



下载APP



08 | Embedding实战：如何使用Spark生成Item2vec和Graph Embedding?

2020-10-19 王喆

深度学习推荐系统实战

[进入课程 >](#)



讲述：王喆

时长 11:35 大小 10.61M



你好，我是王喆。

前面两节课，我们一起学习了从 Item2vec 到 Graph Embedding 的几种经典 Embedding 方法。在打好了理论基础之后，这节课就让我们从理论走向实践，看看到底**如何基于 Spark 训练得到物品的 Embedding 向量。**

通过特征工程部分的实践，我想你已经对 Spark 这个分布式计算平台有了初步的认识。^甘 实除了一些基本的特征处理方法，在 Spark 的机器学习包 Spark MLlib 中，还包含了[☆]成熟的机器学习模型，这其中就包括我们讲过的 Word2vec 模型。基于此，这节课我们会在 Spark 平台上，完成 **Item2vec 和基于 Deep Walk 的 Graph Embedding** 的训练。

对其他机器学习平台有所了解的同学可能会问，TensorFlow、PyTorch 都有很强大的深度学习工具包，我们能不能利用这些平台进行 Embedding 训练呢？当然是可以的，我们也会在之后的课程中介绍 TensorFlow 并用它实现很多深度学习推荐模型。但是 Spark 作为一个原生的分布式计算平台，在处理大数据方面还是比 TensorFlow 等深度学习平台更具有优势，而且业界的很多公司仍然在使用 Spark 训练一些结构比较简单的机器学习模型，再加上我们已经用 Spark 进行了特征工程的处理，所以，这节课我们继续使用 Spark 来完成 Embedding 的实践。

首先，我们来看看怎么完成 Item2vec 的训练。

Item2vec：序列数据的处理

我们知道，Item2vec 是基于自然语言处理模型 Word2vec 提出的，所以 Item2vec 要处理的是类似文本句子，观影序列之类的序列数据。那在真正开始 Item2vec 的训练之前，我们还要先为它准备好训练用的序列数据。在 movieLens 数据集中，有一张叫 rating（评分）的数据表，里面包含了用户对看过电影的评分和评分的时间。既然时间和评分历史都有了，我们要用的观影序列自然就可以通过处理 rating 表得到啦。

userId	movieId	rating	timestamp
1	2	3.5	1112486027
1	29	3.5	1112484676
1	32	3.5	1112484819
1	47	3.5	1112484727
1	50	3.5	1112484580
1	112	3.5	1094785740
1	151	4.0	1094785734
1	223	4.0	1112485573
1	253	4.0	1112484940
1	260	4.0	1112484826
1	293	4.0	1112484703
1	296	4.0	1112484767
1	318	4.0	1112484798
1	337	3.5	1094785709
1	367	3.5	1112485980

图1 movieLens数据集中的rating评分表

不过，在使用观影序列编码之前，我们还要再明确两个问题。一是 movieLens 这个 rating 表本质上只是一个评分的表，不是真正的“观影序列”。但对用户来说，当然只有看过这

部电影才能够评价它，所以我们几乎可以把评分序列当作是观影序列。二是我们是应该把所有电影都放到序列中，还是只放那些打分比较高的呢？


这里，我是建议对评分做一个过滤，只放用户打分比较高的电影。为什么这么做呢？我们要思考一下 Item2vec 这个模型本质上是要学习什么。我们是希望 item2vec 能够学习到物品之间的近似性。既然这样，我们当然是希望评分好的电影靠近一些，评分差的电影和评分好的电影不要在序列中结对出现。

好，那到这里我们明确了样本处理的思路，就是对一个用户来说，我们先过滤掉他评分低的电影，再把他评论过的电影按照时间戳排序。这样，我们就得到了一个用户的观影序列，所有用户的观影序列就组成了 Item2vec 的训练样本集。

那这个过程究竟该怎么在 Spark 上实现呢？其实很简单，我们只需要明白这 5 个关键步骤就可以实现了：

1. 读取 ratings 原始数据到 Spark 平台。
2. 用 where 语句过滤评分低的评分记录。
3. 用 groupBy userId 操作聚合每个用户的评分记录，DataFrame 中每条记录是一个用户的评分序列。
4. 定义一个自定义操作 sortUdf，用它实现每个用户的评分记录按照时间戳进行排序。
5. 把每个用户的评分记录处理成一个字符串的形式，供后续训练过程使用。

具体的实现过程，我还是建议你来参考我下面给出的代码，重要的地方我也都加上了注释，方便你来理解。

 复制代码


```
1 def processItemSequence(sparkSession: SparkSession): RDD[Seq[String]] = {  
2   //设定rating数据的路径并用spark载入数据  
3   val ratingsResourcesPath = this.getClass.getResource("/webroot/sampledata/ra  
4   val ratingSamples = sparkSession.read.format("csv").option("header", "true")  
5  
6  
7   //实现一个用户定义的操作函数(UDF)，用于之后的排序  
8   val sortUdf: UserDefinedFunction = udf((rows: Seq[Row]) => {  
9     rows.map { case Row(movieId: String, timestamp: String) => (movieId, times  
10      .sortBy { case (movieId, timestamp) => timestamp }  
11      .map { case (movieId, timestamp) => movieId }
```

```

12    })
13
14
15    //把原始的rating数据处理成序列数据
16    val userSeq = ratingSamples
17        .where(col("rating") >= 3.5) //过滤掉评分在3.5一下的评分记录
18        .groupBy("userId")           //按照用户id分组
19        .agg(sortUdf(collect_list(struct("movieId", "timestamp")))) as "movieIds")
20        .withColumn("movieIdStr", array_join(col("movieIds"), " "))
21        //把所有id连接成一个String, 方便后续word2vec模型处理
22
23
24    //把序列数据筛选出来, 丢掉其他过程数据
25    userSeq.select("movieIdStr").rdd.map(r => r.getAs[String]("movieIdStr").spli

```

通过这段代码生成用户的评分序列样本中，每条样本的形式非常简单，它就是电影 ID 组成的序列，比如下面就是 ID 为 11888 用户的观影序列：

 复制代码


```

1  296 380 344 588 593 231 595 318 480 110 253 288 47 364 377 589 410 597 539 39
2

```

Item2vec：模型训练

训练数据准备好了，就该进入我们这堂课的重头戏，模型训练了。手写 Item2vec 的整个训练过程肯定是一件让人比较“崩溃”的事情，好在 Spark MLlib 已经为我们准备好了方便调用的 Word2vec 模型接口。我先把训练的代码贴在下面，然后再带你一步步分析每一行代码是在做什么。

 复制代码

```

1  def trainItem2vec(samples : RDD[Seq[String]]): Unit = {
2      //设置模型参数
3      val word2vec = new Word2Vec()
4      .setVectorSize(10)
5      .setWindowSize(5)
6      .setNumIterations(10)
7
8
9      //训练模型
10     val model = word2vec.fit(samples)
11
12
13     //训练结束, 用模型查找与item"592"最相似的20个item

```



```
14 val synonyms = model.findSynonyms("592", 20)
15 for((synonym, cosineSimilarity) <- synonyms) {
16     println(s"$synonym $cosineSimilarity")
17 }
18
19 //保存模型
20 val embFolderPath = this.getClass.getResource("/webroot/sampledData/")
21 val file = new File(embFolderPath.getPath + "embedding.txt")
22 val bw = new BufferedWriter(new FileWriter(file))
23 var id = 0
24 //用model.getVectors获取所有Embedding向量
25 for (movieId <- model.getVectors.keys){
26     id+=1
27     bw.write( movieId + ":" + model.getVectors(movieId).mkString(" ") + "\n")
28 }
29 bw.close()
```

从上面的代码中我们可以看出，Spark 的 Word2vec 模型训练过程非常简单，只需要四五行代码就可以完成。接下来，我就按照从上到下的顺序，依次给你解析其中 3 个关键的步骤。

首先是创建 Word2vec 模型并设定模型参数。我们要清楚 Word2vec 模型的关键参数有 3 个，分别是 `setVectorSize`、`setWindowSize` 和 `setNumIterations`。其中，`setVectorSize` 用于设定生成的 Embedding 向量的维度，`setWindowSize` 用于设定在序列数据上采样的滑动窗口大小，`setNumIterations` 用于设定训练时的迭代次数。这些超参数的具体选择就要根据实际的训练效果来做调整了。

其次，模型的训练过程非常简单，就是调用模型的 `fit` 接口。训练完成后，模型会返回一个包含了所有模型参数的对象。

最后一步就是提取和保存 Embedding 向量，我们可以从最后的几行代码中看到，调用 `getVectors` 接口就可以提取出某个电影 ID 对应的 Embedding 向量，之后就可以把它们保存到文件或者其他数据库中，供其他模块使用了。

在模型训练完成后，我们再来验证一下训练的结果是不是合理。我在代码中求取了 ID 为 592 电影的相似电影。这部电影叫 Batman 蝙蝠侠，我把通过 item2vec 得到相似电影放到了下面，你可以从直观上判断一下这个结果是不是合理。

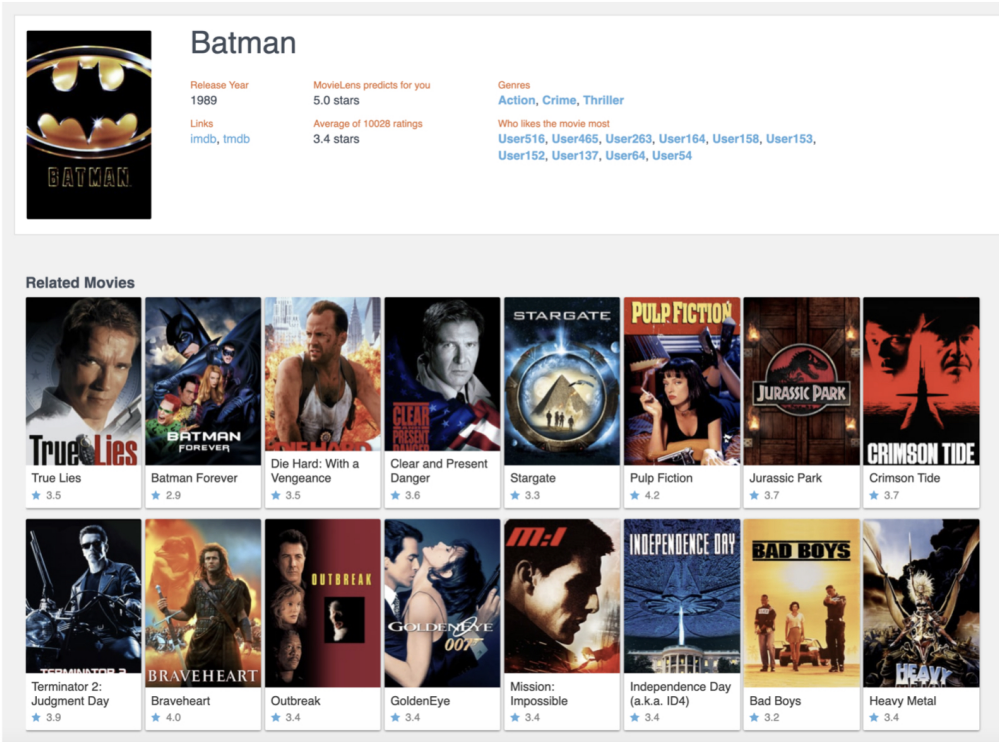


图2 通过Item2vec方法找出的电影Batman的相似电影

当然，因为 SparrowRecsys 在演示过程中仅使用了 1000 部电影和部分用户评论集，所以我们得出的结果不一定非常准确，如果你有兴趣优化这个结果，可以去 movieLens 下载全部样本进行重新训练。

Graph Embedding：数据准备

到这里，我相信你已经熟悉了 Item2vec 方法的实现。接下来，我们再来说说基于随机游走的 Graph Embedding 方法，看看如何利用 Spark 来实现它。这里，我们选择 Deep Walk 方法进行实现。

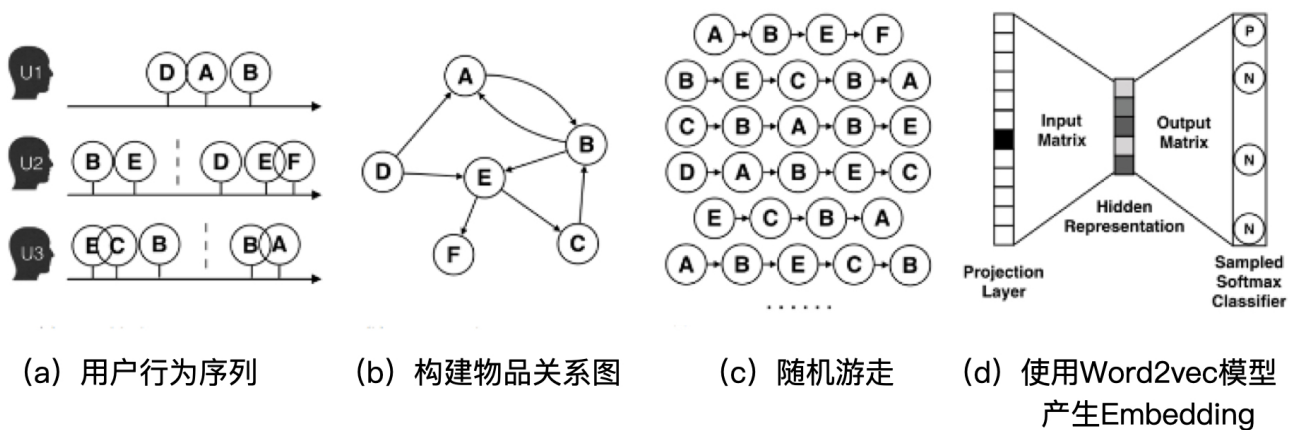


图3 Deep Walk的算法流程

在 Deep Walk 方法中，我们需要准备的最关键数据是物品之间的转移概率矩阵。图 3 是 Deep Walk 的算法流程图，转移概率矩阵表达了图 3 (b) 中的物品关系图，它定义了随机游走过程中，从物品 A 到物品 B 的跳转概率。所以我们首先来看一下如何利用 Spark 生成这个转移概率矩阵。

复制代码

```

1 //samples 输入的观影序列样本集
2 def graphEmb(samples : RDD[Seq[String]], sparkSession: SparkSession): Unit = {
3   //通过flatMap操作把观影序列打碎成一个个影片对
4   val pairSamples = samples.flatMap[String]( sample => {
5     var pairSeq = Seq[String]()
6     var previousItem:String = null
7     sample.foreach((element:String) => {
8       if(previousItem != null){
9         pairSeq = pairSeq :+ (previousItem + ":" + element)
10      }
11      previousItem = element
12    })
13    pairSeq
14  })
15  //统计影片对的数量
16  val pairCount = pairSamples.countByValue()
17  //转移概率矩阵的双层Map数据结构
18  val transferMatrix = scala.collection.mutable.Map[String, scala.collection.mutable.Map[String, Long]]()
19  val itemCount = scala.collection.mutable.Map[String, Long]()
20
21

```

```
22 //求取转移概率矩阵
23 pairCount.foreach( pair => {
24     val pairItems = pair._1.split(":")
25     val count = pair._2
26     lognumber = lognumber + 1
27     println(lognumber, pair._1)
28
29
30     if (pairItems.length == 2){
31         val item1 = pairItems.apply(0)
32         val item2 = pairItems.apply(1)
33         if(!transferMatrix.contains(pairItems.apply(0))){
34             transferMatrix(item1) = scala.collection.mutable.Map[String, Long]()
35         }
36
37
38         transferMatrix(item1)(item2) = count
39         itemCount(item1) = itemCount.getOrElse[Long](item1, 0) + count
40     }
41
42 }
```

生成转移概率矩阵的函数输入是在训练 Item2vec 时处理好的观影序列数据。输出的是转移概率矩阵，由于转移概率矩阵比较稀疏，因此我没有采用比较浪费内存的二维数组的方法，而是采用了一个双层 map 的结构去实现它。比如说，我们要得到物品 A 到物品 B 的转移概率，那么 transferMatrix(itemA)(itemB) 就是这一转移概率。

在求取转移概率矩阵的过程中，我先利用 Spark 的 flatMap 操作把观影序列“打碎”成一个个影片对，再利用 countByValue 操作统计这些影片对的数量，最后根据这些影片对的数量求取每两个影片之间的转移概率。

在获得了物品之间的转移概率矩阵之后，我们就可以进入图 3（c）的步骤，进行随机游走采样了。

Graph Embedding：随机游走采样过程

随机游走采样的过程是利用转移概率矩阵生成新的序列样本的过程。这怎么理解呢？首先，我们要根据物品出现次数的分布随机选择一个起始物品，之后就进入随机游走的过程。在每次游走时，我们根据转移概率矩阵查找到两个物品之间的转移概率，然后根据这个概率进行跳转。比如当前的物品是 A，从转移概率矩阵中查找到 A 可能跳转到物品 B 或

物品 C，转移概率分别是 0.4 和 0.6，那么我们就按照这个概率来随机游走到 B 或 C，依次进行下去，直到样本的长度达到了我们的要求。

根据上面随机游走的过程，我用 Scala 进行了实现，你可以参考下面的代码，在关键的位置我也给出了注释：

[复制代码](#)

```
1 //随机游走采样函数
2 //transferMatrix 转移概率矩阵
3 //itemCount 物品出现次数的分布
4 def randomWalk(transferMatrix : scala.collection.mutable.Map[String, scala.col
5 //样本的数量
6 val sampleCount = 20000
7 //每个样本的长度
8 val sampleLength = 10
9 val samples = scala.collection.mutable.ListBuffer[Seq[String]]()
10
11 //物品出现的总次数
12 var itemTotalCount:Long = 0
13 for ((k,v) <- itemCount) itemTotalCount += v
14
15
16 //随机游走sampleCount次，生成sampleCount个序列样本
17 for( w <- 1 to sampleCount) {
18     samples.append(oneRandomWalk(transferMatrix, itemCount, itemTotalCount, sa
19 }
20
21
22 Seq(samples.toList : _*)
23 }
24
25
26 //通过随机游走产生一个样本的过程
27 //transferMatrix 转移概率矩阵
28 //itemCount 物品出现次数的分布
29 //itemTotalCount 物品出现总次数
30 //sampleLength 每个样本的长度
31 def oneRandomWalk(transferMatrix : scala.collection.mutable.Map[String, scala.
32 val sample = scala.collection.mutable.ListBuffer[String]()
33
34
35 //决定起始点
36 val randomDouble = Random.nextDouble()
37 var firstElement = ""
38 var culCount:Long = 0
39 //根据物品出现的概率，随机决定起始点
40 breakable { for ((item, count) <- itemCount) {
41     culCount += count
```

```
42     if (culCount >= randomDouble * itemTotalCount){
43         firstElement = item
44         break
45     }
46 }}
47
48
49 sample.append(firstElement)
50 var curElement = firstElement
51 //通过随机游走产生长度为sampleLength的样本
52 breakable { for( w <- 1 until sampleLength) {
53     if (!itemCount.contains(curElement) || !transferMatrix.contains(curElement)
54         break
55     }
56     //从curElement到下一个跳的转移概率向量
57     val probDistribution = transferMatrix(curElement)
58     val curCount = itemCount(curElement)
59     val randomDouble = Random.nextDouble()
60     var culCount:Long = 0
61     //根据转移概率向量随机决定下一跳的物品
62     breakable { for ((item, count) <- probDistribution) {
63         culCount += count
64         if (culCount >= randomDouble * curCount){
65             curElement = item
66             break
67         }
68     }}
69     sample.append(curElement)
70 }}
71 Seq(sample.toList : _
72
```

通过随机游走产生了我们训练所需的 sampleCount 个样本之后，下面的过程就和 Item2vec 的过程完全一致了，就是把这些训练样本输入到 Word2vec 模型中，完成最终 Graph Embedding 的生成。你也可以通过同样的方法去验证一下通过 Graph Embedding 方法生成的 Embedding 的效果。

小结

这节课，我们运用 Spark 实现了经典的 Embedding 方法 Item2vec 和 Deep Walk。它们的理论知识你应该已经在前两节课的学习中掌握了，这里我就总结一下实践中应该注意的几个要点。

关于 Item2vec 的 Spark 实现，你应该注意的是训练 Word2vec 模型的几个参数 VectorSize、WindowSize、NumIterations 等，知道它们各自的作用。它们分别是用来

设置 Embedding 向量的维度，在序列数据上采样的滑动窗口大小，以及训练时的迭代次数。

而在 Deep Walk 的实现中，我们应该着重理解的是，生成物品间的转移概率矩阵的方法，以及通过随机游走生成训练样本过程。

最后，我还是把这节课的重点知识总结在了一张表格中，希望能帮助你进一步巩固。

知识点	关键描述
Spark中的自定义函数	用于实现一些Spark自带聚合函数外的复杂操作， 比如在Item2vec中实现的排序函数sortUdf: UserDefinedFunction
Word2vec中的关键参数	VectorSize、WindowSize和NumIterations
flatMap操作和map操作的区别	flatMap操作能够实现单条输入、多条输出， 而map操作只能够进行单条输入、单条输出
Deep Walk中随机游走的过程	是利用转移概率矩阵生成新的序列样本的过程， 其中最关键的两个函数是randomWalk和oneRandomWalk
Deep Walk中 转移概率矩阵的求取	使用到graphEmb函数，输入的是观影序列数据， 输出的是转移概率矩阵，并且采用了双层map的结构



这里，我还想再多说几句。这节课，我们终于看到了深度学习模型的产出，我们用 Embedding 方法计算出了相似电影！对于我们学习这门课来说，它完全可以看作是一个里程碑式的进步。接下来，我希望你能总结实战中的经验，跟我继续同行，一起迎接未来更多的挑战！

课后思考

上节课，我们在讲 Graph Embedding 的时候，还介绍了 Node2vec 方法。你能尝试在 Deep Walk 代码的基础上实现 Node2vec 吗？这其中，我们应该着重改变哪部分的代码呢？

欢迎把你的思考和答案写在留言区，如果你掌握了 Embedding 的实战方法，也不妨把它分享给你的朋友吧，我们下节课见！

提建议

更多学习推荐

机器学习训练营

成为能落地的实干型机器学习工程师

王然 众微科技 AI Lab 负责人

前100名秒杀 ¥3649  加赠书籍

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 07 | Embedding进阶：如何利用图结构数据生成Graph Embedding？

下一篇 答疑 | 基础架构篇+特征工程篇常见问题解答

精选留言 (11)

 写留言

你笑起来真好看

2020-10-20

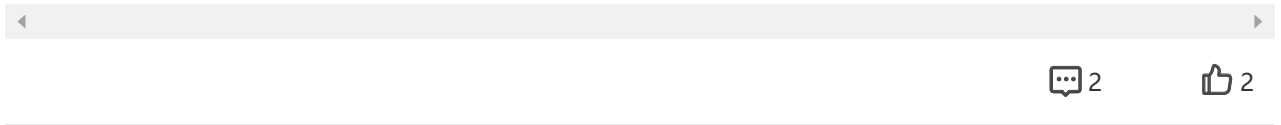
transitionMatrixAndItemDis 在生成中这样定义的话，会不会造成driver端oom？

作者回复：非常非常好的问题。细心的话可以发现transitionMatrixAndItemDis这个矩阵完全是在driver端。

难点在于如何分布式地处理转移概率矩阵，解决方案确实是业界的难点。另外在随机游走的时候

因为肯定要访问全部的转移矩阵，所以理论上来讲需要把这个矩阵broadcast到所有节点，这又是一个容易oom的问题。

有好的思路可以欢迎分享！但不是一个短时间能够解决的业界难题。

**WiFeng**

2020-10-26

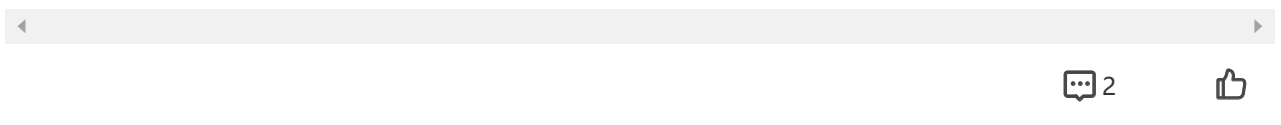
你们执行都没有问题吗？

```
/Users/leeco/github/wzhe06/SparrowRecSys/src/main/java/com/wzhe/sparrowrec  
sys/offline/spark/embedding/Embedding.scala:34:43  
No TypeTag available for scala.collection.Seq[String]...
```

展开 ▾

作者回复: 有可能是scala版本的问题，项目中使用了scala 2.11版本，可以检查一下是否一致。

如果还不能解决的话，需要自行百度或google一下解决方案。

**Todd**

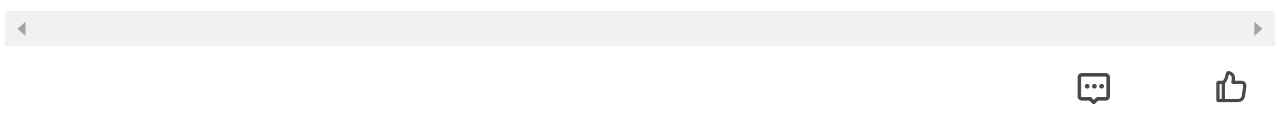
2020-10-22

最后一行代码bug，有没有github？

展开 ▾

作者回复: 没有从github上clone SparrowRecSys项目吗？这节课的所有实践代码都在这个项目里面。

这节课的内容请参考com.wzhe.sparrowrecsys.offline.spark.embedding类

**王发庆**

2020-10-22

老师，您好，请教您一个问题，在生成Embedding的时候我们都是全量生成的，在生产环境下我们能增量的去生成新节点的Embedding么？

展开 ▾

作者回复: 不可以。Embedding冷启动的问题大家问的比较多，我找时间统一回复一下。

简单来说生成环境下对于冷启动物品需要指定一个默认embedding，或者基于一些其他的相似条件取相似embedding的平均。



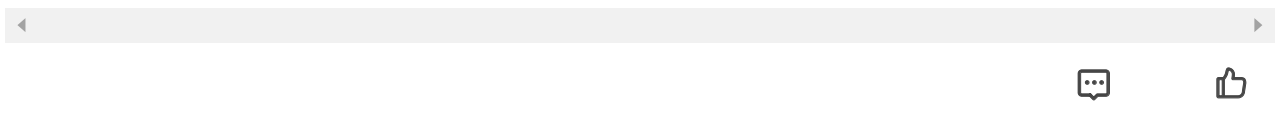
Geek_63ee39

2020-10-22

请教一下老师，从效率角度上讲，可以把转移矩阵transferMatrix放在executor端进行分布式随机游走吧，因为这是可以并行计算的。

展开 ∨

作者回复: 前提是把transferMatrix广播出去，项目里面的transferMatrix大小没问题，如果特别海量数据形成的transferMatrix可能一些worker node的内存会有问题。



聪聪呀

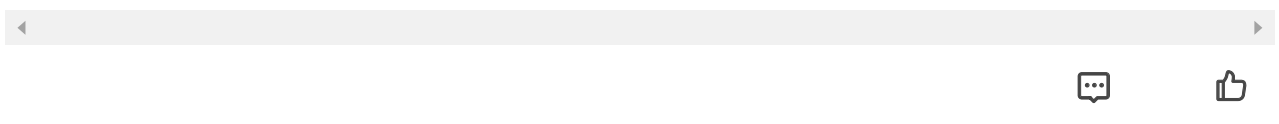
2020-10-21

老师，我最近在研究使用graph embedding，根据网上的git 代码跑了item2vec，EGES 代码，我的推荐场景是视频推荐，同样的训练数据，我发现item2vec 推荐效果较好，但我发现EGES推荐效果不好（只用了ID，没有加其他特征）推荐结果相似度很低。所以想请教您，您觉得可能是什么原因引起的呢，您有没有EGES的demo

展开 ∨

作者回复: EGES是希望融合更多side information，只用id信息并不是它的意义所在。而且除了这些原理上的分析外，我不主张给任何人模型效果好坏的建议，决定的因素太多了。

但总的来说，如果你只有id类的历史行为信息，使用EGES的意义不大。

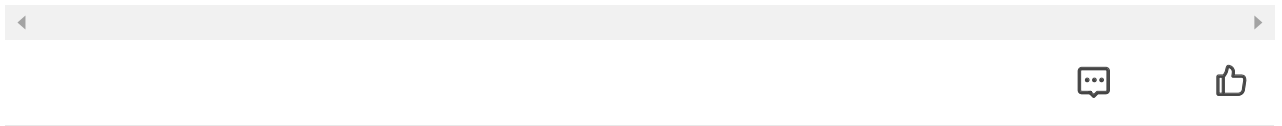


芳菲菲兮满堂

2020-10-20

老师 生产环境对spark的调用都是用scala吗？有用python的吗？

作者回复: 用pyspark肯定也没问题。但因为spark是原生支持scala，所以大部分团队，特别是专门的数据团队是以scala为主。



Huntley

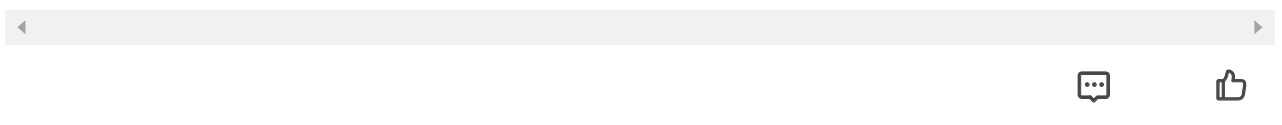
2020-10-19

想请教下老师，user embedding 也可以用相同的方法获得吗？如何构建用户关系图呢？是不是看过同一部电影的两个用户之间由一条无向边进行连接？和item embedding相比，有什么区别或注意事项吗？

作者回复: 在项目中用最简单的average pooling的方法生成了user embedding。也就是把用户评价过的高分高分电影的embedding进行平均。

user embedding的其他生成方式也很多，像你说的，也可以根据历史行为构建用户-物品的关系图，然后直接在其上进行随机游走，直接一同生成user 和item emb。

或者采用其他双塔结构的模型生成item和user emb等等。



大魔王

2020-10-19

对scala语法不太熟,scala方法返回值居然不用写return...



浣熊当家

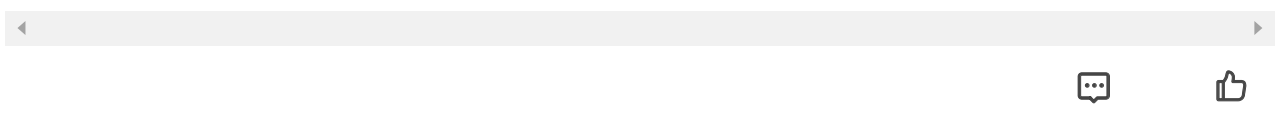
2020-10-19

老师，请问我平时工作中是用PySpark，这节课的代码是不是也都可以用python的来写？有什么好的学习资料来学习怎么把他们翻译成python吗？谢谢老师！！

展开 ∨

作者回复: 我一个人能力有限，欢迎大家提供java，python的版本，贡献到开源项目中！

从scala到python应该没有完全自动的转换工具，还是需要参照官方文档不同语言的用法来进行转换。



**WiFeng**

2020-10-19

由于之前没有实际基础过机器学习开发，请问老师，上面这些代码也是在 spark-shell 中执行吗？

作者回复: 不是，通过IDE的Spark环境执行，接近真实的生产环境。如果安装好了SparrowRecsys，直接执行Embedding object的主函数就可以了。

