MapReduce数据挖掘基础算法(II)

分类算法和频繁项集挖掘

摘要

- 分类问题的基本描述
- KNN最近邻分类算法
- 朴素贝叶斯分类算法
- 支持向量机SVM分类
- 频繁项集挖掘算法

摘要

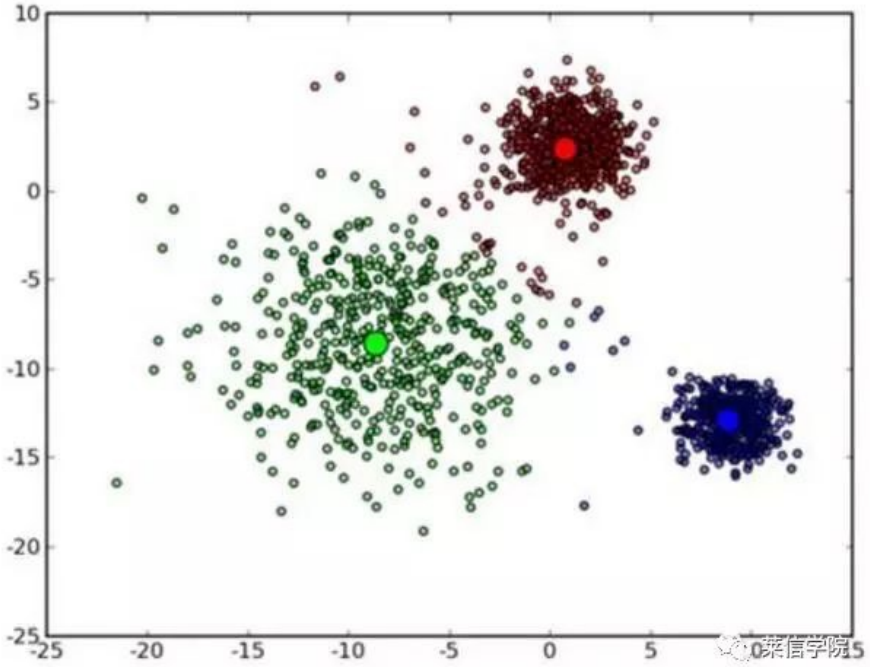
- 分类问题的基本描述
- KNN最近邻分类算法
- 朴素贝叶斯分类算法
- 支持向量机SVM分类
- 频繁项集挖掘算法

聚类

- 聚类与分类的不同在于，聚类所要求划分的类是未知的。聚类的目的是使得属于同类别的对象之间的差别尽可能的小，而不同类别上的对象的差别尽可能的大。因此，聚类的意义就在于将观察到的内容依据相应算法组织成类分层结构，把类似的事物组织在一起。
- 在机器学习中，聚类是一种无监督学习。也就是说，聚类是在预先不知道欲划分类的情况下，根据信息相似度原则进行信息聚类的一种方法。通过聚类，人们能够识别密集的和稀疏的区域，因而发现全局的分布模式，以及数据属性之间的有趣的关系。

聚类

- 从统计学的观点看，聚类分析是通过数据建模简化数据的一种方法。常见的聚类算法包括：K-Means聚类算法、K-中心点聚类算法、层次聚类、DBScan、EM聚类等。

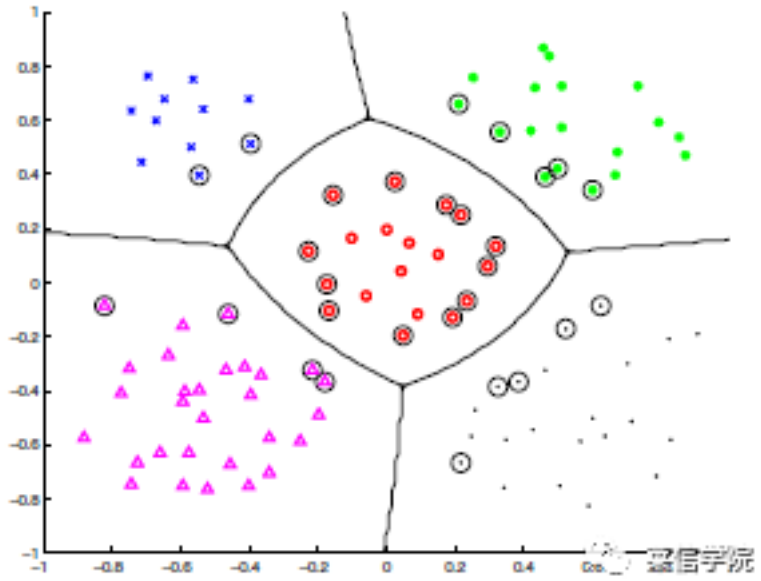


分类

- 分类（Classification）是机器学习和数据挖掘中最重要、最频繁使用的算法。
- 分类算法的基本作用是：从一组已经带有分类标记的训练样本数据集来预测一个测试样本的分类结果。
- 从机器学习的观点，分类属于监督学习，每个训练样本的数据对象已经有类标识，通过学习可以形成表达数据对象与类标识间对应的知识。
- 分类具有广泛的应用，例如医疗诊断、信用卡的信用分级、图像模式识别、营销用户画像等。

分类

- 分类挖掘所获的分类模型可以采用多种形式加以描述输出。其中主要的表示方法有：分类规则、决策树、数学公式和神经网络等。



分类 vs. 聚类

- 分类
  - 监督学习
  - 类别已知，通过对已知分类的数据进行训练和学习，找到不同类的特征，再对未分类的数据进行分类
  - 根据文本的特征或属性，划分到已有类别中。
- 聚类
  - 无监督学习
  - 类别未知，通过聚类分析将数据聚合成几个群体
  - 需要分析人员找出各类用户的重要特征
  - 需要通过各类别的特征解释含义以及为各类别命名。

分类问题基本描述

- 分类算法的基本描述：
  - 一个训练数据集TR={tr1, tr2, tr3, ...}
  - 每个训练样本tri是一个三元组(tid, A, y)
  - 其中tid是每个训练样本的标识符，A是一组特征属性值：A={a1, a2, a3, ...}，而y是训练样本已知的分类标记。

tid	age	sex	cm	kg	发育
1	2	M	80	14	良好
2	3	M	82	12	不良
3	2	F	68	12	良好
…	…	…	…	…	…

分类问题基本描述

- 对于一个测试样本数据集TS={ts1, ts2, ts3, ...}，每个测试样本ts也是一个三元组(tid, A, y')，其中y'未知。
- 需要解决的问题是：根据训练数据集来预测出每个ts的未知的分类标记y'。
- 训练数据集越大，对测试样本分类标记的预测越准确。

分类问题基本描述

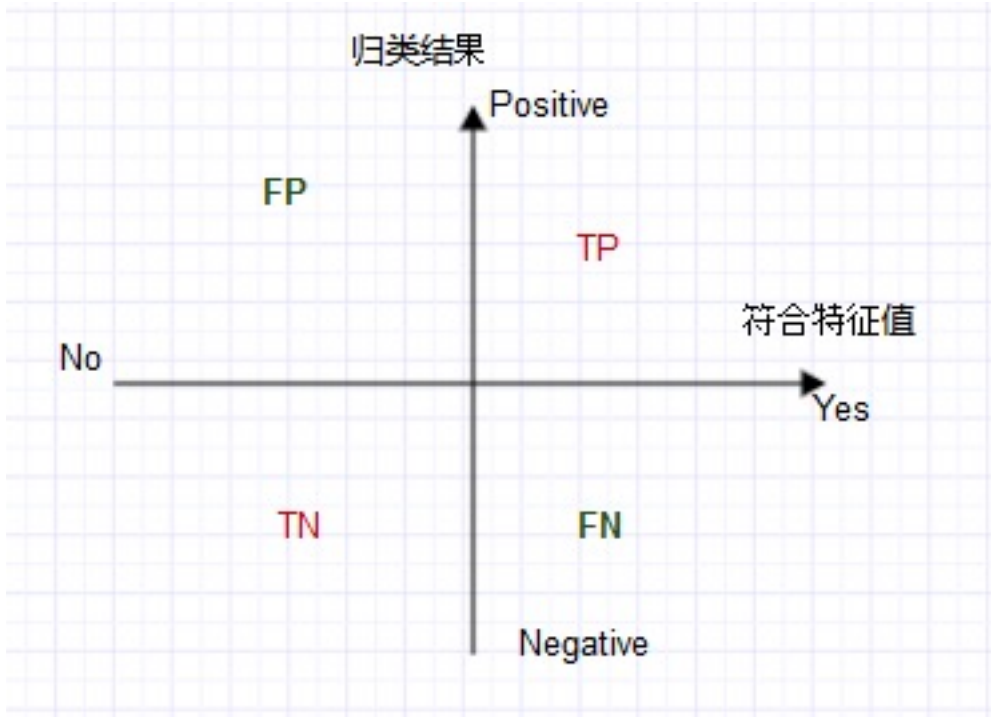
训练样本数据

tid	age	sex	cm	kg	发育
1	2	M	80	14	良好
2	3	M	82	12	不良
3	2	F	68	12	良好
4	2	F	75	17	肥胖
…	…	…	…	…	…

测试样本数据

tid	age	sex	cm	kg	发育
1	2	F	70	15	?
2	2	M	82	12	?
3	3	F	68	12	?
4	2	M	75	17	?
…	…	…	…	…	…

分类算法的评估



分类算法的评估

- Accuracy准确率
  - 对于给定的测试数据集，分类器正确分类的样本数与总样本数之比

$$A(M) = \frac{TN + TP}{TN + FP + FN + TP}$$

- Precision精确率
  - 预测结果中符合实际值的比例，可以理解为没有“误报”的情形

$$P(M) = \frac{TP}{TP + FP}$$

分类算法的评估

- Recall召回率
  - 正确分类的数量与所有“应该”被正确分类（符合目标标签）的数量的比例，可以理解为召回率对应的没有“漏报”的情形。

$$R(M) = \frac{TP}{TP + FN}$$

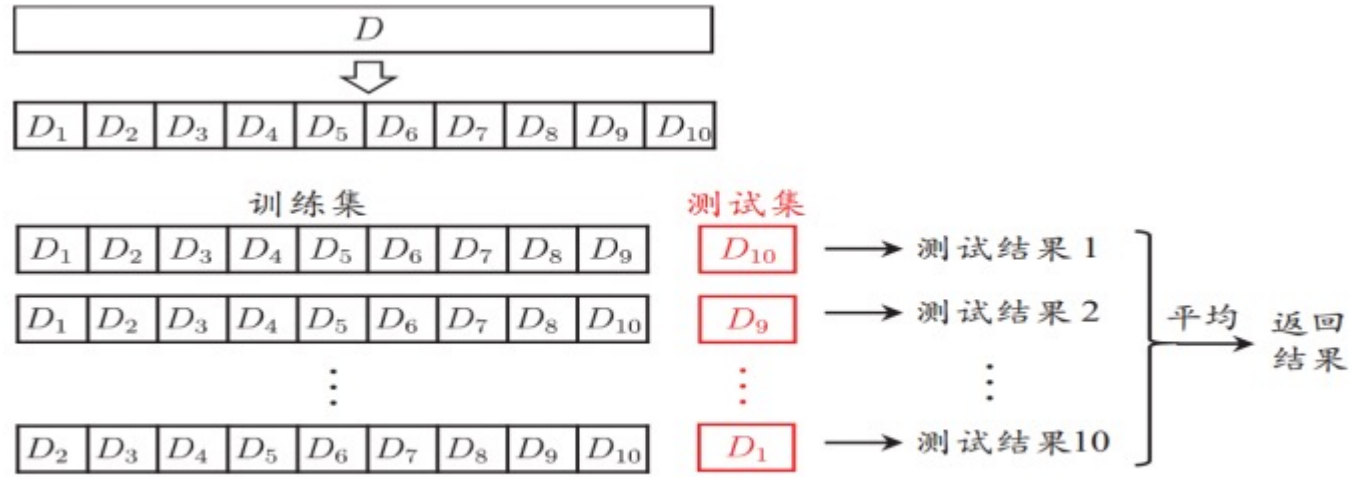
- F1 Score：精确率和召回率的调和均值

$$\frac{2}{F1} = \frac{1}{P} + \frac{1}{R} \quad \Rightarrow \quad F1 = \frac{2TP}{2TP + FP + FN}$$

分类算法的评估方法

- 通常将包含m个样本的数据集$D=\{(x_1,y_1),(x_2,y_2),\ldots,(x_m,y_m)\}$拆分成训练集S和测试集T:
- 留出法:
  - 直接将数据集划分为两个互斥集合
  - 训练/测试集划分要尽可能保持数据分布的一致性
  - 一般若干次随机划分、重复实验取平均值
  - 训练/测试样本比例通常为2:1~4:1
- 交叉验证法:
  - 将数据集分层采样划分为k个大小相似的互斥子集，每次用k-1个子集的并集作为训练集，余下的子集作为测试集，最终返回k个测试结果的均值，k最常用的取值是10。

分类算法的评估方法



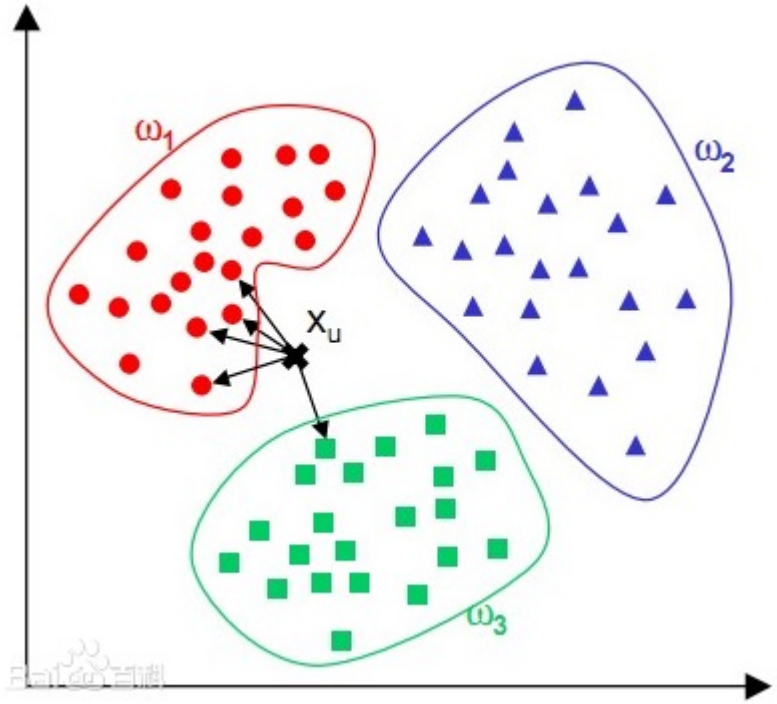
10 折交叉验证示意图

- 为了减小因样本划分不同而引入的差别，k折交叉验证通常随机使用不同的划分重复p次，最终的评估结果是这p次k折交叉验证结果的均值，例如常见的“10次10折交叉验证”。
- 假设数据集D包含m个样本，若令k=m，则得到留一法。不受随机样本划分方式的影响，结果往往比较准确，但当数据集比较大时，计算开销难以忍受。

摘要

- 分类问题的基本描述
- KNN最近邻分类算法
- 朴素贝叶斯分类算法
- 支持向量机SVM分类
- 频繁项集挖掘算法

K-最近邻(KNN)分类并行化算法



K-最近邻(KNN)分类并行化算法

- 基本算法设计思想
  - K-最近邻是分类器算法中最通俗易懂的一种，计算测试样本到各训练样本的距离，取其中距离最小的K个，并根据这K个训练样本的标记进行投票得到测试样本的标记。
  - 加权K-最近邻分类算法的思路是，在根据测试样本的标记进行投票表决时，将根据测试样本与每个训练样本间距离（或相似度）的大小决定训练样本标记的作用大小，基本原则是：距离越近的训练样本其标记的作用权重越大，反之则越小。据此，可以建立一个带加权的投票表决计算模型（比如y' = ΣSi*yi/Σsi，k=[0,k-1]，Si为取值0-1的相似度数值，yi为选取出的最邻近训练样本的分类标记值）决定以最终的测试样本的分类标记。
- 算法的思路清晰简单，然而对于海量数据计算量很大，耗费时间较长。

MapReduce并行化算法设计思路

- 基本处理思路是：将测试样本数据分块后分布在不同的节点上进行处理，将训练样本数据文件放在DistributedCache中供每个节点共享访问。
- Map阶段对每个读出的测试样本数据ts(trid, A', y')
  - 计算其与每个训练样本数据tr(trid, A, y)之间的相似度S=Sim(A', A) （1：相似度最大，0：相似度最小）
  - 检查S是否比目前的k个S值中最小的大，若是则将(S, y)计入k个最大者
  - 根据所保留的k个S值最大的(S, y)，根据模型y' =ΣSi*yi/Σsi计算出ts的分类标记值y'，发射出(tsid, y')
- Reduce阶段直接输出(tsid, y')

MapReduce并行化算法实现

- Mapper伪代码

```
class Mapper{
        setup(…){ //读取全局训练样本数据文件，转入本地内存的数据表TR中 }
        map(key, ts) { // ts为一个测试样本
          Φ → MaxS(k)
          ts →tsid, A', y'
          for i=0 to TR.lenghth {
              TR[i] → trid, A, y
              S = Sim(A, A');
              若S属于k个最大者，(S, y) → MaxS;
          }
          根据MaxS和带加权投票表决模型计算出y' =ΣSi*yi/ΣSi
           emit(tsid, y')
    }
}
```

摘要

- 分类问题的基本描述
- KNN最近邻分类算法
- 朴素贝叶斯分类算法
- 支持向量机SVM分类
- 频繁项集挖掘算法

贝叶斯定理

- 在一个论域中，某事件A发生的概率用P(A)表示，事件的条件概率P(A|B)的定义为：在事件B已经发生的前提下事件A发生的概率。
  - P(A|B) = P(A)*P(B|A)/P(B)

- 朴素贝叶斯分类器基于一个简单的假定：给定目标值时属性之间相互条件独立。

朴素贝叶斯分类算法

- 基本问题描述
  - 设每个数据样本用一个n维特征向量来描述n个属性的值，即：$X = \{x1, x2, \ldots, xn\}$，假定有$m$个类，分别用$Y1, Y2, \ldots Ym$表示
  - 给定一个未分类的数据样本X，若朴素贝叶斯分类将未知的样本X分配给类Yi，则一定有$P(Yi|X) > P(Yj|X),\ 1 \le j \le m,\ j \ne i$
  - 根据贝叶斯定理
  - $P(Yi|X) = \frac{P(Yi)*P(X|Yi)}{\sum_{i=1}^{k} P(Yi)*P(X|Yi)} = \frac{P(Yi)*P(X|Yi)}{P(X)}$
  - 由于$P(X)$对于所有类为常数，概率$P(Yi|X)$可转化为概率$P(X|Yi) * P(Yi)$。

朴素贝叶斯分类算法

- 如果训练数据集中有很多具有相关性的属性，计算$P(X|Yi)$将非常复杂，为此，通常假设各属性是互相独立的，这样$P(X|Yi)$的计算可简化为求$P(x1|Yi)$，$P(x2|Yi)$，…，$P(xn|Yi)$之积；而每个$P(xj|Yi)$可以从训练数据集近似求得。

- 据此，对一个未知类别的样本X，可以先分别计算出X属于每一个类别Yi的概率$P(X|Yi)*P(Yi)$，然后选择其中概率最大的Yi作为其类别。

朴素贝叶斯分类并行算法

- 哪些部分可以并行，而结果又如何汇总？
- 根据前述的思路，判断一个未标记的测试样本属于哪个类Yi的核心任务成为：根据训练数据集计算Yi出现的频度和所有属性值xj在Yi中出现的频度。
- 据此，并行化算法设计的基本思路是：用MapReduce扫描训练数据集，计算每个分类Yi出现的频度FYi(即$P(Yi)$)、以及每个属性值出现在Yi中的频度$FxYij$ (即$P(xj|Yi)$)

朴素贝叶斯分类并行算法

- 而在MapReduce中对训练数据集进行以上的频度计算时，实际上就是简单地统计Yi和每个xj在Yi中出现的频度
- 在进行分类预测时，对一个未标记的测试样本X，根据其包含的每个具体属性值xj，根据从训练数据集计算出的$FxYij$进行求积得到$FxYij$(即$P(X|Yi)$，再乘以FYi即可得到X在各个Yi中出现的频度$P(X|Yi) * P(Yi)$，取得最大频度的Yi即为X所属的分类。此过程在Map阶段实现。
- Reduce过程只需将各节点的统计频度汇总。

朴素贝叶斯分类并行算法

- 训练算法的Mapper伪代码

```
//输入数据：整个训练样本数据集
//输出数据：各Map节点输出的局部频度数据FYi和FxYij
class TrainMapper{
   map(key, TR){
       //TR为一个训练样本，由一个样本标志trid、属性x以及分类值y组成
      TR→trid, X, y
      emit(y,1)
      for j=0 to X.length(){
         X[j] → 属性名xnj和属性值xvj
         emit(<y, xnj, xvj>, 1)
      }
   }
}
```

朴素贝叶斯分类并行算法

- 训练算法的Reducer伪代码

```
//输入数据：Map节点输出的局部频度数据FYi和FxYij
//输出数据：全局的频度数据FYi和FxYij
class TrainReducer{
   reduce(key, value_list){
     //key或为分类标记y，或为<y, xnj, xvj>
     //value_list中的每个值是key出现的次数
     //累加后即为Fyi或者FxYij
     //reduce只是简单地汇总两个频度即可
      sum = 0
      while (value_list.hasNext()){  sum+= value_list.next().get();  }
      emit(key, sum)
  }
}   //Reduce完成后，FYi和FxYij频度数据将输出并保存到HDFS文件中
```

- 输出结果为所有Yi的出现频度FYi，以及所有属性值xj在Yi中的出现频度；
- 进行未标记测试样本X的分类预测时，可以从这个输出数据文件中快速查找相应的频度值并最终计算出X在各个Yi中出现的频度P(X|Yi)P(Yi)，最后取得最大频度的Yi即为X所属的分类

朴素贝叶斯分类并行算法

- 分类预测算法的Mapper伪代码

```
//输入数据：测试样本数据集
//输出数据：各测试样本及其分类结果
class TestMapper{
    setup(…){
        //初始化时读取从训练数据集得到的频度数据
        //分别装入频度表FY和FxY供各个map节点共享访问
        //分类频度表FY = { (Yi, 每个Yi的频度FYi) }
        //属性频度表FxY= { (<Yi, xnj, xvj>, 出现频度FxYij)}

    }
```

朴素贝叶斯分类并行算法


```
map(key, ts){ // ts为一个测试样本
   ts →tsid, A
   MaxF= MIN_VALUE; idx= -1;
   for (i=0 to FY.length){
      FXYi= 1.0; Yi = FY[i].Yi; FYi= FY[i].FYi;
      for (j=0 to A.length){
         xnj= A[j].xnj; xvj= A[j].xvj
         根据<Yi, xnj, xvj>扫描FxY表, 取得FxYij
         FXYi= FXYi* FxYij;
      } //执行到此，FXYi等价于公式中的P(X|Yi)
      if(FXYi* FYi>MaxF) { MaxF= FXYi*FYi; idx= i; }
    }
    emit(tsid, FY[idx].Yi)
}
```

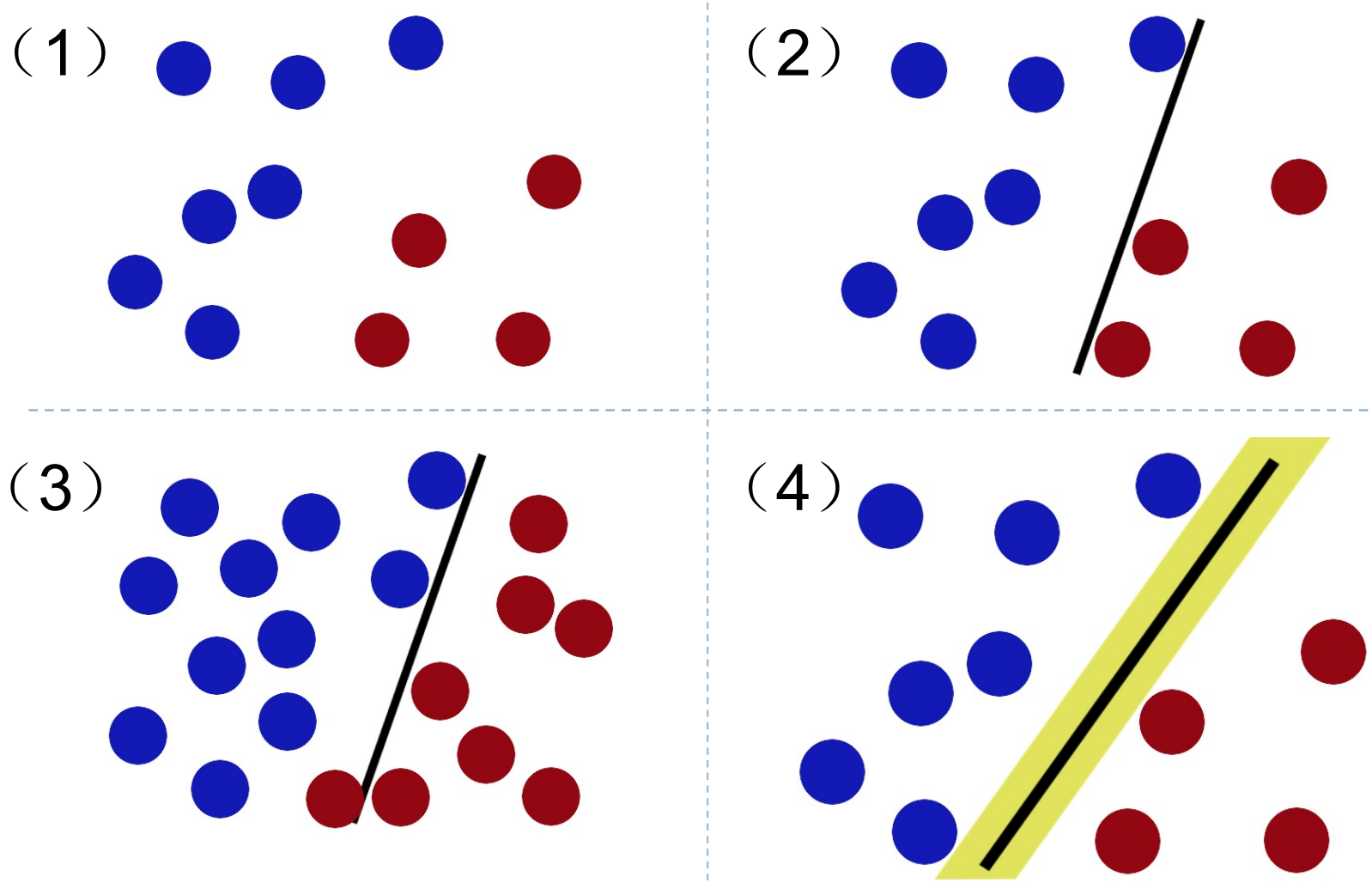
摘要

- 分类问题的基本描述
- KNN最近邻分类算法
- 朴素贝叶斯分类算法
- 支持向量机SVM分类
- 频繁项集挖掘算法

支持向量机(SVM)

- SVM（Support Vector Machine）指的是支持向量机，是常见的一种判别方法。在机器学习领域，是一个有监督的学习模型，通常用来进行模式识别、分类以及回归分析。
- SVM的主要思想可以概括为两点：
  - 它是针对线性可分情况进行分析，对于线性不可分的情况，通过使用非线性映射算法将低维输入空间线性不可分的样本转化为高维特征空间使其线性可分，从而使得高维特征空间采用线性算法对样本的非线性特征进行线性分析成为可能。
  - 它基于结构风险最小化理论之上在特征空间中构建最优超平面，使得学习器得到全局最优化，并且在整个样本空间的期望以某个概率满足一定上界。

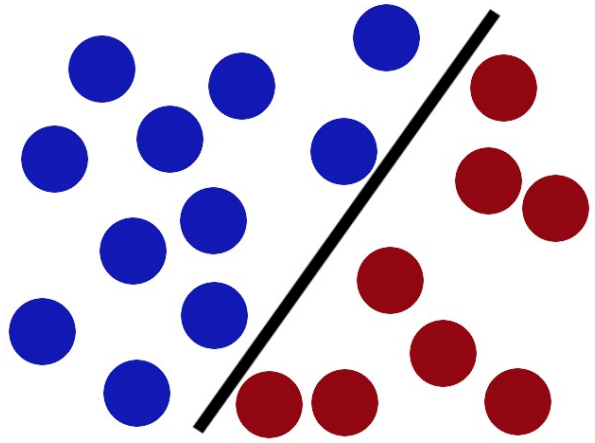
支持向量机(SVM)



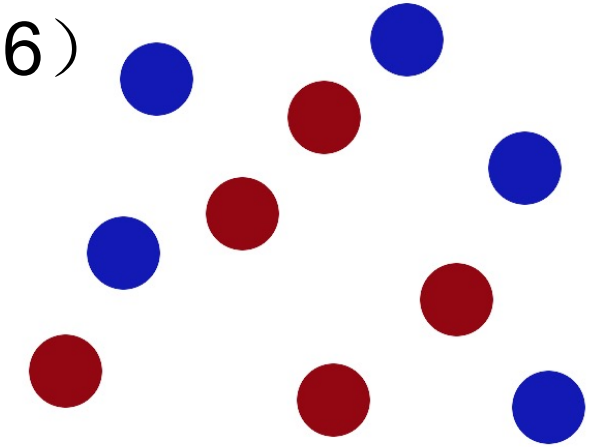
引自： https://www.reddit.com/r/MachineLearning/comments/15zrpp/please_explain_support_vector_machines_svm_like_i/

支持向量机(SVM)

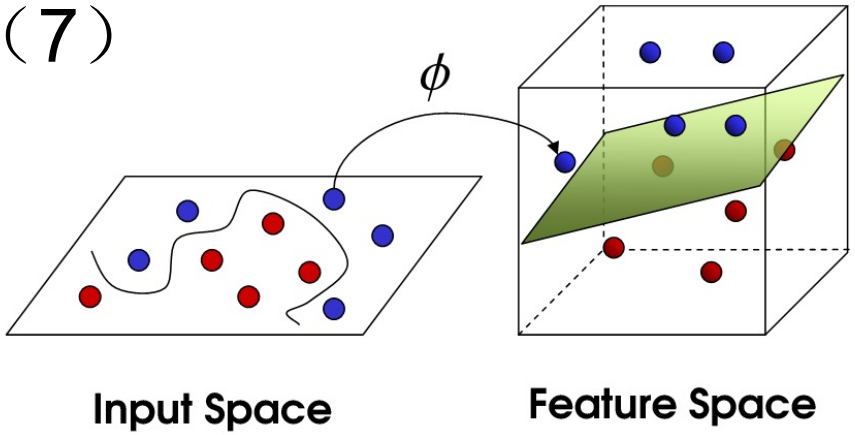
（5）



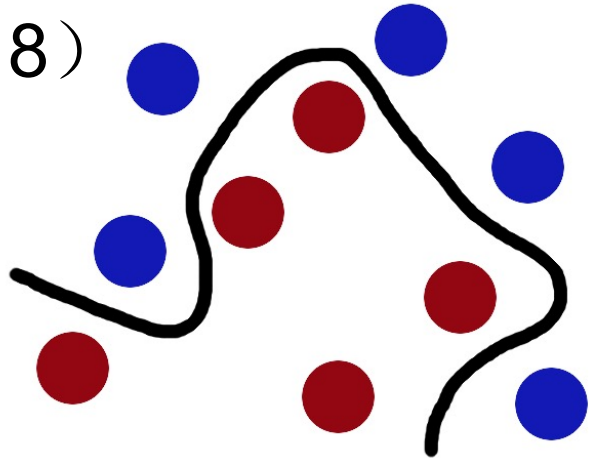
（6）



（7）



（8）



支持向量机(SVM)应用

- 有助于文本和超文本分类
- 图像的分类
- 识别手写字符
- 生物科学和其他科学领域，比如对高达90%正确分类的化合物进行蛋白质分类。
- ……

如何使用SVM

- 入门级：实现并使用各种版本的SVM
- 专业级：尝试、组合核函数
- 专家级：根据问题而设计目标函数、替代损失...

- 常用软件包
  - LIBSVM：https://www.csie.ntu.edu.tw/~cjlin/libsvm/
  - MATLAB： SVM and Kernel Methods MATLAB Toolbox
  - R： e1071 - Machine learning library for R
  - Python： numpy，scipy

SVM短文本多分类并行化算法

- 基本问题描述
  - 本问题是2012年中国第一届“云计算与移动互联网大奖赛”指定的4个大数据并行处理赛题之一，本系研究生组队参加，经过角逐获得1、2、3等奖各一项。
  - 基于MapReduce的查询短文本多分类并行化算法研究。提供了1万条已经标注出所属类别的短文本样本数据作为训练样本，一共有480个类别。测试数据有1000万条查询短文本样本数据，其中有少数不属于这480类的异类测试样本，需要对这些大量的短文本进行分类，并能标识出不属于以上480类的异类样本。
  - 每个短文本样本数据由一个n维的高维特征向量构成

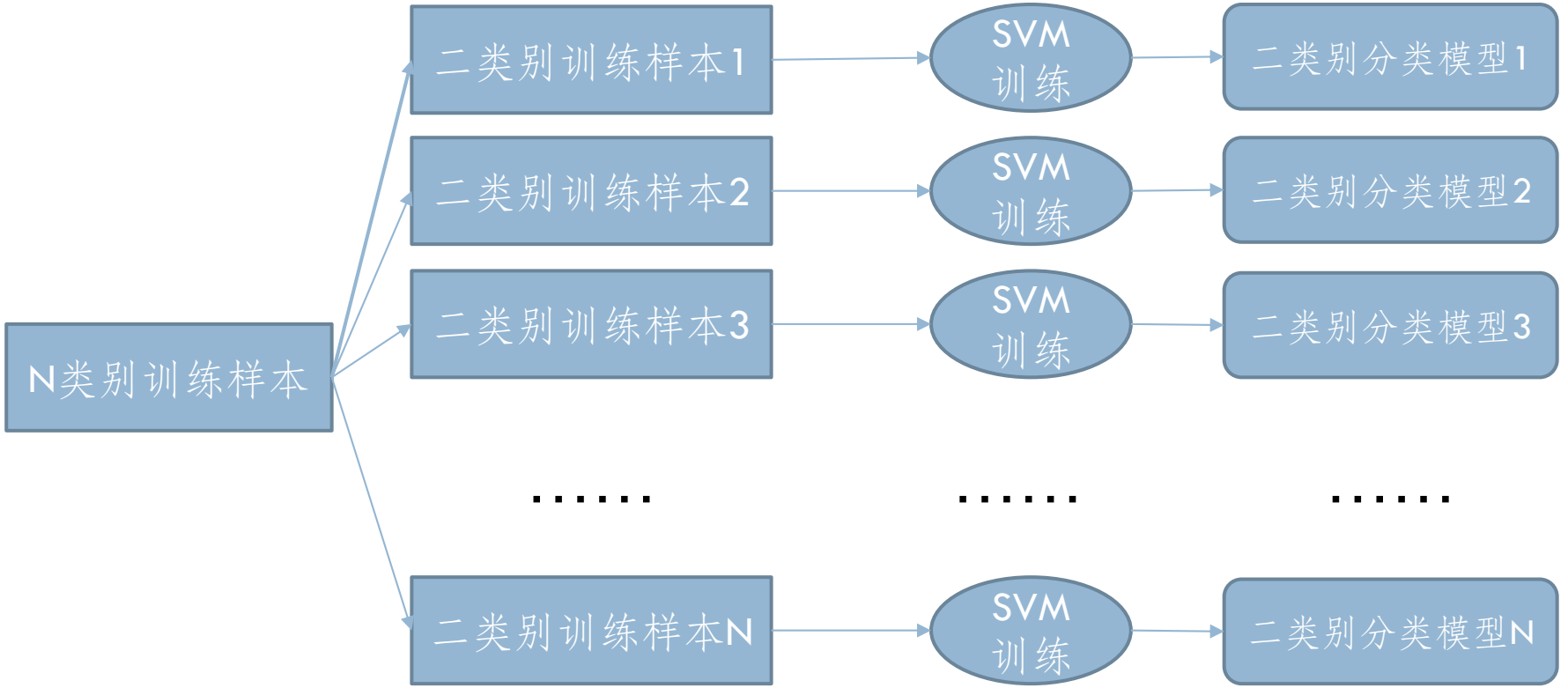
SVM短文本多分类并行化算法

- 特点：
  - 短文本包含的词语相对较少，因此其构造出来的特征向量往往具有高维稀疏性；
  - 短文本的目标分类种类很多，因此不再是一个简单的二分类问题，而直接采用一个多类别分类器可获得的精度较差，且其训练流程难以并行化。

SVM短文本多分类并行化算法

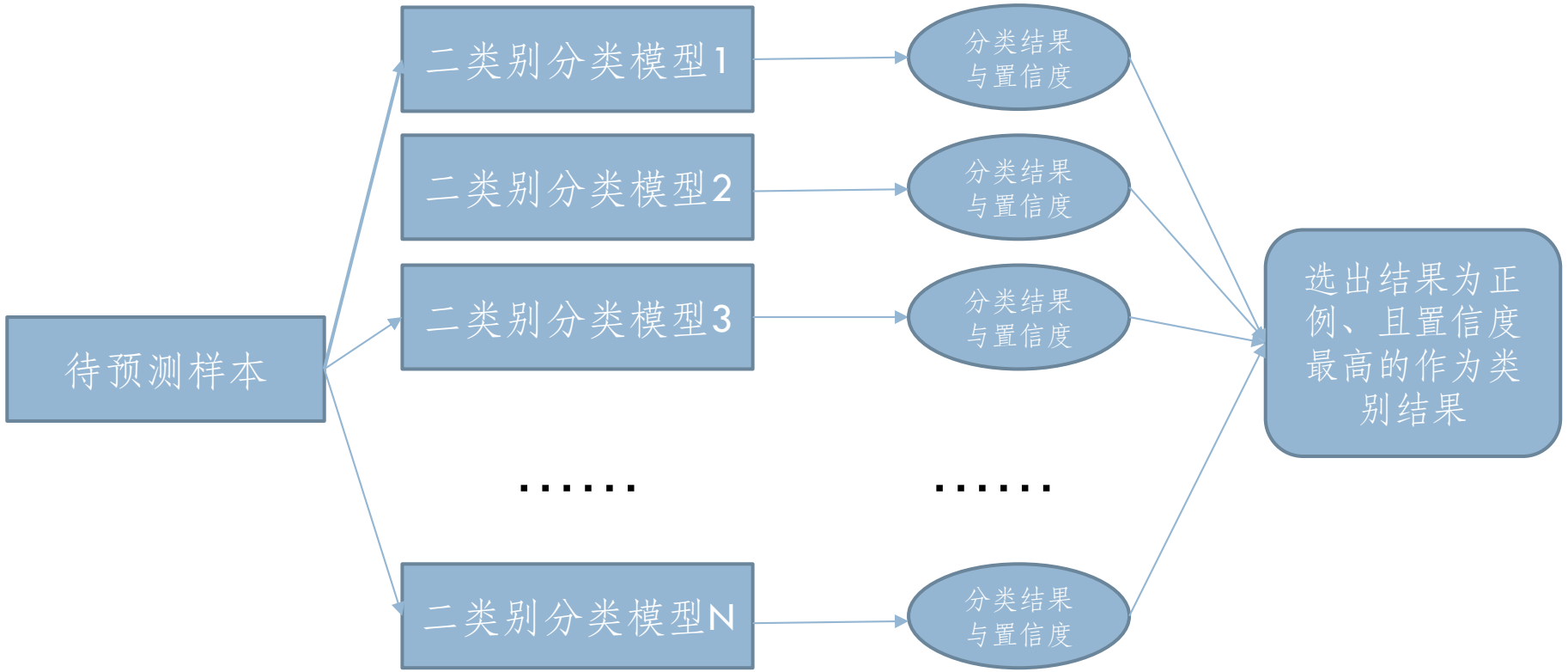
- 基本算法思路
  - 本道题目是高维稀疏空间文本的分类问题。由于大量实践证实SVM 针对高维空间数据训练效果较好，而且分类器的速度较快，因此将使用linear SVM进行处理
    - 训练阶段，对于多类（480 类）问题，为了提高分类精度，首先针对每个类做一个2-Class两类分类器；
    - 预测阶段，分别用480 个分类器对每个待预测的样本进行分类并打分，选择分类为“是”且打分最高的类别作为该样本可能的预测类别；如打分低于最低阈值，则将该测试样本判定为不属于480类的异类

SVM短文本多分类并行化算法



N类训练样本训练出N个二类别分类器的流程图

SVM短文本多分类并行化算法



对待预测样本确定其类别的流程图

SVM短文本多分类并行化算法

- 训练样本很大，样本类别多，待预测的样本数量巨大等问题会导致训练时间很长。
- 实现基于MapReduce并行化的多分类器，包括：
  - 数据预处理
  - 训练N个二类别分类器
  - 预测样本

基本算法设计思路

- 第一步：用训练数据产生480个2-class分类器模型
  - Map：将每个训练样本的分类标签ClassID作为主键，输出(ClassID, <true/false，特征向量>)
  - Reduce: 具有相同ClassID的键值对进入同一个Reduce，然后训练出一个2-Class SVM分类模型共输出480个2-Class SVM分类模型

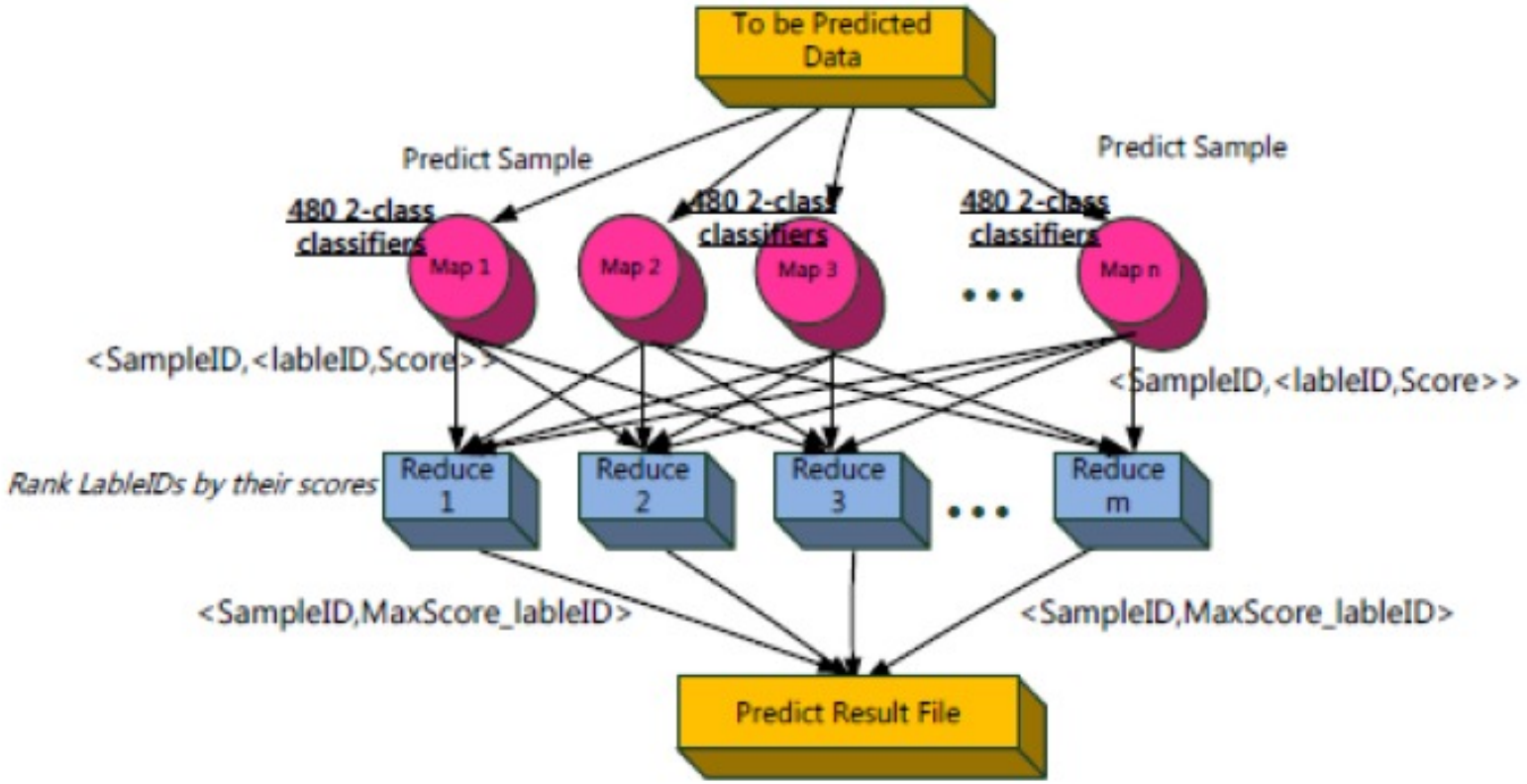
基本算法设计思路

基于MapReduce的480个两类分类器并行训练算法

基本算法设计思路

- 第二步：用480个2-Class分类器模型处理测试数据
  - Map：将每个测试样本，以SampleID作为主键，输出(SampleID, <LableID，评分Score>)
  - Reduce: 具有相同SampleID的键值对进入同一个Reduce，然后以最高评分者对应的标记作为该样本最终的标记；虽然是最高评分，但仍然低于最小阈值，则判定为不属于已知的480个类

基本算法设计思路



基于MapReduce的480个两类分类器并行预测算法

摘要

- 分类问题的基本描述
- KNN最近邻分类算法
- 朴素贝叶斯分类算法
- 支持向量机SVM分类
- 频繁项集挖掘算法

频繁项集挖掘

- 关联规则或频繁项集
  - 频繁项可以看作是两个或多个对象的“亲密”程度，如果同时出现的次数很多，那么这两个或多个对象可以认为是高关联性的。当这些高关联性对象的项集出现次数满足一定阈值时即称其为频繁项。
- 经典实例：购物篮分析
  - 啤酒和尿布

频繁项集

- m个项的集合： $I = \{I_1, I_2, \ldots, I_m\}$
- n个事务的数据库： $D = \{T_1, T_2, \ldots, T_n\}$，其中$T_i$是$I$的非空子集。
- 关联规则： $X \Rightarrow Y$ ，其中X，Y是I的子集，并且$X \cap Y = \emptyset$
- 关联规则的$X \Rightarrow Y$的支持度是指D中事务包含$X \cup Y$的百分比，置信度是指包含X的事务中同时包含Y的百分比，也就是条件概率$P(Y|X)$
- 如果某一规则$X \Rightarrow Y$的置信度和支持度都高于某个阈值，那么可以认为X，Y具有关联性。

频繁项集

- 把满足上述条件的X，Y组成一个项集A，则可以认为，数据库D中包含A的事务超过了某个最小支持度。而所谓频繁项集挖掘，就是将所有满足项数为某个固定整数k，且支持度高于某阈值的项集计算出来。
- 经典算法：Apriori，FP-growth，SON等。

Apriori算法

- 通过多轮迭代的方法来逐步挖掘频繁项集
- 先验原理：
  - 频繁项集的任何非空子集都是频繁的。
  - 非频繁项集的任何超集都是非频繁的。

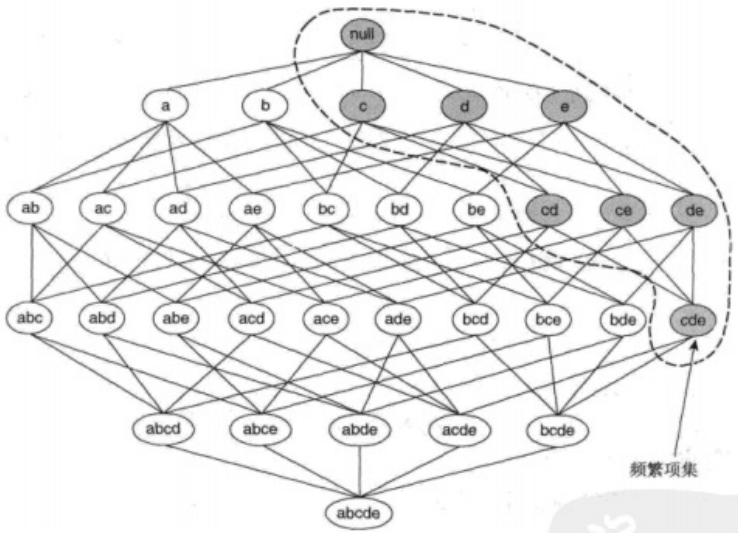


图 6-3 先验原理的图示。如果{c, d, e}是频繁的，则它的所有子集也是频繁的

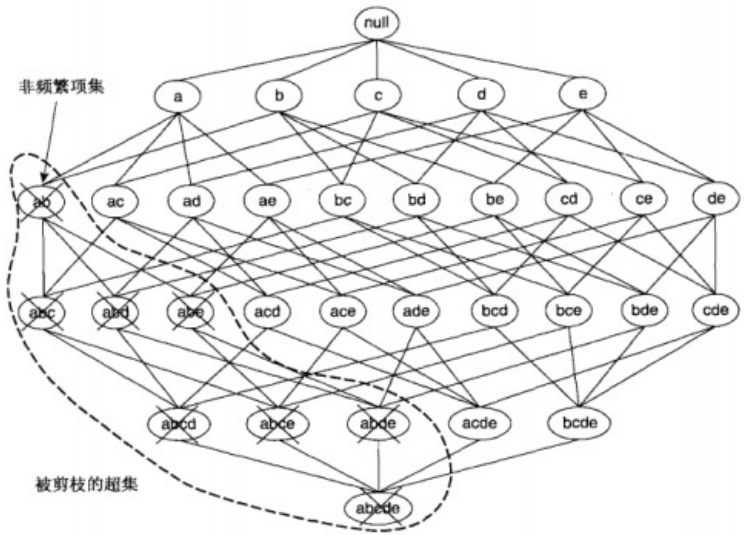


图 6-4 基于支持度的剪枝的图示。如果{a,b}是非频繁的，则它的所有超集也是非频繁的

Apriori算法

L_1={frequent 1-itemsets};

For (k=2; $L_{k-1} \neq$ null; k++} do begin　　//多轮迭代过程

　C_k=apriori-gen(L_{k-1});　　//候选k-项集

　for each transaction t ∈ D do begin

　　$C_t = \text{subset}(C_k, t)$;　　//生成子集

　　for each candidate c ∈ C_t do

　　c.count ++;　　//计算实际的支持度

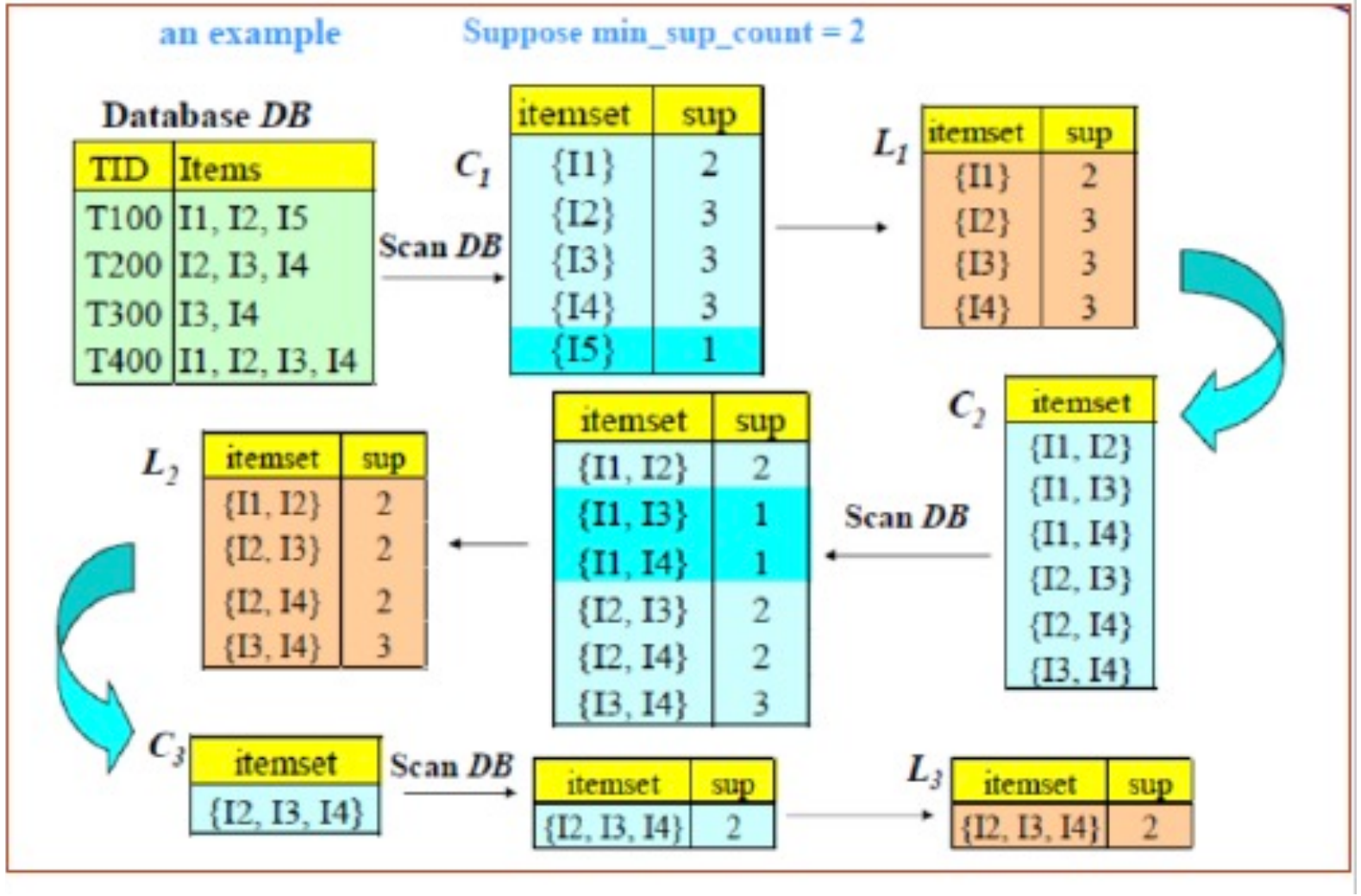
　　end

　　L_k={ c= C_k | c.count ≥ minsup }

　end

Answer = $U_k L_k$

Apriori算法


an example
Suppose min_sup_count = 2
Database DB
TID Items
T100 I1, I2, I5
T200 I2, I3, I4
T300 I3, I4
T400 I1, I2, I3, I4
Scan DB
C1
itemset sup
{I1} 2
{I2} 3
{I3} 3
{I4} 3
{I5} 1
L1
itemset sup
{I1} 2
{I2} 3
{I3} 3
{I4} 3
C2
itemset
{I1, I2}
{I1, I3}
{I1, I4}
{I2, I3}
{I2, I4}
{I3, I4}
Scan DB
itemset sup
{I1, I2} 2
{I1, I3} 1
{I1, I4} 1
{I2, I3} 2
{I2, I4} 2
{I3, I4} 3
L2
itemset sup
{I1, I2} 2
{I2, I3} 2
{I2, I4} 2
{I3, I4} 3
C3
itemset
{I2, I3, I4}
Scan DB
itemset sup
{I2, I3, I4} 2
L3
itemset sup
{I2, I3, I4} 2

Apriori并行化算法设计

- 主进程扫描事务数据库，从原始数据集中产生候选1-项集
- 由候选1-项集与原始数据集进行对比，算出每个项集的支持度，这些支持度与程序中给定的支持度进行对比，得出频繁1-项集
- 由频繁1-项集产生候选2-项集，并通过与原始数据集比较得到频繁2-项集
- 逐次迭代直到产生候选k-项集，候选k-项集与原始数据集对比，如果存在频繁k-项集，则继续迭代执行；如果不存在，则最终得到频繁(k-1)-项集

Apriori并行化算法设计

- 第一轮迭代：计算频繁1-项集
- Map阶段
  - 输入key：事务数据库每行首字符相对于文本文件的首地址偏移；输入value：存储在事务数据库文件中的一行数据
  - 输出key：候选1-项集　输出value：1

```
Map Task:
    For each transaction t in Si
      Map(line offset, t)
        For each item I in t
            Emit (I, 1);
        End For each
      End Map()
    End For each
End Map
```

Apriori并行化算法设计

- Reduce阶段
  - 输出key：频繁1-项集；输出value：出现的次数

```
Reduce Task:
    Reduce(key2, value2)
        Sum = 0;
    While (value2.hasNext())
        Sum += value2.hasNext();
     End While
     If (Sum>=Minsup)
         Emit (key2, sum)            //得到频繁1-项集
     End If
End Reduce
```

Apriori并行化算法设计

- 第k轮MapReduce迭代：计算k-频繁项集
- Map阶段
  - 输入<key,value>对为：事务数据分片中的每个事务数据。这个阶段会从频繁(k-1)-项集得到候选k-项集，频繁(k-1)-项集存放在Distributed Cache中以便每个Map节点共享读取和使用。为了产生候选k-项集，包括连接和剪枝过程。
  - 输出key为候选k-项集，value为其出现的次数。

Apriori并行化算法设计

Map Task:

Read L_{k-1} from HDFS file

C_k = ap_gen(L_{k-1})　　//self join连接过程

For each transaction t in Si

Map(line offset, t)

C_t = subset (C_k , t)　　//获取在原始事务数据中出现的候选项集

For each item c in C_t

Emit (c, 1);

End For each

End Map()

End For each

End Map

Apriori并行化算法设计

- Reduce阶段
  - 输出key：频繁k-项集；输出value：出现的次数

```
Reduce Task:
    Reduce(key2, value2)
        Sum = 0;
    While (value2.hasNext())
        Sum += value2.hasNext();
     End While
     If (Sum>=Minsup)
         Emit (key2, sum)            //得到频繁k-项集
     End If
End Reduce
```

基于子集求取的频繁项集挖掘算法

- 一个频繁项集F必然是D中某个事务的子集，即只要根据事务数据本身就可以计算频繁项集。
- 假设数据库为D，事务为T，项集大小为k
  - 首先扫描数据库，对数据库中的每一个事务做如下操作：将该事务所有大小为k的子集求出来(Mapper)
  - 然后，统计输出所有子集的个数，如果某个子集的个数超过了某一阈值S，那么就可以认为这个子集是频繁项集，将所有这样的子集输出即可(Reducer)

基于子集求取的频繁项集挖掘算法

- Mapper

```
Class MiningMapper{
    map(T){
      //T为数据库中的一个事务
      //求出T所有大小为k的子集
      //若T的项数小于k，则子集数为0
      Subsets = FindSubsets(T, k);
      for (i=0; i<Subsets.size(); i++)
          subset = Subsets.get(i)
          emit(subset, 1)
      }
}
```

基于子集求取的频繁项集挖掘算法

- Reducer

```
Class MiningReducer{
    reduce(key, value_list){
        sum = 0;
        while(value_list.hasNext())
            sum += value_list.next().get();
        if(sum > minSupport)
            emit(key, sum);
    }
}
```

SON算法

- 有限扫描：最多两遍之内发现全部或者大部分频繁项集。只利用数据抽样样本而不是全部数据集进行计算。
- 基本思路：先采样部分样本，在其上运行Apriori算法，并调整相应支持度阈值。采样得到的子数据集直接存入主存，之后扫描数据自己不用再次进行I/O操作。
  - 例如选择1%的样本，支持度阈值为s/100
- 抽样可能导致完整数据集上的频繁项在样本中不频繁（false negative），样本中频繁但在整个数据集上不频繁（false positive）

* A. Savasere, E. Omiecinski, and S. Navathe, “An efficient algorithm for mining association rules in large databases,” in proceedings of the 21st VLDB Conference Zurich, Swizerland, 1995

SON算法

- SON算法同时避免了false negatives和false positives，所带来的代价是需要两次扫描。
- 基本思想：将输入文件划分成1/p个块（p为样本占全部数据集的比例），将每个文件块作为一个样本，并执行Apriori算法。同样适用p*s作为其阈值。将每个块找到的频繁项集放到磁盘上。

SON算法

- 第一步：找到局部频繁项集
  - 一旦所有的块按此方式被处理，将那些在一个或多个块中被选中的频繁项集收集起来作为候选频繁项集。注意，如果一个项集在所有的块中都不是频繁项集，即它在每个块中的支持度都低于p*s。因为块数为1/p，所以，整体的支持度也就低于(1/p)*p*s=s。这样，每个频繁项集必然会出现在至少一个块的频繁项集中，于是，我们可以确定，真正的频繁项一定全在候选频繁项集中，因此这里没有false negatives。
- 第二步：找出全局频繁项集
  - 计算所有的候选频繁项集，选择那些支持度至少为s的作为频繁项集。

PSON：基于MapReduce的并行化SON算法

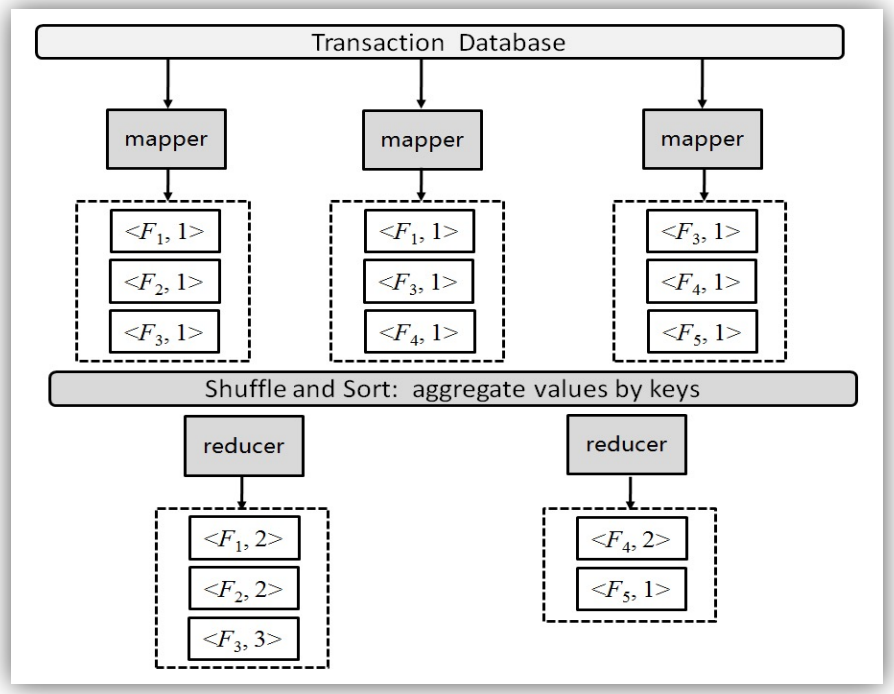
- 第一个Map 函数：使用子集，并在子集上使用Apriori算法找出每个项集的频繁度。将支持度从s降为p*s，输出(F,1)，F为该样本的频繁项集。
- 第一个Reduce函数：产生那些出现一次或多次的key（即项集），输出候选项集。
- 第二个Map函数：接受第一个Reduce函数的所有输出和输入数据文件的一部分，每个Map任务计算每个候选项集在当前数据集上出现的次数，输出为(C,v)组成的集合，其中C是一个候选集，v是C在本Map任务所分配数据上的支持度。
- 第二个Reduce函数：将分配的项集作为Key，并将关联的值求和。最终得到每个Reduce任务所分配的每个项集在整个数据集上的全部支持度。那些值求和后不低于s的项集是整个数据集上的频繁项集，Reduce任务会将这些项集及其计数值输出。

* Tao Xiao, Shuai Wang, Chunfeng Yuan, Yihua Huang. PSON: A Parallelized SON Algorithm with MapReduce for Mining Frequent Sets, The Fourth International Symposium on Parallel Architectures, Algorithms and Programming (PAAP 2011), Tianjin，Dec. 9-11, 2011

1st MapReduce Job: generate all global candidate itemsets

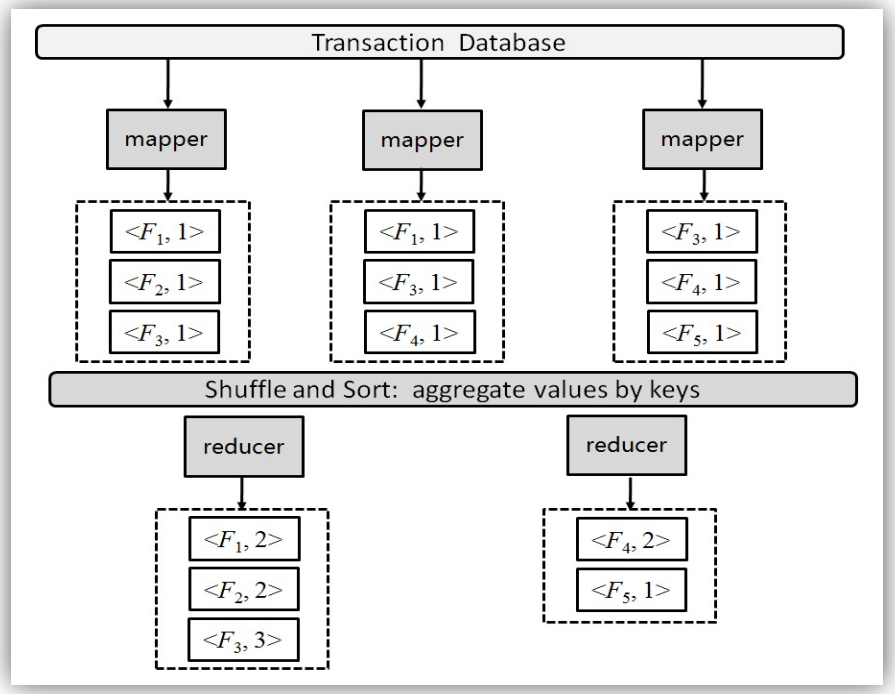
- **Map phase**
  - Each map node takes in one partition and generates local frequent itemsets for that partition using Apriori algorithm.
  - For each local frequent itemset *F*, emits key-value pair <*F*, 1>. Here, the value 1 is only to indicate that *F* is a local frequent itemset for that partition.
- **Shuffle and Sort phase**
  - The same local frequent itemsets are sent to one reduce node.
- **Reduce phase**
  - Each reduce node emits one and only one key-value pair

    <F, 1> to DFS
- **Finally**
  - Merging all the pairs in DFS gives us all global candidate itemsets



2nd MapReduce Job: identify global frequent itemsets

- **Assumption**
  - Each node is given a full duplicate of the global candidate itemsets generated by the 1st MapReduce job beforehand
- **Map phase**
  - Each map node counts for each of the global candidate itemsets in the partition the map node is assigned
  - Then emits pairs like <C, v> where C is a global candidate itemset and v is the count of it in that partition
- **Shuffle and Sort phase**
  - Each global candidate itemset and its counts in all the partitions are sent to one reduce node
- **Reduce phase**
  - For each global candidate itemset C, reduce node adds up all the associative counts for C and emits only the actual global frequent itemsets to DFS



THANK YOU

南京大学
NANJING UNIVERSITY

南京大学计算机软件研究所
Institute of Computer Software, Nanjing University