Heidelberg University
Winter semester 2017/2018

Group Parallel and Distributed Systems (PVS)
Artur Andrzejak, Diego Costa

Problem Set 7 for lecture Mining Massive Datasets

Due December 11, 2017, 11:59 pm

Exercise 1 (2 points)

Explain the details of the data structure `HashMap` from Java standard library by answering the following questions (to this aim, analyze the source code and/or documentation of the implementation).

- Describe briefly the internal data structure used by `HashMap` for storing of map entries.
- How is a new <key, value> pair added to a map? Consider the possibility that a collision of hash values may occur.
- How is an entry retrieved given a search key? Consider the possibility that a bucket can have more than one entry.

Exercise 2 (8 points)

In this exercise you should implement in Spark the computation of minhash signatures of sets. The input of your implementation consists of the following data:

- a parameter K determining the number of random hash functions (i.e., the number of rows of the signature matrix)
- a parameter N specifying the maximum number of elements in any of the considered sets
- an RDD where each record contains one set represented as a sorted list of 32-bit integers with possible values ranging from $\{1, \dots, N\}$. These integers are IDs of elements present in the set (e.g., hash values of shingles). Note that this representation is different from the one used in lecture 8 (where a set `S` is represented by an array `B(S)` of bits indicating whether an element is present in the set, or not).

The output should be an RDD containing the signature matrix, stored column-wise. That is, one record holds the K entries that correspond to the signature of one set. The algorithm should at first choose a set of K random hash functions $h_1, \dots, h_K$ (described in lecture 8 on slide "Implementation /1"). As a value for the prime number p you might use $2^{31} - 1 = 2,147,483,647$ (the greatest signed 32-bit int value and - conveniently - a prime number).

Notes:

- when programming in Python, use the signed int data type `numpy.int32` for storage
- beware of overflows while computing the hash functions
- think about an efficient way to distribute a list of K random hash functions to perform the needed transformation(s) of the input RDD.

a) Implement the algorithm described above in Spark and attempt to provide a scalable implementation in terms of number of sets.

b) Validate your implementation. To this aim generate an RDD with 100 sets, each with 20000 elements, as follows. The first set S_0 contains 20000 elements represented as random integers with possible values ranging from $\{1, \dots, 10^5\}$ (sort them!). Each S_i (for $i = 1, \dots, 99$) is generated from $S_{(i-1)}$ by replacing (by other random elements) about 2% of randomly chosen elements of $S_{(i-1)}$. Make sure to always sort the newly created sets. Submit as a solution:

- your code
- the signature matrix M for $K = 30$ (i.e., with 30 rows) for a single run
- the number (or fraction) of identical values in M for column 0 (i.e. set S_0) and each of the columns $1, 2, \dots, 99$ (essentially, this compares the similarity of set pairs (S_0 vs. S_1), (S_0 vs. S_2), ... (S_0 vs. S_{99})).

Note: The preparation of input data and the comparison of the output may be implemented sequentially, i.e. without Spark.

Exercise 3 (2 points)

Plot the S-curve $1-(1-s^r)^b$ for $s = 0.1, 0.2, \dots, 0.9$, for each of the 16 pairs of parameters $r = 2, 3, 5, 7$ and $b = 10, 20, 50, 100$. For each plot, estimate by your visual judgment, roughly where on the x-axis the curve passes the y-value of 0.5. Submit the plots and your results.

Exercise 4 (4 points)

This exercise is the first in a series of tasks related to processing data with Spark dataframes. Your implementation will be reused in future exercises.

Take a look at a dataset[1] with prices of so-called spot instances from Amazon Elastic Compute Cloud (EC2) service. This dataset contains prices collected in the past *four* weeks for *seven* availability zones, grouped in 28 zipped files. Inside of each compressed file there is a tab-delimited text file with the structure

<Type>\t<Price>\t<Timestamp>\t<InstanceType>\t<ProductDescription>\t<AvailabilityZone>

containing the prices of EC2 spot instances of the last 100 days.

a) Implement a subroutine in Python/Spark which takes as an argument a file name of a compressed (*.gz) file with the structure as described above, reads its content into a new Spark dataframe, and returns a reference to this dataframe. To this aim define an appropriate schema with a correct name and (memory-efficient) data type for each column (e.g. use *TimestampType*() and not *StringType*() for column *Timestamp*, see also https://goo.gl/rkxENJ). The resulting dataframe should **not** contain the column *Type* (as each row has the value “SPOTINSTANCEPRICE”). Submit your code as the solution.

b) To test your implementation, read the file *prices-ap-northeast-2-2017-11-10.txt.gz* into a dataframe, and then calculate and output the average price per each combination *InstanceType*/*ProductDescription*.

[1] Available at https://heibox.uni-heidelberg.de/d/78ea9d73e2/