软件分析

聚类和挖掘

熊英飞
北京大学
2017

聚类

聚类

- 把数据按相似度分类
  - 最常见的无监督学习方法
- 如何衡量相似度
  - 欧氏距离
  - 协方差
- 如何分类
  - k-means
  - 层次聚类

$$d_{euc}(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$ 欧氏距离

$$\rho(\mathbf{x},\mathbf{y}) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$ 协方差

K-Means

- 输入：数据集，聚类数量k
- 算法
  - 随机分类k个数据到k个类别，每个数据成为该类别的中点
  - 对于每个数据
    - 计算该数据到每个类别中心的距离
    - 将该数据加入最近的类别
    - 重新计算该类别的中点

K-Means优缺点

- 优点：简单快速
- 缺点：
  - 需要指定k
  - 一开始的随机选点对结果影响很大
  - 必须能计算中点
- 针对前两个缺点有若干改进算法
  - 尝试多组k或者随机选点，选出较好的（即每个类别中的点互相接近）
  - 允许拆分和重新组合类别

层次聚类

- 一开始每一个数据是一个类别
- 每次合并两个最相似的类别
  - 如何衡量类别的相似性？
    - 中点的距离
    - 任意两点的最短距离
    - 任意两点的最大距离
    - ……
- 到达停止条件时结束
  - 类别数量少于等于k
  - 类别之间的距离都大于等于d

层次聚类优缺点

- 优点：
  - 可不计算中点
  - 终止条件更灵活
- 缺点：
  - 计算比较慢

聚类的应用：编译器测试排序

- 首先对选择特征，然后基于选择后的特征聚类
- 利用分类器排序的时候，先对类别按类别中最大值进行排序，然后依次从各个类别中选择最优测试输入
- 整体可以获得更稳定的输出

频繁模式挖掘

频繁模式挖掘

- 从一个（大型）数据库中挖掘频繁出现的子项
- 常见频繁模式挖掘
  - 频繁子集(itemset)挖掘
  - 频繁序列挖掘
  - 频繁子树挖掘
  - 频繁子图挖掘

Transaction	Items
t_1	Bread,Jelly,PeanutButter
t_2	Bread,PeanutButter
t_3	Bread,Milk,PeanutButter
t_4	Beer,Bread
t_5	Beer,Milk

{Bread, PeanutButter}是一个频繁子集

频繁模式挖掘通式

- 考虑某种数据结构S
  - S上存在包含关系
  - 任意数据项$s \in S$有大小$d(s)$，$d(s)$为一个包含0的自然数
- 输入：
  - 一组数据项的集合I
  - 一个频率的阈值t
  - 定义：一个数据项s的频率为I中包含s的数据项的个数
  - 定义：一个数据项s是频繁当且仅当s的频率 $>$ t
- 输出：一个频繁数据项的集合F，满足
  - $\forall s, s$的频率 $\geq$ t $\Leftrightarrow s \in F$

Apriori算法

- 频繁子项性质：如果一个子项是非频繁的，那么其超项也是非频繁的
- 算法：

n=1;

找到大小为1的所有频繁项，记为L_1;

do {

n++;

C = $\{ s \mid d(s) = n \wedge \exists s' \in L_{n-1}, s' \subseteq s \}$

L_n=$\{s \mid s \in C \wedge s$是频繁的$\}$

}

优化：
产生C的时候过滤掉包含非频繁子项的数据项

Apriori算法小结

- 优点：
  - 非常容易并行
  - 非常容易实现
- 缺点：
  - 需要反复扫描数据集，效率不高
- 针对特定数据结构有各种专门算法，在使用时可直接找到最新算法使用

关联规则挖掘

- 找到所有频繁且可靠的关联规则
- 关联规则$X \Rightarrow Y$
  - $X \cap Y = \emptyset$，即X和Y没有公共子项
- 频繁的关联规则
  - 同时包含X和Y的数据项数目超过阈值t
- 可靠的关联规则
  - $\frac{\text{同时包含}x\text{和}y\text{的数据项数目}}{\text{包含x的数据项数目}} >$ 阈值c

关联规则挖掘

对任意频繁子项l

对任意数据项$x \subset l$

检查x=>(l-x)是否是可靠的

基于关联规则的API使用错误查找

- Zhenmin Li and Yuanyuan Zhou. PR-Miner: Automatically Extracting Implicit Programming Rules and Detecting Violations in Large Software Code. FSE 2005.
- 很多函数是一起调用的。如果漏了一个函数就是Bug。

```
postgresql-8.0.1/src/backend/catalog/dependency.c:
1733 getRelationDescription (StringInfo buffer, Oid relid)
1734 {
1735    HeapTuple  relTup;
          ......
1740    relTup = SearchSysCache(...);
          ......
1796    ReleaseSysCache(relTup);
1797 }
```

基于关联规则的API使用错误查找


```
postgresql-8.0.1/src/backend/commands/tablecmds.c:
5686 AlterTableCreateToastTable(Oid relOid, bool silent)
5687 {
        ......
5692   Relation  class_rel;
        ......
5853   class_rel = heap_openr (...);
        ......
5863   simple_heap_update(class_rel, ...);
        ......
5866   CatalogUpdateIndexes(class_rel, ...);
        ......
5870   heap_close (class_rel, ...);
        ......
5891 }
```

```
linux-2.6.11/drivers/isdn/hisax/config.c:
771 void ll_stop(struct IsdnCardState *cs)
772 {
773     isdn_ctrl ic;
775     ic.command= ISDN_STAT_STOP;
776     ic.driver= cs->myid;
777     cs->iif.statcallb(&ic);
779 }
```

基于关联规则的API使用错误查找

- 从代码中提取API调用集合
- 将API编码成整数
- 应用关联规则挖掘
- 用挖掘出的规则检查代码中的规则违反
- 在Linux代码中检查了60处违反，发现了16个Bug