sparkmllib协同过滤推荐算法

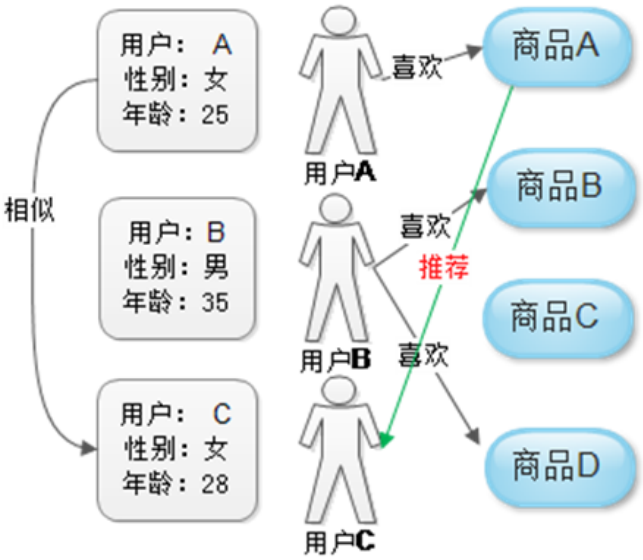
标签： sparkmllib 协同过滤算法

2016-11-20 15:39 2111人阅读 评论(0) 收藏 举报

目录(?) [+]

协同过滤推荐**算法**，是最经典、最常用的推荐算法。通过分析用户兴趣，在用户群中找到相似用户，综合这些相似用户对某一信息的评价，形成系统关于该指定用户对此信息的喜好程度预测。要实现协同过滤，需要以下几个步骤：

```
1）收集用户偏好；
2）找到相似的用户或物品；
3）计算推荐。
```



用户评分

从用户的行为和偏好中发现规律，并基于此进行推荐，所以收集用户的偏好信息成为系统推荐效果最基础的决定因素。用户有很多种方式向系统提供自己的偏好信息，比如：评分、投票、转发、保存书签、购买、点击流、页面停留时间等。

1. 将不同的行为分组

一般可以分为查看和购买，然后基于不同的用户行为，计算不同用户或者物品的相似度。

2. 对不同行为进行加权

对不同行为产生的用户喜好进行加权，然后求出用户对物品的总体喜好。当我们收集好用户的行为数据后，还要对 数据进行预处理，最核心的工作就是减噪和归一化。

表 14-1 用户评分表

用户/物品	物品A	物品B	物品C
用户A	0.1	0.8	1
用户B	0.1	0	0.02
用户C	0.5	0.3	0.1

对用户的行为分析得到用户的偏好后，可以根据用户的偏好计算相似用户和物品，然后可以基于相似用

sparkmllib算法实例 (2)
Spark相关 (0)
scala学习 (0)
CDH安装 (0)
Elasticsearch线程池配置 (0)
Java执行js文件 (0)

	物品1	物品2	物品3	物品4	物品5	物品6
用户A	1	1	0	0	0	1
用户B	1	0	1	1	0	0
用户C	1	1	1	0	0	0
用户D	0	0	1	1	1	0
用户E	0	0	0	1	1	1
用户F	1	0	1	0	1	0

1. 同现相似度

物品 i 和物品 j 的同现相似度公式定义：

$$w_{i,j}=\frac{|N(i)\cap N(j)|}{|N(i)|}$$

其中，分母$|N(i)|$是喜欢物品 i 的用户数，而分子$|N(i)\cap N(j)|$是同时喜欢物品 i [illegible] 用户数据。因此，上述公式可以理解为喜欢物品 i 的用户中有多少比例的用户也喜欢物品 i。

上述公式存在一个问题，如果物品 j 是热门物品，很多人都喜欢，那么 $w_{i,j}$ 就会很大，接近1。因此，该公式会造成任何物品都会和热门物品有很大的相似度。为了避免推荐出热可以用如下公式：

$$w_{i,j}=\frac{|N(i)\cap N(j)|}{\sqrt{|N(i)||N(j)|}}$$

这个公式惩罚了物品 j 的权重，因此减轻了热门物品与很多物品相似的可能性。

2. 欧氏距离（Euclidean Distance）

最初用于计算欧几里得空间中两个点的距离，假设 x、y 是 n 维空间的两个点，它们之间的欧几里得距离是：

$$d(x,y)=\sqrt{\sum(x_i-y_i)^2}$$

可以看出，当 n=2 时，欧几里得距离就是平面上两个点的距离。

当用欧几里得距离表示相似度时，一般采用以下公式进行转换：距离越小，相似度越大。

$$\mathrm{sim}(x,y)=\frac{1}{1+d(x,y)}$$

3. 皮尔逊相关系数（Pearson Correlation Coefficient）
4. Cosine 相似度（Cosine Similarity）
5. Tanimoto 系数（Tanimoto Coefficient）

推荐计算

1. 基于用户的CF（User CF）

   基于用户的 CF 的基本思想相当简单：基于用户对物品的偏好找到相邻的邻居用户，然后将邻居用户喜欢的推荐给当前用户。在计算上，就是将一个用户对所有物品的偏好作为一个向量来计算用户之间的相似度，找到K 邻居后，根据邻居的相似度权重及其对物品的偏好，预测当前用户没有偏好的未涉及物品，计算得到一个排序的物品列表作为推荐。图14-1 给出了一个例子，对于用户A，根据用户的历史偏好，这里只计算得到一个邻居-用户C，然后将用户C 喜欢的物品D 推荐给用户

A。

用户/物品	物品A	物品B	物品C	物品D
用户A	√		√	推荐
用户B		√		
用户C	√		√	√

2. 基于物品的CF（Item CF）

基于物品的CF 的原理和基于用户的CF 类似，只是在计算邻居时采用物品本身，而不是从用户的角度。即基于用户对物品的偏好找到相似的物品，然后根据用户的历史偏好，推荐相似的物品给他。从计算的角度看，就是将所有用户对某个物品的偏好作为一个向量来计算物品之间的相似度，得到物品的相似物品后，根据用户历史的偏好预测当前用户还没有表示偏好的物品，计算的物品列表作为推荐。

用户/物品	物品A	物品B	物品C
用户A	√		√
用户B	√	√	√
用户C	√		推荐

图 14-2 基于物品的 CF 的基本原理

根据用户评分矩阵采用同现相似度计算物品相似度矩阵。

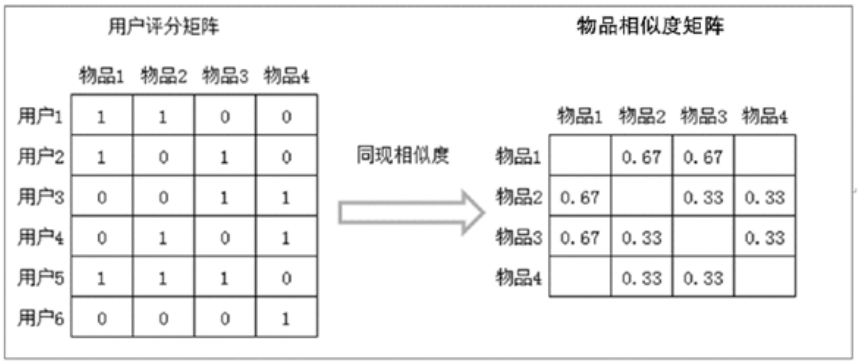


图 14-3 物品相似度矩阵

其相似度计算实现了分布式计算，实现过程如下：

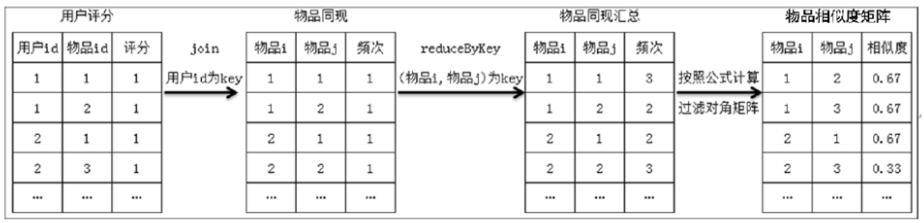


图 14-4 分布式同现相似度矩阵计算过程

对于欧氏相似度的计算，采用离散计算公式d(x, y) = sqrt(Σ((x(i)-y(i)) * (x(i)- y(i))))。其中，i 只取x、y 同现的点，未同现的点不参与相似度计算；sim(x, y) = m / (1 + d(x, y))，m 为x、y 重叠数，同现次数

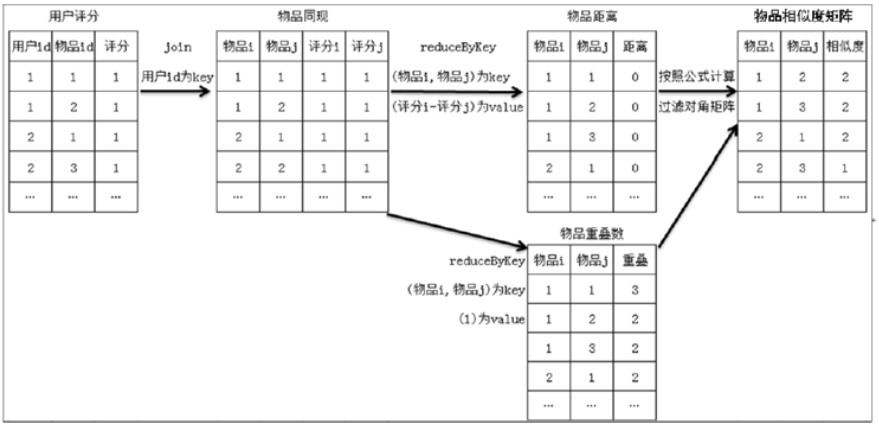


图 14-5 分布式欧氏距离相似度矩阵计算过程

根据物品相似度矩阵和用户评分计算用户推荐列表，计算公式是R=W*A，取推荐计
分过的物品，并且按照计算结果倒序推荐给用户。

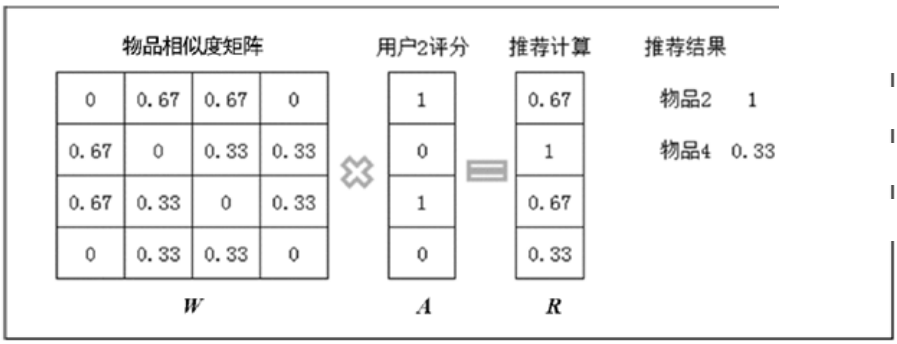


图 14-6 协同推荐计算

其推荐计算实现了分布式计算。

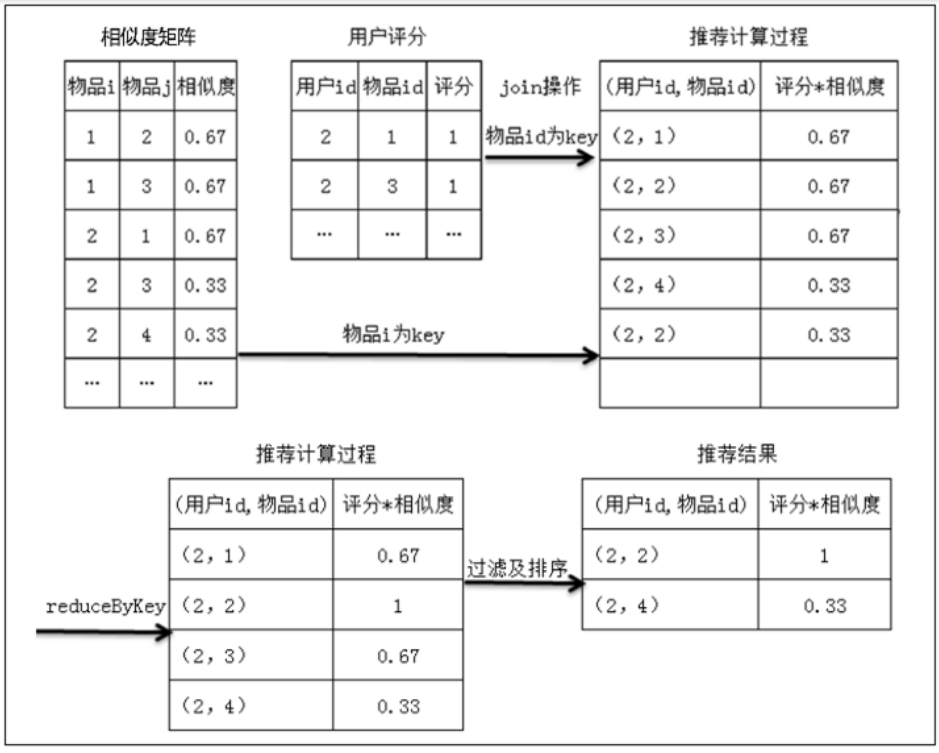


图 14-7 分布式协同推荐计算过程

源码分析

表 14-2　ItemCF协同过滤源码分解

源码分解说明		
1. 物品相似度类	ItemSimilarity	ItemSimilarity类
1.1　相似度计算	Similarity	ItemSimilarity类的Similarity方法。该方法计算物品与物品之间的相似度
2. 基于物品协同推荐类	RecommendedItem	RecommendedItem类
2.1　推荐计算	Recommend	RecommendedItem类的Recommend方法。该方法根据物品与物品的相似度模型和用户评分，计算用户的推荐物品列表

ALS算法:

```
def train(
      ratings: RDD[Rating],
      rank: Int,
      iterations: Int,
      lambda: Double,
      blocks: Int,
      seed: Long
    ): MatrixFactorizationModel = {
    new ALS(blocks, blocks, rank, iterations, lambda, false, 1.0, seed).run(
  }
```

```
 def run(ratings: RDD[Rating]): MatrixFactorizationModel = {
    require(!ratings.isEmpty(), s"No ratings available from $ratings")

    val sc = ratings.context

    val numUserBlocks = if (this.numUserBlocks == -1) {
      math.max(sc.defaultParallelism, ratings.partitions.length / 2)
    } else {
      this.numUserBlocks
    }
    val numProductBlocks = if (this.numProductBlocks == -1) {
      math.max(sc.defaultParallelism, ratings.partitions.length / 2)
    } else {
      this.numProductBlocks
    }

    val (floatUserFactors, floatProdFactors) = NewALS.train[Int](
      ratings = ratings.map(r => NewALS.Rating(r.user, r.product, r.rating.toFloat)),
      rank = rank,
      numUserBlocks = numUserBlocks,
      numItemBlocks = numProductBlocks,
      maxIter = iterations,
      regParam = lambda,
      implicitPrefs = implicitPrefs,
      alpha = alpha,
      nonnegative = nonnegative,
      intermediateRDDStorageLevel = intermediateRDDStorageLevel,
      finalRDDStorageLevel = StorageLevel.NONE,
      checkpointInterval = checkpointInterval,
      seed = seed)

    val userFactors = floatUserFactors
      .mapValues(_.map(_.toDouble))
      .setName("users")
      .persist(finalRDDStorageLevel)
    val prodFactors = floatProdFactors
      .mapValues(_.map(_.toDouble))
      .setName("products")
      .persist(finalRDDStorageLevel)
    if (finalRDDStorageLevel != StorageLevel.NONE) {
      userFactors.count()
      prodFactors.count()
    }
    new MatrixFactorizationModel(rank, userFactors, prodFactors)
  }
```

```
private def recommend(
      recommendToFeatures: Array[Double],
      recommendableFeatures: RDD[(Int, Array[Double])],
      num: Int): Array[(Int, Double)] = {
    val scored = recommendableFeatures.map { case (id, features) =>
      (id, blas.ddot(features.length, recommendToFeatures, 1, features, 1))
    }
    scored.top(num)(Ordering.by(_._2))
  }
```

实例：

```
import org.apache.spark.mllib.recommendation.ALS
import org.apache.spark.mllib.recommendation.MatrixFactorizationModel
import org.apache.spark.mllib.recommendation.Rating

// Load and parse the data
val data = sc.textFile("data/mllib/als/test.data")
val ratings = data.map(_.split(',') match { case Array(user, item, rate) =>
  Rating(user.toInt, item.toInt, rate.toDouble)
})

// Build the recommendation model using ALS
val rank = 10
val numIterations = 10
val model = ALS.train(ratings, rank, numIterations, 0.01)

// Evaluate the model on rating data
val usersProducts = ratings.map { case Rating(user, product, rate) =>
  (user, product)
}
val predictions =
  model.predict(usersProducts).map { case Rating(user, product, rate) =>
    ((user, product), rate)
  }
val ratesAndPreds = ratings.map { case Rating(user, product, rate) =>
  ((user, product), rate)
}.join(predictions)
val MSE = ratesAndPreds.map { case ((user, product), (r1, r2)) =>
  val err = (r1 - r2)
  err * err
}.mean()
println("Mean Squared Error = " + MSE)

// Save and load model
model.save(sc, "target/tmp/myCollaborativeFilter")
val sameModel = MatrixFactorizationModel.load(sc, "target/tmp/myCollaborativeFilter")
```


上一篇 sparkmllib关联规则算法（FPGrowth,Apriori）
下一篇 QQ邮箱模拟登录

相关文章推荐

- Spark MLlib系列(二):基于协同过滤的电影推荐系统
- 基于Spark构建推荐引擎之一：基于物品的协同过...
- spark基于用户的协同过滤算法与坑点，提交job
- spark中协同过滤算法分析

- 《Spark MLlib 机器学习》第二章代码
- spark/MLlib 协同过滤算法
- 基于Spark MLlib平台的协同过滤算法---电影推荐…
- 协同过滤itembase增量计算Spark实现(一)
- Spark学习笔记 - 推荐系统（协同过滤算法为用户…
- 基于Spark MLlib平台的协同过滤算法---电影推荐…

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场