Hands-On Exercise: Implement an Iterative Algorithm with Apache Spark

Files and Data Used in This Exercise:

Exercise directory:	`$DEVSH/exercises/spark-iterative`
Data files (HDFS):	`/loudacre/devicestatus_etl/*`
Stubs:	`KMeansCoords.pyspark`
	`KMeansCoords.scalaspark`

In this exercise, you will practice implementing iterative algorithms in Spark by calculating k-means for a set of points.

Reviewing the Data

1. If you completed the bonus section of the "Process Data Files with Apache Spark" exercise, you used Spark to extract the date, maker, device ID, latitude and longitude from the `devicestatus.txt` data file, and store the results in the HDFS directory `/loudacre/devicestatus_etl`.

   If you did not complete that bonus exercise, upload the solution file from the local filesystem to HDFS now. (If you have run the course catch-up script, this is already done for you.)

   ```
   $ hdfs dfs -put $DEVDATA/static_data/devicestatus_etl \
     /loudacre/
   ```

2. Examine the data in the dataset. Note that the latitude and longitude are the 4th and 5th fields, respectively, as shown in the sample data below:

```
2014-03-15:10:10:20,Sorrento,8cc3b47e-bd01-4482-b500-
  28f2342679af,33.6894754264,-117.543308253
2014-03-15:10:10:20,MeeToo,ef8c7564-0a1a-4650-a655-
  c8bbd5f8f943,37.4321088904,-121.485029632
```

Calculating k-means for Device Location

3. Start with the provided `KMeansCoords` stub file, which contains the following convenience functions used in calculating k-means:

- `closestPoint`: given a (latitude/longitude) point and an array of current center points, returns the index in the array of the center closest to the given point
- `addPoints`: given two points, return a point which is the sum of the two points—that is, `(x1+x2, y1+y2)`
- `distanceSquared`: given two points, returns the squared distance of the two—this is a common calculation required in graph analysis

Note that the stub code sets the variable K equal to `5`—this is the number of means to calculate.

4. The stub code also sets the variable `convergeDist`. This will be used to decide when the k-means calculation is done—when the amount the locations of the means changes between iterations is less than `convergeDist`. A "perfect" solution would be 0; this number represents a "good enough" solution. For this exercise, use a value of `0.1`.

5. Parse the input file—which is delimited by commas—into `(latitude,longitude)` pairs (the 4th and 5th fields in each line). Only include known locations—that is, filter out `(0,0)` locations. Be sure to persist the resulting RDD because you will access it each time through the iteration.

6. Create a K-length array called `kPoints` by taking a random sample of K location points from the RDD as starting means (center points). For example, in Python:

```
data.takeSample(False, K, 42)
```

Or in Scala:

```
data.takeSample(false, K, 42)
```

7. Iteratively calculate a new set of K means until the total distance between the means calculated for this iteration and the last is smaller than `convergeDist`. For each iteration:

    a. For each coordinate point, use the provided `closestPoint` function to map that point to the index in the `kPoints` array of the location closest to that point. The resulting RDD should be keyed by the index, and the value should be the pair: `(point, 1)`. (The value `1` will later be used to count the number of points closest to a given mean.) For example:

`(1, ((37.43210, -121.48502), 1))`
`(4, ((33.11310, -111.33201), 1))`
`(0, ((39.36351, -119.40003), 1))`
`(1, ((40.00019, -116.44829), 1))`
…

    b. Reduce the result: for each center in the `kPoints` array, sum the latitudes and longitudes, respectively, of all the points closest to that center, and also find the number of closest points. For example:

`(0, ((2638919.87653,-8895032.182481), 74693))`
`(1, ((3654635.24961,-12197518.55688), 101268))`
`(2, ((1863384.99784,-5839621.052003), 48620))`
`(3, ((4887181.82600,-14674125.94873), 126114))`
`(4, ((2866039.85637,-9608816.13682), 81162))`

c. The reduced RDD should have (at most) K members. Map each to a new center point by calculating the average latitude and longitude for each set of closest points: that is, map `(index,(totalX,totalY),n)` to `(index,(totalX/n, totalY/n))`.

d. Collect these new points into a local map or array keyed by index.

e. Use the provided `distanceSquared` method to calculate how much the centers "moved" between the current iteration and the last. That is, for each center in `kPoints`, calculate the distance between that point and the corresponding new point, and sum those distances. That is the delta between iterations; when the delta is less than `convergeDist`, stop iterating.

f. Copy the new center points to the `kPoints` array in preparation for the next iteration.

8. When all iterations are complete, display the final K center points.

This is the end of the exercise