ALS 实战 从行为数据到评分再到预测

1.ML 库 ALS 简介及实战

1.1 协同过滤内容

- 协同过滤
  - 显性与隐性反馈
  - 缩放正则化参数
  - 冷启动策略

1.2 协同过滤

协同过滤 通常用于推荐系统。这些技术旨在填写用户项关联矩阵的缺失条目。 spark.ml 目前支持基于模型的协同过滤，其中用户和产品由一小组可用于预测缺失条目的潜在因素描述。 spark.ml 使用交替最小二乘（ALS） 算法来学习这些潜在因素。实现中 spark.ml 包含以下参数：

- *numBlocks* 是用户和项目将被分区为块以便并行计算的块数（默认为 10）。
- *rank* 是模型中潜在因子的数量（默认为 10）。
- *maxIter* 是要运行的最大迭代次数（默认为 10）。
- *regParam* 指定 ALS 中的正则化参数（默认为 1.0）。
- *implicitPrefs* 指定是使用*显式反馈* ALS 变体还是使用适用于*隐式反馈*数据的变体 （默认值 false 表示使用*显式反馈*）。
- *alpha* 是适用于 ALS 的隐式反馈变量的参数，其控制偏好观察中的 *基线*置信度（默认为 1.0）。
- *nonnegative* 指定是否对最小二乘使用非负约束（默认为 false）。

注意：基于 DataFrame 的 ALS API 目前仅支持用户和项 ID 的整数。user 和 item id 列支持其他数字类型，但 id 必须在整数值范围内。

1.2.1 显性与隐性反馈

基于矩阵分解的协同过滤的标准方法将用户项目矩阵中的条目视为用户对项目给出的*显式*偏好，例如，给予电影评级的用户。

在许多现实世界的用例中，通常只能访问*隐式反馈*（例如，观看，点击，购买，喜欢，分享等）。用于 spark.ml 处理此类数据的方法来自隐式反馈数据集的协作过滤。本质上，

这种方法不是直接对评级矩阵进行建模，而是将数据视为代表*强度的*数字用户行为的观察（例如点击次数或某人花在观看电影上的累积持续时间）。然后，这些数字与观察到的用户偏好的置信水平相关，而不是与项目的明确评级相关。然后，该模型试图找到可用于预测用户对项目的预期偏好的潜在因素。

1.2.2 缩放正则化参数

我们 `regParam` 通过用户在更新用户因素时产生的评级数或在更新产品因子时收到的产品评级数来缩小正则化参数以解决每个最小二乘问题。这种方法被命名为“ALS-WR”，并在“ Netflix 奖的大规模并行协同过滤 ”一文中进行了讨论。它 `regParam` 更少依赖于数据集的规模，因此我们可以将从采样子集中学习的最佳参数应用于完整数据集，并期望获得类似的性能。

1.2.3 冷启动策略

在使用 a 进行预测时 `ALSModel`，通常会遇到测试数据集中的用户和/或项目，这些用户和/或项目在训练模型期间不存在。这通常发生在两种情况中：

1. 在生产中，对于没有评级历史且未对模型进行过培训的新用户或项目（这是“冷启动问题”）。
2. 在交叉验证期间，数据在训练和评估集之间分配。当 Spark 中的使用简单随机拆分为 `CrossValidator` 或者 `TrainValidationSplit`，它实际上是非常普遍遇到的评估组不是在训练组用户和/或项目

默认情况下，Spark 会在模型中不存在用户和/或项目因子时指定 `NaN` 预测 `ALSModel.transform`。这在生产系统中很有用，因为它表示新用户或项目，因此系统可以决定使用某些后备作为预测。

但是，这在交叉验证期间是不合需要的，因为任何 `NaN` 预测值都将导致 `NaN` 评估度量的结果（例如在使用时 `RegressionEvaluator`）。这使得模型选择不可能。

Spark 允许用户将 `coldStartStrategy` 参数设置为“drop”，以便删除 `DataFrame` 包含 `NaN` 值的预测中的任何行。然后将根据非 `NaN` 数据计算评估度量并且该评估度量将是有效的。以下示例说明了此参数的用法。

注意：目前支持的冷启动策略是“nan”（上面提到的默认行为）和“drop”。将来可能会支持进一步的策略。

1.2.4 代码

```
package cn.apple.mltest

import org.apache.spark.ml.evaluation.RegressionEvaluator
import org.apache.spark.ml.recommendation.ALS
import org.apache.spark.sql.SparkSession
```

```
/**
  * Created by zhao-chj on 2018/7/15.
  */
case class Rating(userId: Int, movieId: Int, rating: Float, timestamp: Long)
object MovieReconmentALS {
  def parseRating(str: String): Rating = {
    val fields = str.split("::")
    assert(fields.size == 4)
    Rating(fields(0).toInt, fields(1).toInt, fields(2).toFloat, fields(3).toLong)
  }

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

    val spark: SparkSession = SparkSession.builder()
      .appName("SparkMlilb")
      .master("local[2]")
      .getOrCreate()
    spark.sparkContext.setLogLevel("WARN")

    import spark.implicits._

    val ratings =
spark.read.textFile("D:\\BigData\\Workspace\\Spark_Test\\src\\main\\scala\\cn\\a
pple\\mltest\\sample_movielens_ratings.txt")
      .map(x=>parseRating(x))
      .toDF()
    val Array(training, test) = ratings.randomSplit(Array(0.8, 0.2))

    // Build the recommendation model using ALS on the training data
    val als = new ALS()
      .setMaxIter(5)
      .setRegParam(0.01)
      .setUserCol("userId")
      .setItemCol("movieId")
      .setRatingCol("rating")

    val model = als.fit(training)
    model.setColdStartStrategy("drop")
    // Evaluate the model by computing the RMSE on the test data
    val predictions = model.transform(test)

    val evaluator = new RegressionEvaluator()
      .setMetricName("rmse")
```

```
      .setLabelCol("rating")
      .setPredictionCol("prediction")
    val rmse = evaluator.evaluate(predictions)
    println(s"Root-mean-square error = $rmse")

    // Generate top 10 movie recommendations for each user 为每个用户生成前
10 个电影推荐
    val userRecs = model.recommendForAllUsers(10)
    userRecs.show()

    // Generate top 10 user recommendations for each movie 为每部电影生成前
10 个用户推荐
    val movieRecs = model.recommendForAllItems(10)
    movieRecs.show()
    // Generate top 10 movie recommendations for a specified set of users
    val users = ratings.select(als.getUserCol).distinct().limit(3)
    val userSubsetRecs = model.recommendForUserSubset(users, 10)
    userSubsetRecs.show()
    // Generate top 10 user recommendations for a specified set of movies
    val movies = ratings.select(als.getItemCol).distinct().limit(3)
    val movieSubSetRecs = model.recommendForItemSubset(movies, 10)
    movieSubSetRecs.show()

    movieSubSetRecs.registerTempTable("t") //creatR
    spark.sql("select * from t").foreach(x=>println(x))
  }
}
```

1.3MLLIB 库补充

```
package cn.apple.mltest

import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.mllib.recommendation.ALS
import org.apache.spark.mllib.recommendation.Rating
/**
  * Created by zhao-chj on 2018/7/15.
  */
object ALSMLLIBTest {
  def main(args: Array[String]): Unit = {
```

```
    val sparkconf = new SparkConf().setAppName("ALSMLLIBTest").setMaster("local[2]")
    val sc = new SparkContext(sparkconf)
    // Load and parse the data
    val data = sc.textFile("D:\\BigData\\Workspace\\Spark_Test\\src\\main\\scala\\cn\\apple\\mltest\\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(s"Mean Squared Error = $MSE")
  }
}
```

1.4 总结

ALS 通过 ml 和 mllib 部分分别提供了基于模型推荐代码，首先在明确参数的基础上，要明确不用包调用方式的差别。

2.电商数据实战

大数据组的同学采集到用户的动态数据转化为格式化的数据，提交给算法工程师进一步建立推荐模型。

1. 字段说明

[用户 id,商品 id,用户行为类型,时间戳,该行为在这一天的次数]

（公司的现有数据是只记录某用户一天内的行为次数，没有给每次行为记录一条数据。比如用户一天内浏览了商品 3 次，只会有一条 pv 数据，且次数为 3。）

2. 输入数据

存在 UserBehaviorData.txt 文件中。

```
usfvm1223ds,1,pv,1536940800,3
uhf34sdcfe3,2,pv,1536940800,1
dsfcds2332f,3,pv,1536940800,7
vfcv4356fvf,4,pv,1536940800,1
usfvm1223ds,1,fav,1536940800,1
dvbgf909gbn,1,pv,1536940800,1
dsfcds2332f,2,fav,1536940800,1
vfcv4356fvf,3,pv,1536940800,2
bvdfv487fer,5,pv,1536940800,1
usfvm1223ds,1,buy,1536940800,1
```

3. 其他说明

1. 用户浏览次数，统计的是 pv（不确定是用 pv 还是 uv，这里代码是用的 pv）。

2. 因为 ALS 算法值接收(userId:Int, itemId:Int, rating:Float)型数据，但是我们的

userid 是 string

3. 这里用了 new StringIndexer().fit(dataFrame).transform(dataFrame)来将 string 类型的 userId 转成 Int。

4. 也可以建立一张中间表（用户 id 和 int 型 id）。

5. 这里是读取文本文件的数据，还可以用 spark sql 来读取数据。

6. 在预测的评分中，会有负分，这是因为用了乔里斯基（Cholesky）分解来解（还有一种是非负最小二乘（NNLS））。可参考：

https://endymecy.gitbooks.io/spark-ml-source-analysis/content/%E6%8E%A8%E8%8D%90/ALS.html

4. pom. xml 文件

新建 maven 工程，需要一个正确的 pom.xml 文件，写代码之前，要保证 pom 文件的正确。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.asin.als</groupId>
    <artifactId>scala-als</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <scala.version>2.11.8</scala.version>
        <hadoop.version>2.7.4</hadoop.version>
        <spark.version>2.0.2</spark.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.scala-lang</groupId>
```

```
            <artifactId>scala-library</artifactId>
            <version>${scala.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-core_2.11</artifactId>
            <version>${spark.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-client</artifactId>
            <version>${hadoop.version}</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.38</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-sql_2.11</artifactId>
            <version>2.0.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-hive_2.11</artifactId>
            <version>2.0.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-streaming_2.11</artifactId>
            <version>2.0.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-streaming-flume_2.11</artifactId>
            <version>2.0.2</version>
        </dependency>
        <!--
https://mvnrepository.com/artifact/org.apache.spark/spark-streaming-kafka-0-8 -->
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-streaming-kafka-0-8_2.11</artifactId>
            <version>2.1.0</version>
```

```
        </dependency>
        <dependency>
            <groupId>com.aliyun.emr</groupId>
            <artifactId>emr-logservice_2.11</artifactId>
            <version>1.4.1</version>
        </dependency>
        <dependency>
            <groupId>com.aliyun.emr</groupId>
            <artifactId>emr-maxcompute_2.11</artifactId>
            <version>1.4.1</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-mllib_2.11</artifactId>
            <version>2.0.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-math3</artifactId>
            <version>3.5</version>
        </dependency>
    </dependencies>
    <build>
        <sourceDirectory>src/main/scala</sourceDirectory>
        <testSourceDirectory>src/test/scala</testSourceDirectory>
        <plugins>
            <plugin>
                <groupId>net.alchim31.maven</groupId>
                <artifactId>scala-maven-plugin</artifactId>
                <version>3.2.0</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>compile</goal>
                            <goal>testCompile</goal>
                        </goals>
                        <configuration>
                            <args>
                                <arg>-dependencyfile</arg>
                                <arg>${project.build.directory}/.scala_dependencies</arg>
                            </args>
                        </configuration>
                    </execution>
                </executions>
```

```
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>2.3</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <filters>
                                <filter>
                                    <artifact>*:*</artifact>
                                    <excludes>
                                        <exclude>META-INF/*.SF</exclude>
                                        <exclude>META-INF/*.DSA</exclude>
                                        <exclude>META-INF/*.RSA</exclude>
                                    </excludes>
                                </filter>
                            </filters>
                            <transformers>
                                <transformer
implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                                    <mainClass>ScalaDemo2.scala</mainClass>
                                </transformer>
                            </transformers>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

5. ALS 代码

```scala
package com.xin

import org.apache.spark.ml.feature.StringIndexer
import org.apache.spark.mllib.recommendation.{ALS, Rating}
import org.apache.spark.sql.{DataFrame, SparkSession}
import org.apache.spark.{SparkConf, SparkContext}

import scala.collection.mutable.{ArrayBuffer, Map}

case class UserActionIndex(user: String, item: Int, action: String, timestamp: String, num: Int, index: Double)

case class UserAction(user: String, item: Int, action: String, timestamp: String, num: Int)

object ScalaDemo {

  val ACTION_PV: String = "pv"
  val ACTION_FAV: String = "fav"
  val ACTION_BUY: String = "buy"

//行为对应的分数，根据实际情况定义，这里是随便定义的
  val RATE_PV: Double = 1
  val RATE_FAV: Double = 3
  val RATE_BUY: Double = 10

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

    val conf = new SparkConf().setMaster("local[2]").setAppName("als")
    val sc = SparkContext.getOrCreate(conf)
    sc.setLogLevel("ERROR") // 控制台只打印 ERROR 级别日志

    val spark = SparkSession.builder.getOrCreate()

    //读取行为日志文件
    val data_user_action = sc.textFile("file:///D:/code/data/UserBehaviorData.txt")

    // 转成 userActionsRDD
    val userActionsRDD = data_user_action.map(_.split(',') match { case Array(user, item, action, timestamp, num) =>
      UserAction(user.toString, item.toInt, action.toString, timestamp.toString, num.toInt)
    })

    //转换字符串类型的 userId  转换前的样子：usfvm1223ds,1,pv,1536940800,3
```

```
    val dataFrame = spark.createDataFrame(userActionsRDD)
    val indexedDf: DataFrame = new
StringIndexer().setInputCol("user").setOutputCol("user_code").fit(dataFrame).transform
(dataFrame)
    // 转换后的样子：usfvm1223ds,1,pv,1536940800,3,0.0
    val actionDFRDD = indexedDf.rdd.collect() // 获取 RDD 里的内容，不然后面遍历取不到值

    println("actionDFRDD")
    actionDFRDD foreach println

    //    indexedDf.rdd.saveAsTextFile("file:///D:/code/data/out")  // 保存为文件

    //构建 map，用于存放评分。 key：userId-itemId（字符串拼接），value：评分
    var countMap: Map[String, Double] = scala.collection.mutable.Map() //新建空 map

    //获得所有用户，同时获取“用户-商品”对
    val users = actionDFRDD.map { x =>

      var userId = x(5).toString
      userId = userId.substring(0, userId.length - 2)

      //往 map 里面添加值
      var key = userId + "-" + x(1) // 用户序号-商品 ID

      var value = 0.0 // 评分
      countMap.put(key, value) // map 的长度=日志数据的条数

      userId.toInt // 以 userId 作为后面的 distinct 操作数据

    }.distinct

    println("users")
    users foreach println

    //获得所有的商品
    val items = actionDFRDD.map(x => x(1).toString.toInt).distinct

    println("items")
    items foreach println

    println("countMap")
    countMap foreach println

    //用户-商品 笛卡尔积，用于给所有“用户-商品”组合预测评分
```

```scala
val userRdd = sc.makeRDD(users)
val itemRdd = sc.makeRDD(items)

val userItems = userRdd.cartesian(itemRdd)

//获取所有值
val userItemsRel = userItems.collect()

//遍历行为，累计评分
indexedDf.rdd.collect().map { x =>

  // 将 userId 转成 int 型对应的值
  var index_temp = x(5).toString
  var index = index_temp.substring(0, index_temp.length - 2)

  var item = x(1) //map 的 key
  var action = x(2).toString //行为类型
  var num = x(4).toString.toInt //行为次数

  var key = index + "-" + item

  var rate = countMap.getOrElse(key, 0.0) //已有的分值

  //判断行为类型，乘以次数，再加上原有的分数
  if (ACTION_PV.equals(action)) {
    rate += RATE_PV * num
  }
  if (ACTION_FAV.equals(action)) {
    rate += RATE_FAV * num
  }
  if (ACTION_BUY.equals(action)) {
    rate += RATE_BUY * num
  }

  countMap.put(key, rate)
}

println("countMap")
println(countMap)

//把评分的 map 转成 Rating(user, product, rate)格式的 RDD
val ab = ArrayBuffer[Rating]()
for (a <- countMap) {  // (4-1,0.0)
  var key = a._1  // 4-1
```

```
      var value = a._2  // 0.0

      var arr = key.split("-")
      var userId = arr(0).substring(0, arr.length - 1).toInt  // 4
      var itemId = arr(1).toInt  // 1

      var myRating = new Rating(userId, itemId, value)  // 4,1,0.0
      ab += myRating
    }

    var ratingRDD = sc.makeRDD(ab)

    //模型与训练
    val (rank, iterations, lambda) = (50, 5, 0.01)
    val model = ALS.train(ratingRDD, rank, iterations, lambda)

    // 预测
    val predictions =
      model.predict(userItems).map { case Rating(user, product, rate) =>
        ((user, product), rate)
      }

    println("预测结果")
    predictions foreach println

    // 评估模型
    val ratesAndPreds = ratingRDD.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(s"Mean Squared Error = $MSE")

  }
}
```

6. 代码输出

7.代码输出

actionDFRDD

```
[usfvm1223ds,1,pv,1536940800,3,0.0]

[uhf34sdcfe3,2,pv,1536940800,1,5.0]

[dsfcds2332f,3,pv,1536940800,7,2.0]

[vfcv4356fvf,4,pv,1536940800,1,1.0]

[usfvm1223ds,1,fav,1536940800,1,0.0]

[dvbgf909gbn,1,pv,1536940800,1,4.0]

[dsfcds2332f,2,fav,1536940800,1,2.0]

[vfcv4356fvf,3,pv,1536940800,2,1.0]

[bvdfv487fer,5,pv,1536940800,1,3.0]

[usfvm1223ds,1,buy,1536940800,1,0.0]
```

users

```
0

5

2

1

4

3
```

items

```
1
```

```
2

3

4

5
```

countMap

```
(4-1,0.0)

(1-4,0.0)

(2-3,0.0)

(3-5,0.0)

(1-3,0.0)

(2-2,0.0)

(5-2,0.0)

(0-1,0.0)
```

countMap

```
Map(4-1 -> 1.0, 1-4 -> 1.0, 2-3 -> 7.0, 3-5 -> 1.0, 1-3 -> 2.0, 2-2 -> 3.0, 5-2 -> 1.0, 0-1 -> 16.0)
```

预测结果

```
((1,4),0.9814112189944704)

((1,2),0.8377005876757513)

((4,4),-6.031333549009407E-4)

((1,1),-0.12653905579369407)

((4,2),-0.029346542176744587)

((1,3),1.9999906220495713)
```

```
((4,1),0.999953169108017)

((1,5),-0.022793612338590882)

((4,3),-0.06829971098682966)

((3,4),-0.024335730305469788)

((4,5),0.008401785930320831)

((3,2),-0.33523222751490345)

((3,1),1.8090521764939655)

((0,4),-0.009650133678415052)

((3,3),-0.7787258512439097)

((0,2),-0.4695446748279134)

((3,5),0.989916688466003)

((0,1),15.999250705728272)

((5,4),0.11681138852564343)

((0,3),-1.0927953757892745)

((5,2),0.9986909552413586)

((0,5),0.1344285748851333)

((5,1),-0.8203296698520387)

((5,3),2.325605848116709)

((2,4),0.3643438094017086)

((5,5),-0.043520892539204846)

((2,2),3.003574732008076)

((2,1),-2.4651438840544175)

((2,3),6.99508362749611)

((2,5),-0.13042296358488986)
```

MSE

```
Mean Squared Error = 6.080533743173865E-5
```

排序后的结果

(0,1,15.999288742209364)
(2,3,6.999784544956694)
(2,2,2.992834174076609)
(2,1,2.616216773366991)
(5,3,2.3262092113157196)
(1,3,1.9999757313090603)
(0,3,1.0270070748265063)
(4,1,0.9999555463880853)
(5,2,0.9987206358861549)
(3,5,0.9899167018830504)
(1,4,0.9814318823467407)
(5,1,0.8786316172092582)
(0,2,0.4884032798762953)
(1,2,0.18810143415045233)
(2,4,0.08665553476647699)
(3,4,0.06506845231277508)
(4,3,0.06418794217665665)
(1,5,0.05711188450789009)
(4,2,0.030525204992268455)
(5,4,0.0228821756785125)
(4,5,-0.002684599259042535)
(4,4,-0.0036242848429082317)
(0,5,-0.04295358814468056)
(0,4,-0.05798855748653171)
(5,5,-0.1345513671654361)
(2,5,-0.4016796629824251)
(3,1,-0.6090266058733799)
(1,1,-0.7395180567704943)
(3,2,-1.0604656589392074)
(3,3,-2.3271817555020666)