Big Data Predictive Analytics with RevoScaleR

A Yugandhar Reddy
Department of CSE, JNTU College of Engineering Anantapur, Anantapur-515-002, India.

A. Suresh Babu
Department of CSE, JNTU College of Engineering Anantapur, Anantapur-515-002, India.

Abstract – R is becoming the leading language in data science. Because of thousands of packages to handle data science in an easy programmable manner. When it comes to the larger data sets its showing poor performance in high end servers also. This is only because of its style of handling the data. In R entire data need to hold in the memory for processing. Due to this memory bound of R, original R is not suitable to handle larger data sets. This paper addresses how the problem of R can overcome with RevoScaleR.

Index Terms – Big Data, RevoScaleR, Analytics.

1. INTRODUCTION

R is hot, R is rapidly becoming the leading language in statistics and data science. Today, R is the tool of choice for data science professionals in every industry and field. R applications span the universe from the hard sciences such as astronomy, chemistry and computational statistics and genomics to practical applications in business, medicine drug development, marketing, finance, health care, and all manner of forecast analytics. Because R has thousands of packages many of which are devoted to particular applications. Mastering of R is not need to develop your own applications.

Memory-bound of the R is the main barrier to it .All data need to be held in the memory to perform calculations. This constraint becomes a problem when data is very large. Even for modern computers with high configuration (like, 64-bit address spaces and huge amounts of RAM) dealing with data sets that are hundreds of millions of observation is a significant challenge. The problem is not just simply being capable to house the data in memory for examination but performance and scalability are prime considerations

Revolution Analytics has addressed these challenges with its Big Data to extend the reach of R into the kingdom of production data analysis with very large data sets. This paper address the Revolution Analytics new package named RevoScaleR TM, which delivers extraordinary levels of performance and capacity for statistical analysis in the R environment. Now without deploying specialized or expensive R users can model and visualize larger data sets.

2. LIMITATIONS OF IN-MEMORY DATA ANALYSIS

In the world of fatly increasing size of data, organizations are discreet to deploy R beyond research due to this memory bound limit. This drawback of R does not only impacts the ability of data that can be processed, but also badly affects the performance and scalability. For instance, allotting a data frame as small as 5,000,000 observations and 3 variables will throw an error like the following due to memory limitations. Even though high-end servers can manage to held big data in memory, the performance of original R on such big data files will be expensively slow

```
>lm(ArrDelayUniqueCarrier+DayOfWeek, data=myData)
Error: cannot allocate vector of size 751.8 Mb
```

To overcome this limitation, efforts have been made in improving R to scale for Big data. The best way to achieve it is by implementing parallel external memory storage and parallel processing mechanisms in R.

3. HOW REVOSCALER PROCESS DATA

3.1 Storing Data

One of the answers to being scalable is the capacity to handle additional data than can fit into memory at same time. This equates to being capable to work with chunks of data as an alternative of requiring the whole dataset to be housed in memory at same time. In the case of RevoScaleR, chunks are defined as consecutive blocks of observations for a given projection of variables.

Though RevoScaleR can process data from various sources, it has its specific fully optimized file structure (theXDF format) that is particularly right for chunking. Data in an XDF file can be read rapidly by observations or by variables. In addition, blocks of observations for selected variables can be read consecutively, instead of randomly. Consecutive accesses can be tens to thousands of times faster than random access. Also, in an XDF file the time it takes to access a block of observations for a variable is fundamentally independent of the total number of variables and observations in the file. This means even in the very files, only the data for the actual variables required for an analysis needs to be accessed and processed, and this may only be a few hundreds to few thousands megabytes. The time it takes to do that is basically the same as if only that data were stored in the file; storing the additional idle data does not add to the processing time.

Data in an XDF file is held in the same binary format that is employed in memory, so no conversion is needed when it is

brought into memory. In order to reduce wasted space, it can also be held in a suitable -sized way. For example, a variable that have no more than 256 values can be stored in a byte per number. Floating point values with a precision of less than 6 or 7 decimal digits can be stored in 4 bytes per number.

New columns and new observations can be added to the file without rewriting the whole file. Thus, the cost of creating new variables and of adding more rows is greatly reduced.

3.2 Reading Data

When data is accessed in chunks, the feasible chunk size depends on the variety of reasons, such as the speed of RAM, the speed of the disk, speed of cores, the number of cores, and the variety of calculations being done. RevoScaleR permits the size of chunks to change depending upon circumstances. A bottleneck for data handling is data I/O: accessing the data from disk (external). RevoScaleR devotes one core to accessing the data from disk. Meanwhile, the other cores are allocated to processing the chunk of data read into memory from the last access. If it is possible to hold all data into memory on once, RevoScaleR permits that. It then assigns all available cores to handle that data.

3.3 Handling Data in Memory

As on disk, usage of the appropriate-sized data type in memory reduces the space needed and also the time taken to move the data in memory. In other technologies, before adding a set of integers and a set of double floating point numbers together, the set of integers is first converted and copied into a set of doubles. This it is time and space consuming. In RevoScaleR avoid such conversions and coping of the data values regardless of the type of the operation and size of the data until the data values are really given to CPU.

3.4 Use of Multiple Cores on a Single Computer

Virtually all computations that encompass data in RevoScaleR are spontaneously threaded that they use several cores on an engine when they are available. This is done efficiently by minimalizing the overhead of shifting the computations to multiple threads, by minimizing the quantity of data that must be copied, by doing as much work as possible on each thread to remunerate the cost of initializing the computations, and by minimizing inter-thread synchronization and communication. Loading huge chunks of data to each of the multiple cores is vital for efficiency. For analytic routines such as crosstabs, logistic regression, descriptive statistics, K-means clustering and linear regression (in which large number of variables are naturally used) a huge chunk of observations perhaps billions for all of the variables is read into memory by one core. Concurrently, the data chunk from the previous read is virtually split among the left behind cores for the required processing. The code doing the processing on each core (thread) only desires to know what its allocated task is, and no inter-thread synchronization and communication is required.

4. EFFICIENT PARALLELIZATION OF STATISTICAL AND BIG DATA PREDICTIVE ANALYTICS

RevoScaleR is built upon a platform designed to efficiently and automatically external memory algorithms. These are the class of algorithms that do not need all data to be in memory at same time, and such algorithms are existing for a wide range of data mining and statistical routines. The way in which these algorithms are automatically parallelized is such that, in general, the quickest algorithms per core are also the quickest when parallelized. (This happy situation is not the case for some other class of parallel algorithms). Since the load of worrying about parallelization is detached from the engineers applying these algorithms, they can concentrating on getting feasible speed on each core. This encompasses several things. Most obviously, it involves using fast algorithms, and carefully coding those using C++ templates, which can produce very fast code. Other issues are important as well. Categorical data is very common in statistical computations, and they are dialed in ways that save memory, upturn speed, and increase computational exactness as well. It is often the case in statistical models that the same values are required in different parts of the computation. RevoScaleR has a erudite algorithm for pre-analyzing models to detect such repetition, so that the number of computations can be diminished. Multiple models can be analyzed jointly. This algorithm can also detect collinearities in models, which can lead to wasted computations or even computational failures, and can remove them prior to doing any computations.

5. USING REVOSCALER FOR BIG DATA PREDICTION ANALYTICS

This section focuses on applying RevoScaleR package to big data prediction analysis. This show how to read a data set in the text format, how to convert text file to .xdf format, how to construct big data decision tree, how to prune the decision tree, how to predict the future and finally how to plot the graph.

5.1 Importing the dataset

Here is a sample RevoScaleR analysis that uses a subset of the airline on-time data reported each month to the U.S. Department of Transportation (DOT) and Bureau of Transportation Statistics (BTS) by the 16 U.S. air carriers. This data comprises three columns: one categorical variable, DayOfWeek and two numeric variables, ArrDelay and CRSDepTime. It is positioned in the SampleData directory of the RevoScaleR package, so you can easily exucute this example in your Revolution R Enterprise session

Import the example airline data from a csv to an .xdf file. When we load the data, we transform the string variable to a (categorical) factor variable by stringsAsFactors:

inFile<-file.path(rxGetOption("sampleDataDir"), "AirlineDemoSmall.csv") rxTextToXdf(inFile = inFile, outFile = "airline.xdf", stringsAsFactors = T, rowsPerRead = 200000)

There are a total of 600,000 observation in the dataset file. Specifying the argument rowsPerRead lets us to read and write the data in 3 blocks of 200,000 observations each.

View elementary data information. The rxGetInfoXdf function lets you to quickly view some elementary information regarding variables data set and the data set.

rxGetInfoXdf("airline.xdf", getVarInfo = TRUE, numRows = 20)

Function rxHistogram is used to show the distribution of flight delay by the day of week.

5.2 Exploring the data

Use the rxHistogram function to show the distribution of flight delay by the day of week.

rxHistogram(ArrDelay|DayOfWeek, data = "airline.xdf")

Next, we calculate summary statistics to the arrival delay variable

rxSummary(ArrDelay, data = "airline.xdf")

5.3 rxDTree for prediction

Decision trees are effective algorithms widely used for regression and classification. Classical algorithms for constructing a decision tree sort all continuous variables to decide where to divide the data. This sorting step becomes memory and time prohibitive when handling with large data. Numerous techniques have been projected to overcome the sorting problem, which can be roughly classified into two classes: using approximate summary statistics or performing data pre-sorting of the data. While pre-sorting procedures follow classical decision tree algorithms more closely, they cannot house very huge data sets. These big data decision trees are generally parallelized in numerous ways to allow large scale learning: data parallelism splits the data either vertically or horizontally so that different processors see different variables or rows and task parallelism builds diverse tree nodes on diverse processors.

The rxDTree algorithm is an estimated decision tree algorithm with horizontal data parallelism specially designed for handling very large data sets. It calculates histograms to create empirical distribution functions of the data and constructs the decision tree in a breadth-first fashion. The algorithm can be run in parallel environments such as a distributed (grid or cluster) environment or a multicore machine. Each core gets only a part of the observations of the data, but has a view of the whole tree built so far. It constructs a histogram from the observations it sees, which basically compresses the data to a static amount of memory. This estimated description of the data is then sent to a master with constant little communication complexity autonomous of the length of the data set. The master integrates the information received from each of the workers and spots which terminal tree nodes to split and how. Since the histogram is built in parallel, it can be rapidly constructed even for very large data sets.

With rxDTree , you can regulator the balance between prediction accuracy and time complexity by setting the maximum amount of bins for the histogram. The algorithm constructs the histogram with roughly equal number of rows in each bin and takes the limits of the bins as the candidate splits for the terminal tree nodes. Since only a limited number of split sites are inspected, it is probable that a suboptimal split point is chosen causing the entire tree to be different from the one constructed by a classical algorithm. However, it has been exposed analytically that the error rate of the parallel tree reaches the error rate of the serial tree, even if the trees are not identical .You can set the number of bins in the histograms to regulate the balance between speed and accuracy: a huge number of bins allows a more precise description of the data and thus more precise results, whereas a little number of bins diminishes time complexity and memory usage. In the case of integer forecasters for which the count of bins equals or exceeds the number of unique observations, the rxDTree algorithm produces the identical results as classical sorting algorithms since the empirical distribution function precisely represents the data set. To forecast the class label of the query pattern we build a classifier called decision tree for training and testing the classifier we need training set and testing set. Genarally 70% observations of dataset are used to train the classifier remaining records are used for testing. Following code used to load and split the dataset in to training and test sets.

inDataFile <- file.path("AirlineDemoSmall.csv") rxImport(inDataFile,outFile="airline", overwrite = TRUE) inDataFile <- file.path("airline.xdf")

split data set

rxSplit(inData = inDataFile, outFilesBase = paste0("airline"), outFileSuffixes = c("Train", "Test"), splitByFactor = "s", overwrite = TRUE, transforms = list(splitVar = factor(sample(c("Train", "Test"), size = .rxNumRows, replace = TRUE, prob = c(.80, .20)), levels = c("Train", "Test"))), rngSeed = 17, consoleOutput = TRUE)

training and testing data sets

training <- file.path("airline.s.Train.xdf") testing <- file.path("airline.s.Test.xdf")

For the given data set Late is a dependent variable.Setting the complexity parameter (cp) to it's default value results in the very large number of splits. Specifying cp = 1e-5 produces a

more manageable set of splits in this model.Following code builds the factors for the tree

```
rxFactors(inData=training,factorInfo = list(defaultFctor=list(varName="default")), outFile=training,overwrite = TRUE)
rxFactors(inData=testing,factorInfo = list(defaultFctor=list(varName="default")), outFile=testing,overwrite = TRUE)
```

Following code sets the parameters for the decision tree

```
control <- list(minsplit=20,cp=0.01,xval=2, maxdepth=5, maxcompete=0,maxsurrogate=0, usesurrogate=2 ,surrogatestye=0) treeC <- rxDTree(formula = Late CRSDepTime + DayOfWeek ,data=training,control=control,maxNumBins = 15000)
```

Prediction and confusion matrix

```
rxPredict(treeC,data=testing,outData = testing, overwrite = TRUE,predVarNames = "Pred_C",type="vector") conf.mat <- rxCrossTabs( defaultFctor:Pred_CF, data=testing) print(conf.mat) print(prop.table(conf.mat$counts[[1]]))
```

6. CONCLUSION

RevoScaleR is a library included in Revolution R Enterprise that provides enormously fast statistical analysis on very very large data sets, without needing specialized hardware. Using only a commodity multi-processor computer with modest amounts of RAM, data processing and predictive modeling techniques can easily be performed on data sets with hundreds of millions of rows and hundreds of variables, at speeds suitable for interactive processing. Extending the system to a small cluster of similar computers commensurately reduces processing time. These achievements are the result of the design of fthe RevoScaleR platform, constructed from the ground up for speed and scalability.

REFERENCES

[1] Ben-Haim, Y., & Tom-Tov, E. (2010). A streaming parallel decision tree algorithm. Journal of Machine Learning Research, 849-872.
[2] Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). Classification and Regression Trees. Pacific Grove: Wadsworth.
[3] Frank, A., & Asuncion, A. (2010). (University of California, Irvine, School of Information and Computer Science) Retrieved August 2012, from UCI Machine Learning Repository: http://archive.ics.uci.edu/ml
[4] Kohavi, R. (1996). Scaling Up the Accuracy of Naive-Bayes Classifiers: a Decision-Tree Hybrid. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining.
[5] Therneau, T.,& Atkinson, E. (1997). An Introduction to Recursive Partitioning Using the RPART Routines. Rochester, MN: Mayo Clinic

Authors

A Yugandhar Reddy Post graduate student in the stream of Computer Science in Depart of Computer Science and Engineering. Obtained his bachelor degree in Information Technology From JNT University Anantapur.

Dr. A Sureshbabu Presently working as Additional control of examinations in JNT University Anantapur. Obtained his PhD from JNT University Anantapur. He published several papers in national and international journal. His research interests are Data Mining, Big Data Analytics, Cloud Computing.