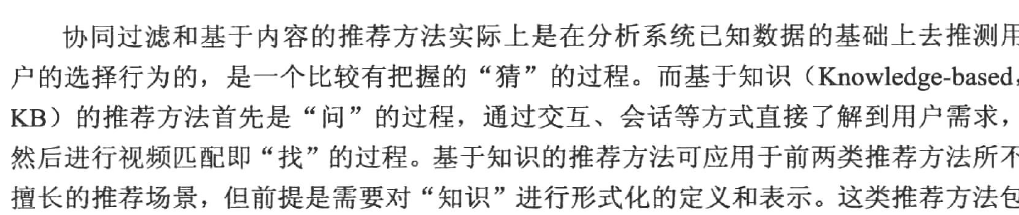
# 推荐系统关联规则算法实战

# 今日目标

通过关联规则部分学习，掌握如何使用关联规则进行推荐算法应用，助力称为一名算法工程师。

# 1.基于知识的推荐方法KB



## 3.1基于约束的推荐算法

该算法重点使用**用户需求的形式化条件(约束条件)**进行搜索。

## 3.2基于实例的推荐算法

该算法侧重于在**已有的实例(关联规则)的基础上进行匹配和调试**。

# 2.关联挖掘推荐算法

参考关联挖掘推荐课件。

基于知识的推荐

* 基于约束的推荐

根据一些约束条件进行匹配推荐

* + 基于实例的推荐

根据已有的关联规则或规则匹配和推荐

* 关联规则算法：找一些重复出现的购买规则

eg:尿布---->啤酒

* + 购物篮数据：用户的一次购买就是一个购物篮
* 关联分析：从大量的数据集中发现存在的关系或模式
  + 商品数据的二元表示-------01表示
  + 项集：对事务中出现项的集合，如果项集的个数为K，指定的是K项集
  + 支持度计数：在项集中出现次数的综合
  + 支持度：同时出现的次数/事务个数

目的：用于频繁项集挖掘

设定一个阈值，小于阈值直接删除

* + 置信度：支持度计数/前置项出现的次数

目的：用于产生规则

设定一个阈值，小于阈值直接删除

* + 关联挖掘：

基于支持度进行频繁项集挖掘

基于置信度进行规则的产生

* + 先验定理：

如果一个超集是频繁项集，那么它的子集一定是频繁项集

如果一个子集是非频繁项集，那么它的超集一定是非频繁项集

* + Apriori算法：

算法思想：根据设定的支持度阈值，计算支持度计数，通过支持度计数求解满足条件的候选项集，从候选项集中删除不满足最小支持度阈值的项构建频繁项集，在设定置信度，求解满足最小置信度的规则

算法步骤：

首先，扫描数据库，进行支持度计数的统计，得到候选1-项集

根据设定的支持度阈值计算支持度计数，删除不满足最小支持度计数的特征项，从而选择频繁-1项集

依次执行上述步骤，产生频繁2项集，频繁3-项集等过程

再次，根据置信度计算规则是否满足最小置信度规则，选择对应的规则作为最终待选规则

算法实现：python可实现

算法优化:

每次构建频繁项集的时候，都需要扫描数据库，扫描数据库次数过多

提出FP-Groupth算法改进Apriori算法

* + FP-Groupth算法

算法思想：在Apriori算法基础上做了一些改进；

算法只需要两次扫描数据库：

第一次扫描数据库：求解项集的支持度计数，筛选非频繁项集，选择频繁项集

第二次扫描数据库：构建FP树

统计支持度计数之后，支持度计数必须按照由大到小的顺序进行排列

算法步骤：

首先，扫描数据库，形成候选1项集，从中选择小于支持度阈值的项，从大到小的顺序产生支持度的排序后的项的序列，对项集中项进行重新排序

其次，第二次扫描数据库，构建FP树，依次选择其中的每条路径，构建FP树

构建以后缀节点为条件的条件FP树，通过条件FP树提取频繁项集

通过置信度计算筛选规则

算法优化：算法需要构建很多条件FP树，计算复杂度较高

* **总结：Apriori算法和FPGroup算法都属于频繁项集产生算法，规则的产生根据置信度产生**的。

# 基于Spark的FP-Groupth算法实战

## 3.0SparkAPI实战

#### 3.0.1MLLIB库FPGrowth算法-频繁项集代码

package cn.apple.mltest  
import org.apache.spark.{SparkConf, SparkContext}  
import org.apache.spark.mllib.fpm.FPGrowth  
import org.apache.spark.rdd.RDD  
/\*\*  
 \* Created by zhao-chj on 2018/7/15.  
 \*/  
object FPGroupth\_PatternMinging {  
 def main(args: Array[String]): Unit = {  
  
 val sparkconf = new SparkConf().setAppName("SparkGrowthDemo").setMaster("local[2]")  
 val sc = new SparkContext(sparkconf)  
 val data = sc.textFile("D:\\BigData\\Workspace\\Spark\_Test\\src\\main\\scala\\cn\\apple\\mltest\\sample\_fpgrowth.txt")  
  
 val transactions: RDD[Array[String]] = data.map(s => s.trim.split(' '))  
  
 val fpg = new FPGrowth()  
 .setMinSupport(0.2)  
 .setNumPartitions(10)  
 val model = fpg.run(transactions)  
  
 model.freqItemsets.collect().foreach { itemset =>  
 println(itemset.items.mkString("[", ",", "]") + ", " + itemset.freq)  
 }  
  
 val minConfidence = 0.8  
 model.generateAssociationRules(minConfidence).collect().foreach { rule =>  
 println(  
 rule.antecedent.mkString("[", ",", "]")  
 + " => " + rule.consequent .mkString("[", ",", "]")  
 + ", " + rule.confidence)  
 }  
 }  
}

#### 3.0.2MLLIB库FPGrowth算法-规则产生代码

import org.apache.spark.{SparkConf, SparkContext}  
import org.apache.spark.mllib.fpm.AssociationRules  
import org.apache.spark.mllib.fpm.FPGrowth.FreqItemset  
/\*\*  
 \* Created by zhao-chj on 2018/7/15.  
 \*/  
object AsscoiateRules {  
 def main(args: Array[String]): Unit = {  
 val sparkconf = new SparkConf().setAppName("SparkGrowthDemo").setMaster("local[2]")  
 val sc = new SparkContext(sparkconf)  
  
 val freqItemsets = sc.parallelize(Seq(  
 new FreqItemset(Array("a"), 15L),  
 new FreqItemset(Array("b"), 35L),  
 new FreqItemset(Array("a", "b"), 12L)  
 ))  
  
 val ar = new AssociationRules()  
 .setMinConfidence(0.8)  
 val results = ar.run(freqItemsets)  
  
 results.collect().foreach { rule =>  
 println("[" + rule.antecedent.mkString(",")  
 + "=>"  
 + rule.consequent.mkString(",") + "]," + rule.confidence)  
 }  
 }  
}

#### 3.0.3ML库FPGrowth算法实战

## 3.1推荐项目代码实战

import org.apache.spark.mllib.fpm.FPGrowth

import org.apache.spark.rdd.RDD

import org.apache.spark.sql.SparkSession

import org.apache.spark.{SparkConf, SparkContext}

import scala.collection.mutable.ListBuffer

case class UserAction1(user: String, item: Int, action: String, timestamp: String, num: Int)

object FPGrowth {

val ITEM\_COUNT: Int = 1 // 大于这个数量

val path = "D:/10.1-10.11/fpg/3"

def main(args: Array[String]): Unit = {

val conf = new SparkConf().setMaster("local[12]").setAppName("FPG").set("spark.executor.memory", "10g")

val sc = SparkContext.getOrCreate(conf)

val spark = SparkSession.builder.getOrCreate()

sc.setLogLevel("ERROR")

//读取行为日志

val actionData = sc.textFile("D:\\BigData\\Workspace\\Spark\_Test\\SparkALSScalamaster\\src\\main\\scala\\als\\ActionData.txt")

// count:1162490

var buyActionsRDD = actionData.map(\_.split("\\001")

match { case Array(user, item, action, timestamp, num) =>

UserAction1(user.toString, item.toInt, action.toString,

timestamp.toString, num.toInt)

}).filter(x => x.action == "buy") // count:60550

val buyItem = buyActionsRDD.map(x => x.item).distinct.count()

println("buyItem",buyItem) // 4

// 购买数量超过阈值的

var userItems = buyActionsRDD.map(x => (x.user, x.item)).groupByKey().map(\_.\_2).filter(x => x.size > ITEM\_COUNT) //count:10804

userItems = userItems.sortBy(x => x.size, ascending = false)

userItems foreach println

// userItems.repartition(1).saveAsTextFile(path + "-orl")

// userItems的格式：Array[Iterable[Int]] = Array(CompactBuffer(1600, 3770, 2758, 6681, 917), ......

// 整理成： Array[String] = Array(1600 3770 2758 6681 917,

val listRDD = userItems.map(x =>

x.toArray.toList.toString().substring(4)

.replace("(", "")

.replace(")", "")

.replace(", ", " "))

// 格式：Array[Array[String]] = Array(Array(1600, 3770, 2758, 6681, 917),

val transactions: RDD[Array[String]] = listRDD.map(

s => s.trim.split(' ').distinct)

println("orl-count", transactions.count())

// setMinSupport 支持度support：项集在总项集里面出现的次数。

// minConfidence 置信度confidence：包含X的项集中，Y出现的概率。u(X,Y)/u(x)。

val fpg = new FPGrowth().setMinSupport(0.6).setNumPartitions(2)

val model = fpg.run(transactions)

val lst1 = new ListBuffer[String]

model.freqItemsets.collect().foreach { itemset =>

val items = itemset.items

if (items.length > 1) {

val str = itemset.items.mkString("[", ",", "]") + "," + itemset.freq

lst1 += str

}

println(s"${itemset.items.mkString("[", ",", "]")},${itemset.freq}")

}

// sc.makeRDD(lst1).repartition(1).saveAsTextFile(path + "-recomm-item-freq")

val lst2 = new ListBuffer[String]

val minConfidence = 0.8

model.generateAssociationRules(minConfidence).collect().foreach { rule =>

val str2 = rule.antecedent.mkString("[", ",", "]") + " " + rule.consequent.mkString("[", ",", "]") + " " + rule.confidence

lst2 += str2

println(s"${rule.antecedent.mkString("[", ",", "]")}=> " +

s"${rule.consequent.mkString("[", ",", "]")},${rule.confidence}")

}

// sc.makeRDD(lst2).repartition(1).saveAsTextFile(path + "-confidence")

}

}

## 3.2结果解释

原始数据集：

Userid-itemid-type(pv\fav\buy)-timestamp-times

usfvm1223ds1pv15391008001

uhf34sdcfe32pv15391008001

dsfcds2332f6pv15391008002

vfcv4356fvf8pv15391008001

usfvm1223ds1pv15391008001

dvbgf909gbn7pv15391008001

dsfcds2332f2pv15391008002

vfcv4356fvf8pv15391008001

bvdfv487fer2pv15391008003

usfvm1223ds3pv15391008001

usfvm1223ds1buy15391008001

uhf34sdcfe35buy15391008001

uhf34sdcfe31buy15391008001

dsfcds2332f3buy15391008002

vfcv4356fvf5buy15391008001

usfvm1223ds5buy15391008001

dvbgf909gbn5buy15391008001

dsfcds2332f1buy15391008002

vfcv4356fvf1buy15391008001

dvbgf909gbn1buy15391008003

usfvm1223ds2buy15391008001

usfvm1223ds3buy15391008001

我们只要行为类型是buy的，那么数据只有这些

用户ID -商品ID-行为类型-时间戳-行为次数（浏览了多少次，或者购买了几件，或者收藏了）

usfvm1223ds1buy15391008001

uhf34sdcfe35buy15391008001

uhf34sdcfe31buy15391008001

dsfcds2332f3buy15391008002

vfcv4356fvf5buy15391008001

usfvm1223ds5buy15391008001

dvbgf909gbn5buy15391008001

dsfcds2332f1buy15391008002

vfcv4356fvf1buy15391008001

dvbgf909gbn1buy15391008003

usfvm1223ds2buy15391008001

usfvm1223ds3buy15391008001

在上述数据集中，出现的商品ID分别是：1、5、3、2

被购买过的商品数量是：

(buyItem,4) 4指的是购买的物品的数量

每个用户的购买商品是（购物篮）

CompactBuffer(5, 1)

CompactBuffer(1, 5, 2, 3)

CompactBuffer(5, 1)

CompactBuffer(5, 1)

CompactBuffer(3, 1)

购物篮数量

(orl-count,5)

// setMinSupport 支持度support：项集在总项集里面出现的次数。

// minConfidence 置信度confidence：包含X的项集中，Y出现的概率。u(X,Y)/u(x)。

达到最小支持度阈值的项集

[1],5 # 商品1出现5次

[5],4 # 商品5出现4次

[5,1],4 # 项集[5,1]出现4次

[5]=> [1],1.0 # 商品5相对于商品1的置信度是1.0

[1]=> [5],0.8 # 商品1相对于商品5的置信度是0.8

# 4.基于知识推荐特点

