# **1.频繁模式挖掘**

挖掘频繁项集，项集的子集，子序列或其他子结构通常是分析大规模数据集的第一步，这是数据挖掘多年来一直活跃的研究课题。我们将用户引用到维基百科的[关联规则学习](http://en.wikipedia.org/wiki/Association_rule_learning)以获取更多信息。

****目录****

* [FP-G](http://spark.apache.org/docs/latest/ml-frequent-pattern-mining.html" \l "fp-growth)rowth
* [PrefixSpan](http://spark.apache.org/docs/latest/ml-frequent-pattern-mining.html" \l "prefixspan)

## **1.1FP-Growth**

FP-growth算法在[Han等人](http://dx.doi.org/10.1145/335191.335372)的文章中描述 [，挖掘频繁模式而没有](http://dx.doi.org/10.1145/335191.335372)**[候选生成](http://dx.doi.org/10.1145/335191.335372)**(区别于Apriori算法)，其中“FP”代表频繁模式。给定训练数据集，FP-growth的第一步是计算item频率并识别频繁项。与为相同目的而设计[的类Apriori](http://en.wikipedia.org/wiki/Apriori_algorithm)算法不同，FP-growth的第二步使用后缀树（FP-tree）结构来编码事务，**而不显式生成候选集**，这通常很难生成。

在第二步之后，可以从FP树中提取频繁项集。在spark.mllib，我们实现了一个名为PFP的FP-growth的并行版本，如[Li等人](http://dx.doi.org/10.1145/1454008.1454027)所述[，PFP：Parallel FP-growth for query recommendation](http://dx.doi.org/10.1145/1454008.1454027)。PFP基于事务的后缀分配增长FP树的任务，因此比单机实现更具可扩展性。我们将用户引向论文以获取更多详细信息。

spark.ml，FP-growth实现采用以下（超）参数：

* + minSupport：对项目集的最小支持被识别为频繁。例如，如果一个项目出现在5个交易中的3个中，则它具有3/5 = 0.6的支持。
  + minConfidence：生成关联规则的最小置信度。置信度表明关联规则被发现的真实程度。例如，如果在交易项目集中**X出现4次，X 并且Y只出现2次**，则规则的置信度为X => Y ，2/4 = 0.5。该参数不会影响频繁项集的挖掘，但会指定从频繁项集生成关联规则的最小置信度。
  + numPartitions：用于分发工作的分区数。默认情况下，不设置参数，并使用输入数据集的分区数。

FPGrowthModel规定：

* freqItemsets：频繁的项目集格式为DataFrame（“items”[Array]，“freq”[Long]）
* associationRules：上面以置信度生成的关联规则minConfidence，格式为DataFrame（“antecedent”[Array]，“结果”[Array]，“confidence”[Double]）。
* transform：对于每个事务itemsCol，**该transform方法将其项目与每个关联规则的前提进行比较。**如果记录包含特定关联规则的所有前提，则该规则将被视为适用，并且其结果将被添加到预测结果中。变换方法将所有适用规则的结果总结为预测。预测列具有相同的数据类型，itemsCol并且不包含现有项目itemsCol。

## 1.2ML

import org.apache.spark.sql.SparkSession

import org.apache.spark.ml.fpm.FPGrowth

/\*\*

\* Created by zhao-chj on 2018/11/13.

\*/

object MLFPGrowth {

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

val spark: SparkSession = SparkSession.builder()

.appName("SparkMlilb")

.master("local[2]")

.getOrCreate()

spark.sparkContext.setLogLevel("WARN")

import spark.implicits.\_

val dataset = spark.createDataset(Seq(

"1 2 5",

"1 2 3 5",

"1 2")

).map(t => t.split(" ")).toDF("items")

val fpgrowth = new FPGrowth().setItemsCol("items").setMinSupport(0.5).setMinConfidence(0.6)

val model = fpgrowth.fit(dataset)

// Display frequent itemsets.

model.freqItemsets.show()

// antecedent ===> consequent

// Display generated association rules.

model.associationRules.show()

// transform examines the input items against all the association rules and summarize the

// consequents as prediction

model.transform(dataset).show()

}

}

## 1.3Mllib

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

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

import org.apache.spark.rdd.RDD

import org.apache.spark.mllib.fpm.AssociationRules

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

/\*\*

\* Created by zhao-chj on 2018/11/13.

\*/

object MllibFPGrowth {

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

// 环境基础配置

val sparkconf = new SparkConf().setAppName("SparkGrowthDemo").setMaster("local[2]")

val sc = new SparkContext(sparkconf)

//导入数据

val data = sc.textFile("D:\\BigData\\Workspace\\Saprk\_Test\_ML\\src\\main\\scala\\cn\\apple\\Bigdata11\\sample\_fpgrowth.txt")

//切分数据

val transactions: RDD[Array[String]] = data.map(s => s.trim.split(' '))

//使用关联挖掘产生算法

val fpg = new FPGrowth()

.setMinSupport(0.2)

.setNumPartitions(10)

val model = fpg.run(transactions)

//输出预测结果

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

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

}

//定义最小置信度产生规则

val minConfidence = 0.8

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

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

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

}

// =====================================================================

//自定义频繁项集产生规则

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

s" ${rule.confidence}")

}

}

}

# 基于Spark的FP-Growth算法实战

大数据组的同学采集到用户的动态数据转化为格式化的数据，提交给算法工程师进一步建立推荐模型。

## 2.1.字段说明

[用户id\001商品id\001用户行为类型\001时间戳\001该行为在这一天的次数]

公司的现有数据是只记录某用户一天内的行为次数，没有给每次行为记录一条数据。比如用户一天内浏览了商品3次，只会有一条pv数据，且次数为3。

## 2.2.输入数据

存在UserBehaviorData.txt文件中。



首先将上述数据转化为购物篮数据(行-事务、列-项集)：如

CompactBuffer(5, 1)

CompactBuffer(1, 5, 2, 3)

CompactBuffer(5, 1)

CompactBuffer(5, 1)

CompactBuffer(3, 1)

通过购物篮数据，使用FPGrowth算法得到支持度和置信度，在进行频繁项挖掘和规则产生。

## 2.3.实战代码

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.实战结果

原始数据集：

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)**

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-事务个数

**(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(频繁项)

**[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**

关联规则

[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