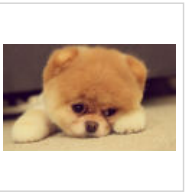
996440550 的BLOG



发私信 加友情链接

博主的更多文章>>

原创 基于Spark MLlib平台的协同过滤算法---电影推荐系统

2015-06-15 21:48:57

标签：协同过滤 推荐系统 Spark MLlib

基于Spark MLlib平台的协同过滤算法---电影推荐系统

又好一阵子没有写文章了，阿弥陀佛...最近项目中要做理财推荐，所以，回过头来回顾一下协同过滤算法在推荐系统中的应用。

说到推荐系统，大家可能立马会想到协同过滤算法。本文基于Spark MLlib平台实现一个向用户推荐电影的简单应用。其中，主要包括三部分内容：

协同过滤算法概述

基于模型的协同过滤应用---电影推荐

实时推荐架构分析

一、协同过滤算法概述

本人对算法的研究，目前还不是很深入，这里简单的介绍下其工作原理。

通常，协同过滤算法按照数据使用，可以分为：

1）基于用户（UserCF）

2）基于商品（ItemCF）

3）基于模型（ModelCF）

按照模型，可以分为：

1）最近邻模型：基于距离的协同过滤算法

2）Latent Factor Mode（SVD）：基于矩阵分解的模型

3）Graph：图模型，社会网络图模型

文中，使用的协同过滤算法是基于矩阵分解的模型。

意见反馈

1、基于用户（UserCF）---基于用户相似性

基于用户的协同过滤，通过不同用户对物品的评分来评测用户之间的相似性，基于用户之间的相似性做出推荐。简单来讲，就是给用户推荐和他兴趣相似的其他用户喜欢的物品。

举个例子：

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	√		√	√

如图，有三个用户A、B、C，四个物品A、B、C、D，需要向用户A推荐物品。这里，由于用户A和用户C都买过物品A和物品C，所以，我们认为用户A和用户C非常相似，同时，用户C又买过物品D，那么就需要给A用户推荐物品D。

基于UserCF的基本思想相当简单，基于用户对物品的偏好，找到相邻邻居用户，然后将邻居用户喜欢的商品推荐给当前用户。

计算上，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以及他们对物品的偏好，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。

2、基于商品（ItemCF）---基于商品相似性

基于商品的协同过滤，通过用户对不同item的评分来评测item之间的相似性，基于item之间的相似性做出推荐。简单来将，就是给用户推荐和他之前喜欢的物品相似的物品。

例如：

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

如图，有三个用户A、B、C和三件物品A、B、C，需要向用户C推荐物品。这里，由于用户A买过物品A和C，用户B买过物品A、B、C，用户C买过物品A，从用户A和B可以看出，这两个用户都买过物品A和C，说明物品A和C非常相似，同时，用户C又买过物品A，所以，将物品C推荐给用户C。

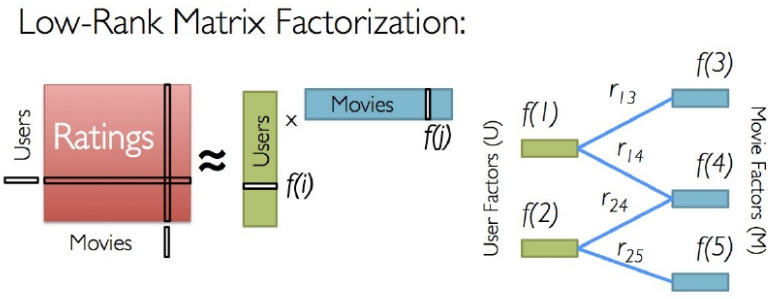
基于ItemCF的原理和基于UserCF类似，只是在计算邻居时采用物品本身，而不是从用户的角度，即基于用户对物品的偏好找到相似的物品，然后根据用户的历史偏好，推荐相似的物品给他。

从计算角度，即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，计算得到一个排序的物品列表作为推荐。

3、基于模型（ModelCF）

基于模型的协同过滤推荐就是基于样本的用户喜好信息，训练一个推荐模型，然后根据实时的用户喜好的信息进行预测，计算推荐。

本文使用的基于矩阵分解的模型，算法如图：

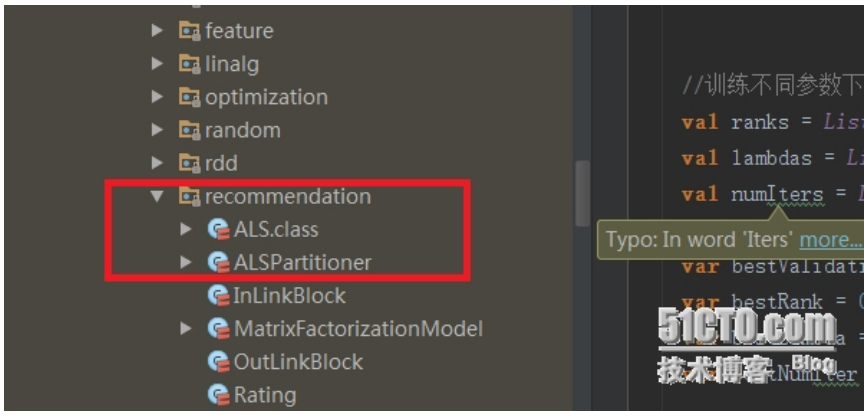


Iterate:

$$f[i] = \arg\min_{w \in \mathbb{R}^d} \sum_{j \in \mathrm{Nbrs}(i)} \left(r_{ij} - w^T f[j]\right)^2 + \lambda ||w||_2^2$$

Spark MLlib当前支持基于模型的协同过滤，其中用户和商品通过一小组隐性因子进行表达，并且这些因子也用于预测缺失的元素。MLlib使用交替最小二乘法（ALS）来学习这些隐性因子。

如果有兴趣，可以阅读Spark的这部分源代码：



二、基于模型的协同过滤应用---电影推荐

本文实现对用户推荐电影的简单应用。

1、测试数据描述

本次测试数据主要包括四个数据文件：（详细的数据描述参见README文件）

1）用户数据文件

用户ID::性别::年龄::职业编号::邮编

```
[root@hdbdap4t ml-1m]# tail users.dat
6031::F::18::0::45123
6032::M::45::7::55108
6033::M::50::13::78232
6034::M::25::14::94117
6035::F::25::1::78734
6036::F::25::15::32603
6037::F::45::1::76006
6038::F::56::1::14706
6039::F::45::0::01060
6040::M::25::6::11106
```

2）电影数据文件

电影ID::电影名称::电影种类

```
[root@hdbdap4t ml-1m]# tail -f movies.dat
3943::Bamboozled (2000)::Comedy
3944::Bootmen (2000)::Comedy|Drama
3945::Digimon: The Movie (2000)::Adventure|Animation|Children's
3946::Get Carter (2000)::Action|Drama|Thriller
3947::Get Carter (1971)::Thriller
3948::Meet the Parents (2000)::Comedy
3949::Requiem for a Dream (2000)::Drama
3950::Tigerland (2000)::Drama
3951::Two Family House (2000)::Drama
3952::Contender, The (2000)::Drama|Thriller
```

3）评分数据文件

用户ID::电影ID::评分::时间

```
Terminal
[root@hdbdap4t ml-1m]# tail ratings.dat  -f
6040::2022::5::956716207
6040::2028::5::956704519
6040::1080::4::957717322
6040::1089::4::956704996
6040::1090::3::956715518
6040::1091::1::956716541
6040::1094::5::956704887
6040::562::5::956704746
6040::1096::4::956715648
6040::1097::4::956715569
```

4）测试数据

用户ID::电影ID::评分::时间

这里，前三个数据文件用于模型训练，第四个数据文件用于测试模型。

2、实现代码：

```
import org.apache.log4j.{Level, Logger}
import org.apache.spark.mllib.recommendation.{ALS, MatrixFactorizationModel, Rating}
import org.apache.spark.rdd._
import org.apache.spark.{SparkContext, SparkConf}
import org.apache.spark.SparkContext._

import scala.io.Source

object MovieLensALS {
   def main(args:Array[String]) {

      //屏蔽不必要的日志显示在终端上
      Logger.getLogger("org.apache.spark").setLevel(Level.WARN)
      Logger.getLogger("org.apache.eclipse.jetty.server").setLevel(Level.OFF)

      //设置运行环境
      val sparkConf = new SparkConf().setAppName("MovieLensALS").setMaster("local[5]")
      val sc = new SparkContext(sparkConf)

      //装载用户评分，该评分由评分器生成(即生成文件personalRatings.txt)
      val myRatings = loadRatings(args(1))
      val myRatingsRDD = sc.parallelize(myRatings, 1)

      //样本数据目录
      val movielensHomeDir = args(0)

      //装载样本评分数据，其中最后一列Timestamp取除10的余数作为key，Rating为值，即(Int，Rating)
      val ratings = sc.textFile(movielensHomeDir + "/ratings.dat").map {
         line =>
            val fields = line.split("::")
            // format: (timestamp % 10, Rating(userId, movieId, rating))
            (fields(3).toLong % 10, Rating(fields(0).toInt, fields(1).toInt, fields(2).toDouble))
      }

      //装载电影目录对照表(电影ID->电影标题)
      val movies = sc.textFile(movielensHomeDir + "/movies.dat").map {
         line =>
            val fields = line.split("::")
            // format: (movieId, movieName)
```

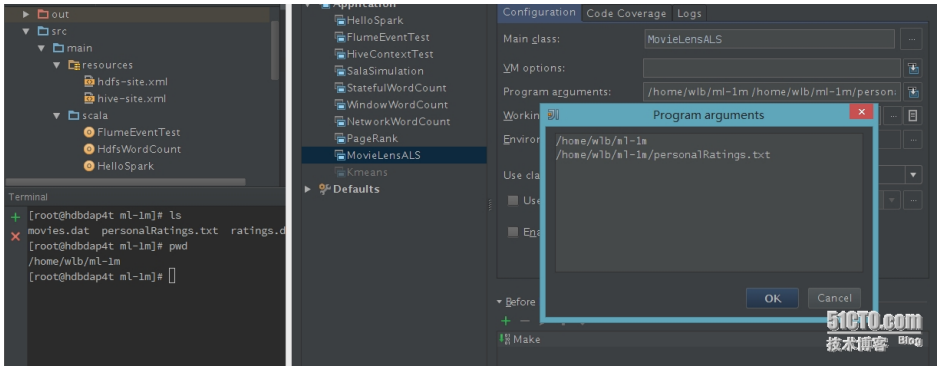
```
            (fields(0).toInt, fields(1))
        }.collect().toMap

          //统计有用户数量和电影数量以及用户对电影的评分数目
        val numRatings = ratings.count()
        val numUsers = ratings.map(_._2.user).distinct().count()
        val numMovies = ratings.map(_._2.product).distinct().count()
        println("Got " + numRatings + " ratings from " + numUsers + " users " + numMovies + " movies")

        //将样本评分表以key值切分成3个部分，分别用于训练 (60%，并加入用户评分)， 校验 (20%)， and 测试 (20%)
        //该数据在计算过程中要多次应用到，所以cache到内存
        val numPartitions = 4
        val training = ratings.filter(x => x._1 < 6).values.union(myRatingsRDD).repartition(numPartitions).persist()
        val validation = ratings.filter(x => x._1 >= 6 && x._1 < 8).values.repartition(numPartitions).persist()
        val test = ratings.filter(x => x._1 >= 8).values.persist()

        val numTraining = training.count()
        val numValidation = validation.count()
        val numTest = test.count()
        println("Training: " + numTraining + " validation: " + numValidation + " test: " + numTest)


        //训练不同参数下的模型，并在校验集中验证，获取最佳参数下的模型
        val ranks = List(8, 12)
        val lambdas = List(0.1, 10.0)
        val numIters = List(10, 20)
        var bestModel: Option[MatrixFactorizationModel] = None
        var bestValidationRmse = Double.MaxValue
        var bestRank = 0
        var bestLambda = -1.0
        var bestNumIter = -1

        for (rank <- ranks; lambda <- lambdas; numIter <- numIters) {
           val model = ALS.train(training, rank, numIter, lambda)
           val validationRmse = computeRmse(model, validation, numValidation)
           println("RMSE(validation) = " + validationRmse + " for the model trained with rank = "
              + rank + ",lambda = " + lambda + ",and numIter = " + numIter + ".")

           if (validationRmse < bestValidationRmse) {
              bestModel = Some(model)
              bestValidationRmse = validationRmse
              bestRank = rank
              bestLambda = lambda
              bestNumIter = numIter
           }
        }

        //用最佳模型预测测试集的评分，并计算和实际评分之间的均方根误差（RMSE）
        val testRmse = computeRmse(bestModel.get, test, numTest)
```

```
println("The best model was trained with rank = " + bestRank + " and lambda = " + bestLambda
    + ", and numIter = " + bestNumIter + ", and its RMSE on the test set is " + testRmse + ".")

//create a naive baseline and compare it with the best model
val meanRating = training.union(validation).map(_.rating).mean
val baselineRmse = math.sqrt(test.map(x => (meanRating - x.rating) * (meanRating -
x.rating)).reduce(_ + _) / numTest)
val improvement = (baselineRmse - testRmse) / baselineRmse * 100
println("The best model improves the baseline by " + "%1.2f".format(improvement) + "%.")

//推荐前十部最感兴趣的电影，注意要剔除用户已经评分的电影
val myRatedMovieIds = myRatings.map(_.product).toSet
val candidates = sc.parallelize(movies.keys.filter(!myRatedMovieIds.contains(_)).toSeq)
val recommendations = bestModel.get
    .predict(candidates.map((0, _)))
    .collect
    .sortBy(-_.rating)
    .take(10)
var i = 1
println("Movies recommended for you:")
recommendations.foreach { r =>
    println("%2d".format(i) + ": " + movies(r.product))
    i += 1
}

sc.stop()
}


/** 校验集预测数据和实际数据之间的均方根误差 **/
def computeRmse(model:MatrixFactorizationModel,data:RDD[Rating],n:Long):Double = {

    val predictions:RDD[Rating] = model.predict((data.map(x => (x.user,x.product))))
    val predictionsAndRatings = predictions.map{ x =>((x.user,x.product),x.rating)}
                                    .join(data.map(x => ((x.user,x.product),x.rating))).values
    math.sqrt(predictionsAndRatings.map( x => (x._1 - x._2) * (x._1 - x._2)).reduce(_+_)/n)
}

/** 装载用户评分文件 personalRatings.txt **/
def loadRatings(path:String):Seq[Rating] = {
    val lines = Source.fromFile(path).getLines()
    val ratings = lines.map{
        line =>
            val fields = line.split("::")
            Rating(fields(0).toInt,fields(1).toInt,fields(2).toDouble)
    }.filter(_.rating > 0.0)
    if(ratings.isEmpty){
        sys.error("No ratings provided.")
    }else{
        ratings.toSeq
    }
}
```

```
}
```

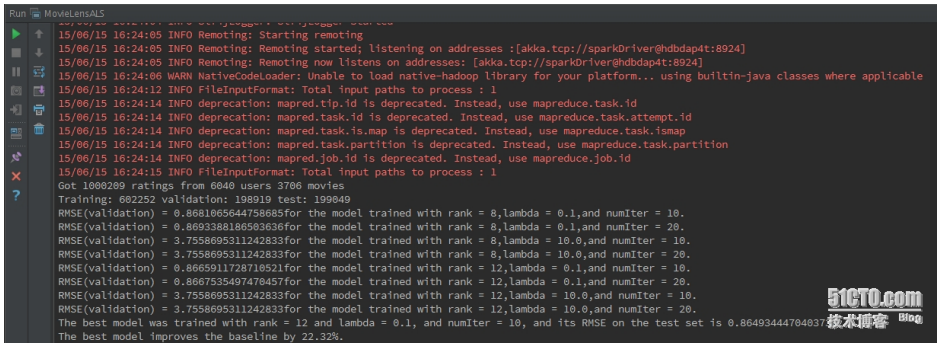
3、运行程序

1）设置参数，运行程序



这里有两个输入参数：第一个是数据文件目录，第二个是测试数据。

2）程序运行效果---模型训练过程



从运行效果来看，总共有6040个用户，3706个电影（已经去重），1000209条评分数据；如程序，我们把所有数据分为三部分：60%用于训练、20%用户校验、20%用户测试模型；接下来是模型在不同参数下的均方根误差（RMSE）值，以及对应的参数，最优的参数选择均方根误差（RMSE---0.8665911...）最小的参数值---即最优参数模型建立；接着，使用20%的测试模型数据来测试模型的好坏，也就是均方根误差（RMSE），这里计算的结果为0.86493444...，在最优参数模型基础上提升了22.32%的准确率。

说明下，其实在数据的划分上（60%+20%+20%），最好随机划分数据，这样得到的结果更有说服力。

3）程序运行效果---电影推荐结果

最后，给用户推荐10部自己未看过的电影。

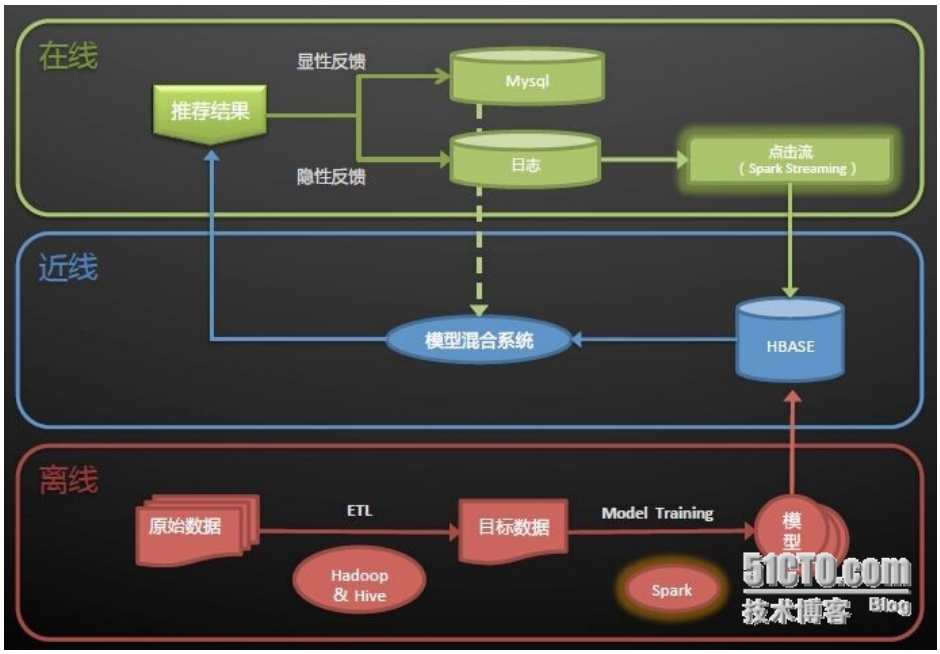
4、总结

这样，一个简单的基于模型的电影推荐应用就算OK了。

三、实时推荐架构分析

上面，实现了简单的推荐系统应用，但是，仅仅实现用户的定向推荐，在实际应用中价值不是非常大，如果体现价值，最好能够实现实时或者准实时推荐。

下面，简单介绍下实时推荐的一个架构：



该架构图取自淘宝Spark On Yarn的实时架构，这里，给出一些个人的观点：

架构图分为三层：离线、近线和在线。

离线部分：主要实现模型的建立。原始数据通过ETL加工清洗，得到目标数据，目标业务数据结合合适的算法，学习训练模型，得到最佳的模型。

近线部分：主要使用HBase存储用户行为信息，模型混合系统综合显性反馈和隐性反馈的模型处理结果，将最终的结果推荐给用户。

在线部分：这里，主要有两种反馈，显性和隐性，个人理解，显性反馈理解为用户将商品加入购物车，用户购买商品这些用户行为；隐性反馈理解为用户在某个商品上停留的时间，用户点击哪些商品这些用户行为。这里，为了实现实时/准实时操作，使用到了Spark Streaming对数据进行实时处理。（有可能是Flume+Kafka+Spark Streaming架构）

这里是个人的一些理解，不足之处，望各位指点。

本文出自 “一步.一步” 博客，请务必保留此出处http://snglw.blog.51cto.com/5832405/1662153

类别：机器学习应用 ┆ 阅读(6409) ┆ 评论(4) ┆ 返回博主首页 ┆ 返回博客首页

上一篇 spark MLlib之零 构建通用的解析矩阵程序

文章评论

[1楼] gaojibin14 意见反馈

2015-08-20 15:05:11

楼主，可否提供下README四个文件，供实验参考，我的邮箱是568301727@qq.com

[2楼]楼主 996440550 回复

2015-08-21 13:24:23

回复 gaojibin14:[1楼]

从这可以下载：
http://down.51cto.com/data/2065052

[3楼] rongcui986 回复

2015-11-13 11:01:56

我最近也要做理财的推荐，博主可否提供一些思路？

[4楼] nnnnajius 回复

2015-12-26 23:04:26

博主 请问你这些是不是照着Codementor上的那篇文章：Building a Movie Recommendation Service with Apache Spark & Flask 来做的？

发表评论

昵　　称: 登录 快速注册

验证码: 请点击后输入验证码 博客过2级，无需填写验证码

内　　容: