Spark 实战，第 4 部分

内容

使用 Spark MLlib 做 K-means 聚类

王 龙
2015 年 9 月 24 日发布

引言

提起机器学习 (Machine Learning)，相信很多计算机从业者都会对这个技术方向感到兴奋数据却是一项复杂的工作，需要充足的知识储备，如概率论，数理统计，数值逼近，最优人类一样的学习能力和模仿能力，这也是实现人工智能的核心思想和方法。传统的机器学只能在少量数据上使用，随着 HDFS(Hadoop Distributed File System) 等分布式文件系统而由于 MapReduce 自身的限制，使得使用 MapReduce 来实现分布式机器学习算法非常机器学习算法参数学习的过程都是迭代计算的，即本次计算的结果要作为下一次迭代的输MapReduce，我们只能把中间结果存储磁盘，然后在下一次计算的时候从新读取，这对于频。Spark 立足于内存计算，天然的适应于迭代式计算，相信对于这点，读者通过前面几即便这样，对于普通开发者来说，实现一个分布式机器学习算法仍然是一件极具挑战的事机器学习变得更加简单，它提供了常用机器学习算法的分布式实现，开发者只需要有 Sp以及方法相关参数的含义，就可以轻松的通过调用相应的 API 来实现基于海量数据的机器指标提取，调节参数并优化学习过程，这依然需要有足够的行业知识和数据敏感度，这往读者介绍如何使用 MLlib 机器学习库提供的 K-means 算法做聚类分析，这是一个有意义发意义。

Spark 机器学习库简介

Spark 机器学习库提供了常用机器学习算法的实现，包括聚类，分类，回归，协同过滤，

做机器学习工作，可以说是非常的简单，通常只需要在对原始数据进行处理后，然后直接选择合适的算法，高效准确地对数据进行分析，您可能还需要深入了解下算法原理，以及义。

需要提及的是，Spark 机器学习库从 1.2 版本以后被分为两个包，分别是：

- spark.mllib

Spark MLlib 历史比较长了，1.0 以前的版本中已经包含了，提供的算法实现都是基于原始容易上手。如果您已经有机器学习方面的经验，那么您只需要熟悉下 MLlib 的 API 就可以提供的工具构建完整并且复杂的机器学习流水线是比较困难的。

- spark.ml

Spark ML Pipeline 从 Spark1.2 版本开始，目前已经从 Alpha 阶段毕业，成为可用并且较弥补了原始 MLlib 库的不足，向用户提供了一个基于 DataFrame 的机器学习工作流式 API以很方便的把数据处理，特征转换，正则化，以及多个机器学习算法联合起来，构建一个种新的方式给我们提供了更灵活的方法，而且这也更符合机器学习过程的特点。

从官方文档来看，Spark ML Pipeline 虽然是被推荐的机器学习方式，但是并不会在短期内包含了丰富稳定的算法实现，并且部分 ML Pipeline 实现基于 MLlib。而且就笔者看来，成一个流水线，有时候原始数据格式整齐且完整，而且使用单一的算法就能实现目标，我单且容易理解的方式才是正确的选择。

本文基于 Spark 1.5，向读者展示使用 MLlib API 进行聚类分析的过程。读者将会发现，比较简单的，相信本文可以使读者建立起信心并掌握基本方法，以便在后续的学习和工作

K-means 聚类算法原理

聚类分析是一个无监督学习 (Unsupervised Learning) 过程, 一般是用来对数据对象按照其分群，欺诈检测，图像分析等领域。K-means 应该是最有名并且最经常使用的聚类算法良好，有着广泛的使用。

和诸多机器学习算法一样，K-means 算法也是一个迭代式的算法，其主要步骤如下:

- 第一步，选择 K 个点作为初始聚类中心。
- 第二步，计算其余所有点到聚类中心的距离，并把每个点划分到离它最近的聚类中心

般有多个函数可以选择，最常用的是欧几里得距离 (Euclidean Distance), 也叫欧式距

$$D(C,X) = \sqrt{\sum_{i=1}^{n}(c_i - x_i)^2}$$

其中 C 代表中心点，X 代表任意一个非中心点。

- 第三步，重新计算每个聚类中所有点的平均值，并将其作为新的聚类中心点。
- 最后，重复 (二)，(三) 步的过程，直至聚类中心不再发生改变，或者算法达到预定的设定的阀值。

在实际应用中，K-means 算法有两个不得不面对并且克服的问题。

1. 聚类个数 K 的选择。K 的选择是一个比较有学问和讲究的步骤，我们会在后文专门描
2. 初始聚类中心点的选择。选择不同的聚类中心可能导致聚类结果的差异。

Spark MLlib K-means 算法的实现在初始聚类点的选择上，借鉴了一个叫 K-means||的类初始点选择上遵循一个基本原则: 初始聚类中心点相互之间的距离应该尽可能的远。基本

- 第一步，从数据集 X 中随机选择一个点作为第一个初始点。
- 第二步，计算数据集中所有点与最新选择的中心点的距离 D(x)。
- 第三步，选择下一个中心点，使得

$$P(x) = \frac{D(x)^2}{\sum_{x \in X} D(x)^2}$$

  最大。
- 第四部，重复 (二),(三) 步过程，直到 K 个初始点选择完成。

MLlib 的 K-means 实现

Spark MLlib 中 K-means 算法的实现类 (KMeans.scala) 具有以下参数，具体如下。

图 1. MLlib K-means 算法实现类预览

```
class KMeans private (
    private var k: Int,
    private var maxIterations: Int,
    private var runs: Int,
    private var initializationMode: String,
    private var initializationSteps: Int,
    private var epsilon: Double,
    private var seed: Long) extends Serializable with Logging
```

通过下面默认构造函数，我们可以看到这些可调参数具有以下初始值。

图 2. MLlib K-means 算法参数初始值

```
/**
 * Constructs a KMeans instance with default parameters: {k: 2, maxIterations: 20, runs: 1,
 * initializationMode: "k-means||", initializationSteps: 5, epsilon: 1e-4, seed: random}.
 */
@Since("0.8.0")
def this() = this(2, 20, 1, KMeans.K_MEANS_PARALLEL, 5, 1e-4, Utils.random.nextLong())
```

参数的含义解释如下：

- *k* 表示期望的聚类的个数。
- *maxInterations* 表示方法单次运行最大的迭代次数。
- *runs* 表示算法被运行的次数。K-means 算法不保证能返回全局最优的聚类结果，所以
  法，有助于返回最佳聚类结果。
- *initializationMode* 表示初始聚类中心点的选择方式, 目前支持随机选择或者 K-means
- *initializationSteps*表示 K-means||方法中的部数。
- *epsilon* 表示 K-means 算法迭代收敛的阀值。
- *seed* 表示集群初始化时的随机种子。

通常应用时，我们都会先调用 KMeans.train 方法对数据集进行聚类训练，这个方法会返
以使用 KMeansModel.predict 方法对新的数据点进行所属聚类的预测，这是非常实用的

KMeans.train 方法有很多重载方法，这里我们选择参数最全的一个展示。

图 3. KMeans.train 方法预览

```
@Since("1.3.0")
def train(
    data: RDD[Vector],
    k: Int,
    maxIterations: Int,
    runs: Int,
    initializationMode: String,
    seed: Long): KMeansModel = {
  new KMeans().setK(k)
    .setMaxIterations(maxIterations)
    .setRuns(runs)
    .setInitializationMode(initializationMode)
    .setSeed(seed)
    .run(data)
}
```

KMeansModel.predict 方法接受不同的参数，可以是向量，或者 RDD，返回是入参所属

图 4. KMeansModel.predict 方法预览

```
/**
 * Returns the cluster index that a given point belongs to.
 */
@Since("0.8.0")
def predict(point: Vector): Int = {
  KMeans.findClosest(clusterCentersWithNorm, new VectorWithNorm(point))._1
}

/**
 * Maps given points to their cluster indices.
 */
@Since("1.0.0")
def predict(points: RDD[Vector]): RDD[Int] = {
  val centersWithNorm = clusterCentersWithNorm
  val bcCentersWithNorm = points.context.broadcast(centersWithNorm)
  points.map(p => KMeans.findClosest(bcCentersWithNorm.value, new VectorWithNorm(p))._1)
}
```

聚类测试数据集简介

在本文中，我们所用到目标数据集是来自 UCI Machine Learning Repository 的 Wholesa
机器学习测试数据的下载中心站点，里面包含了适用于做聚类，分群，回归等各种机器

Wholesale customer Data Set 是引用某批发经销商的客户在各种类别产品上的年消费数
式转化成了两个文本文件，分别是训练用数据和测试用数据。

图 5. 客户消费数据格式预览

Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
2	3	12669	9656	7561	214	2674	1338
2	3	7057	9810	9568	1762	3293	1776
2	3	6353	8808	7684	2405	3516	7844
1	3	13265	1196	4221	6404	507	1788
2	3	22615	5410	7198	3915	1777	5185
2	3	9413	8259	5126	666	1795	1451
2	3	12126	3199	6975	480	3140	545
2	3	7579	4956	9426	1669	3321	2566
1	3	5963	3648	6192	425	1716	750
2	3	6006	11093	18881	1159	7425	2098
2	3	3366	5403	12974	4400	5977	1744
2	3	13146	1124	4523	1420	549	497
2	3	31714	12319	11757	287	3881	2931
2	3	21217	6208	14982	3095	6707	602
2	3	24653	9465	12091	294	5058	2168
1	3	10253	1114	3821	397	964	412
2	3	1020	8816	12121	134	4508	1080

读者可以从标题清楚的看到每一列代表的含义，当然读者也可以到 UCI 网站上去找到关于
可以自由获取并使用，但是我们还是在此声明，该数据集的版权属 UCI 以及其原始提供组

案例分析和编码实现

本例中，我们将根据目标客户的消费数据，将每一列视为一个特征指标，对数据集进行聚

清单 1. 聚类分析实现类源码

```
1  import org.apache.spark.{SparkContext, SparkConf}
2  import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}
3  import org.apache.spark.mllib.linalg.Vectors
4  object KMeansClustering {<br> def main (args: Array[String]) {<br> if (a
5      println("Usage:KMeansClustering trainingDataFilePath testDataFilePat
6      numIterations runTimes")<br> sys.exit(1)<br> }<br><br> val conf = ne
7      SparkConf().setAppName("Spark MLlib Exercise:K-Means Clustering")<br
8     /**<br> *Channel Region Fresh Milk Grocery Frozen Detergents_Paper De
9       12669 9656 7561 214 2674 1338<br> * 2 3 7057 9810 9568 1762 3293 17
10      7684 2405 3516 7844<br> */<br>
11     val rawTrainingData = sc.textFile(args(0))<br> val parsedTrainingDat
12     rawTrainingData.filter(!isColumnNameLine(_)).map(line => {<br>
13     Vectors.dense(line.split("\t").map(_.trim).filter(!"".equals(_)).map
14     // Cluster the data into two classes using KMeans<br>
15     val numClusters = args(2).toInt<br> val numIterations = args(3).toIn
16     args(4).toInt<br> var clusterIndex:Int = 0<br> val clusters:KMeansMo
17     KMeans.train(parsedTrainingData, numClusters, numIterations,runTimes
18     println("Cluster Number:" + clusters.clusterCenters.length)<br>
19     println("Cluster Centers Information Overview:")<br> clusters.cluste
20     x => {<br>
21     println("Center Point of Cluster " + clusterIndex + ":")<br>
22     println(x)<br> clusterIndex += 1<br> })<br>
23     //begin to check which cluster each test data belongs to based on th
24     val rawTestData = sc.textFile(args(1))<br> val parsedTestData = rawT
25     {<br>
26     Vectors.dense(line.split("\t").map(_.trim).filter(!"".equals(_)).map
27     })<br> parsedTestData.collect().foreach(testDataLine => {<br> val pr
28     Int = clusters.predict(testDataLine)<br>
29     println("The data " + testDataLine.toString + " belongs to cluster
30     predictedClusterIndex)<br> })<br>
```

```
31      println("Spark MLlib K-means clustering test finished.")<br> }<br><
32      isColumnNameLine(line:String):Boolean = {<br> if (line != null &&
33      line.contains("Channel")) true<br> else false<br> }
```

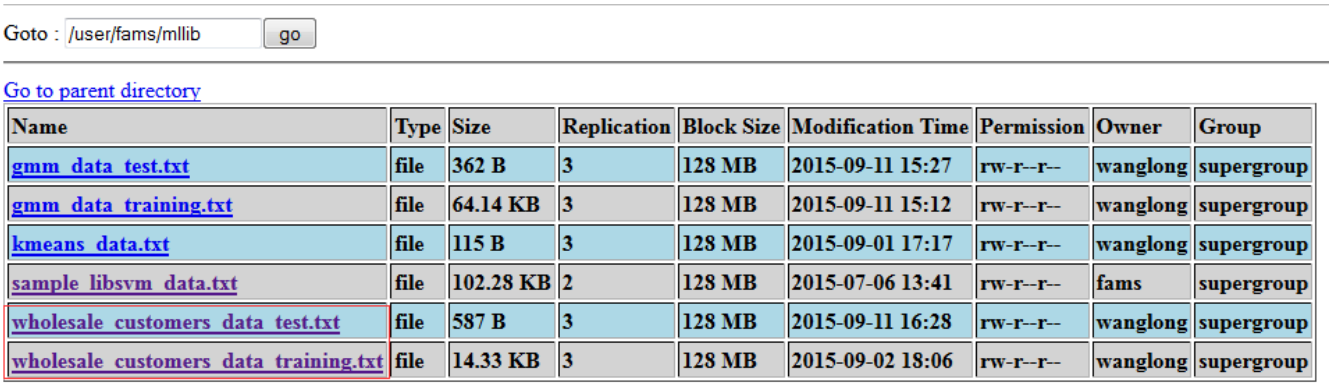
该示例程序接受五个入参，分别是

- 训练数据集文件路径
- 测试数据集文件路径
- 聚类的个数
- K-means 算法的迭代次数
- K-means 算法 run 的次数

运行示例程序

和本系列其他文章一样，我们依然选择使用 HDFS 存储数据文件。运行程序之前，我们需
到 HDFS。

图 6. 测试数据的 HDFS 目录


Goto : /user/fams/mllib go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
gmm_data_test.txt	file	362 B	3	128 MB	2015-09-11 15:27	rw-r--r--	wanglong	supergroup
gmm_data_training.txt	file	64.14 KB	3	128 MB	2015-09-11 15:12	rw-r--r--	wanglong	supergroup
kmeans_data.txt	file	115 B	3	128 MB	2015-09-01 17:17	rw-r--r--	wanglong	supergroup
sample_libsvm_data.txt	file	102.28 KB	2	128 MB	2015-07-06 13:41	rw-r--r--	fams	supergroup
wholesale_customers_data_test.txt	file	587 B	3	128 MB	2015-09-11 16:28	rw-r--r--	wanglong	supergroup
wholesale_customers_data_training.txt	file	14.33 KB	3	128 MB	2015-09-02 18:06	rw-r--r--	wanglong	supergroup

清单 2. 示例程序运行命令

```
./spark-submit --class com.ibm.spark.exercise.mllib.KMeansClustering \
 --master spark://<spark_master_node_ip>:7077 \
 --num-executors 6 \
--driver-memory 3g \
--executor-memory 512m \
--total-executor-cores 6 \
 /home/fams/spark_exercise-1.0.jar \
 hdfs://<hdfs_namenode_ip>:9000/user/fams/mllib/wholesale_customers_data
 hdfs://<hdfs_namenode_ip>:9000/user/fams/mllib/wholesale_customers_data
 8 30 3
```

图 7. K-means 聚类示例程序运行结果

```
Cluster Number:8
Cluster Centers Information Overview:
Center Point of Cluster 0:
[1.21,2.52,21048.47,3774.7200000000003,5067.36,3777.39,1136.94,1673.4]
Center Point of Cluster 1:
[1.0,3.0,112151.0,29627.0,18148.0,16745.0,4948.0,8550.0]
Center Point of Cluster 2:
[2.0,2.3125,8964.0625,20052.375,32380.1875,2185.9375,15713.4375,3184.875]
Center Point of Cluster 3:
[1.1287128712871288,2.5198019801980202,6174.608910891089,3044.792079207921,3564.9059405940593,2546.267326732673,969.3811881188119,965.7970297029703]
Center Point of Cluster 4:
[2.0,3.0,29862.5,53080.75,60015.75,3262.25,27942.25,3082.25]
Center Point of Cluster 5:
[1.0952380952380951,2.7142857142857144,46865.142857142855,3534.2857142857144,4976.571428571428,5285.190476190476,834.7619047619047,2137.7619047619046]
Center Point of Cluster 6:
[1.8674698795180724,2.5421686746987953,4095.686746987952,9714.674698795181,15269.819277108434,1393.3855421686749,6622.530120481928,1453.9518072289156]
Center Point of Cluster 7:
[1.0,2.6666666666666665,26959.333333333332,21274.666666666664,11952.666666666666,44137.33333333333,527.3333333333333,18750.0]

The data [1.0,3.0,3097.0,4230.0,16483.0,575.0,241.0,2080.0] belongs to cluster 6
The data [1.0,3.0,8533.0,5506.0,5160.0,13486.0,1377.0,1498.0] belongs to cluster 3
The data [1.0,3.0,21117.0,1162.0,4754.0,269.0,1328.0,395.0] belongs to cluster 0
The data [1.0,3.0,1982.0,3218.0,1493.0,1541.0,356.0,1449.0] belongs to cluster 3
The data [1.0,3.0,16731.0,3922.0,7994.0,688.0,2371.0,838.0] belongs to cluster 0
The data [1.0,3.0,29703.0,12051.0,16027.0,13135.0,182.0,2204.0] belongs to cluster 0
The data [1.0,3.0,39228.0,1431.0,764.0,4510.0,93.0,2346.0] belongs to cluster 5
The data [2.0,3.0,14531.0,15488.0,30243.0,437.0,14841.0,1867.0] belongs to cluster 2
The data [1.0,3.0,10290.0,1981.0,2232.0,1038.0,168.0,2125.0] belongs to cluster 3
The data [1.0,3.0,2787.0,1698.0,2510.0,65.0,477.0,52.0] belongs to cluster 3
The data [2.0,3.0,24653.0,9465.0,12091.0,294.0,5058.0,2168.0] belongs to cluster 0
The data [1.0,3.0,10253.0,1114.0,3821.0,397.0,964.0,412.0] belongs to cluster 3
The data [2.0,3.0,1020.0,8816.0,12121.0,134.0,4508.0,1080.0] belongs to cluster 6
The data [1.0,3.0,5876.0,6157.0,2933.0,839.0,370.0,4478.0] belongs to cluster 3
The data [2.0,3.0,18601.0,6327.0,10099.0,2205.0,2767.0,3181.0] belongs to cluster 0
The data [1.0,3.0,7780.0,2495.0,9464.0,669.0,2518.0,501.0] belongs to cluster 3
The data [2.0,3.0,17546.0,4519.0,4602.0,1066.0,2259.0,2124.0] belongs to cluster 0
Spark MLlib K-means clustering test finished.
```

如何选择 K

前面提到 K 的选择是 K-means 算法的关键，Spark MLlib 在 KMeansModel 类里提供了 c
数据点到其最近的中心点的平方和来评估聚类的效果。一般来说，同样的迭代次数和算法
越好。但是在实际情况下，我们还要考虑到聚类结果的可解释性，不能一味的选择使 co

清单 3. K 选择示例代码片段

```
1  val ks:Array[Int] = Array(3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
2  ks.foreach(cluster => {
3   val model:KMeansModel = KMeans.train(parsedTrainingData, cluster,30,1)
4   val ssd = model.computeCost(parsedTrainingData)
5   println("sum of squared distances of points to their nearest center whe
6  })
```

图 8. K 选择示例程序运行结果

```
sum of squared distances of points to their nearest center when k=3 -> 8.041528713958105E10
sum of squared distances of points to their nearest center when k=4 -> 6.399194766877159E10
sum of squared distances of points to their nearest center when k=5 -> 5.19118107261774E10
sum of squared distances of points to their nearest center when k=6 -> 4.677162566965356E10
sum of squared distances of points to their nearest center when k=7 -> 4.092327309195996E10
sum of squared distances of points to their nearest center when k=8 -> 3.524987845177904E10
sum of squared distances of points to their nearest center when k=9 -> 3.5591490710820656E10
sum of squared distances of points to their nearest center when k=10 -> 2.9344365153578773E10
sum of squared distances of points to their nearest center when k=11 -> 2.9141246577663395E10
sum of squared distances of points to their nearest center when k=12 -> 2.783892279441648E10
sum of squared distances of points to their nearest center when k=13 -> 2.487433836743413E10
sum of squared distances of points to their nearest center when k=14 -> 2.3794284805679573E10
sum of squared distances of points to their nearest center when k=15 -> 2.1595034146423157E10
sum of squared distances of points to their nearest center when k=16 -> 2.032208546757111E10
sum of squared distances of points to their nearest center when k=17 -> 1.9285998648052174E10
sum of squared distances of points to their nearest center when k=18 -> 2.0547990749057438E10
sum of squared distances of points to their nearest center when k=19 -> 1.6395225248852257E10
sum of squared distances of points to their nearest center when k=20 -> 1.709278950973962E10
```

从上图的运行结果可以看到，当 K=9 时，cost 值有波动，但是后面又逐渐减小了，所以当然可以多跑几次，找一个稳定的 K 值。理论上 K 的值越大，聚类的 cost 越小，极限情 cost 是 0，但是显然这不是一个具有实际意义的聚类结果。

结束语

通过本文的学习，读者已经初步了解了 Spark 的机器学习库，并且掌握了 K-means 算法构建自己的机器学习应用。机器学习应用的构建是一个复杂的过程，我们通常还需要对数清洗等，然后才能利用算法来分析数据。Spark MLlib 区别于传统的机器学习工具，不仅的是 Spark 在处理大数据上的高效以及在迭代计算时的独特优势。虽然本文所采用的测试场景，但是对于掌握基本原理已经足够，并且如果读者拥有更大的数据集就可以轻松的将景下，因为 Spark MLlib 的编程模型都是一致的，无非是数据读取和处理的方式略有不同趣的知识，相信这对读者今后深入学习是有帮助的。另外，读者在阅读本文的过程中，如赐教，在文末留言，共同交流学习，谢谢。

相关主题

- 参考 Spark MLlib 官方网页，查看关于 Spark MLlib 的基本介绍。
- 查看 WIKIPEDIA K-means++ 介绍，了解更多 K-means 算法的原理。
- developerWorks 开源技术主题：查找丰富的操作信息、工具和项目更新，帮助您掌

评论

添加或订阅评论，请先登录或注册。

有新评论时提醒我

developerWorks

站点反馈

我要投稿

投稿指南

报告滥用

第三方提示

关注微博

加入

ISV 资源 (英语)

选择语言

English

中文

日本語

Русский

Português (Brasil)

Español

한글

技术文档库

dW 中国时事通讯

博客

活动

社区

开发者中心

视频

订阅源

软件下载

Code patterns