Exercise set 3

Hou, Jue

014695647

* Exercise 1 & 2

|  |  |  |  |
| --- | --- | --- | --- |
| pair | Correlation | time RDD | time DF |
| energyRate, cpuUsage | 0.007296950505 | 91.3560550213 | 9.85823893547 |
| energyRate, screenBrightness | 0.048568574534 | 92.5550849438 | 10.5577869415 |
| energyRate, wifiLinkSpeed | 0.000305233520 | 91.3492290974 | 10.2188339233 |
| energyRate, wifiSignalStrength | -0.02730869430 | 94.5271809101 | 11.1153469086 |

Dataframe spent significantly less time on calculation correlation if compared to conventional RDD calculation.

The schema of caratDF before assigning column name:



|  |  |  |
| --- | --- | --- |
| Item | Unique Element | Outlier |
| batteryTemperature | 117 | 298776 |
| batteryVoltage | 6064 | 7127 |

* Exercise 3

An RDD consists of multiple partitions. After reading data, it will first automatically assign into difference partitions. One partition will never be distributed into multiple nodes, but a node can have multiple partitions. The number of partitions can be manually set but usually is determined by the number of cores. There are two different partitioning approaches in Spark: hash and range. Hash partitioning will partition data according to the hash value of key, while range partitioning will partition data according to the order of keys and their range. Both partition methods will lead to partitions with different size. Since partitions might be distributed into different nodes, performance can be optimized by balancing the number of partitions of each node. In other words, better performance can achieve when each node has more or less same amount of workload or partitions.

* Exercise 5
  + To build an effective ML pipeline, I will first try to optimize the algorithm to minimize the time cost of the algorithm. Next thing I will do is to make sure input data is parallelized into each node uniformly so that workload can be kept in balance among all the nodes. Third, I will use as much as possible embedded function, since some of them might get optimized already while implementation of mine may cost more. Additionally, if possible, I will implement function like PCA to decrease the amount of data so that further computations can be easier.
  + To apply collaborative filtering, I will first obviously need to get dataset ready and parsed. Next I will split a relatively small dataset from the original one. Split the original dataset into training, validation and test dataset. Use the smaller datasets to determine the best ALS parameter. Apply cross-validation if necessary. At last, use the entire dataset to build recommender model with the ALS parameter which previously determined.
  + HaLoop and Hadoop with Spark have many common features. They are both solutions for processing enormous amount of data. And they are both built on distributed system and share almost same structure. They all have a master that responsible for assigning and managing tasks and workers that actually run the code. HaLoop is modified from Hadoop, while Spark can get data from HDFS. So, they can be placed above the same distributed storage system and read from the same data source. And they both support a better programming model if compared to standard Google’s MapReduce model. Also, they both support iterative MapReduce.

They also have many different properties. First of all, HaLoop as a single distributed system is modified from Hadoop, while Hadoop and Spark is two different system but having different function and working together. HaLoop will cache data partitions on the physical nodes’ local disk. RDD of Spark will usually stay in nodes’ memory. Also, RDD is immutable once it got created. HaLoop is specialized in iterative MapReduce. It explicitly has a loop control in its system and its master will monitor and keep tracking iterations. And its task scheduling knows iterations so that Inter-Iteration locality can be achieved. Spark does not have such implementation inside its task scheduler. But, for Spark, it provides a higher level of general abstraction for programming. RDDs provide a distributed storage abstraction explicitly and can thus support applications that HaLoop does not capture, such as interactive data mining.