Apriori算法是一种用于从大规模数据中挖掘频繁项集（Frequently Bought Together）和关联规则（如果A那么B） 的经典算法。

它的核心思想非常直观：“一个频繁项集的所有子集也一定是频繁的”。反过来，如果一个项集不是频繁的，那么它的所有超集也不可能是频繁的。这个性质大大减少了需要计算的候选集数量。

为什么需要Apriori？

想象一下超市的购物数据。我们想知道“哪些商品经常被一起购买”。最笨的方法是计算所有可能的商品组合（如{啤酒,尿布}、{啤酒,牛奶}...）的出现次数。但如果有1万种商品，组合数会是一个天文数字（2^10000 - 1），计算根本不可行。

Apriori算法通过一种“智能”的、逐层搜索的方法，高效地解决了这个问题。

核心概念与指标

在理解算法步骤前，需要先了解三个核心指标：

支持度 (Support)

含义：一个项集（商品组合）在所有交易中出现的频率。

公式：Support(A) = (包含A的交易数) / (总交易数)

作用：衡量项集的普遍性。支持度太低说明这个组合不常见，没有商业价值。

置信度 (Confidence)

含义：在包含A的交易中，同时包含B的概率。即规则 A → B 的可靠性。

公式：Confidence(A → B) = Support(A ∪ B) / Support(A)

例子：置信度(尿布 → 啤酒) = 0.6 意味着买尿布的人中，有60%也买了啤酒。

提升度 (Lift)

含义：衡量A和B之间的相关性，而不仅仅是条件概率。

公式：Lift(A → B) = Confidence(A → B) / Support(B)

解读：

提升度 = 1：A和B独立，购买A对是否购买B没有影响。

提升度 > 1：A和B正相关，规则有效且有价值。值越大，相关性越强。

提升度 < 1：A和B负相关，购买A反而可能降低购买B的概率。

Apriori算法两步走

Apriori算法主要分为两个阶段：

找出所有频繁项集（满足最小支持度的商品组合）。

从频繁项集中生成强关联规则（满足最小置信度的规则）。

第一阶段：找出频繁项集 (Frequent Itemsets)

这一步的目标是找到所有支持度高于我们设定阈值（min\_support）的商品组合。

算法步骤（迭代过程）：

扫描所有交易，计算每个单一商品的支持度。保留那些支持度 ≥ min\_support的商品，形成频繁1-项集集合 L1。

基于 L1，生成所有可能的候选2-项集（即商品两两组合），记为 C2。

再次扫描数据库，计算每个候选2-项集的支持度。保留支持度 ≥ min\_support的，形成频繁2-项集集合 L2。

基于 L2，生成候选3-项集 C3。这里就用到Apriori的核心思想了：要生成 {A, B, C} 作为候选，其前提是 {A,B}, {A,C}, {B,C} 都必须在 L2 中（即都是频繁的）。如果 {A,B} 不频繁，那么任何包含 {A,B} 的超集（如{A,B,C}）绝不可能是频繁的，直接剪掉！

重复这个过程（扫描数据库计算支持度 -> 剪枝 -> 生成更高阶的候选集），直到不能再生成新的频繁项集为止（即 Ck 为空）。

这个过程大大减少了需要考察的候选集数量，是Apriori高效的关键。

第二阶段：生成关联规则 (Association Rules)

从上一步找到的频繁项集中，我们可以生成关联规则。

算法步骤（对于每一个频繁项集I）：

对于频繁项集I，生成所有可能的非空子集。例如，对于项集 {A, B, C}，其子集有 {A}, {B}, {C}, {A,B}, {A,C}, {B,C}。

对于每一个子集 s，可以形成一条规则 s → (I - s)。例如，子集 {A,B} 可以生成规则 {A,B} → {C}。

计算这条规则的置信度：Confidence = Support(I) / Support(s)

如果该置信度 ≥ 我们设定的阈值（min\_confidence），并且提升度(Lift) > 1，那么我们就保留这条规则，认为它是一条强关联规则。

举例说明（经典“啤酒与尿布”）

假设我们有5条交易记录：

{牛奶, 面包}

{面包, 尿布, 啤酒, 鸡蛋}

{牛奶, 尿布, 啤酒, 可乐}

{面包, 牛奶, 尿布, 啤酒}

{面包, 牛奶, 尿布, 可乐}

目标：找出强关联规则（设 min\_support = 0.4, min\_confidence = 0.6）

第一步：找频繁项集

计算所有1-项集支持度：

Support(牛奶)=4/5=0.8

Support(面包)=4/5=0.8

Support(尿布)=4/5=0.8

Support(啤酒)=3/5=0.6

Support(可乐)=2/5=0.4

Support(鸡蛋)=1/5=0.2

保留支持度≥0.4的，得到 L1 = {牛奶, 面包, 尿布, 啤酒, 可乐}

由L1生成候选2-项集C2并计算支持度，保留支持度≥0.4的：

{牛奶,面包}=0.6

{牛奶,尿布}=0.8

{牛奶,啤酒}=0.4

{面包,尿布}=0.6

{面包,啤酒}=0.4

{尿布,啤酒}=0.6

...其他组合支持度较低

得到 L2 = { {牛奶,面包}, {牛奶,尿布}, {牛奶,啤酒}, {面包,尿布}, {面包,啤酒}, {尿布,啤酒} }

由L2生成候选3-项集。例如，要生成{牛奶,面包,尿布}，需要检查其子集{牛奶,面包}, {牛奶,尿布}, {面包,尿布}是否都在L2中。是的，所以它是一个候选。

计算候选3-项集支持度：

Support({牛奶,面包,尿布}) = 3/5 = 0.6 (交易2,4,5)

Support({牛奶,尿布,啤酒}) = 2/5 = 0.4 (交易3,4)

Support({面包,尿布,啤酒}) = 2/5 = 0.4 (交易2,4)

Support({牛奶,面包,啤酒}) = 1/5 = 0.2 (交易4) -> 小于0.4，丢弃！

得到 L3 = { {牛奶,面包,尿布}, {牛奶,尿布,啤酒}, {面包,尿布,啤酒} }

生成4-项集...最终无法生成频繁4-项集。

第二步：从频繁项集生成规则  
以频繁项集 I = {尿布,啤酒} (Support=0.6) 为例：

它的所有非空子集：{尿布}, {啤酒}

规则1: {尿布} → {啤酒}

置信度 = Support(I) / Support({尿布}) = 0.6 / 0.8 = 0.75 (≥0.6，是强规则)

提升度 = (0.6) / (0.8 \* 0.6) ≈ 1.25 (>1，正相关)

规则2: {啤酒} → {尿布}

置信度 = Support(I) / Support({啤酒}) = 0.6 / 0.6 = 1.0 (≥0.6，是强规则)

提升度 = (0.6) / (0.8 \* 0.6) ≈ 1.25 (>1，正相关)

结论：我们发现了“尿布和啤酒”之间存在强关联规则，可以建议超市将这两种商品摆放在一起或进行联合促销。

优缺点

优点：

原理简单，易于理解和实现。

利用支持度剪枝，大大减少了搜索空间。

缺点：

需要多次扫描数据库（I/O开销大），如果数据库很大，效率会较低。

可能产生大量的候选集（尤其是当1-项集就很多的时候）。

需要谨慎选择min\_support和min\_confidence，设置不当要么找不到规则，要么找到大量无用规则。

为了解决这些缺点，后续提出了更高效的算法，如FP-Growth，它通常只需要扫描两次数据库，并使用一种称为FP树的数据结构来压缩数据，效率更高。