目录

[目录 1](#_Toc410675400)

[MLlib卡方检验 2](#_Toc410675401)

[一、卡方检验理论 2](#_Toc410675402)

[1、简介 2](#_Toc410675403)

[2、卡方检验基本思想 2](#_Toc410675404)

[3、卡方检验作用 2](#_Toc410675405)

[二、MLlib对卡方检验的实现 3](#_Toc410675406)

[1、输入数据类型分析 3](#_Toc410675407)

[2、MLLib卡方检验API 3](#_Toc410675408)

[3、MLlib为什么要提供RDD[LabeledPoint]类型的接口 4](#_Toc410675409)

[三、RDD方式卡方检验分析 4](#_Toc410675410)

[1、概述 4](#_Toc410675411)

[2、数据准备过程 4](#_Toc410675412)

[3、计算过程 5](#_Toc410675413)

[附录：附加注释代码 5](#_Toc410675414)

MLlib卡方检验

## 一、卡方检验理论

##### 1、简介

总体的分布函数完全未知或只知形式、但不知其参数的情况，为了推断总体的某些未知特性，提出某些关于总体的假设。我们要根据样本对所提出的假设作出是接受，还是拒绝的决策。假设检验是作出这一决策的过程。卡方检验即是假设检验的一种。

##### 2、卡方检验基本思想

首先假设*H*0成立，基于此前提计算出χ2值，它表示观察值与理论值之间的偏离程度。根据χ2分布及自由度可以确定在*H*0假设成立的情况下获得当前统计量及更极端情况的[概率](http://wiki.mbalib.com/wiki/%E6%A6%82%E7%8E%87)P。如果P值很小，说明观察值与理论值偏离程度太大，应当拒绝无效假设，表示比较资料之间有[显著差异](http://wiki.mbalib.com/wiki/%E6%98%BE%E8%91%97%E5%B7%AE%E5%BC%82)；否则就不能拒绝无效假设，尚不能认为[样本](http://wiki.mbalib.com/wiki/%E6%A0%B7%E6%9C%AC)所代表的实际情况和理论假设有差别

##### 3、卡方检验作用

卡方检验主要有以下两种作用：

1） 皮尔森独立性检验（Pearson's independence test）

验证从两个变量抽出的配对观察值组是否互相独立。例如：例如：每次都从A国和B国各抽一个人，看他们的反应是否与国籍无关。

2）适度检验（Goodness of Fit test）

实际执行多项式试验而得到的观察次数，与虚无假设的期望次数相比较，称为卡方适度检验，即在于检验二者接近的程度，利用样本数据以检验总体分布是否为某一特定分布的统计方法。

## 二、MLlib对卡方检验的实现

##### 1、输入数据类型分析

MLlib自1.1版本起实现了卡方检验（Pearson），包括独立性检验、适度检验。MLlib对二种检验的实现接口依据二种检验的特性互不相同：适度检验是检验一系列特征值是否符合某种指标，故选的向量（Vector）作为参数；而独立性检验是多个率之间的相互关系的分析，故适合用矩阵（Matrix）作为输入。

同时MLlib还对独立性检验提供了以RDD[LabeledPoint]类形参数的接口。LabeledPoint格式的数据多用于机器学习方面，如回归分析等。其数据格式为label:features，MLLib实现中label为Double, features为Vector类型，数据样例如下（以下是两条来自MLlib库官方样例数据）：

-9.490009878824548 1:0.4551273600657362 2:0.36644694351969087 3:-0.38256108933468047 4:-0.4458430198517267 5:0.33109790358914726 6:0.8067445293443565 7:-0.2624341731773887 8:-0.44850386111659524 9:-0.07269284838169332 10:0.5658035575800715

0.2577820163584905 1:0.8386555657374337 2:-0.1270180511534269 3:0.499812362510895 4:-0.22686625128130267 5:-0.6452430441812433 6:0.18869982177936828 7:-0.5804648622673358 8:0.651931743775642 9:-0.6555641246242951 10:0.17485476357259122

##### 2、MLLib卡方检验API

根据上述对卡方输入数据的分析，MLlib库关于卡方对外接供的接口有：

1）def chiSqTest(data: RDD[LabeledPoint]): Array[ChiSqTestResult]

2）def chiSqTest(observed: Matrix): ChiSqTestResult

3）def chiSqTest(observed: Vector): ChiSqTestResult

其中1),2)用于独立性检验，3）用于适度检验。

接口 1）主要的程序逻辑是将RDD类型的数据转换为Matrix(矩阵)类型的数据，然后掉用接口 2）。

接口 2）主要程序逻辑是根据卡方定义进行数学计算，由于卡方检验数据量要求较小，此方法为串行执行。

接口 3）同接口二类似。

##### 3、MLlib为什么要提供RDD[LabeledPoint]类型的接口

如本小节 1 中所述,机器学习常用label : features类型的数据。以房屋价格情况为例：

Label features

2015/2/2日 面积：30年 房龄：20年 价格：20000元/平 地区……

因假设检验是由部分对总体的估计，矩阵或向量方式参数的接口的需要数据应是对原始数据计数之后的数据，数据量一般不会很大，因此程序只是串行方式执行数学计算。但这两种类型的方法，面对其它统计分析方法使用的数据确束手无策，虽假设检验是部分对总体的估计，但作为样本的原始数据量也不会太小，所以对采集的原始数据样本中每个特征出现频次统计的计算量也将很大，适合并行处理。综上所述, 易知其提供这种方法的原因。

## 三、RDD方式卡方检验分析

##### 1、概述

MLlib提供的RDD 方式的卡方检验的实现大体可以分为两个过程：

1）数据准备过程，即：将RDD方式的数据转换成可以直接用来进行数学计算的矩阵数据。

2）计算过程，即：根据卡方数学公式，利用数学库进行数学计算。

##### 2、数据准备过程

RDD方式数据转换为矩阵过种存在着数据类型变换，因此其过程中也伴随着RDD的转换操作。以下结合源码对数据变换做出分析如下：

首先结合第二部分第3小节中举例，对程序中数据格式定义作如下说明，

Col(n)表示某种特征值所在的列，如col(1)表示面积所在的列。

Label代表一条数据的某种无关属性， 其形成上述的日期，或其它调查对象。

Feature 代表某种特征，如 面积，房价等。

程序一开始使用mapPartitions对RDD[LabeledPoint]的每个分区的数据转分成内部原素为三元组（col, feature, label）的MapPartitionsRDD.

然后调用MapPartitionsRDD的countByValue()方法。 RDD的countByValue()方法实际是先执行一个Map转换将数据转换为（value, null）的型式，然后再执行countByKey()操作，然而countByKey() 其实质又是执行mapValues(),

reduceByKey(),collect() 这一系列操作。

其中

mapValues()操作将生成MappedValuesRDD;

reduceByKey() 又调用的combineByKey，是一个shuffle过程 生成ShuffledRDD

collect()是RDD的action算子，提交作业，收集数据。最后通过toMap变成Map数据，之后的一切操作皆是针对变换后的map<三元组, 个数>。其具体转换过程详见附录代码。

RDD转换过程用图形表示如下：

ShuffleRDD

原始RDD MapPartitionsRDD MapedRDD MapedValueRDD

LabeledPoint

<C,F,L>:1

<C,F,L>:Int

<C,F,L>:Int

<C,F,L >

三元组

<C,F,L>:1

<C,F,L>:Int

<C,F,L>

LabeledPoint

mapPartitions map mapValues combineBykey

##### 3、计算过程

利用数学函数库依据卡方公式进行计算。

## 附录：附加注释代码

/\*\*

\* 皮尔森独立性检验：验证从两个变量抽出的配对观察值组是否互相独立

\* （例如：每次都从A国和B国各抽一个人，看他们的反应是否与国籍无关）

\* **@param** data 待检测的数据 RDD[LabeledPoint] 类型

\* **@param** methodName 使用的检验方法

\* **@result** Array[ChiSqTestResult] 返回值

\*/

**def** chiSquaredFeatures(data: RDD[LabeledPoint],

methodName: String = PEARSON.name): Array[ChiSqTestResult] = {

**val** maxCategories = 10000 //最大分类

**val** numCols = data.first().features.size //特征值中有多少种数据

**val** results = **new** Array[ChiSqTestResult](numCols) //存储卡方检验结果：numCols个元素的卡方检验结果数组

**var** labels: Map[Double, Int] = **null**

// at most 1000 columns at a time

**val** batchSize = 1000 //每批数据量

**var** batch = 0 //批次

**while** (batch \* batchSize < numCols) { //处理

// The following block of code can be cleaned up and made public as

// chiSquared(data: RDD[(V1, V2)])

**val** startCol = batch \* batchSize //开始执行位置

**val** endCol = startCol + math.min(batchSize, numCols - startCol) //相对startCol的偏移量

**val** pairCounts = data.mapPartitions { iter =>

**val** distinctLabels = mutable.HashSet.empty[Double] //不同的标签(分类)

/\*创建一个Map对象，并初始化：key为列的编号，value用空的HashSet\*/

**val** allDistinctFeatures: Map[Int, mutable.HashSet[Double]] =

Map((startCol until endCol).map(col => (col, mutable.HashSet.empty[Double])): \_\*)

**var** i = 1 //计数器，避免频繁进行判断

/\*对此分片进行flatMap操作， 将LabeledPoint(label, features) 转换成三元组\*/

iter.flatMap { **case** LabeledPoint(label, features) =>

**if** (i % 1000 == 0) {

/\*若分类过多，导致理论频数(即期望计数)小于5的太多，会导至皮尔森方法失效\*/

**if** (distinctLabels.size > maxCategories) {

**throw** **new** SparkException(s"Chi-square test expect factors (categorical values) but "

+ s"found more than $maxCategories distinct label values.")

}

allDistinctFeatures.foreach { **case** (col, distinctFeatures) =>

**if** (distinctFeatures.size > maxCategories) {

**throw** **new** SparkException(s"Chi-square test expect factors (categorical values) but "

+ s"found more than $maxCategories distinct values in column $col.")

}

}

}

i += 1

distinctLabels += label

/\*将features，加上索引，然后切片，再转将其通过map 操作 赋值到allDistinctFeatures\*/

features.toArray.view.zipWithIndex.slice(startCol, endCol).map { **case** (feature, col) =>

allDistinctFeatures(col) += feature

(col, feature, label)

}

}

}.countByValue()

**if** (labels == **null**) {

// Do this only once for the first column since labels are invariant across features.

/\*

\*取出col 是startCol（实质每行一个分类，此filter即相当于从原数据每一行中拿出第一个）的数据，

\*将label取出加入labels

\* 方法仅执行一次。

\*/

labels =

pairCounts.keys.filter(\_.\_1 == startCol).map(\_.\_3).toArray.distinct.zipWithIndex.toMap

}

/\*标签，标识矩阵的列， （因为矩阵是列优先存储的）\*/

**val** numLabels = labels.size

pairCounts.keys.groupBy(\_.\_1).map { **case** (col, keys) =>

**val** features = keys.map(\_.\_2).toArray.distinct.zipWithIndex.toMap

**val** numRows = features.size //特征值个数， 矩阵的行

**val** contingency = **new** BDM(numRows, numLabels, **new** Array[Double](numRows \* numLabels)) //创建（空）矩阵

/\*赋值操作\*/

keys.foreach { **case** (\_, feature, label) =>

**val** i = features(feature)

**val** j = labels(label)

contingency(i, j) += pairCounts((col, feature, label))

}

/\*转换为Spark mllib库中的矩阵，并对矩阵作卡方检验\*/

results(col) = chiSquaredMatrix(Matrices.fromBreeze(contingency), methodName)

}

batch += 1

}

results

}

/\*\*

\* 独立性检验

\* **@param** counts 要检测的数据（矩阵）

\* **@param** methodName 使用的检测方法（默认：皮尔森）

\*/

**def** chiSquaredMatrix(counts: Matrix, methodName:String = PEARSON.name): ChiSqTestResult = {

**val** method = methodFromString(methodName) //获取Method (case类)

**val** numRows = counts.numRows //矩阵行数

**val** numCols = counts.numCols //矩阵列数

// get row and column sums

**val** colSums = **new** Array[Double](numCols) //存放每列数据和（用于计算概率）

**val** rowSums = **new** Array[Double](numRows) //存放每行数据和

**val** colMajorArr = counts.toArray //存在所有数据（用于计算 样本总数）

**var** i = 0

**while** (i < colMajorArr.size) {

**val** elem = colMajorArr(i)

**if** (elem < 0.0) {

**throw** **new** IllegalArgumentException("Contingency table cannot contain negative entries.")

}

colSums(i / numRows) += elem //赋值

rowSums(i % numRows) += elem //赋值

i += 1

}

**val** total = colSums.sum //计算样本总量

// second pass to collect statistic

**var** statistic = 0.0

**var** j = 0

**while** (j < colMajorArr.size) {

**val** col = j / numRows

**val** colSum = colSums(col)

**if** (colSum == 0.0) {

**throw** **new** IllegalArgumentException("Chi-squared statistic undefined for input matrix due to"

+ s"0 sum in column [$col].")

}

**val** row = j % numRows

**val** rowSum = rowSums(row)

**if** (rowSum == 0.0) {

**throw** **new** IllegalArgumentException("Chi-squared statistic undefined for input matrix due to"

+ s"0 sum in row [$row].")

}

**val** expected = colSum \* rowSum / total //计算期望值

statistic += method.chiSqFunc(colMajorArr(j), expected) //计算卡方值、并叠加至statistic中

j += 1

}

**val** df = (numCols - 1) \* (numRows - 1) //计算自由度

**if** (df == 0) {

// 1 column or 1 row. Constant distribution is independent of anything.

// pValue = 1.0 and statistic = 0.0 in this case.

**new** ChiSqTestResult(1.0, 0, 0.0, methodName, NullHypothesis.independence.toString)

} **else** {

/\*

\* cumulativeProbability() 依据下面理论进行实现

\* <a href="http://mathworld.wolfram.com/Chi-SquaredDistribution.html">

\* 自由度为n的卡方分布是 a=n/2 和 λ = 1/2 的伽马分布

\* 此方法计算的概率是 P{X <= x} ;

\*

\* 当x是某个常数，选择x是为了得到想要的检验显著性水平a, 也就是说x的选择满足Ho为真时， 应为：P(X > x) = a;

\* 与上述方法计算值互补

\* 又pValue = P{X > x} = a;

\* pValue 为拒绝原假设的最小显著性水平

\*/

**val** pValue = 1.0 - **new** ChiSquaredDistribution(df).cumulativeProbability(statistic)

//生成结果（根据pValue值进行比较，得出独立性强弱）

**new** ChiSqTestResult(pValue, df, statistic, methodName, NullHypothesis.independence.toString)

}

}