记（分类）预测。这里，所有的标记（分类）是已知的。因此，训练样本的岐义性低。

**非监督学习**： 由于在很多实际应用中，缺少所研究对象类别形成过程的知识，或者为了判断各个样本(模式)所属的类别需要很大的工作量(例如卫星遥感照片上各像元所对应的地面情况)，因此往往只能用无类别标答的样本集进形学习。通过无监督式学习，把样本集划分为若干个子集(类别)，从而直接解决看样本的分类问题，或者把它作为训练样本集，再用监督学习的方法进行分类器设计。

对没有概念标记（分类）的训练样本进行学习，以发现训练样本集中的结构性知识。这里，所有的标记（分类）是未知的。因此，训练样本的岐义性高。聚类就是典型的无监督学习。（ 将物理或抽象对象的集合分成由类似的对象组成的多个类的过程被称为聚类。）

5.找机器学习挖掘的案例，包含java实现代码 ，理解，课堂讲解。

注意：了解算法的本质，主要关注这个算法能做什么，怎么应用？

knn、朴素贝叶斯、 线性回归 （逻辑回归）、svm，决策树、关联规则……

答：

KNN是一种分类算法。算法的主要过程为：

1、给定一个训练集数据，每个训练集数据都是已经分好类的。

2、设定一个初始的测试数据a，计算a到训练集所有数据的欧几里得距离，并排序。

3、选出训练集中离a距离最近的K个训练集数据。

4、比较k个训练集数据，选出里面出现最多的分类类型，此分类类型即为最终测试数据a的分类。

KNN算法实现

首先测试数据需要2块，1个是训练集数据，就是已经分好类的数据，比如上图中的非绿色的点。还有一个是测试数据，就是上面的绿点，当然这里的测试数据不会是一个，而是一组。这里的数据与数据之间的距离用数据的特征向量做计算，特征向量可以是多维度的。通过计算特征向量与特征向量之间的欧几里得距离来推算相似度。定义训练集数据trainInput.txt:

a 1 2 3 4 5

b 5 4 3 2 1

c 3 3 3 3 3

d -3 -3 -3 -3 -3

a 1 2 3 4 4

b 4 4 3 2 1

c 3 3 3 2 4

d 0 0 1 1 -2

待测试数据testInput，只有特征向量值：

1 2 3 2 4

2 3 4 2 1

8 7 2 3 5

-3 -2 2 4 0

-4 -4 -4 -4 -4

1 2 3 4 4

4 4 3 2 1

3 3 3 2 4

0 0 1 1 -2

下面是主程序：

package DataMing\_KNN;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collection;

import java.util.Collections;

import java.util.Comparator;

import java.util.HashMap;

import java.util.Map;

import org.apache.activemq.filter.ComparisonExpression;

/\*\*

 \* k最近邻算法工具类

 \*

 \* @author lyq

 \*

 \*/

public class KNNTool {

// 为4个类别设置权重，默认权重比一致

public int[] classWeightArray = new int[] { 1, 1, 1, 1 };

// 测试数据地址

private String testDataPath;

// 训练集数据地址

private String trainDataPath;

// 分类的不同类型

private ArrayList<String> classTypes;

// 结果数据

private ArrayList<Sample> resultSamples;

// 训练集数据列表容器

private ArrayList<Sample> trainSamples;

// 训练集数据

private String[][] trainData;

// 测试集数据

private String[][] testData;

public KNNTool(String trainDataPath, String testDataPath) {

this.trainDataPath = trainDataPath;

this.testDataPath = testDataPath;

readDataFormFile();

}

/\*\*

\* 从文件中阅读测试数和训练数据集

\*/

private void readDataFormFile() {

ArrayList<String[]> tempArray;

tempArray = fileDataToArray(trainDataPath);

trainData = new String[tempArray.size()][];

tempArray.toArray(trainData);

classTypes = new ArrayList<>();

for (String[] s : tempArray) {

if (!classTypes.contains(s[0])) {

// 添加类型

classTypes.add(s[0]);

}

}

tempArray = fileDataToArray(testDataPath);

testData = new String[tempArray.size()][];

tempArray.toArray(testData);

}

/\*\*

\* 将文件转为列表数据输出

\*

\* @param filePath

\*            数据文件的内容

\*/

private ArrayList<String[]> fileDataToArray(String filePath) {

File file = new File(filePath);

ArrayList<String[]> dataArray = new ArrayList<String[]>();

try {

BufferedReader in = new BufferedReader(new FileReader(file));

String str;

String[] tempArray;

while ((str = in.readLine()) != null) {

tempArray = str.split(" ");

dataArray.add(tempArray);

}

in.close();

} catch (IOException e) {

e.getStackTrace();

}

return dataArray;

}

/\*\*

\* 计算样本特征向量的欧几里得距离

\*

\* @param f1

\*            待比较样本1

\* @param f2

\*            待比较样本2

\* @return

\*/

private int computeEuclideanDistance(Sample s1, Sample s2) {

String[] f1 = s1.getFeatures();

String[] f2 = s2.getFeatures();

// 欧几里得距离

int distance = 0;

for (int i = 0; i < f1.length; i++) {

int subF1 = Integer.parseInt(f1[i]);

int subF2 = Integer.parseInt(f2[i]);

distance += (subF1 - subF2) \* (subF1 - subF2);

}

return distance;

}

/\*\*

\* 计算K最近邻

\* @param k

\* 在多少的k范围内

\*/

public void knnCompute(int k) {

String className = "";

String[] tempF = null;

Sample temp;

resultSamples = new ArrayList<>();

trainSamples = new ArrayList<>();

// 分类类别计数

HashMap<String, Integer> classCount;

// 类别权重比

HashMap<String, Integer> classWeight = new HashMap<>();

// 首先讲测试数据转化到结果数据中

for (String[] s : testData) {

temp = new Sample(s);

resultSamples.add(temp);

}

for (String[] s : trainData) {

className = s[0];

tempF = new String[s.length - 1];

System.arraycopy(s, 1, tempF, 0, s.length - 1);

temp = new Sample(className, tempF);

trainSamples.add(temp);

}

// 离样本最近排序的的训练集数据

ArrayList<Sample> kNNSample = new ArrayList<>();

// 计算训练数据集中离样本数据最近的K个训练集数据

for (Sample s : resultSamples) {

classCount = new HashMap<>();

int index = 0;

for (String type : classTypes) {

// 开始时计数为0

classCount.put(type, 0);

classWeight.put(type, classWeightArray[index++]);

}

for (Sample tS : trainSamples) {

int dis = computeEuclideanDistance(s, tS);

tS.setDistance(dis);

}

Collections.sort(trainSamples);

kNNSample.clear();

// 挑选出前k个数据作为分类标准

for (int i = 0; i < trainSamples.size(); i++) {

if (i < k) {

kNNSample.add(trainSamples.get(i));

} else {

break;

}

}

// 判定K个训练数据的多数的分类标准

for (Sample s1 : kNNSample) {

int num = classCount.get(s1.getClassName());

// 进行分类权重的叠加，默认类别权重平等，可自行改变，近的权重大，远的权重小

num += classWeight.get(s1.getClassName());

classCount.put(s1.getClassName(), num);

}

int maxCount = 0;

// 筛选出k个训练集数据中最多的一个分类

for (Map.Entry entry : classCount.entrySet()) {

if ((Integer) entry.getValue() > maxCount) {

maxCount = (Integer) entry.getValue();

s.setClassName((String) entry.getKey());

}

}

System.out.print("测试数据特征：");

for (String s1 : s.getFeatures()) {

System.out.print(s1 + " ");

}

System.out.println("分类：" + s.getClassName());

}

}

}

Sample样本数据类：

package DataMing\_KNN;

/\*\*

 \* 样本数据类

 \*

 \* @author lyq

 \*

 \*/

public class Sample implements Comparable<Sample>{

// 样本数据的分类名称

private String className;

// 样本数据的特征向量

private String[] features;

//测试样本之间的间距值，以此做排序

private Integer distance;

public Sample(String[] features){

this.features = features;

}

public Sample(String className, String[] features){

this.className = className;

this.features = features;

}

public String getClassName() {

return className;

}

public void setClassName(String className) {

this.className = className;

}

public String[] getFeatures() {

return features;

}

public void setFeatures(String[] features) {

this.features = features;

}

public Integer getDistance() {

return distance;

}

public void setDistance(int distance) {

this.distance = distance;

}

@Override

public int compareTo(Sample o) {

// TODO Auto-generated method stub

return this.getDistance().compareTo(o.getDistance());

}

}

测试场景类：

/\*\*

 \* k最近邻算法场景类型

 \* @author lyq

 \*

 \*/

public class Client {

public static void main(String[] args){

String trainDataPath = "C:\\Users\\lyq\\Desktop\\icon\\trainInput.txt";

String testDataPath = "C:\\Users\\lyq\\Desktop\\icon\\testinput.txt";

KNNTool tool = new KNNTool(trainDataPath, testDataPath);

tool.knnCompute(3);

}

}

执行的结果为：

测试数据特征：1 2 3 2 4 分类：a

测试数据特征：2 3 4 2 1 分类：c

测试数据特征：8 7 2 3 5 分类：b

测试数据特征：-3 -2 2 4 0 分类：a

测试数据特征：-4 -4 -4 -4 -4 分类：d

测试数据特征：1 2 3 4 4 分类：a

测试数据特征：4 4 3 2 1 分类：b

测试数据特征：3 3 3 2 4 分类：c

测试数据特征：0 0 1 1 -2 分类：d

KNN算法的注意点：

1、knn算法的训练集数据必须要相对公平，各个类型的数据数量应该是平均的，否则当A数据由1000个B数据由100个，到时无论如何A数据的样本还是占优的。

2、knn算法如果纯粹凭借分类的多少做判断，还是可以继续优化的，比如近的数据的权重可以设大，最后根据所有的类型权重和进行比较，而不是单纯的凭借数量。

3、knn算法的缺点是计算量大，这个从程序中也应该看得出来，里面每个测试数据都要计算到所有的训练集数据之间的欧式距离，时间复杂度就已经为O(n\*n)，如果真实数据的n非常大，这个算法的开销的确态度，所以KNN不适合大规模数据量的分类。

3/7

第八次作业

1.监督式学习：基本概念深入理解，用自己的话描述一个案例。

前提：要有带标签的数据。什么是标签？训练集，测试集？（怎么得到，怎么用的）

模型学习：找一种分类算法讲解如何学习？（贝叶斯、knn、SVM等）

模型评价：欠拟合，过拟合？

答：

**监督式学习的概念：**使用预定义的“**训练示例**”集合，训练系统，便于其在**新数据**被馈送时也能得出结论。系统一直被训练，直到达到所需的精度水平

**前提：**

**标签（label）**：数据的标签。举例*：*书属于的类别，例如“计算机”“图形学”“英文书”“教材”等。

**训练集** 用来估计模型的数据

**测试集** 则检验最终选择最优的模型的性能如何的数据。

一般两者都是从总样本数据集中随机产生的，训练集会多于测试集

**学习（learning）** ：将很多数据丢给计算机分析，以此来训练该计算机，培养计算机给数据分类的能力。换句话说，学习指的就是找到特征与标签的映射（mapping）关系。这样当有特征而无标签的未知数据输入时，我们就可以通过已有的关系得到未知数据标签。

**监督式学习**不仅把**训练数据**丢给计算机，而且还把分类的结果（数据具有的**标签**）也一并丢给计算机分析。 计算机进行学习之后，再丢给它新的**未知的数据**，它也能计算出该数据导致各种结果的概率，给你一个最接近正确的结果。 由于计算机在学习的过程中不仅有训练数据，而且有训练结果（标签），因此训练的效果通常不错。

**举例**：不仅把书给学生进行训练给书本分类的能力，而且把分类的结果（哪本书属于哪些类别）也给了学生做标准参考。老师给学生一堆书，然后告诉学生们把这些书分为漫画书、科学书、教科书三类。已知书的类别然后对书进行分类。标签是漫画书、科学书、教科书这三个类别，训练集是老师给的这一堆书，测试集是再给出一些其他的书，然后进行分类

**定义**：指有求知欲的学生从老师那里获取知识、信息，老师提供对错指示、告知最终答案的学习过程。 在机器学习中，计算机 = 学生，周围的环境 = 老师。

**最终目标**：根据在学习过程中获得的经验技能，对没学习过的问题也可以做出正确解答，使计算机获得这种泛化能力。

**应用**：手写文字识别、声音处理、图像处理、垃圾邮件分类与拦截、网页检索、基因诊断、股票预测等。

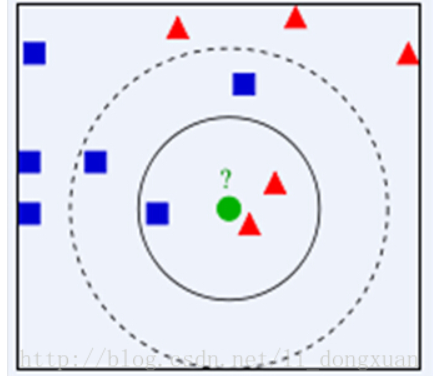
**典型任务**：预测数值型数据的回归、预测分类标签的分类、预测顺序的排列。

**模型算法：knn算法进行分类 如何进行学习呢？**

对于已知的图书类别，把图书进行分类，然后根据类别对于未知的图书进行分类

**K-近邻算法（k-Nearest Neighbors，KNN）**

K-近邻是一种分类算法，其思路是：如果一个样本在特征空间中的k个最相似(即特征空间中最邻近)的样本中的大多数属于某一个类别，则该样本也属于这个类别。K通常是不大于20的整数。KNN算法中，所选择的邻居都是已经正确分类的对象。该方法在定类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。



如上图，绿色圆要被决定赋予哪个类，是红色三角形还是蓝色四方形？如果K=3，由于红色三角形所占比例为2/3，绿色圆将被赋予红色三角形那个类，如果K=5，由于蓝色四方形比例为3/5，因此绿色圆被赋予蓝色四方形类。

算法的步骤为：

（1）计算**[测试](http://lib.csdn.net/base/softwaretest" \t "_blank)**数据与各个训练数据之间的距离；

（2）按照距离的递增关系进行排序；

（3）选取距离最小的K个点；

（4）确定前K个点所在类别的出现频率；

（5）返回前K个点中出现频率最高的类别作为测试数据的预测分类。

**模型评价：**

knn算法在分类时主要的不足是，当样本不平衡时，如果一个类的样本容量很大，而其他类样本容量很小时，有可能导致当输入一个新样本时，该样本的 K 个邻居中大容量类的样本占多数。

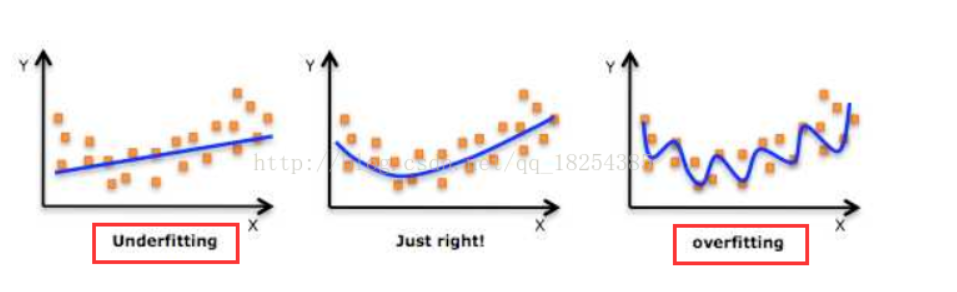
K 值会对算法的结果产生重大影响。K值较小意味着只有与输入实例较近的训练实例才会对预测结果起作用，容易发生过拟合；如果 K 值较大，优点是可以减少学习的估计误差，缺点是学习的近似误差增大，这时与输入实例较远的训练实例也会对预测起作用，是预测发生错误。**在实际应用中，K 值一般选择一个较小的数值，通常采用交叉验证的方法来选择最有的 K 值。**

**欠拟合：** 是指模型拟合程度不高，数据距离拟合曲线较远，或指模型没有很好地捕捉到数据特征，不能够很好地拟合数据。

**过拟合：** 过拟合是指为了得到一致假设而使假设变得过度严格。避免过拟合是分类器设计中的一个核心任务。通常采用增大数据量和测试样本集的方法对分类器性能进行评价。

例如下图，根据散点来模拟曲线，图一中的曲线拟合度过低，大部分数据未能贴合

而图三则过分贴合每个点，使得曲线过分不规律，也不能较为准确的预测数据走势



2.无监督学习：基本概念深入学习，用自己的话描述一个案例。

前提：无标签的数据。是否需要划分训练集和测试集?

模型学习：找一种聚类算法讲解如何学习？（层次聚类，k-means等）

模型评价：针对k-means方法（聚类算法），讲解如何评价无监督学习方法。

答：

**无监督式学习的概念：**

给系统一堆无标签数据，它必须自己检测模式和关系。 系统要用推断功能来描述未分类数据的模式。

只给计算机**训练数据**，不给结果（标签），因此计算机无法准确地知道哪些数据具有哪些标签，只能凭借强大的计算能力分析数据的特征，从而得到一定的成果，通常是得到一些集合，集合内的数据在某些特征上相同或相似。

**举例**：只给学生进行未分类的书本进行训练，不给标准参考，学生只能自己分析哪些书比较像，根据相同与相似点列出清单，说明哪些书比较可能是同一类别的。老师给学生一堆书，不告诉学生该怎么分类，学生需要自己分析哪些书比较相似，然后自己总结类别，进行分类。训练集是老师给的一堆书，需要学生自己分类。

**定义**：指在没有老师的情况下，学生自学的过程。 在机器学习中，计算机从互联网中自动收集信息，并获取有用信息。

**最终目标**：无监督学习不局限于解决有正确答案的问题，所以目标可以不必十分明确。

**应用**：人造卫星故障诊断、视频分析、社交网站解析、声音信号解析、数据可视化、监督学习的前处理工具等。

**典型任务**：聚类、异常检测。

**模型学习：K-means：**通常被称为劳埃德算法，这在数据聚类中是最经典的，也是相对容易理解的模型。算法执行的过程分为4个阶段。

1.首先，随机设K个特征空间内的点作为初始的聚类中心。（人群中选k个代表）

2.然后，对于根据每个数据的特征向量，从K个聚类中心中寻找距离最近的一个，并且把该数据标记为这个聚类中心。（其余人离哪个代表最近就归为哪个代表的团队）

3.接着，在所有的数据都被标记过聚类中心之后，根据这些数据新分配的类簇，通过取分配给每个先前质心的所有样本的平均值来创建新的质心重,新对K个聚类中心做计算。（第一次站队完成后，再从每个小团体中重选最适中的新代表）

4.最后，计算旧和新质心之间的差异,如果所有的数据点从属的聚类中心与上一次的分配的类簇没有变化，那么迭代就可以停止，否则回到步骤2继续循环。（新代表选完，再次进行步骤2，若每个团队成员不变了，则聚类结束；若成员发生变动，则再次站队，再次重选团队代表）

**K-Means算法的优缺点：**

1、首先优点当然是算法简单，快速，易懂，没有涉及到特别复杂的数据结构。

2、缺点1是最开始K的数量值以及K个聚类中心点的设置不好定，往往开始时不同的k个中心点的设置对后面迭代计算的走势会有比较大的影响，这时候可以考虑根据类的自动合并和分裂来确定这个k。

3、缺点2由于计算是迭代式的，而且计算距离的时候需要完全遍历一遍中心点，当数据规模比较大的时候，开销就显得比较大了。

3/11

第九次作业

**1.理解关联规则的基本概念，举3个关联规则的例子**

答:

**关联规则基本概念**：关联规则是形如X→Y的蕴涵式，其中， X和Y分别称为关联规则的先导和后继 ，反映一个事物与其他事物之间的相互依存性和关联性。

**例子1：购物篮分析**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TID | 网球拍 | 网 球 | 运动鞋 | 羽毛球 |
| 1 | 1 | 1 | 1 | 0 |
| 2 | 1 | 1 | 0 | 0 |
| 3 | 1 | 0 | 0 | 0 |
| 4 | 1 | 0 | 1 | 0 |
| 5 | 0 | 1 | 1 | 1 |
| 6 | 1 | 1 | 0 | 0 |

用一个简单的例子说明。表1是顾客购买记录的数据库D，包含6个事务。项集I={网球拍,网球,运动鞋,羽毛球}。考虑关联规则（频繁二项集）：网球拍与网球，事务1,2,3,4,6包含网球拍，事务1,2,6同时包含网球拍和网球，X^Y=3, D=6，支持度(X^Y)/D=0.5；X=5, 置信度(X^Y)/X=0.6。若给定最小支持度α = 0.5，最小置信度β = 0.6，认为购买网球拍和购买网球之间存在关联。

**例子2：穿衣搭配推荐**

      穿衣搭配是服饰鞋包导购中非常重要的课题，基于搭配专家和达人生成的搭配组合数据，百万级别的商品的文本和图像数据，以及用户的行为数据。期待能从以上行为、文本和图像数据中挖掘穿衣搭配模型，为用户提供个性化、优质的、专业的穿衣搭配方案，预测给定商品的搭配商品集合。

**例子3：毒蘑菇特征**

带有毒性的蘑菇通常会具有一些相似的特征，例如：气味、菌环大小、菌身颜色，特定生长环境等等，对于一类我们未知的菌类，基于这些关联规则的归总我们就可以大致推断出它的毒性

**例子4：基于兴趣的实时新闻推荐**

     随着近年来互联网的飞速发展，个性化推荐已成为各大主流网站的一项必不可少服务。提供各类新闻的门户网站是互联网上的传统服务，但是与当今蓬勃发展的电子商务网站相比，新闻的个性化推荐服务水平仍存在较大差距。初步研究发现，同一个用户浏览的不同新闻的内容之间会存在一定的相似性和关联，物理世界完全不相关的用户也有可能拥有类似的新闻浏览兴趣。此外，用户浏览新闻的兴趣也会随着时间变化，这给推荐系统带来了新的机会和挑战。

     因此，希望通过对带有时间标记的**用户浏览行为**和**新闻文本内容**进行分析，挖掘用户的新闻浏览模式和变化规律，设计及时准确的推荐系统预测用户未来可能感兴趣的新闻。

**2.描述apriori算法整体结构，什么叫做支持度，可信度**

答：Apriori算法是用来**发现频繁项集**的一种方法

Apriori算法的名字正是基于这样的事实：算法使用频繁项集性质的先验性质，即频繁项集的所有非空子集也一定是频繁的。Apriori算法使用一种称为**逐层搜索**的迭代方法，其中k项集用于探索(k+1)项集。首先，通过扫描数据库，累计每个项的计数，并收集满足最小支持度的项，找出频繁1项集的集合。该集合记为L1。然后，使用L1找出频繁2项集的集合L2，使用L2找出L3，如此下去，直到不能再找到频繁k项集。每找出一个Lk需要一次数据库的完整扫描。Apriori算法使用频繁项集的先验性质来压缩搜索空间。

输入项：最小支持度和数据集

假设现在的问题是判断A和B之间相关关系的强弱；

**支持度**：说的是所有事务中，A和B同时出现的次数与总的事务数的比例。

例如：support（A,B）=A,B同时出现事务数/样品的中总事务数

**可信度**：A⇒B的置信度说的是包含A的事务中，同时也包含B的事务所占的比例。用以下公式计算：

confidence(A->B)=A,B同时出现的事务数/含有A的事务数

比如说，总共100个事务，有50个包含A，而这50个事务当中，又有20个同时也包含B，那么，confidence(A⇒B)=40%

**算法主要流程：**

首先，找出频繁“1项集”的集合，该集合记作L1。L1用于找频繁“2项集”的集合L2，而L2用于找L3。如此下去，直到不能找到“K项集”。找每个Lk都需要一次数据库扫描。

**核心思想**是：连接步和剪枝步。连接步是自连接，原则是保证前k-2项相同，并按照字典顺序连接。剪枝步，是使任一频繁项集的所有非空子集也必须是频繁的。反之，如果某个候选的非空子集不是频繁的，那么该候选肯定不是频繁的，从而可以将其从CK中删除。

**3.找一个关联规则的数据集，通过apriori案例的搜索，找到一些关联规则**

当气味是almond、anise、none时,判断该蘑菇的type是edible（可食用）的蘑菇；

当气味是creosote、fishy、foul、musty、pungent、spicy时,判断该蘑菇的type是poisonous（有毒）的蘑菇。

其中，6500条记录中有6402条是通过规则判断正确的。

采用80%的数据作为训练样本集，最终可得共9条规则：

¬规则1：当odor（气味）为foul时，蘑菇是poisonous（有毒的）。其中有2160条记录，每条记录都是正确的，无错误记录；

¬规则2：gill\_size（菌褶的大小）为narrow（狭窄的）及gill\_color（菌褶的颜色）是buff（浅黄色）时，蘑菇是poisonous（有毒的）。其中有1152条记录，无错误记录；

¬规则3：gill\_size（菌褶的大小）为narrow（狭窄的）及odor（气味）是pungent（辛辣的）时，蘑菇是poisonous（有毒的）。其中有256条记录无错误记录；

¬规则4：odor（气味）是creosote（碳酸）时，蘑菇是poisonous（有毒的）。其中有192条记录，无错误记录；

¬规则5：spore\_print\_color（孢子颜色）是green时，蘑菇是poisonous（有毒的）。其中有72条记录，无错误记录；

¬规则6：stalk\_surface\_below\_ring（茎的下表面）是scaly（粗糙）和stalk\_surface\_above\_ring（茎的上表面）是silky（光滑）时，蘑菇是poisonous（有毒的）。其中有68条记录，无错误记录；

¬规则7：habitat（产地）是leaves（树叶）和cap\_color（蘑菇帽颜色）是white时，蘑菇是poisonous（有毒的）。其中有8条记录，无错误记录；

¬规则8：stalk\_color\_above\_ring（上茎颜色）是yellow时，蘑菇是poisonous（有毒的）。其中有8条记录，无错误记录；

¬规则9：其余的均为edible（可食用）的蘑菇。其中有4208条记录，无错误记录。

剩余20%作为测试样本集的预测结果:

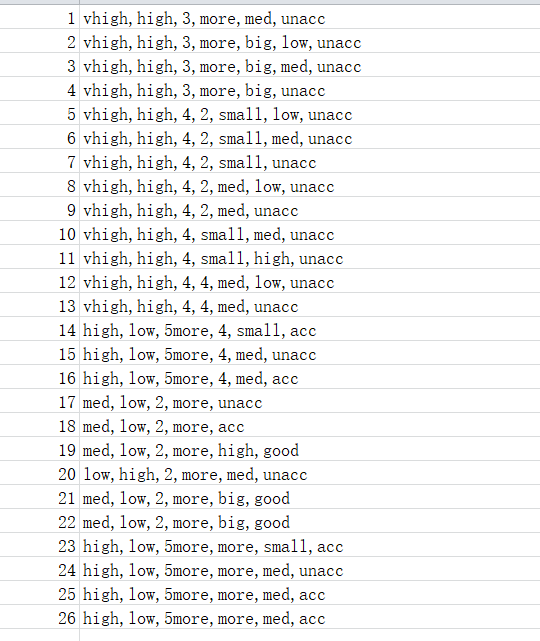
通过该结果可看出预测结果中的type为edible的准确率达100%，而判断type为poisonous的有761条数据是正确的，有22条数据判断错误，准确率达97.2%

3/12

1、Apriori的算法步骤：

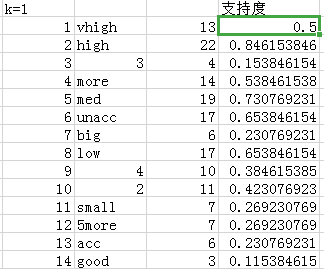
1-1 找所有的候选k项集（频繁k项集是由频繁k-1项集组成的）

描述汽车的一些样本数据（包含购价，卖出价，门数，可乘人数，后备箱大小，安全系数）



以一项集为例：

1-2得到所有的候选k项集的支持度或可信度



1-3根据最小支持度或可信度阈值来进行频繁k项集筛选

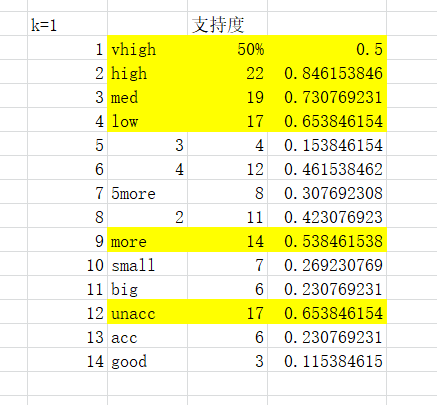
设定最小支持度：0.50

频繁一项集筛选结果：vhigh、high、med、more、low、unacc

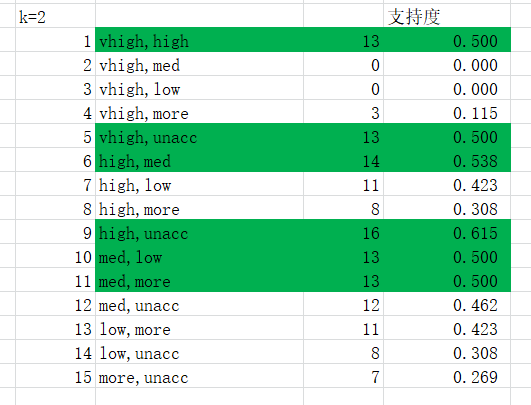
2、构造频繁项集的TDB数据库，设置一个支持度阈值，通过ppt上第21页的例子，自己手动完成频繁多项集的筛选和检索。

支持度阈值：大于等于0.5

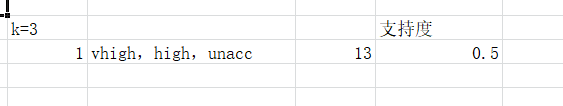
频繁一项集：黄色为筛选完后的



频繁二项集：标绿色的为筛选后的频繁二项集



连接剪枝后的频繁三项集为：



3、找一个关联规则的数据集，通过Apriori案例的搜索，找到一些关联规则，根据我们上课的知识把实现代码读懂。

/\*\*

 \* Apriori算法实现 最大模式挖掘，涉及到支持度，但没有置信度计算

 \*

 \* AssociationRulesMining()函数实现置信度计算和关联规则挖掘

 \*/

public class AprioriMyself {

    public static  int times=0;//迭代次数

    private static  double MIN\_SUPPROT = 0.02;//最小支持度百分比

    private static   double MIN\_CONFIDENCE=0.6;//最小置信度

    private static boolean endTag = false;//循环状态，迭代标识

    static List<List<String>> record = new ArrayList<List<String>>();//数据集

    static  List<List<String>> frequentItemset=new ArrayList<>();//存储所有的频繁项集

    static List<Mymap> map = new ArrayList();//存放频繁项集和对应的支持度技术

    public static void main(String args[]){

        System.out.println("请输入最小支持度（如0.05）和最小置信度（如0.6）");

        Scanner in=new Scanner(System.in);

        MIN\_SUPPROT=in.nextDouble();

        MIN\_CONFIDENCE=in.nextDouble();

        /\*\*\*\*\*\*\*\*\*\*\*\*\*读取数据集\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

        record = getRecord("top1000data");

        //控制台输出记录

        System.out.println("读取数据集record成功===================================");

        ShowData(record);

        Apriori();//调用Apriori算法获得频繁项集

        System.out.println("频繁模式挖掘完毕。\n\n\n\n\n进行关联度挖掘，最小支持度百分比为："+MIN\_SUPPROT+"  最小置信度为："+MIN\_CONFIDENCE);

         AssociationRulesMining();//挖掘关联规则

    }

    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

     \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*读取数据\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

    public static List<List<String>> getRecord(String url) {

        List<List<String>> record = new ArrayList<List<String>>();

        try {

            String encoding = "UTF-8"; // 字符编码(可解决中文乱码问题 )

            File file = new File(url);

            if (file.isFile() && file.exists()) {

                InputStreamReader read = new InputStreamReader(

                        new FileInputStream(file), encoding);

                BufferedReader bufferedReader = new BufferedReader(read);

                String lineTXT = null;

                while ((lineTXT = bufferedReader.readLine()) != null) {//读一行文件

                    String[] lineString = lineTXT.split(",");

                    List<String> lineList = new ArrayList<String>();

                    for (int i = 0; i < lineString.length; i++) {

                        lineList.add(lineString[i]);

                    }

                    record.add(lineList);

                }

                read.close();

            } else {

                System.out.println("找不到指定的文件！");

            }

        } catch (Exception e) {

            System.out.println("读取文件内容操作出错");

            e.printStackTrace();

        }

        return record;

    }

 public static void Apriori()           /\*\*实现apriori算法\*\*/

    {

        //\*\*\*\*\*\*\*\*\*\*\*\*获取候选1项集\*\*\*\*\*\*\*\*\*\*\*\*\*\*

        System.out.println("第一次扫描后的1级 备选集CandidateItemset");

        List<List<String>> CandidateItemset = findFirstCandidate();

        ShowData(CandidateItemset);

        //\*\*\*\*\*\*\*\*\*\*\*\*获取频繁1项集\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

        System.out.println("第一次扫描后的1级 频繁集FrequentItemset");

        List<List<String>> FrequentItemset = getSupprotedItemset(CandidateItemset);

        AddToFrequenceItem(FrequentItemset);//添加到所有的频繁项集中

        //控制台输出1项频繁集

        ShowData(FrequentItemset);

         //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*迭代过程\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

        times=2;

        while(endTag!=true){

            System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*第"+times+"次扫描后备选集");

            //\*\*\*\*\*\*\*\*\*\*连接操作\*\*\*\*获取候选times项集\*\*\*\*\*\*\*\*\*\*\*\*\*\*

            List<List<String>> nextCandidateItemset = getNextCandidate(FrequentItemset);

            //输出所有的候选项集

            ShowData(nextCandidateItemset);

            /\*\*\*\*\*\*\*\*\*\*\*\*\*\*计数操作\*\*\*由候选k项集选择出频繁k项集\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

            System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*第"+times+"次扫描后频繁集");

            List<List<String>> nextFrequentItemset = getSupprotedItemset(nextCandidateItemset);

            AddToFrequenceItem(nextFrequentItemset);//添加到所有的频繁项集中

            //输出所有的频繁项集

            ShowData(nextFrequentItemset);

            //\*\*\*\*\*\*\*\*\*如果循环结束，输出最大模式\*\*\*\*\*\*\*\*\*\*\*\*\*\*

            if(endTag == true){

                System.out.println("\n\n\nApriori算法--->最大频繁集==================================");

                ShowData(FrequentItemset);

            }

            //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*下一次循环初值\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

            FrequentItemset = nextFrequentItemset;

            times++;//迭代次数加一

        }

    }

    public static void AssociationRulesMining()//关联规则挖掘

    {

        for(int i=0;i<frequentItemset.size();i++)

        {

            List<String> tem=frequentItemset.get(i);

            if(tem.size()>1) {

                List<String> temclone=new ArrayList<>(tem);

                List<List<String>> AllSubset = getSubSet(temclone);//得到频繁项集tem的所有子集

                for (int j = 0; j < AllSubset.size(); j++) {

                    List<String> s1 = AllSubset.get(j);

                    List<String> s2 = gets2set(tem, s1);

                    double conf = isAssociationRules(s1, s2, tem);

                    if (conf > 0)

                        System.out.println("置信度为：" + conf);

                }

            }

            }

        }

    public  static  double isAssociationRules(List<String> s1,List<String> s2,List<String> tem)//判断是否为关联规则

    {

        double confidence=0;

        int counts1;

        int countTem;

        if(s1.size()!=0&&s1!=null&&tem.size()!=0&&tem!=null)

        {

            counts1= getCount(s1);

            countTem=getCount(tem);

            confidence=countTem\*1.0/counts1;

            if(confidence>=MIN\_CONFIDENCE)

            {

                System.out.print("关联规则："+ s1.toString()+"=>>"+s2.toString()+"   ");

                return confidence;

            }

            else

                return 0;

        }

        else

            return 0;

    }

    public static int getCount(List<String> in)//根据频繁项集得到其支持度计数

    {

        int rt=0;

        for(int i=0;i<map.size();i++)

        {

            Mymap tem=map.get(i);

            if(tem.isListEqual(in)) {

                rt = tem.getcount();

                return rt;

            }

        }

        return rt;

    }

    public static  List<String> gets2set(List<String> tem, List<String> s1)//计算tem减去s1后的集合即为s2

    {

        List<String> result=new ArrayList<>();

        for(int i=0;i<tem.size();i++)//去掉s1中的所有元素

        {

            String t=tem.get(i);

            if(!s1.contains(t))

                result.add(t);

        }

        return  result;

    }

    public static List<List<String>> getSubSet(List<String> set){

        List<List<String>> result = new ArrayList<>(); //用来存放子集的集合，如{{},{1},{2},{1,2}}

        int length = set.size();

        int num = length==0 ? 0 : 1<<(length); //2的n次方，若集合set为空，num为0；若集合set有4个元素，那么num为16.

        //从0到2^n-1（[00...00]到[11...11]）

        for(int i = 1; i < num-1; i++){

            List<String> subSet = new ArrayList<>();

            int index = i;

            for(int j = 0; j < length; j++){

                if((index & 1) == 1){     //每次判断index最低位是否为1，为1则把集合set的第j个元素放到子集中

                    subSet.add(set.get(j));

                }

                index >>= 1;      //右移一位

            }

            result.add(subSet);       //把子集存储起来

        }

        return result;

    }

    public  static  boolean  AddToFrequenceItem(List<List<String>> fre)

    {

        for(int i=0;i<fre.size();i++)

        {

            frequentItemset.add(fre.get(i));

        }

        return true;

    }

    public static  void ShowData(List<List<String>> CandidateItemset)//显示出candidateitem中的所有的项集

    {

        for(int i=0;i<CandidateItemset.size();i++){

            List<String> list = new ArrayList<String>(CandidateItemset.get(i));

            for(int j=0;j<list.size();j++){

                System.out.print(list.get(j)+" ");

            }

            System.out.println();

        }

    }

    /\*\*

     \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 有当前频繁项集自连接求下一次候选集

     \*/

    private static List<List<String>> getNextCandidate(List<List<String>> FrequentItemset) {

        List<List<String>> nextCandidateItemset = new ArrayList<List<String>>();

        for (int i=0; i<FrequentItemset.size(); i++){

            HashSet<String> hsSet = new HashSet<String>();

            HashSet<String> hsSettemp = new HashSet<String>();

            for (int k=0; k< FrequentItemset.get(i).size(); k++)//获得频繁集第i行

                hsSet.add(FrequentItemset.get(i).get(k));

            int hsLength\_before = hsSet.size();//添加前长度

            hsSettemp=(HashSet<String>) hsSet.clone();

            for(int h=i+1; h<FrequentItemset.size(); h++){//频繁集第i行与第j行(j>i)连接   每次添加且添加一个元素组成    新的频繁项集的某一行，

                hsSet=(HashSet<String>) hsSettemp.clone();//！！！做连接的hasSet保持不变

                for(int j=0; j< FrequentItemset.get(h).size();j++)

                    hsSet.add(FrequentItemset.get(h).get(j));

                int hsLength\_after = hsSet.size();

                if(hsLength\_before+1 == hsLength\_after && isnotHave(hsSet,nextCandidateItemset)){

                    //如果不相等，表示添加了1个新的元素       同时判断其不是候选集中已经存在的一项

                    Iterator<String> itr = hsSet.iterator();

                    List<String>  tempList = new ArrayList<String>();

                    while(itr.hasNext()){

                        String Item = (String) itr.next();

                        tempList.add(Item);

                    }

                    nextCandidateItemset.add(tempList);

                }

            }

        }

        return nextCandidateItemset;

    }

    /\*\*

     \* 判断新添加元素形成的候选集是否在新的候选集中

     \*/

    private static boolean isnotHave(HashSet<String> hsSet, List<List<String>> nextCandidateItemset) {//判断hsset是不是candidateitemset中的一项

        List<String>  tempList = new ArrayList<String>();

        Iterator<String> itr = hsSet.iterator();

        while(itr.hasNext()){//将hsset转换为List<String>

            String Item = (String) itr.next();

            tempList.add(Item);

        }

        for(int i=0; i<nextCandidateItemset.size();i++)//遍历candidateitemset，看其中是否有和templist相同的一项

            if(tempList.equals(nextCandidateItemset.get(i)))

                return false;

        return true;

    }

    /\*\*

     \* 由k项候选集剪枝得到k项频繁集

     \*/

    private static List<List<String>> getSupprotedItemset(List<List<String>> CandidateItemset) { //对所有的商品进行支持度计数

        // TODO Auto-generated method stub

        boolean end = true;

        List<List<String>> supportedItemset = new ArrayList<List<String>>();

        for (int i = 0; i < CandidateItemset.size(); i++){

            int count = countFrequent1(CandidateItemset.get(i));//统计记录数

            if (count >= MIN\_SUPPROT \* (record.size()-1)){

                supportedItemset.add(CandidateItemset.get(i));

                map.add(new Mymap(CandidateItemset.get(i),count));//存储当前频繁项集以及它的支持度计数

                end = false;

            }

        }

        endTag = end;//存在频繁项集则不会结束

        if(endTag==true)

            System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*无满足支持度的"+times+"项集,结束连接");

        return supportedItemset;

    }

    /\*\*

     \* 统计record中出现list集合的个数

     \*/

    private static int countFrequent1(List<String> list) {//遍历所有数据集record，对单个候选集进行支持度计数

        int count =0;

        for(int i=0;i<record.size();i++)//从record的第一个开始遍历

        {

            boolean flag=true;

            for (int j=0;j<list.size();j++)//如果record中的第一个数据集包含list中的所有元素

            {

                String t=list.get(j);

                if(!record.get(i).contains(t)) {

                    flag = false;

                    break;

                }

            }

            if(flag)

                count++;//支持度加一

        }

        return count;//返回支持度计数

    }

     //获得一项候选集

    private static List<List<String>> findFirstCandidate() {

        // TODO Auto-generated method stub

        List<List<String>> tableList = new ArrayList<List<String>>();

        HashSet<String> hs  = new HashSet<String>();//新建一个hash表，存放所有的不同的一维数据

        for (int i = 1; i<record.size(); i++){  //遍历所有的数据集，找出所有的不同的商品存放到hs中

            for(int j=1;j<record.get(i).size();j++){

                hs.add(record.get(i).get(j));

            }

        }

        Iterator<String> itr = hs.iterator();

        while(itr.hasNext()){

            List<String>  tempList = new ArrayList<String>();

            String Item = (String) itr.next();

            tempList.add(Item);   //将每一种商品存放到一个List<String>中

            tableList.add(tempList);//所有的list<String>存放到一个大的list中

        }

        return tableList;//返回所有的商品

    }

}

class  Mymap{//自定义的map类，一个对象存放一个频繁项集以及其支持度计数

    public List<String> li=new LinkedList<>();

    public  int count;

    public Mymap(List<String> l,int c)//构造函数  新建一个对象

    {

        li=l;

        count=c;

    }

    public int getcount()//返回得到当前频繁项集的支持度计数

    {

        return count;

    }

    public boolean isListEqual(List<String> in)//判断传入的频繁项集是否和本频繁项集相同

    {

        if(in.size()!=li.size())//先判断大小是否相同

            return false;

        else {

            for(int i=0;i<in.size();i++)//遍历输入的频繁项集，判断是否所有元素都包含在本频繁项集中

            {

                if(!li.contains(in.get(i)))

                    return false;

            }

        }

        return true;//如果两个频繁项集大小相同，同时本频繁项集包含传入的频繁项集的所有元素，则表示两个频繁项集是相等的，返回为真

    }

}

4、分类算法的基本流程是什么。决策树ID3算法的流程，朴素贝叶斯的流程。

**分类算法的基本流程：**

数据分类包括学习阶段和分类阶段

学习阶段是用分类算法分析训练数据，构建分类模型

分类阶段是使用模型预测给定数据的类标号

**决策树ID3算法：**

决策树的基本概念：

决策树是一种依托决策而建立起来的一种树。在**[机器学习](http://lib.csdn.net/base/2" \t "_blank)**中，决策树是一种预测模型，代表的是一种对象属性与对象值之间的一种映射关系，每一个节点代表某个对象，树中的每一个分叉路径代表某个可能的属性值，而每一个叶子节点则对应从根节点到该叶子节点所经历的路径所表示的对象的值。决策树仅有单一输出，如果有多个输出，可以分别建立独立的决策树以处理不同的输出。

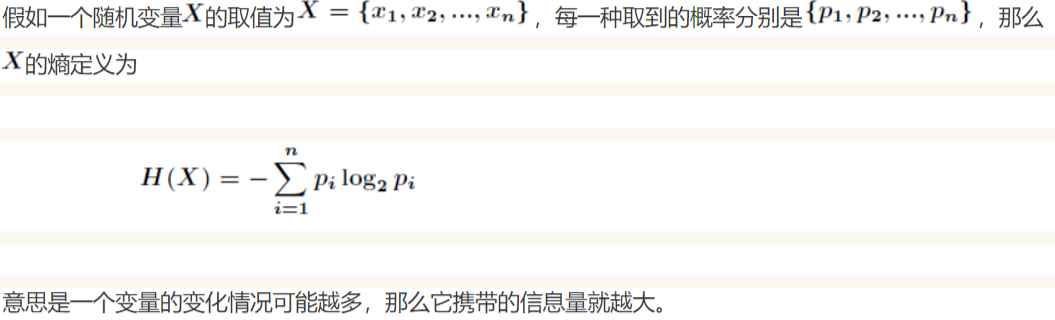
ID3算法简介:

    ID3算法是决策树的一种，它是基于奥卡姆剃刀原理的，即用尽量用较少的东西做更多的事。ID3算法，即Iterative Dichotomiser 3，迭代二叉树3代，是Ross Quinlan发明的一种决策树算法，这个算法的基础就是上面提到的奥卡姆剃刀原理，越是小型的决策树越优于大的决策树，尽管如此，也不总是生成最小的树型结构，而是一个启发式算法。

   在信息论中，期望信息越小，那么信息增益就越大，从而纯度就越高。ID3算法的核心思想就是以信息增益来度量属性的选择，选择分裂后信息增益最大的属性进行分裂。该算法采用自顶向下的贪婪搜索遍历可能的决策空间。

信息熵与信息增益的概念:

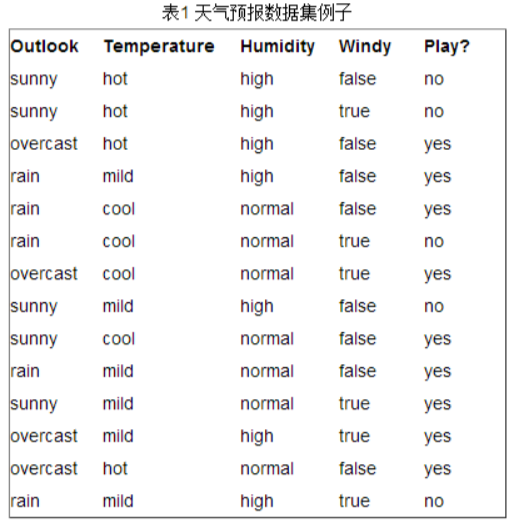
熵这个概念最早起源于物理学，在物理学中是用来度量一个热力学系统的无序程度，而在信息学里面，熵是对不确定性的度量。在1948年，香农引入了信息熵，将其定义为离散随机事件出现的概率，一个系统越是有序，信息熵就越低，反之一个系统越是混乱，它的信息熵就越高。所以信息熵可以被认为是系统有序化程度的一个度量。信息熵的公式：



信息增益是针对一个一个特征而言的，就是看一个特征t，系统有它和没有它时的信息量各是多少，两者的差值就是这个特征给系统带来的信息量，即信息增益。在决策树分类问题中，信息增益就是决策树在进行属性选择划分前和划分后信息熵的差值。

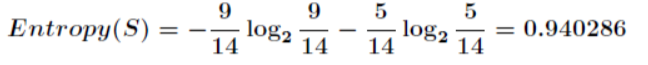
该算法的流程：

以天气预报的例子来说明。下面是描述天气数据表，学习目标是play或者not play。



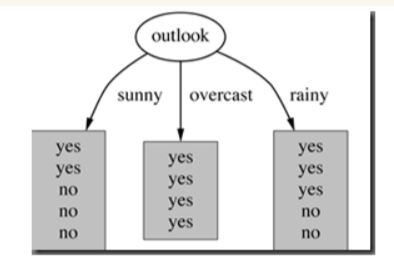
可以看出，一共14个样例，包括9个正例和5个负例。

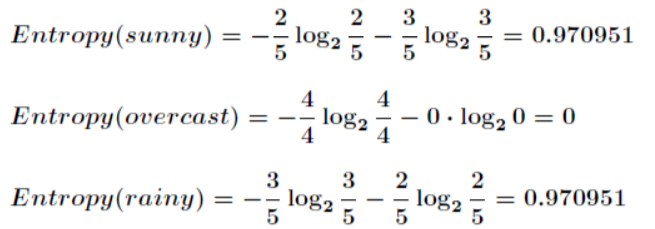
（1）根据提供的数据就可以计算出没有进行属性选择划分前的信息熵



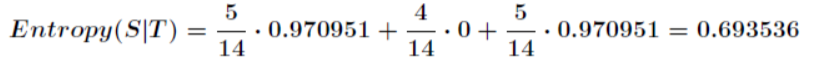
（2）在之后就是计算出进行属性选择划分后的信息熵。

在决策树分类问题中，信息增益就是决策树在进行属性选择划分前和划分后信息的差值。假设利用属性Outlook来分类，那么如下图

划分后，数据被分为三部分了，那么各个分支的信息熵计算如下



 那么划分后的信息熵为

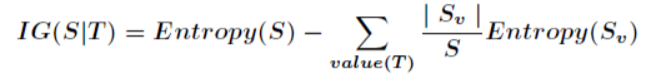


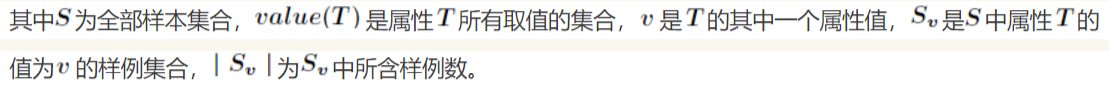
（3）两者的信息熵进行相减。那么最终得到特征属性t带来的信息增益为

https://uploader.shimo.im/f/f4IesRyejrEIMJbu.png!thumbnail

（4）前三步是单个的非子叶结点的算法步骤，下面是全部非子叶结点的集合算法。

信息增益的计算公式





小结：

在决策树的每一个非叶子结点划分之前，先计算每一个属性所带来的信息增益，选择最大信息增益的属性来划分，因为信息增益越大，区分样本的能力就越强，越具有代表性，很显然这是一种自顶向下的贪心策略。以上就是ID3算法的核心思想。

**朴素贝叶斯的流程：**

朴素贝叶斯分类是一种十分简单的分类算法，叫它朴素贝叶斯分类是因为这种方法的思想真的很朴素，朴素贝叶斯的思想基础是这样的：对于**给出的待分类项**，求解在此项出现的条件下各个类别出现的概率，哪个最大，就认为此待分类项属于哪个类别。通俗来说，就好比这么个道理，你在街上看到一个黑人，我问你你猜这哥们哪里来的，你十有八九猜非洲。为什么呢？因为黑人中非洲人的比率最高，当然人家也可能是美洲人或亚洲人，但在没有其它可用信息下，我们会选择**条件概率最大的类别**，这就是朴素贝叶斯的思想基础。

朴素贝叶斯分类的正式**定义**如下：

1、设 https://uploader.shimo.im/f/A8siHVdssQopC9xj.gif为一个待分类项，而每个a为x的一个特征属性。

2、有类别集合 https://uploader.shimo.im/f/dZcAhcPF1bIyu6xw.gif。

3、计算 https://uploader.shimo.im/f/iYFWMr2OoDQVGj2z.gif。

4、如果 https://uploader.shimo.im/f/PtFWceEGHIkVpS7j.gif，则 https://uploader.shimo.im/f/DJH3hPCOFZ0q3i8T.gif。

那么现在的关键就是**如何计算第3步中的各个条件概率**。我们可以这么做：

1、找到一个已知分类的待分类项集合，这个集合叫做训练样本集。

2、统计得到在各类别下**各个特征属性**的条件概率估计。即 https://uploader.shimo.im/f/RjbCUuwlhk8pAtge.gif

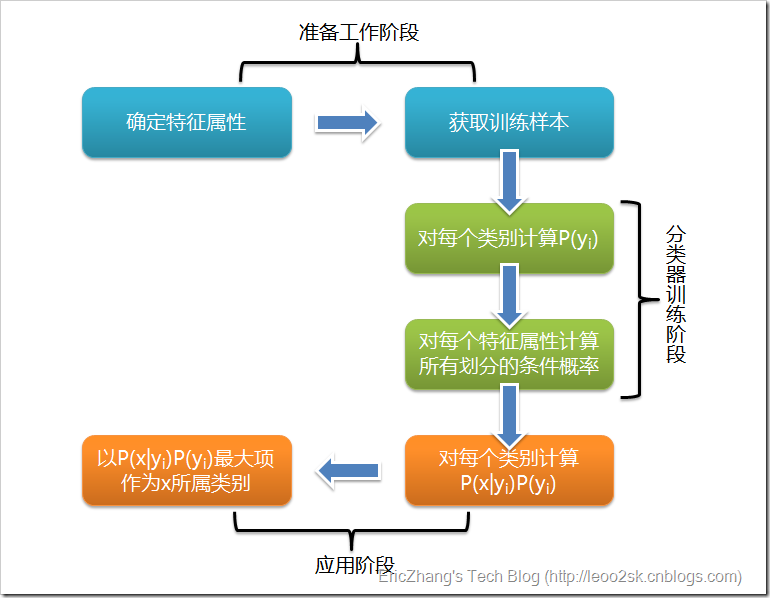
3、如果各个特征属性是**条件独立**的，则根据贝叶斯定理有如下推导：

https://uploader.shimo.im/f/UBSEnM0JtHUUFDXh.gif

因为分母对于所有类别为常数，因为我们只要将分子最大化皆可。又因为各特征属性是条件独立的，所以有：

https://uploader.shimo.im/f/UvLGSeR3yhYYrFpn.gif

根据上述分析，朴素贝叶斯分类的**流程**可以由下图表示（暂时不考虑验证）：



可以看到，整个朴素贝叶斯分类分为**三个阶段**：

      第一阶段——**准备工作阶段**，这个阶段的任务是为朴素贝叶斯分类做必要的准备，主要工作是根据具体情况确定特征属性，并对每个特征属性进行适当划分，然后由人工对一部分待分类项进行分类，形成训练样本集合。这一阶段的输入是所有待分类数据，输出是特征属性和训练样本。这一阶段是整个朴素贝叶斯分类中唯一需要人工完成的阶段，其质量对整个过程将有重要影响，分类器的质量很大程度上由特征属性、特征属性划分及训练样本质量决定。

      第二阶段——**分类器训练阶段**，这个阶段的任务就是生成分类器，主要工作是计算每个类别在训练样本中的出现频率及每个特征属性划分对每个类别的条件概率估计，并将结果记录。其输入是特征属性和训练样本，输出是分类器。这一阶段是机械性阶段，根据前面讨论的公式可以由程序自动计算完成。

      第三阶段——**应用阶段**。这个阶段的任务是使用分类器对待分类项进行分类，其输入是分类器和待分类项，输出是待分类项与类别的映射关系。这一阶段也是机械性阶段，由程序完成。

5、分类算法的评价指标是什么？精度率，回归率。

分类器评价指标：使用混淆矩阵

混淆矩阵是一个表，实际的类有两种买电脑和不买电脑，预测的类有两种买电脑和不买电脑

正元组：感兴趣的主要类的元组比如 买电脑

负元组：其他的元组 不买电脑

真正例TP代表个数：分类器正确分类的正元组，就是测试数据是买电脑，预测是买电脑的个数

真负例TN代表个数：分类器正确分类的负元组，就是测试数据是不买电脑，预测是不买电脑的个数

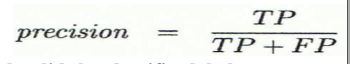
假正例FP代表个数：错误标记为正元组的负元组，就是测试数据是不买电脑，预测是买电脑的个数

假负例FN代表个数：错误标记为负元组的正元组，就是测试数据是买电脑，预测是不买电脑的个数

P代表正样本数就是买电脑的样本数

N代表负样本数就是不买电脑的样本数

**精确度/准确度：**测试样本集中模型正确预测/分类的样本所占的比率



**回归率/召回率：灵敏性，正确识别正元组的百分比**

https://uploader.shimo.im/f/8Ml4p8XVdCImAmAd.png!thumbnail

6、找一个分类的数据集，通过某种分类算法的Java实现进行分类。

**使用朴素贝叶斯实现**

**训练样本的JavaBean**

package NB;

/\*\*

 \* 训练样本的属性 javaBean

 \*

 \*/

public class JavaBean {

  int age;

  String income;

  String student;

  String credit\_rating;

  String buys\_computer;

 public JavaBean(){

 }

public JavaBean(int age,String income,String student,String credit\_rating,String buys\_computer){

       this.age=age;

       this.income=income;

       this.student=student;

       this.credit\_rating=credit\_rating;

       this.buys\_computer=buys\_computer;

}

public int getAge() {

       return age;

}

public void setAge(int age) {

       this.age = age;

}

public String getIncome() {

       return income;

}

public void setIncome(String income) {

       this.income = income;

}

public String getStudent() {

       return student;

}

public void setStudent(String student) {

       this.student = student;

}

public String getCredit\_rating() {

       return credit\_rating;

}

public void setCredit\_rating(String credit\_rating) {

       this.credit\_rating = credit\_rating;

}

public String getBuys\_computer() {

       return buys\_computer;

}

public void setBuys\_computer(String buys\_computer) {

       this.buys\_computer = buys\_computer;

}

@Override

public String toString() {

       return "JavaBean [age=" + age + ", income=" + income + ", student="

                     + student + ", credit\_rating=" + credit\_rating + ", buys\_computer="

                     + buys\_computer + "]";

}

}

**朴素贝叶斯算法实现**

package NB;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.util.ArrayList;

public class TestNB {

       /\*\*data\_length

        \* 算法的思想

        \*/

       public static  ArrayList<JavaBean> list = new ArrayList<JavaBean>();;

       static int data\_length=0;

       public static void main(String[] args) {

              // 1.读取数据，放入list容器中

              File file = new File("E://test.txt");

              txt2String(file);

              //数据测试样本

              testData(25,"Medium","Yes","Fair");

       }

    // 读取样本数据

       public static void txt2String(File file) {

              try {

                     BufferedReader br = new BufferedReader(new FileReader(file));// 构造一个BufferedReader类来读取文件

                     String s = null;

                     while ((s = br.readLine()) != null) {// 使用readLine方法，一次读一行

                            data\_length++;

                            splitt(s);

                     }

                     br.close();

              } catch (Exception e) {

                     e.printStackTrace();

              }

       }

       // 存入ArrayList中

         public static void splitt(String str){

               String strr = str.trim();

               String[] abc = strr.split("[\\p{Space}]+");

               int age=Integer.parseInt(abc[0]);

               JavaBean bean=new JavaBean(age, abc[1], abc[2], abc[3], abc[4]);

               list.add(bean);

           }

**// 训练样本，测试**

         public static void testData(int age,String a,String b,String c){

                //训练样本

                int number\_yes=0;

                int bumber\_no=0;

               // age情况 个数

                int num\_age\_yes=0;

                int num\_age\_no=0;

                // income

                int num\_income\_yes=0;

                int num\_income\_no=0;

                // student

                int num\_student\_yes=0;

                int num\_stdent\_no=0;

                //credit

                int num\_credit\_yes=0;

                int num\_credit\_no=0;

                //遍历List 获得数据

                for(int i=0;i<list.size();i++){

                  JavaBean bb=list.get(i);

                  if(bb.getBuys\_computer().equals("Yes")){ //Yes

                         number\_yes++;

                   if(bb.getIncome().equals(a)){//income

                          num\_income\_yes++;

                   }

                         if(bb.getStudent().equals(b)){//student

                                num\_student\_yes++;

                         }

                         if(bb.getCredit\_rating().equals(c)){//credit

                                num\_credit\_yes++;

                         }

                         if(bb.getAge()==age){//age

                                num\_age\_yes++;

                         }

                  }else {//No

                         bumber\_no++;

                         if(bb.getIncome().equals(a)){//income

                          num\_income\_no++;

                   }

                         if(bb.getStudent().equals(b)){//student

                                num\_stdent\_no++;

                         }

                         if(bb.getCredit\_rating().equals(c)){//credit

                                num\_credit\_no++;

                         }

                         if(bb.getAge()==age){//age

                                num\_age\_no++;

                         }

                     }

                }

                  System.out.println("购买的历史个数:"+number\_yes);

                  System.out.println("不买的历史个数:"+bumber\_no);

                  System.out.println("购买+age:"+num\_age\_yes);

                  System.out.println("不买+age:"+num\_age\_no);

                  System.out.println("购买+income:"+num\_income\_yes);

                  System.out.println("不买+income:"+num\_income\_no);

                  System.out.println("购买+stundent:"+num\_student\_yes);

                  System.out.println("不买+student:"+num\_stdent\_no);

                  System.out.println("购买+credit:"+num\_credit\_yes);

                  System.out.println("不买+credit:"+num\_credit\_no);

**//// 概率判断**

                  double buy\_yes=number\_yes\*1.0/data\_length; // 买的概率

                     double buy\_no=bumber\_no\*1.0/data\_length; //  不买的概率

                  System.out.println("训练数据中买的概率:"+buy\_yes);

                  System.out.println("训练数据中不买的概率:"+buy\_no);

                     /// 未知用户的判断

                  double nb\_buy\_yes=(1.0\*num\_age\_yes/number\_yes)\*(1.0\*num\_income\_yes/number\_yes)\*(1.0\*num\_student\_yes/number\_yes)\*(1.0\*num\_credit\_yes/number\_yes)\*buy\_yes;

                  double nb\_buy\_no=(1.0\*num\_age\_no/bumber\_no)\*(1.0\*num\_income\_no/bumber\_no)\*(1.0\*num\_stdent\_no/bumber\_no)\*(1.0\*num\_credit\_no/bumber\_no)\*buy\_no;

                  System.out.println("新用户买的概率:"+nb\_buy\_yes);

                  System.out.println("新用户不买的概率:"+nb\_buy\_no);

                  if(nb\_buy\_yes>nb\_buy\_no){

                         System.out.println("新用户买的概率大");

                  }else {

                         System.out.println("新用户不买的概率大");

                     }

         }

}

**样本数据**

25  High    No  Fair       No

25  High    No  Excellent  No

33  High    No  Fair       Yes

41  Medium  No  Fair       Yes

41  Low     Yes Fair       Yes

41  Low     Yes Excellent  No

33  Low     Yes Excellent  Yes

25  Medium  No  Fair       No

25  Low     Yes Fair       Yes

41  Medium  Yes Fair       Yes

25  Medium  Yes Excellent  Yes

33  Medium  No  Excellent  Yes

33  High    Yes Fair       Yes

41  Medium  No  Excellent  No

**未知数据预测得到的结果**

购买的历史个数:9

不买的历史个数:5

购买+age:2

不买+age:3

购买+income:4

不买+income:2

购买+stundent:6

不买+student:1

购买+credit:6

不买+credit:2

训练数据中买的概率:0.6428571428571429

训练数据中不买的概率:0.35714285714285715

新用户买的概率:0.028218694885361547

新用户不买的概率:0.006857142857142858

新用户买的概率大

**3/13**

1、分类算法的基本流程，什么叫做 训练集，测试集，验证集

答：分类算法基本流程：

**1.模型建立**

将原始数据的诸多属性进行处理，选定特征，形成**训练集**；选定要使用的分类算法，根据训练集的数据明确算法中的特征属性条件生成分类器

**2.使用模型**，将未知属性的样本（**测试集**）进行分类

对测试集使用分类器，得到模型预测的分类标签，将预测结果与原本已知的分类比较可知这个模型的预测是否准确。

最终可由对测试集的预测正确率来评估这个模型

**训练集：**用于模型拟合的数据样本，一些数据用来训练模型，生成最终的分类器

**测试集：**用来评估模最终模型的泛化能力。但不能作为调参、选择特征等算法相关的选择的依据。未知类别的数据，利用模型来对这些数据进行分类

**验证集：**模型训练过程中单独留出的样本集，它可以用于调整模型的超参数和用于对模型的能力进行初步评估。

训练集-----------学生的课本；学生根据课本里的内容来掌握知识。

验证集------------作业，通过作业可以知道 不同学生学习情况、进步的速度快慢。

测试集-----------考试，考的题是平常都没有见过，考察学生举一反三的能力。

2、如何评价分类算法？混淆矩阵，TP，TN ，FP，FN。精度，回归率

答：

我们主要用**正确率**来评价分类算法

首先介绍几个常见的模型评价术语，现在假设我们的分类目标只有两类，计为正例和负例分别是：

①**TP**: 被正确地划分为正例的个数，即实际为正例且被分类器划分为正例的实例数；

②**FP**: 被错误地划分为正例的个数，即实际为负例但被分类器划分为正例的实例数；

③**FN**:被错误地划分为负例的个数，即实际为正例但被分类器划分为负例的实例数；

④**TN**: 被正确地划分为负例的个数，即实际为负例且被分类器划分为负例的实例数。



**评价指标：**

（1）正确率

正确率是我们**最常见**的评价指标，**正确率 = （TP+TN）/(P+N)**，这个很容易理解，就是被分对的样本数除以所有的样本数，通常来说，正确率越高，分类器越好；

（2）错误率

错误率则与正确率相反，描述被分类器错分的比例，**错误率 = (FP+FN)/(P+N)**，对某一个实例来说，分对与分错是互斥事件，所以**正确率 =1 - 错误率**；

（3）灵敏度

**灵敏度 = TP/P**，表示的是所有正例中被分对的比例，衡量了**分类器对正例的识别能力**；

（4）特效度

**特效度 = TN/N**，表示的是所有负例中被分对的比例，衡量了**分类器对负例的识别能力**；

（5）精度

精度是精确性的度量，表示被分为正例的示例中实际为正例的比例，**精度=TP/（TP+FP）**

**即 精度=TP/P'**；

（6）召回率

召回率是覆盖面的度量，度量有多个正例被分为正例，**召回率=TP/(TP+FN)=TP/P=灵敏度**，可以看到召回率与灵敏度是一样的。

（7）其他评价指标

计算速度：分类器训练和预测需要的时间；

鲁棒性：处理缺失值和异常值的能力；

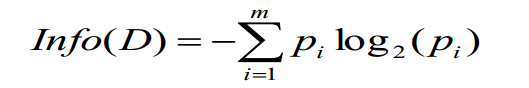
可扩展性：处理大数据集的能力；

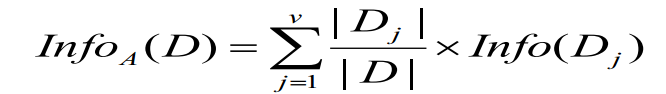
可解释性：分类器的预测标准的可理解性，像决策树产生的规则就是很容易理解的，而神经网络的一堆参数就不好理解，我们只好把它看成一个黑盒子。

3、决策树ID3算法流程

答：

（1）计算信息增益来选择哪一个属性来划分样本，一般选择信息增益大的属性来进行划分

训练集需要的期望信息，其中pi是两个分类（例如：yes，no）分别占的概率： 

若按照A属性进行分类之后的几个分支的信息： 

信息增益： 

所有属性的Gain（）值算出，以值最大的属性为分类的根节点

（2）选择一个属性作为根节点后，分支为分类属性的值，

再重复（1），对其余的属性再进行信息增益的计算，最终以没有属性剩下，没有剩余样本，一个给定节点的所有样本属于一个类别为终止条件，生成决策树。

（3）使用决策树对未知数据进行分类

4、朴素贝叶斯流程

答：**朴素贝叶斯分类的正式定义如下：**

   （1）设 http://latex.codecogs.com/gif.latex?x=%5C%7Ba_1,a_2,...,a_m%5C%7D为一个待分类项，而每个a为x的一个特征属性。

   （2）有类别集合 http://latex.codecogs.com/gif.latex?C=%5C%7By_1,y_2,...,y_n%5C%7D。

   （3）计算 http://latex.codecogs.com/gif.latex?P(y_1%7Cx),P(y_2%7Cx),...,P(y_n%7Cx)。

   （4）如果 http://latex.codecogs.com/gif.latex?P(y_k%7Cx)=max%5C%7BP(y_1%7Cx),P(y_2%7Cx),...,P(y_n%7Cx)%5C%7D，则 http://latex.codecogs.com/gif.latex?x%20%5Cin%20y_k。

**计算第3步中的各个条件概率：**

① 找到一个已知分类的待分类项集合，这个集合叫做训练样本集。

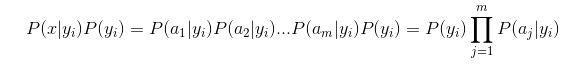
      ② 统计得到在各类别下各个特征属性的条件概率估计。即：

https://uploader.shimo.im/f/kGYBBD6iHaILORH5.png!thumbnail

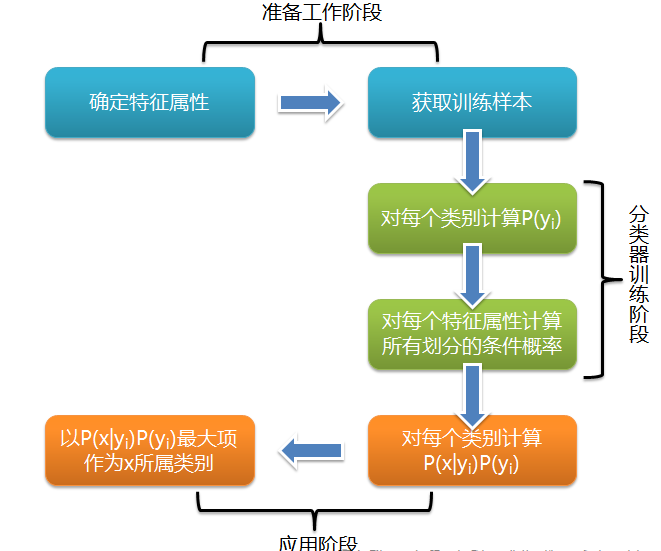
      ③ 如果各个特征属性是条件独立的，则根据贝叶斯定理有如下推导：

https://uploader.shimo.im/f/1KLNwgRfncEy8nm8.png!thumbnail

      因为分母对于所有类别为常数，因为我们只要将分子最大化皆可。又因为各特征属性是条件独立的，所以有：



      根据上述分析，朴素贝叶斯分类的流程可以由下图表示（暂时不考虑验证）：



可以看到，整个朴素贝叶斯分类分为**三个阶段**：

**第一阶段**——**准备工作阶段**，这个阶段的任务是为朴素贝叶斯分类做必要的准备，主要工作是根据具体情况确定特征属性，并对每个特征属性进行适当划分，然后由人工对一部分待分类项进行分类，形成训练样本集合。这一阶段的输入是所有待分类数据，输出是特征属性和训练样本。这一阶段是整个朴素贝叶斯分类中唯一需要人工完成的阶段，其质量对整个过程将有重要影响，分类器的质量很大程度上由特征属性、特征属性划分及训练样本质量决定。

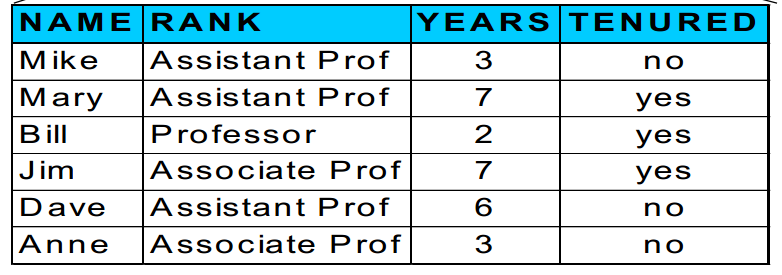
**第二阶段**——**分类器训练阶段**，这个阶段的任务就是生成分类器，主要工作是计算每个类别在训练样本中的出现频率及每个特征属性划分对每个类别的条件概率估计，并将结果记录。其输入是特征属性和训练样本，输出是分类器。这一阶段是机械性阶段，根据前面讨论的公式可以由程序自动计算完成。

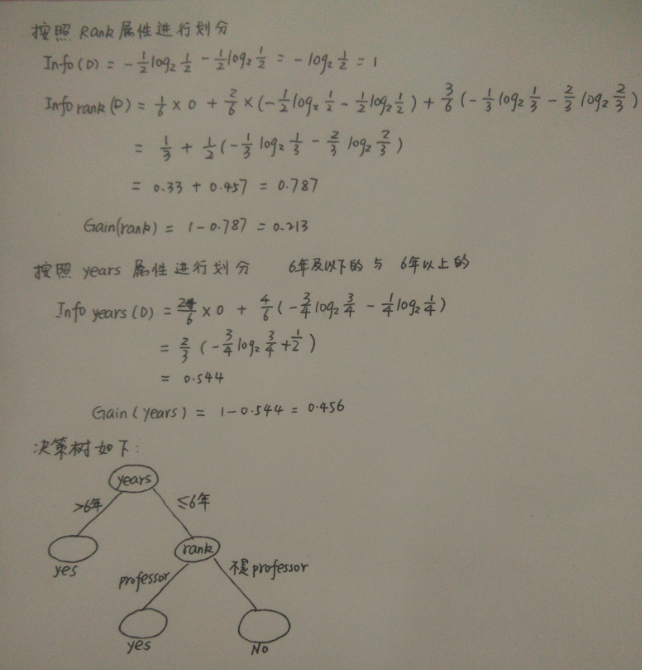
**第三阶段**——**应用阶段**。这个阶段的任务是使用分类器对待分类项进行分类，其输入是分类器和待分类项，输出是待分类项与类别的映射关系。这一阶段也是机械性阶段，由程序完成。

5、决策树ID3的两个案例，手工计算一下，尤其是信息增益的计算

答：

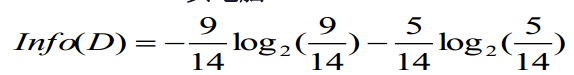
案例一：

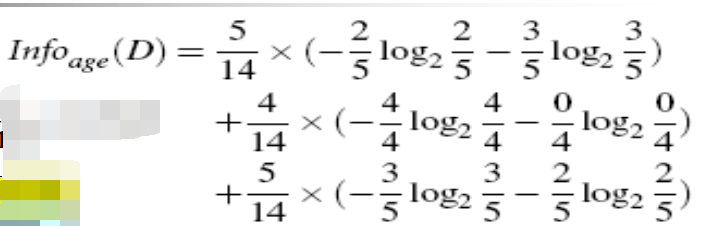


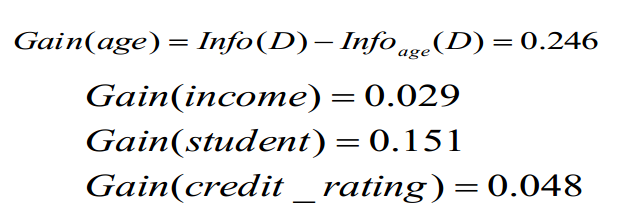


案例二：









6、朴素贝叶斯的案例，手工计算一下，ppt上的例子

答：



（1）分类： 根据是否购买PC分类

（2）C1：购买PC=yes；

C2：购买PC=no；

（3）训练样本集：X = (年龄 <=30,

收入 = medium,

学生 = yes，

信誉 = Fair)

P（Ci）：P（C1）=9/14=0.643

P（C2）=5/14=0.357

设 X1：年龄 <=30

X2：收入= medium

X3：学生= yes

X4：信誉 = Fair

P（Xi|Ci）：P（X1|C1）=2/9=0.222

P（X2|C1）=4/9=0.444

P（X3|C1）=6/9=0.667

P（X4|C1）=6/9=0.667

P（X1|C2）=3/5=0.6

P（X2|C2）=2/5=0.4

P（X3|C2）=1/5=0.2

P（X4|C2）=2/5=0.4

P（X|Ci）： P（X|C1）=0.222\*0.444\*0.667\*0.667=0.0439

P（X|C2）=0.6\*0.4\*0.2\*0.4=0.0192

P（X|Ci）\*P（Ci）： P（X|C1）\*P（C1）=0.0439\*0.643=0.0282

P（X|C2）\*P（C2）=0.0192\*0.357=0.00685

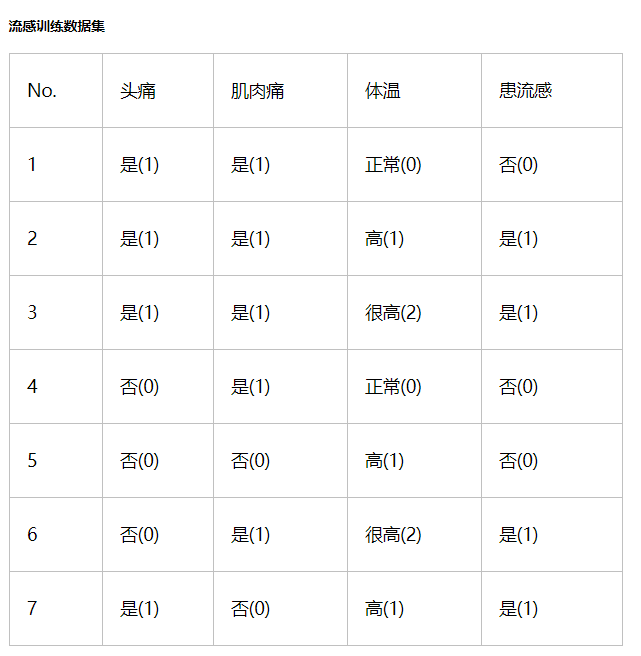
故X属于C1

3/14

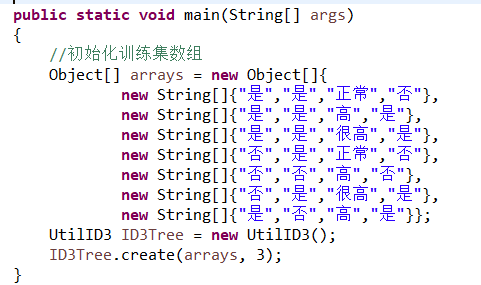
1、决策树ID3算法的Java实现。

答：：

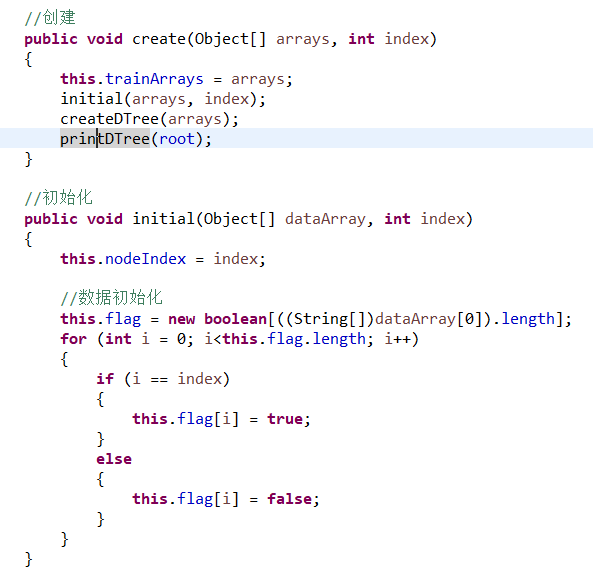
**训练数据如下：**

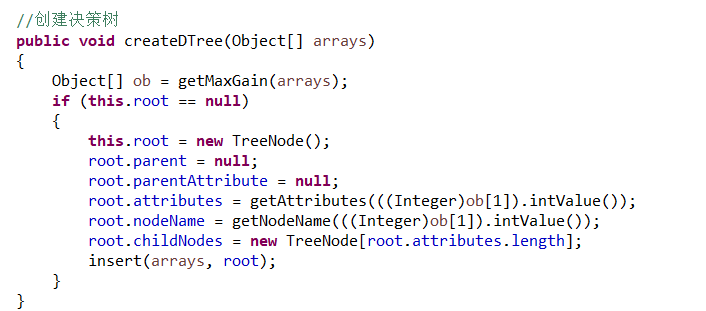


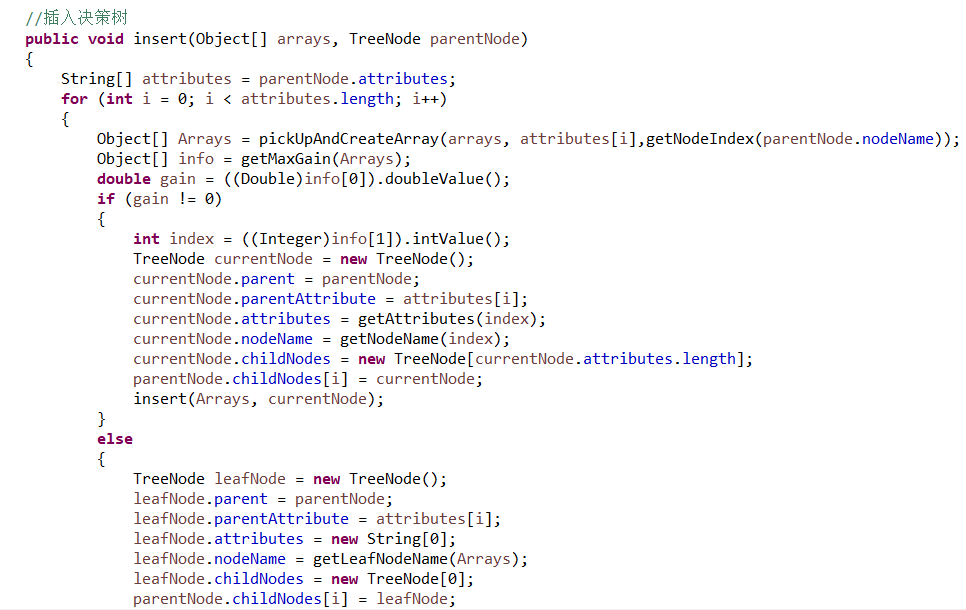
**训练集数据放到数组里面**

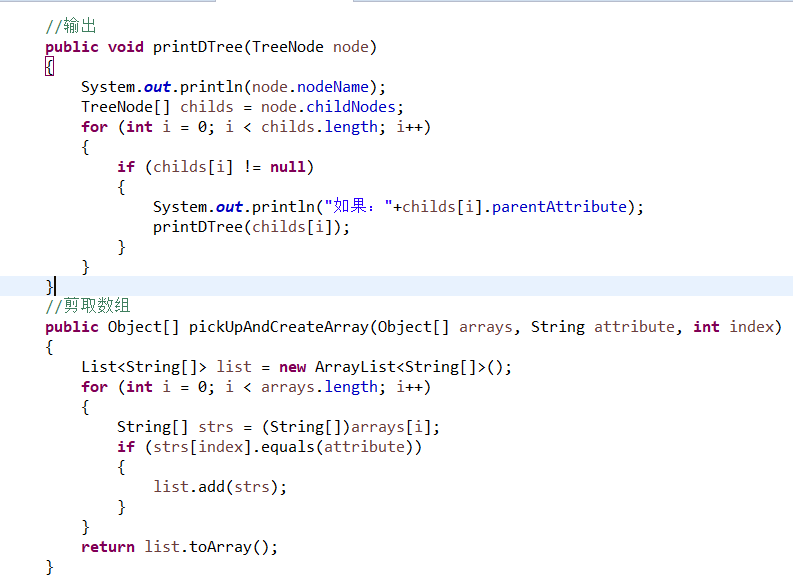


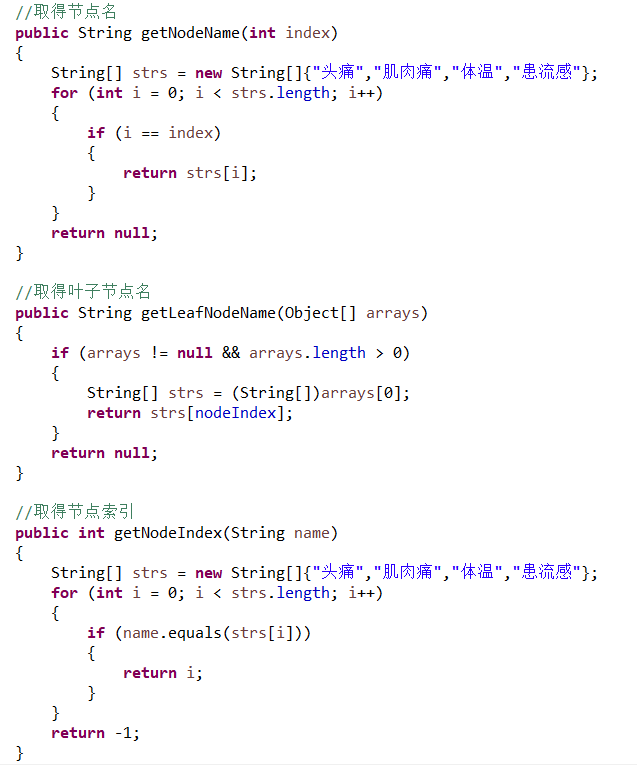
**创建决策树的过程**

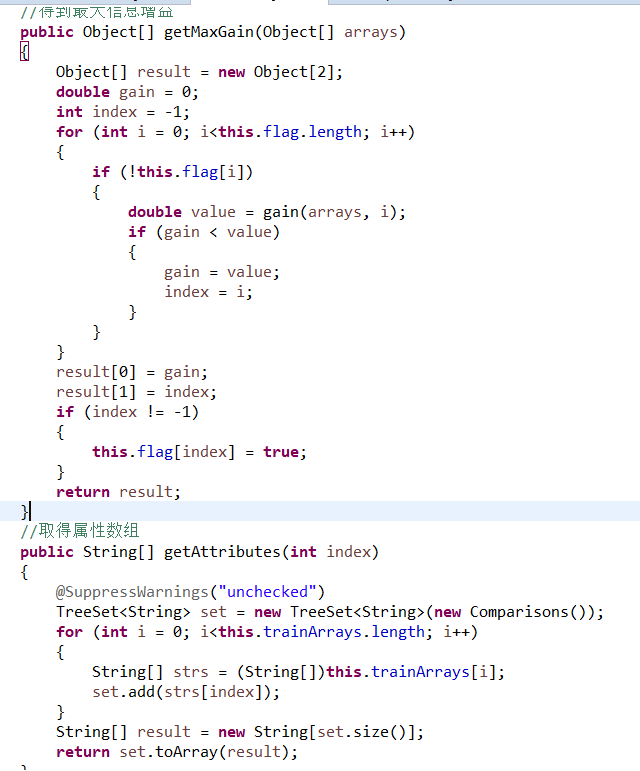




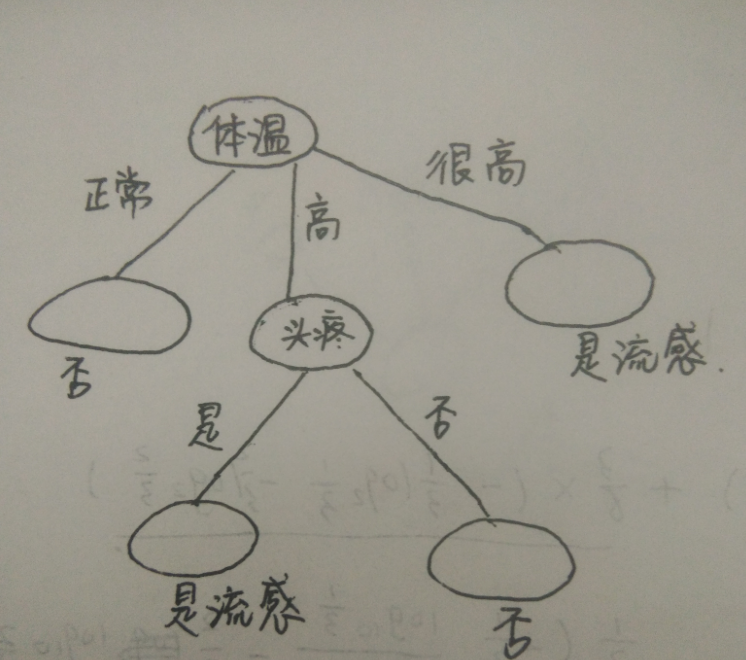




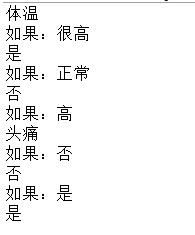




**决策树手画的如下：**



**结果输出：**



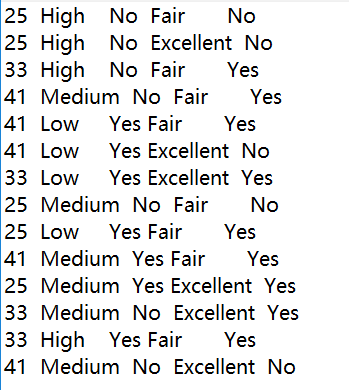
**通过计算信息增益得到，根节点的属性，然后再判断分支节点的属性**

2、朴素贝叶斯算法的Java实现。

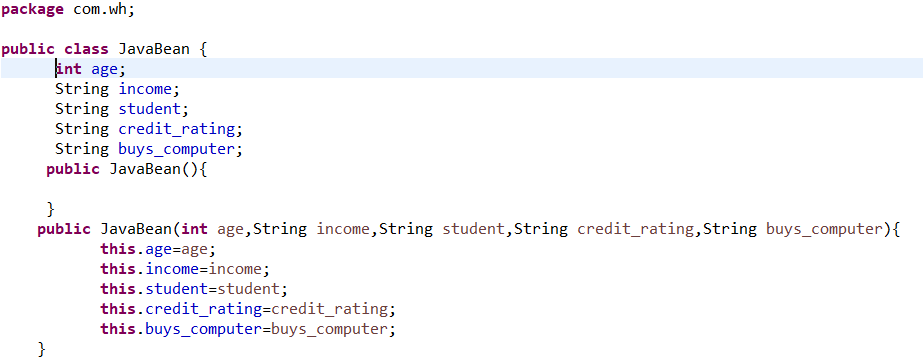
答：

**训练数据如下：**

年龄 收入 学生 信誉 是否买电脑



根据训练数据构造javabean



**测试数据**

https://uploader.shimo.im/f/uffFXy4yUX8ZLoQT.png!thumbnail

**编写算法和测试类**

**全局的变量**

public static  ArrayList<JavaBean> list = new ArrayList<JavaBean>();

static int data\_length=0;

**main方法**

public static void main(String[] args) {

           // 1.读取数据，放入list容器中

           File file = new File("C:\\Users\\14735\\Desktop\\test.txt");

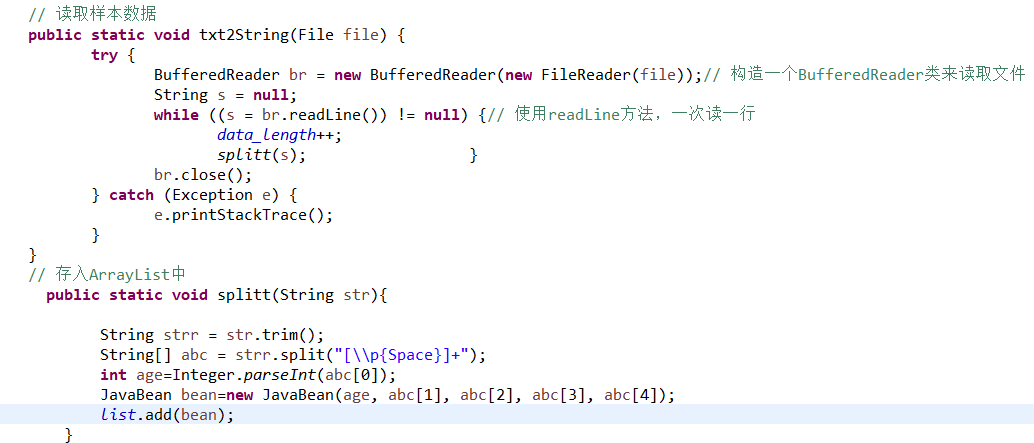
           txt2String(file);

           //数据测试样本

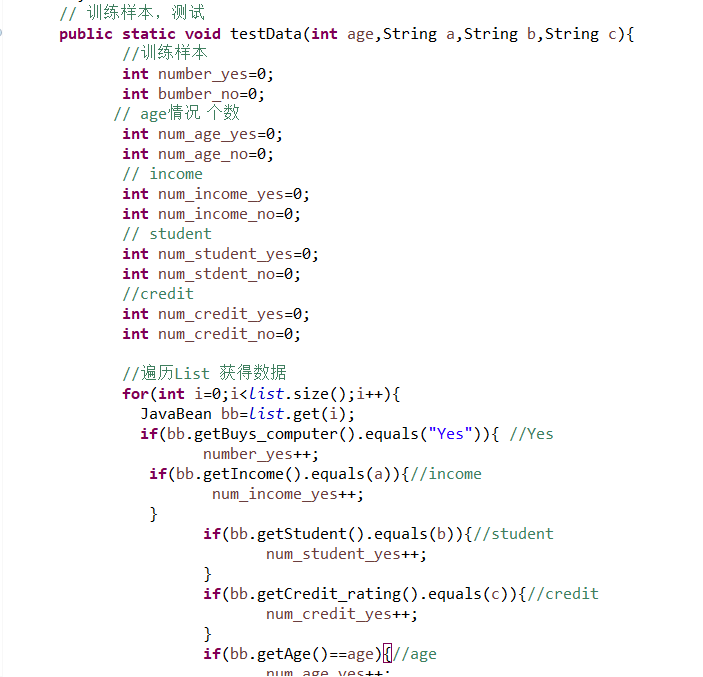
           testData(25,"Medium","Yes","Fair");

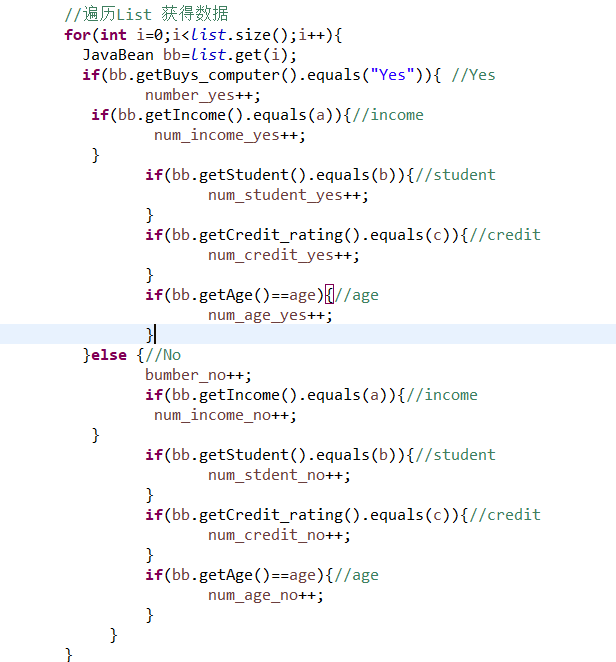
    }

**编写一个读文件的方法，并将数据存放在ArrayList里面**

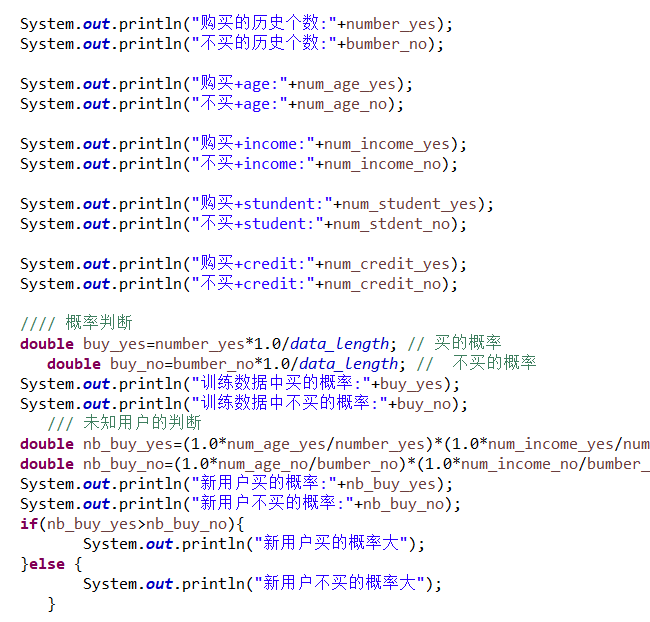


然后遍历数组进行计数



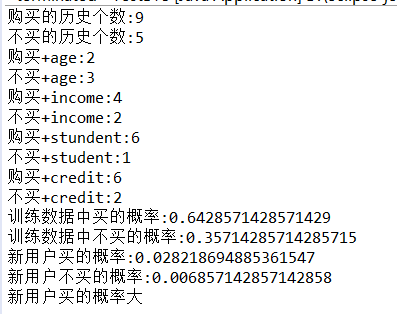


**编写输出信息**



**利用朴素贝叶斯的公式进行计算概率，预测测试数据是否会买电脑，选择概率大的**

**程序输出如下**



**最终计算得出新用户买的概率大**

3、基本概念学习：KNN算法流程；SVM；NN（ANN）；

答：

（1）KNN算法：

在训练集中数据和标签已知的情况下，输入测试数据，将测试数据的特征与训练集中对应的特征进行相互比较，找到训练集中与之最为相似的前K个数据，则该测试数据对应的类别就是K个数据中出现次数最多的那个分类，其算法的描述为：

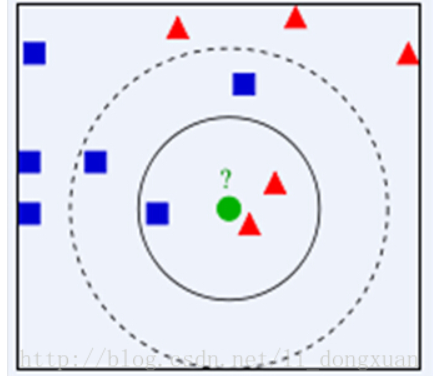
a. 计算测试数据与各个训练数据之间的距离；

b. 按照距离的递增关系进行排序；

c. 选取距离最小的K个点；

d. 确定前K个点所在类别的出现频率；

e. 返回前K个点中出现频率最高的类别作为测试数据的预测分类。



如上图，绿色圆要被决定赋予哪个类，是红色三角形还是蓝色四方形？如果K=3，由于红色三角形所占比例为2/3，绿色圆将被赋予红色三角形那个类，如果K=5，由于蓝色四方形比例为3/5，因此绿色圆被赋予蓝色四方形类。

（2）**SVM：**

SVM又称为支持向量机是一类按监督学习方式对数据进行二元分类的广义线性分类器。

**SVM解决要解决的问题：**

SVM最基本的应用是分类。 求解最优的分类面，然后用于分类。

最优分类面的定义：对于SVM，存在一个分类面，两个点集到此平面的最小距离最大，两个点集中的边缘点到此平面的距离最大。

**基本流程：**

给定训练样本集D={(x1→,y1),(x2→,y2),…,(xn→,yn)}D={(x1→,y1),(x2→,y2),…,(xn→,yn)}，yi∈{+1,−1}yi∈{+1,−1}，ii表示第ii个样本，nn表示样本容量。分类学习最基本的想法就是基于训练集DD在特征空间中找到一个最佳划分超平面将正负样本分开，而SVM算法解决的就是如何找到最佳超平面的问题。超平面可通过如下的线性方程来描述：

w→Tx→+b=0 (1)

其中w→表示法向量，决定了超平面的方向；b表示偏移量，决定了超平面与原点之间的距离。

对于训练数据集D假设找到了最佳超平面w∗→x→+b∗=0，定义决策分类函数

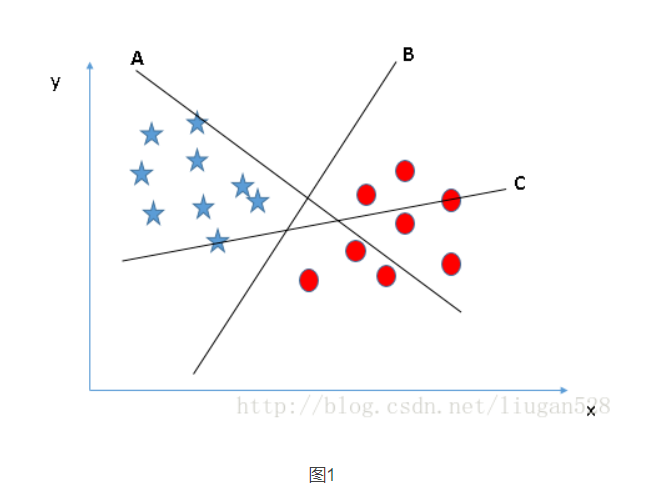
f(x→)=sign(w∗→x→+b∗) (2)

该分类决策函数也称为线性可分支持向量机。

在测试时对于线性可分支持向量机可以用一个样本离划分超平面的距离来表示分类预测的可靠程度，如果样本离划分超平面越远则对该样本的分类越可靠，反之就不那么可靠。

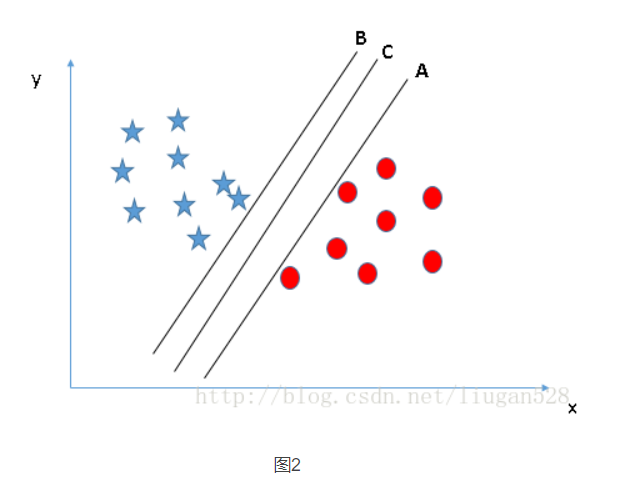
那么，什么样的划分超平面是最佳超平面呢？

对于图1有A、B、C三个超平面，很明显应该选择超平面B，也就是说超平面首先应该能满足将两类样本点分开。

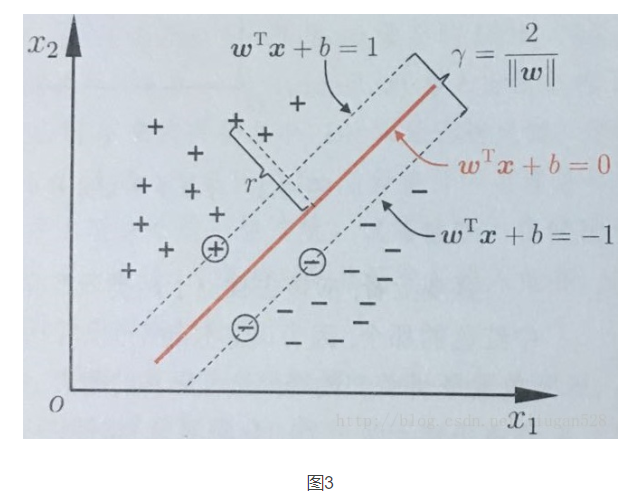


对于图2的A、B、C三个超平面，应该选择超平面C，因为使用超平面C进行划分对训练样本局部扰动的“容忍”度最好，分类的鲁棒性最强。

例如，由于训练集的局限性或噪声的干扰，训练集外的样本可能比图2中的训练样本更接近两个类目前的分隔界，在分类决策的时候就会出现错误，而超平面C受影响最小，也就是说超平面C所产生的分类结果是最鲁棒性的、是最可信的，对未见样本的泛化能力最强。



**示例：**下面以图3中示例进行推导得出最佳超平面。



空间中超平面可记为(w→,b)，根据点到平面的距离公式，空间中任意点 x→到超平面(w→,b)的距离可写为：

r=w→x→+b||w→|| (3)

假设超平面(w→,b)能将训练样本正确分类，那么对于正样本一侧的任意一个样本(xi→,yi)∈D，应该需要满足该样本点往超平面的法向量w→的投影到原点的距离大于一定值cc的时候使得该样本点被预测为正样本一类，即存在数值c使得当w→Txi→>c时yi=+1。w→Txi→>c又可写为w→Txi→+b>0。在训练的时候我们要求限制条件更严格点以使最终得到的分类器鲁棒性更强，所以我们要求w→Txi→+b>1。也可以写为大于其它距离，但都可以通过同比例缩放w→和b来使得使其变为1，因此为计算方便这里直接选择1。同样对于负样本应该有w→Txi→+b<−1时yi=−1。即：

{w→Txi→+b≥+1,yi=+1w→Txi→+b≤−1,yi=−1 (4)

亦即：

yi(w→Txi→+b)≥+1 (5)

如图3所示，距离最佳超平面w→x→+b=0最近的几个训练样本点使上式中的等号成立，它们被称为“支持向量”（support vector）。记超平面w→x→+b=+1和w→x→+b=−1之间的距离为γγ，该距离又被称为“间隔”（margin），SVM的核心之一就是想办法将“间隔”γ最大化。下面我们推导一下γ与哪些因素有关：

记超平面w→x→+b=+1上的正样本为x+→，超平面w→x→+b=−1上的负样本为xx−→，则根据向量的加减法规则x+→减去x−→得到的向量在最佳超平面的法向量w→方向的投影即为“间隔”γ：

γ=(x+→−x−→)w→||w→||=x+→w→||w→||−x−→w→||w→|| (6)

而w→x+→+b=+1，w→x−→+b=−1，即：

{w→x+→=1−bw→x+→=−1−b (7)

将(7)带入(6)可得：

γ=2||w→|| (8)

也就是说使两类样本距离最大的因素仅仅和最佳超平面的法向量有关！

要找到具有“最大间隔”（maximum margin）的最佳超平面，就是找到能满足式(4)中约束的参数w→、b使得γ最大，即：

{maxw→,b2||w→||s.t.yi(w→Txi→+b)≥+1,i=1,2,…,n (9)

显然(9)等价于

{minw→,b12||w→||2s.t.yi(w→Txi→+b)≥+1,i=1,2,…,n (10)

这就是SVM的基本型。

（3）ANN：

**概念**：是指由大量的处理单元(神经元) 互相连接而形成的**复杂网络结构**，是对人脑组织结构和运行机制的某种**抽象、简化和模拟**。人工神经网络（Artificial Neural Network，简称ANN ），以数学模型模拟神经元活动，是基于模仿大脑神经网络结构和功能而建立的一种**信息处理系统**。

**特点及作用**：人工神经网络具有**自学习、自组织、自适应**以及很强的非线性函数逼近能力，拥有强大的容错性。它可以实现**仿真**、**二值图像识别**、**预测**以及**模糊控制**等功能。是处理**非线性系统**的有力工具。

4、聚类算法的基本流程

答：

（1）将数据分割成K个组，保证每个数据属于且仅属于一个分类；

（2）通过反复迭代的方法改变原有分类，直到分类分组的数据不再改变或是其他条件满足，具体改变衡量的条件跟不同算法原理有关，例如：k-means是和每组的质心与同组其他数据的欧式距离有关，最终效果是同一分组间的距离最近；

5、k-means算法的基本流程

（1）首先，随机设K个特征空间内的点作为初始的聚类中心。（人群中选k个代表）

（2）然后，对于根据每个数据的特征向量，从K个聚类中心中寻找距离最近的一个，并且把该数据标记为这个聚类中心。（其余人离哪个代表最近就归为哪个代表的团队）

（3）接着，在所有的数据都被标记过聚类中心之后，根据这些数据新分配的类簇，通过取分配给每个先前质心的所有样本的平均值来创建新的质心重,新对K个聚类中心做计算。（第一次站队完成后，再从每个小团体中重选最适中的新代表）

（4）最后，计算旧和新质心之间的差异,如果所有的数据点从属的聚类中心与上一次的分配的类簇没有变化，那么迭代就可以停止，否则回到步骤2继续循环。（新代表选完，再次进行步骤（2），若每个团队成员不变了，则聚类结束；若成员发生变动，则再次站队，再次重选团队代表）

流程效果如下图

