Spark1.4中DataFrame功能加强,新增科学和数学函数

社区在Spark 1.3中开始引入了DataFrames，使得Apache Spark更加容易被使用。受R和Python中的data frames激发，Spark中的DataFrames提供了一些API，这些API在外部看起来像是操作单机的数据一样，而数据科学家对这些API非常地熟悉。统计是日常数据科学的一个重要组成部分。在即将发布的Spark 1.4中改进支持统计函数和数学函数（statistical and mathematical functions）。

这篇文章中将介绍一些非常重要的函数，包括：
1、随机数据生成(Random data generation)；
2、总结和描述性统计(Summary and descriptive statistics)；
3、样本协方差和相关性(Sample covariance and correlation)；
4、交叉分类汇总表（又称列联表）(Cross tabulation)；
5、频繁项(Frequent items)；
6、数学函数(Mathematical functions)。

下面的例子全部是使用Python语言实现，在Scala和Java中存在类似的API。


微信扫一扫，加关注
即可及时了解Spark、Hadoop或者Hbase等相关的文章
欢迎关注微信公共帐号：iteblog_hadoop

过往记忆博客(http://www.iteblog.com)专注于Hadoop、Spark、Flume、Hbase等技术的博客，欢迎关注。

Hadoop、Hive、Hbase、Flume等交流群：138615359和149892483

如果想及时了解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

一、随机数据生成(Random data generation)

随机数据生成在测试现有的算法和实现随机算法中非常重要，比如随机投影。在sql.functions函数里面提供了生成包含i.i.uniform(rand)和标准的normal(randn)。

```
In [1]: from pyspark.sql.functions import rand, randn
In [2]: # Create a DataFrame with one int column and 10 rows.
In [3]: df = sqlContext.range(0, 10)
```

```
In [4]: df.show()
+--+
|id|
+--+
| 0|
| 1|
| 2|
| 3|
| 4|
| 5|
| 6|
| 7|
| 8|
| 9|
+--+

In [4]: # Generate two other columns using uniform distribution and normal distribution.
In [5]: df.select("id", rand(seed=10).alias("uniform"), randn(seed=27).alias("normal")).show()
+--+------------------+--------------------+
|id|           uniform|              normal|
+--+------------------+--------------------+
| 0| 0.7224977951905031| -0.1875348803463305|
| 1| 0.2953174992603351|-0.26525647952450265|
| 2| 0.4536856090041318| -0.7195024130068081|
| 3| 0.9970412477032209|  0.5181478766595276|
| 4|0.19657711634539565|  0.7316273979766378|
| 5|0.48533720635534006| 0.07724879367590629|
| 6| 0.7369825278894753| -0.5462256961278941|
| 7| 0.5241113627472694| -0.2542275002421211|
| 8| 0.2977697066654349| -0.5752237580095868|
| 9| 0.5060159582230856|  1.0900096472044518|
+--+------------------+--------------------+
```

二、总结和描述性统计(Summary and descriptive statistics)

我们在导入数据之后的第一个操作是想获取一些数据，来看看他到底是不是我们所要的。对于数字列，了解这些数据的描述性统计可以帮助我们理解我们数据的分布。describe函数返回的是一个DataFrame，而这个DataFrame中包含了每个数字列的很多信息，比如不为空的实体总数、平均值、标准差以及最大最小值。

```
In [1]: from pyspark.sql.functions import rand, randn
In [2]: # A slightly different way to generate the two random columns
In [3]: df = sqlContext.range(0, 10).withColumn('uniform', rand(seed=10)).withColumn('normal'
```

, randn(seed=27))

```
In [4]: df.describe().show()
+-------+-----------------+------------------+-------------------+
|summary|               id|           uniform|             normal|
+-------+-----------------+------------------+-------------------+
|  count|               10|                10|                 10|
|   mean|              4.5| 0.5215336029384192|-0.01309370117407197|
| stddev|2.8722813232690143|  0.229328162820653|  0.5756058014772729|
|    min|                0|0.19657711634539565| -0.7195024130068081|
|    max|                9| 0.9970412477032209|  1.0900096472044518|
+-------+-----------------+------------------+-------------------+
```

如果返回的DataFrame含有大量的列，你可以返回其中的一部分列：

```
In [4]: df.describe('uniform', 'normal').show()
+-------+------------------+-------------------+
|summary|           uniform|             normal|
+-------+------------------+-------------------+
|  count|                10|                 10|
|   mean| 0.5215336029384192|-0.01309370117407197|
| stddev|  0.229328162820653|  0.5756058014772729|
|    min|0.19657711634539565| -0.7195024130068081|
|    max| 0.9970412477032209|  1.0900096472044518|
+-------+------------------+-------------------+
```

当然，虽然describe在那些快速探索性数据分析中可以很好的工作，你还可以控制描述性统计的展示以及那些使用DataFrame中简单选择的列(这句话好别扭，请看英文you can also control the list of descriptive statistics and the columns they apply to using the normal select on a DataFrame:)

```
In [5]: from pyspark.sql.functions import mean, min, max
In [6]: df.select([mean('uniform'), min('uniform'), max('uniform')]).show()
+-----------------+------------------+-----------------+
|     AVG(uniform)|      MIN(uniform)|     MAX(uniform)|
+-----------------+------------------+-----------------+
|0.5215336029384192|0.19657711634539565|0.9970412477032209|
+-----------------+------------------+-----------------+
```

三、样本协方差和相关性(Sample covariance and correlation)

协方差表示的是两个变量的总体的误差。正数意味着其中一个增加，另外一个也有增加的趋势；而负数意味着其中一个数增加，另外一个有降低的趋势。DataFrame两列中的样本协方差计算可以如下：

```
In [1]: from pyspark.sql.functions import rand
In [2]: df = sqlContext.range(0, 10).withColumn('rand1', rand(seed=10)).withColumn('rand2', rand(seed=27))

In [3]: df.stat.cov('rand1', 'rand2')
Out[3]: 0.009908130446217347

In [4]: df.stat.cov('id', 'id')
Out[4]: 9.166666666666666
```

正如你从上面看到的，两个随机生成的列之间的协方差接近零；而id列和它自己的协方差非常大。

协方差的值为9.17可能很难解释，而相关是协方差的归一化度量，这个相对更好理解，因为它提供了两个随机变量之间的统计相关性的定量测量。

```
In [5]: df.stat.corr('rand1', 'rand2')
Out[5]: 0.14938694513735398

In [6]: df.stat.corr('id', 'id')
Out[6]: 1.0
```

在上面的例子中，ID那列完全与相关本身；而两个随机生成的列之间的相关性非常低。

四、交叉分类汇总表（又称列联表）(Cross tabulation)

如果同时按几个变量或特征，把数据分类列表时，这样的统计表叫作交叉分类汇总表，其主要用来检验两个变量之间是否存在关系，或者说是否独立。在Spark 1.4中，我们可以计算DataFrame中两列之间的交叉分类汇总表，以便获取计算的两列中不同对的数量，下面是关于如何使用交叉表来获取列联表的例子

```
In [1]: # Create a DataFrame with two columns (name, item)
In [2]: names = ["Alice", "Bob", "Mike"]
```

```
In [3]: items = ["milk", "bread", "butter", "apples", "oranges"]
In [4]: df = sqlContext.createDataFrame([(names[i % 3], items[i % 5]) for i in range(100)], ["nam
e", "item"])

In [5]: # Take a look at the first 10 rows.
In [6]: df.show(10)
+-----+-------+
| name|   item|
+-----+-------+
|Alice|   milk|
|  Bob|  bread|
| Mike| butter|
|Alice| apples|
|  Bob|oranges|
| Mike|   milk|
|Alice|  bread|
|  Bob| butter|
| Mike| apples|
|Alice|oranges|
+-----+-------+

In [7]: df.stat.crosstab("name", "item").show()
+---------+----+-----+------+------+-------+
|name_item|milk|bread|apples|butter|oranges|
+---------+----+-----+------+------+-------+
|      Bob|   6|    7|     7|     6|      7|
|     Mike|   7|    6|     7|     7|      6|
|    Alice|   7|    7|     6|     7|      7|
+---------+----+-----+------+------+-------+
```

我们需要记住，列的基数不能太大。也就是说，name和item
distinct之后的数量不能过多。试想，如果item
distinct之后的数量为10亿，那么你如何在屏幕上显示这个表？？

五、频繁项(Frequent items)

了解列中那些频繁出现的item对于我们了解数据集非常重要。在Spark
1.4中，我们可以通过使用DataFrames来发现列中的频繁项，

```
In [1]: df = sqlContext.createDataFrame([(1, 2, 3) if i % 2 == 0 else (i, 2 * i, i % 4) for i in range(1
00)], ["a", "b", "c"])
```

```
In [2]: df.show(10)
+-+--+-+
|a| b|c|
+-+--+-+
|1| 2|3|
|1| 2|1|
|1| 2|3|
|3| 6|3|
|1| 2|3|
|5|10|1|
|1| 2|3|
|7|14|3|
|1| 2|3|
|9|18|1|
+-+--+-+

In [3]: freq = df.stat.freqItems(["a", "b", "c"], 0.4)
```

对应上面的DataFrame，下面的代码可以计算出每列中出现40%的频繁项

```
In [4]: freq.collect()[0]
Out[4]: Row(a_freqItems=[11, 1], b_freqItems=[2, 22], c_freqItems=[1, 3])
```

正如你所看到的，11和1是列a的频繁值。同样，你也可以获取到列组合的频繁项，我们可以通过struct函数来创建列组合

```
In [5]: from pyspark.sql.functions import struct

In [6]: freq = df.withColumn('ab', struct('a', 'b')).stat.freqItems(['ab'], 0.4)

In [7]: freq.collect()[0]
Out[7]: Row(ab_freqItems=[Row(a=11, b=22), Row(a=1, b=2)])
```

对于上面的例子来说，“a=11 and b=22”和“a=1 and b=2”的组合频繁出现在数据集中。注意“a=11 and b=22”是为假阳性。

六、数学函数(Mathematical functions)

Spark 1.4中增加了一系列的数学函数，用户可以自如地将这些操作应用到他们列。我可以在这里看到所有的数学函数。输入必须是一个列函数，并且这个列函数只能输入一个参数，比如cos, sin, floor, ceil。对于那些需要输入两个参数的列函数，比如pow, hypot，我们可以输入两列或者列的组合。

```
In [1]: from pyspark.sql.functions import *
In [2]: df = sqlContext.range(0, 10).withColumn('uniform', rand(seed=10) * 3.14)

In [3]: # you can reference a column or supply the column name
In [4]: df.select(
   ...:   'uniform',
   ...:   toDegrees('uniform'),
   ...:   (pow(cos(df['uniform']), 2) + pow(sin(df.uniform), 2)). ₩
   ...:     alias("cos^2 + sin^2")).show()

+------------------+-----------------+-----------------+
|           uniform|  DEGREES(uniform)|     cos^2 + sin^2|
+------------------+-----------------+-----------------+
| 0.7224977951905031| 41.39607437192317|              1.0|
| 0.3312021111290707|18.976483133518624|0.9999999999999999|
| 0.2953174992603351|16.920446323975014|              1.0|
|0.018326130186194667| 1.050009914476252|0.9999999999999999|
| 0.3163135293051941|18.123430232075304|              1.0|
| 0.4536856090041318| 25.99427062175921|              1.0|
|  0.873869321369476| 50.06902396043238|0.9999999999999999|
| 0.9970412477032209| 57.12625549385224|              1.0|
| 0.19657711634539565| 11.26303911544332|1.0000000000000002|
| 0.9632338825504894| 55.18923615414307|              1.0|
+------------------+-----------------+-----------------+
```

本文提到的所有函数将在Spark 1.4中可用，并且支持Python、Scala和Java调用。Spark 1.4将在近期发布。如果你等不及了，你可以到https://github.com/apache/spark/tree/branch-1.4里面下载。

本文翻译自：https://databricks.com/blog/2015/06/02/statistical-and-mathematical-functions-with-dataframes-in-spark.html，如果上面翻译不通顺，或者不对请阅读原文。

本博客文章除特别声明，全部都是原创！
原创文章版权归过往记忆大数据（过往记忆）所有，未经许可不得转载。
本文链接:【】（）