**异常点检测算法isolation forest的分布式实现**

[](https://www.jianshu.com/u/cfa3f3cd35c4)

[双er](https://www.jianshu.com/u/cfa3f3cd35c4) 关注

2018.02.03 22:58 字数 1775 阅读 263评论 9喜欢 6赞赏 1

无监督领域有一个准度和效率双佳的异常点检测算法，我在实践中使用过几次，效果奇好，就是最近几年非常流行的isolation forest（孤立森林）。该算法在sklearn中有现成的包，但是如果大数据的集群上跑的话，目前没有封装好的接口，给分布式任务的部署带来了很多不便（话说spark mllib中集成的算法真心太少了），本文用scala从头进行该算法在spark上的分布式实现，并演示任务在集群上的执行全过程。

**一、算法简介**

先说一下算法的最少必要知识，细节部分会揉在代码里进行讲解。

**1、训练过程：构建森林的树木**

iForest由iTree组成。构建每一颗iTree时，从训练数据中抽取N个样本，然后在这些样本中，随机选择一个特征，再随机选择该特征下的一个值，对样本进行二叉划分，然后分别在左右两边的数据集上重复上面的过程，直接达到终止条件，一颗树构建完成。

**2、预测过程：计算样本的异常得分**

把测试数据在每棵树上沿对应的条件分支往下走，直到达到叶子节点，并记录这过程中经过的路径长度path length(用h(x)表示)。并由此得出异常分数，当分数超过某一阈值，即可判定为异常样本。

**二、scala实现**

代码主体非原创，参考自国外的一位大神：[https://github.com/hsperr/first\_steps\_in\_scala](https://link.jianshu.com/?t=https%3A%2F%2Fgithub.com%2Fhsperr%2Ffirst_steps_in_scala)，有部分修改

**1、首先，import编写spark程序所需的包，以及scala的Random模块，用于随机选取功能。**

import org.apache.spark.SparkContext.\_

import org.apache.spark.rdd.RDD

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

import scala.util.Random

**2、定义单颗树iTree，第二、三行意味着，每棵树的左右分支ITreeBranch和叶子节点ITreeLeaf都属于iTree的子类。**

sealed trait ITree

case class ITreeBranch(left: ITree, right: ITree, split\_column: Int, split\_value: Double) extends ITree

case class ITreeLeaf(size: Long) extends ITree

**3、定义孤立森林的类，完成算法的训练部分，即全部树的构建。**

3.1、从样本中抽样，用于构建单个iTree

object IsolationForest {

def getRandomSubsample(data: RDD[Array[Double]], sampleRatio: Double, seed: Long = Random.nextLong): RDD[Array[Double]] = {

data.sample(false, sampleRatio, seed=seed)

}

3.2 、递归构建生成单颗iTree。  
参数：  
data：上步抽出的样本数据；  
maxHeight：树的最大高度即树终止生长的条件；  
numColumns：data的特征数量；  
currentHeight：树的当前高度。  
返回：  
一颗完整的ITree

def growTree(data: RDD[Array[Double]], maxHeight:Int, numColumns:Int, currentHeight:Int = 0): ITree = {

val numSamples = data.count()

//递归终止条件，当前树高大于maxHeight或数据量不大于1

if(currentHeight>=maxHeight || numSamples <= 1){

return new ITreeLeaf(numSamples)

}

//随机选择特征列

val split\_column = Random.nextInt(numColumns)

val column = data.map(s => s(split\_column))

//随机选择该特征列中的值split\_value，用于分割样本

val col\_min = column.min()

val col\_max = column.max()

val split\_value = col\_min + Random.nextDouble()\*(col\_max-col\_min)

//小于分割值的成为左子树，反之右子树

val X\_left = data.filter(s => s(split\_column) < split\_value).cache()

val X\_right = data.filter(s => s(split\_column) >= split\_value).cache()

//递归

new ITreeBranch(growTree(X\_left, maxHeight, numColumns, currentHeight + 1),

growTree(X\_right, maxHeight, numColumns, currentHeight + 1),

split\_column,

split\_value)

}

}

3.3、将多棵iTree组建成完整森林iforest  
参数：  
data：全部训练数据；  
numTrees：森林中树的个数；  
subSampleSize：每棵树采样的大小；  
seed：随机种子。  
返回：  
孤立森林

def buildForest(data: RDD[Array[Double]], numTrees: Int = 2, subSampleSize: Int = 256, seed: Long = Random.nextLong) : IsolationForest = {

val numSamples = data.count()

val numColumns = data.take(1)(0).size

val maxHeight = math.ceil(math.log(subSampleSize)).toInt

val trees = Array.fill[ITree](numTrees)(ITreeLeaf(1))

val trainedTrees = trees.map(s=>growTree(getRandomSubsample(data, subSampleSize/numSamples.toDouble, seed), maxHeight, numColumns))

IsolationForest(numSamples, trainedTrees)

}

**4、定义预测功能类**

4.1 预测功能类定义为IsolationForest的样例类，  
参数  
num\_samples：单课iTree的样本数目  
trees：已经构建好的孤立森林iforest

主函数predict，  
参数x：要预测的单条样本数组，  
返回：异常得分Anomaly Score  
步骤：  
在每一棵iTree上，计算样本达到叶子节点走过的路径长度，然后将得到的不同路径长度按照如下公式进行计算，得到异常得分，走过的路径越短，得分越高，代表越异常。

https://upload-images.jianshu.io/upload_images/9885669-2c775943b14495a4.png?imageMogr2/auto-orient/strip%7CimageView2/2/w/155

image.png

公式中，h(x)代表路径长度，E(h(x))代表在不同的iTree上路径长度的均值，即群体决策，分母是用来归一化的。

case class IsolationForest(num\_samples: Long, trees: Array[ITree]) {

def predict(x:Array[Double]): Double = {

val predictions = trees.map(s => pathLength(x, s, 0)).toList

println(predictions.mkString(","))

math.pow(2, -(predictions.sum/predictions.size)/cost(num\_samples)) //Anomaly Score

}

上面代码用到的cost 方法和pathLength方法定义如下，  
cost方法参数为二叉树中的样本个数，范围该二叉树的平均路径长度，公式为：

https://upload-images.jianshu.io/upload_images/9885669-84b569597ba07ac1.png?imageMogr2/auto-orient/strip%7CimageView2/2/w/314

image.png

def cost(num\_items:Long): Int =

//二叉搜索树的平均路径长度。0.5772156649:欧拉常数

(2\*(math.log(num\_items-1) + 0.5772156649)-(2\*(num\_items-1)/num\_items)).toInt

pathLength方法是一个递归计算，因为每走一步，接下来面对的仍然是一颗树，分支树或者叶子节点。  
参数：样本x，单颗树tree，当前的路径长度path\_length，初始值应传入0。  
返回：最终的路径长度

@scala.annotation.tailrec

final def pathLength(x:Array[Double], tree:ITree, path\_length:Int): Double ={

tree match{ //match方法，让tree进行如下两种模式匹配

//如果ITree匹配到的类型是叶子节点，那么，查看该节点的样本数size，如果size大于1，则加上该size对应的二叉搜索树的平均路径长度，如果size等于1，则直接加1

case ITreeLeaf(size) =>

if (size > 1)

path\_length + cost(size)

else

path\_length + 1

//如果ITree匹配到的类型是一颗分支子树，该子树还会有left分支，right分支，以及分类的依据特征列split\_column，和该特征列的分割值split\_value

case ITreeBranch(left, right, split\_column, split\_value) =>

val sample\_value = x(split\_column) //传入的样本x在该特征上的取值

if (sample\_value < split\_value) //如果小于分割值则在左子树上进行递归计算，如果大于分割值则在右子树上进行递归计算

pathLength(x, left, path\_length + 1)

else

pathLength(x, right, path\_length + 1)

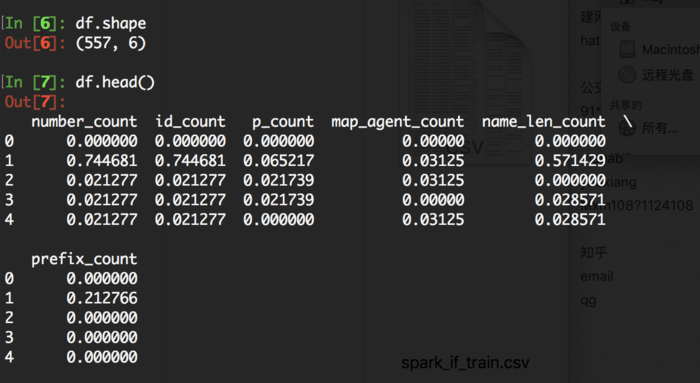
}

}

}

**5、读取数据进行预测**

本节定义最终要调用运行的main方法，我把样例数据放在了本地，也可以放到hdfs上，csv格式，已经做好了标准化，概览如下



训练数据概览.png

5.1、一些对spark的基本设置

object Runner{

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

Random.setSeed(1337)

val conf = new SparkConf()

.setAppName("IsolationTree")

.setMaster("local")

val sc = new SparkContext(conf)

//禁止对输出文件进行压缩

sc.hadoopConfiguration.set("mapred.output.compress", "false")

5.2、读入csv数据并预处理，lines为RDD格式，这是spark处理数据的基本单元

val lines = sc.textFile("file:///tmp/spark\_data/spark\_if\_train.csv") //本地路径

val data = //对每一行数据以逗号为分隔符进行拆分，从第二个数据开始取，因为第一个数字是索引

lines

.map(line => line.split(","))

.map(s => s.slice(1,s.length))

val header = data.first() // 取第一行的数据作为列名

// 去掉列名行并将数据转化为double类型

val rows = data.filter(line => line(0) != header(0)).map(s => s.map(\_.toDouble))

println("Loaded CSV File...")

println(header.mkString("\n")) // 看一下列名

println(rows.take(5).deep.mkString("\n")) // 看一下前5行数据

5.3、进行iforest的构建和对样本的预测

// 构建森林，训练数据rows，森林里树的棵树，这里写10，数据量大的话一般是100

val forest = IsolationForest.buildForest(rows, numTrees=10)

// 对每一行数据进行预测

val result\_rdd = rows.map(row => row ++ Array(forest.predict(row)))

// 将结果存入本地文件

result\_rdd.map(lines => lines.mkString(",")).repartition(1).saveAsTextFile("file:///tmp/predict\_label")

// 看一下前10条数据的预测结果

val local\_rows = rows.take(10)

for(row <- local\_rows){

println("ForestScore", forest.predict(row))

}

println("Finished Isolation")

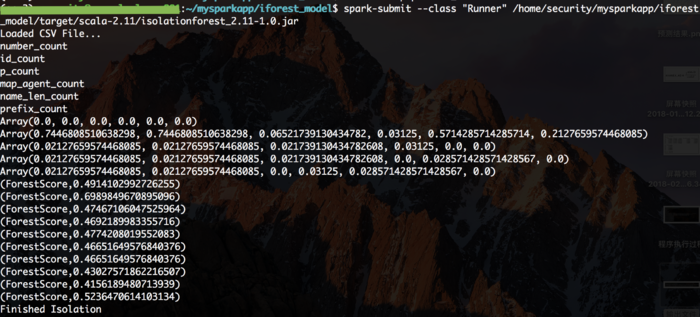
}

}

以上，isolation forest训练部分和预测部分都做好了。

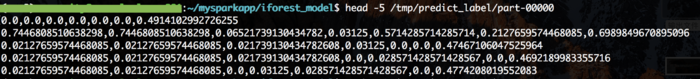
**三、部署到spark上并运行**

开始运行  
我们打印出了数据的列名、前5条数据、以及前10条数据的异常得分如图所示：



程序执行.png

任务执行完毕，看一下输出文件，图示捞出了前五行，最后一个字段即为预测得分，接下来就可以设定一个阈值，原作论文推荐为0.6，大于阈值的即判定为异常啦。



输出文件.png

图中的第二行数据，得分0.69，其他数据得分均为0.5以下，观察一下它前面的字段，比其他数据都要大出很多，确实为一个异常点～

**四、小结**

isolation forest由多棵树构成，而树的生长过程并不受其他树影响，所以是一个非常完美的适合分布式并行的算法。样例数据和代码都放到了[https://github.com/scarlettgin/isolation\_spark](https://link.jianshu.com/?t=https%3A%2F%2Fgithub.com%2Fscarlettgin%2Fisolation_spark" \t "_blank)