Indian Journal of Science and Technology, *Vol 9(37), DOI: 10.17485/ijst/2016/v9i37/85826, October 2016*
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Big Data Mining Techniques

Adeel Shiraz Hashmi* **and Tanvir Ahmad**

Department of Computer Engineering, Jamia Millia Islamia, Delhi, India; ashashmi10@gmail.com

Abstract

Objectives: The objective of this research work is to discuss the various techniques which can be used for mining of big data viz. sampling, incremental learning, and distributed learning. **Methods:** For this study, literature survey was done to identify the various techniques employed by different authors to handle large (and streaming) data sets. For each technique, one or more algorithm was chosen and applied on large data sets. The platform for each technique was standardized (R libraries were used for each algorithm). The algorithms were compared on accuracy and time-consumed. **Findings:** The findings of this research work which conform to the existing literature is that the distributed learning is the best approach in terms of accuracy and time-complexity, for large data sets. However, if the data sets are streaming data sets and we want to perform real-time analysis then sampling or incremental approach are better than distributed approach. Incremental approach provides better accuracy, whereas sampling reduces time-complexity. **Novelty:** This study is important in the sense that it brings all the three techniques together on a single platform, which hasn't been done earlier.

Keywords: Big Data, Data Mining, Distributed Learning, Incremental Learning, Sampling

1. Introduction

The first research paper on 'Big Data' appeared in 2000[1], but the term became widespread as recently as in 2011, after explosion in the amount of data that is being generated daily by sites like Facebook, Twitter, YouTube, etc. along with an increase in networked devices (Internet of Things) like sensors, mobiles, etc. Although 'size' is the first, and most of the times, the only dimension that leaps out at the mention of big data, the term "Big Data" refers to any collection of data which is difficult to handle using traditional database management systems and data processing tools due to its size, complexity or speed at which it is generated. Big Data is characterized by 3 V's: Volume, Velocity, and Variety[2], but other dimensions like Veracity, Value, have also been mentioned.

What should be the volume of a data set to qualify being big data? Microsoft Excel 2007 can't perform analysis on more than 1 million rows, so any data set larger than 1 million instances is "big" data for MS Excel 2007; whereas for other tools like R, the size of RAM is the constraint. So, any data set which is larger than the available processing capabilities needs a big data solution. The data generated by sensors, e-commerce sites, social networking sites etc. has a very high velocity and there may be a need to process this data in real-time, which may be beyond the available processing capabilities, making it "big" not in terms of volume but in terms of velocity. The data may be obtained from various sources, and unlike the traditional data warehouse which deals with only structured data, the data here may be structured, semi-structured, as well as unstructured. This is the third dimension of "Variety" which makes the data too complicated to be handled by traditional data warehousing systems. Applications of big data are found in diverse fields like analyzing and visualizing web-server logs[3], handling illegal parking[4], etc.

2. Big Data Tools

Message Passing Interface[5] is a standard used for developing and running parallel applications on a peer-to-peer network. MPI is available for many programming languages. The major drawback of MPI is fault intolerance, and a single node failure can cause the entire system to shut down. Apache Hadoop, developed by Doug Cutting and Mike Cafarella in 2005, is an open source fault-tolerant framework for distributed processing on cluster of commodity hardware. Hadoop

*Author for correspondence

provides HDFS distributed file system[6] and MapReduce[7] programming model for writing parallel jobs. Apache Pig[8] and Apache Hive[9] are Map-Reduce wrappers which simplify writing of Map-Reduce jobs. Apache Spark[10] is a cluster computing framework, which provides performance up to 100 times faster than Map-Reduce through data caching.

Apache Storm[11] and Apache S4[12] are distributed real-time computation frame work for processing fast, large streams of data. Apache Spark can also handle data streams through Spark Streaming library. General Purpose computing on Graphics Processing Units (GPGPU) takes advantage of large number of cores (typically around 2500+) and high-throughput DDR5 memory present in Graphics Processing Units (GPUs) to speed up the processing multiple times compared to CPUs. Compute Unified Device Architecture (CUDA) is a GPGPU framework for parallel computing on NVIDIA GPUs. The major drawback of GPGPU is the limited size of memory (maximum of 12GB as of now).

So if one desires high performance (like in case of processing high velocity data streams) then he should go for Apache Storm or GPGPU, whereas if data scalability and fault-tolerance is desired then Hadoop or Spark cluster should be setup.

3. Big Data Mining

Traditional data mining algorithms like k-means clustering[14] Naïve Bayes classifier[15], etc. can only handle static data which fits into the memory. These algorithms need to be modified[16] to be able to handle data streams (velocity) or memory limitations (volume). The data streams in which the classification boundary keeps changing as new examples are received are called data streams with concept-drift[17]. There may be situations, where only recent examples are required for building the concept rather than complete data streams. To handle such data streams, we can use the concept of sliding window or weighting policy. In sliding window, we only consider only a specific amount of recent data, and completely discard the old data. In weighted policy, we assign weights to data elements such that older element have lower weight and newest element has highest weight e.g. simply assign weights 1,2,3,4…, and so on.

A simple technique to handle large data sets or data streams is sampling, where we do not consider the whole data set, rather we take sufficient number of samples from the database which can give us approximate statistical measures of the complete data. Reservoir sampling and Hoeffding bounds are the two most popular sampling methods for large data sets.

A technique which can be applied to mine high-velocity data streams as well as massive data sets is incremental learning. Incremental learning processes one instance at a time when generating the model. An instance can be read from the stream or input file, the model can be updated, the next instance can be read, and so on – without ever holding more than one training instance in main memory. Naive Bayes is naturally an incremental algorithm and multi-layer neural networks with stochastic back propagation are also incremental.

Incremental learning tends to be slow, so to speed up the process we can use parallelization, where we split the data set or data stream, process each using a separate processor, and combine the results together using voting or averaging to give a final result. Decision tree learners can be parallelized by letting each processor build a subtree of the complete tree. Bagging and stacking (although not boosting) are naturally parallel algorithms; whereas other popular algorithms can be modified to handle data incrementally.

Which tools we should use for big data mining? Weka and R are the two most popular tools for data mining, but can they handle big data? Massive Online Analysis (MOA) is a Weka-like tool with all the state-of-the-art algorithms for mining of data streams, which can also be used for incremental learning on data streams. R can use package "Rmpi" to handle large data sets or "stream" to handle data streams as well, where Rmpi provides Message-Passing Interface (MPI) for R language and stream package provides data stream mining algorithms. Apache Spark has a machine learning library Spark MLlib which supports many popular machine algorithms. H2O is an open source parallel processing engine for machine learning, and provides support to R and python, along with support for Hadoop, Spark, and Storm.

4. Sampling

A simple straight-forward method for handling large data sets is sampling (dimensionality reduction may also help if there are large number of features). But what should be the size of the sample so as to obtain good results?

Reservoir sampling[18] is sampling without replacement (i.e. an example can't be selected more than once). The

idea is to use a reservoir of size *s*, and initially filling it with first *s* examples; when more examples are received, i^{th} instance in the input stream is placed in the reservoir at a random location with probability *s/i*.

Very Fast Machine Learning[19] toolkit uses **hoeffding-bound** for mining data streams and large data sets. Hoeffding bound states with probability *1-δ*, that the true mean of a random variable of range *R* will not differ from the estimated mean after *n* observations by more than ε,

$$\varepsilon = R * \sqrt{\frac{\ln\left(\frac{1}{\delta}\right)}{2n}} \quad (1)$$

The Very Fast Machine Learning (VFML) toolkit consists of algorithms like Very Fast K-Means (VFKM), hoeffding trees, etc. The VFKM algorithm performs clustering on a sample whose size (n) can be decided by: (1), with confidence 1-δ (δ=5% by default) within error ε.

Hoeffding trees are based on the concept that if the difference in information gains b/w the best two attributes is less than ε then it is not safe to split. However, ε will decrease as n is increased, so it is just a matter of getting adequate number of samples. The random variable being estimated is the difference in information gain b/w the best two attributes, R is the base 2 logarithm of the number of possible class labels (i.e. R=1 for a two-class problem), and δ is taken to be very small ~10^{-7}.

5. Incremental Learning

Incremental learning[20] is different from traditional machine learning in the sense that it does not assume the availability of complete data set before learning process, but the learning takes place whenever new instances emerge and adjusts to what has been learned from these instances.

Incremental k-means[21], Incremental Density Based Spatial Clustering of Applications with Noise (DBSCAN)[22] are variants of popular k-means[14] and DBSCAN[23] clustering algorithms, respectively, which work on dynamic and large databases. There are various other incremental clustering algorithms also in the literature for mining of data streams like CobWeb[24], BIRCH[25], CluStream[26], DenStream[27], etc.

Naïve Bayes[15] needs no major modification to handle data incrementally. In Naïve Bayes, the probability of an example *E* belonging to a class *C* can be calculated as follows:

$$P(C|E) = P(C) * \prod_{i=1}^{k} P(E_i|C)$$

where, P(C) and P(E|C) are priori probabilities obtained from training set, and P(C) = total examples belonging to class C / total examples, whereas P(E|C) = count of attribute value E associated with class C / total examples belonging to class C. In incremental learning, as new examples are received we recalculate P(C) as well as P(E|C) and update the model.

The idea behind **back-propagation neural network**[28] is to adjust the weights of the network to minimize some measure of error on training set. The measure of error is given by the equation:

$$Err = \left[y - g\left(\sum_{i=0}^{n} [w_i x_i)\right]\right]$$

where, *y* is desired output, *g* is the activation function, x_i is the i^{th} input, and w_i is the weight associated with x_i.

The method used to minimize the squared error is gradient descent, which takes all the training examples in each of the iteration to minimize the error,

$$w_i = w_i + \alpha * Err * x_i$$

Where, α is the learning rate, and w is the weight to be adjusted.

For incremental learning, stochastic gradient descent can be used, which instead of taking all of the instances in each of the iteration considers each instance one-by-one.

6. Parallel Learning

Some of the machine learning algorithms like Random Forest work on multiple copies of same data set and can easily be parallelized, whereas algorithms like k-means[14], kNN[29], Naïve Bayes[15], back-propagation neural networks[28], Support Vector Machines (SVM)[30], etc. must be modified for distributed processing.

k-means algorithm can be parallelized by splitting the data into subgroups and clustering samples in each subgroup separately. **Nearest-neighbor** methods can be easily distributed among several processors by splitting the data into parts and letting each processor find the nearest neighbor in its part of the training set. In **Naïve Bayes**, we have to estimate *P(E|C)* from the training data,

so we can parallelize it by computation of $P(E=k|C=0)$ and $P(E=k|C=1)$ for each k in parallel. **Neural Networks** and **SVM** can be parallelized by performing partial batch gradient descent on subgroups of data samples.

Random forest[31] is a collection of decision trees on bagged data sets, but in random forests, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidate from the full set of p predictors. A fresh sample of m predictors is taken at each split, and typically we choose $m \sim \sqrt{p}$. Random forest provides an improvement over bagged trees. As in bagged trees, we use full set of predictors to make the split, therefore each tree may use the strongest predictor in the top split, and as a result all trees may look quite similar to each other. Random forests overcome this problem by forcing each split to consider only a subset of predictors, therefore strongest predictor will not be even considered for few cases, and so other predictors will have more of a chance.

7. Experimental Setup and Results

We want to compare these three learning techniques on large data sets. For this we select representative algorithms for each learning technique; the algorithms selected are hoeffding trees, incremental Naïve Bayes, distributive random forest, and distributive deep learning neural networks. The primary data set used is HIGGS data set of size 8GB, other two data sets being popular kddcup99 and Modified National Institute of Standards and Technology (MNIST) data sets. The summary of experimental setup is given in Table 1.

The time required for building the classifiers from the training sets are listed in Table 2, and the classification errors for each data set by each classifier are listed in Table 3. For hoeffding trees and incremental algorithms, the files were read in chunks of 100000 instances (i.e. chunk size = 100000). MNIST data set is multi-class data set with 10 labels, kddcup99 is a medium-sized data set with binary labels, whereas HIGGS data set is "big" data set with binary labels. The classifiers were trained with the data sets, and then the test data sets of kddcup99 as well as MNIST were used for prediction, and the classification error was calculated. For HIGGS data set with 11 million instances, the last 1 million instances were reserved as test set, whereas the rest were used for training.

Table 1. Experimental setup

Processor	Core i5 @ 3.00GHz (64–bit) with 4 cores
RAM	2GB
OS	Ubuntu 15.04 (64–bit)
Cluster OS	H2O 3.0
Cluster size	6 for HIGGS data set, 1 for kddcup99 and MNIST data sets
Data sets	HIGGS (8GB, with 11 million instances of 29 features) kddcup99 (4 million instances of 42 features) MNIST (42000 instances of 785 features)
Language	R
Packages	RMOA, h2o

Table 2. Training time

Algorithm	MNIST	kddcup99	HIGGS
Hoeffding-tree (single-node)	6.77 min	18.41 min	2.74 hrs
Incremental Bayes (single-node)	6.59 min	18.26 min	2.91 hrs
Distributed Random Forest (20 trees)	1.33 min	7.47 sec	47.24 min
Distributed Deep Learning (200X200 hidden-neurons, 10 epochs)	11.95 min	21.82 min	7.10 min

Table 3. Classification error

Algorithm	MNIST	kddcup99	HIGGS
Hoeffding-tree	7.69%	0.31%	6.72%
Incremental Bayes	6.53%	1.50%	6.54%
Distributed Random Forest	3.35%	0.18%	3.08%
Distributed Deep Learning	2.09%	0.09%	3.02%

8. Conclusion

From the results obtained, we conclude that the performance of sampling and incremental approach is nearly same, both in terms of training time taken and prediction accuracy. However, compared to distribute learning, both sampling and incremental learning techniques are not only much slower, but also have higher classification error. The results of the experiments conducted were not a surprise and were expected. However, the sampling or incremental approach would be better for streaming data.

Hoeffding trees and incremental learning can't handle concept-drift as it can't discard what is learnt from the old data. Ensemble learning can better handle concept drift, by performing bagging using small trees, as smaller trees can easily adapt to concept drift.

9. References

1 Diebold F. Big Data: Dynamic Factor Models for Macroeconomic Measurement and Forecasting. *Eighth World Congress of the Econometric Society*. 2000.
2. Laney D. 3-D Data Management: Controlling Data Volume, Velocity and Variety. *META Group Research Note*. 2001 Feb 6, p. 1–4.
3. Parthiban P, Selvakumar S. Big Data Architecture for Capturing, Storing, Analyzing and Visualizing of Web Server Logs. *Indian Journal of Science and Technology*. 2016 Jan; 9(4). Doi: 10.17485/ijst/2016/v9i4/84173
4. Kim KW, Park WJ, Park ST. A Study on Plan to Improve Illegal Parking using big Data. *Indian Journal of Science and Technology*. 2015 Sep; 8(21). Doi: 10.17485/ijst/2015/v8i21/78274
5. Gropp W, Lusk E, Skjellum A. Using MPI: Portable Parallel Programming with the Message-Passing Interface.MIT Press. 1999.
6. Borthakur D. HDFS architecture guide. *Hadoop Apache Project*. 2008, p.1–13.
7. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. *Communications of ACM*. 2008, 51(1):107–13.
8. Olston C, Reed B, Srivastava U, Kumar R, Tomkins A. Pig latin: a not-so-foreign language for data processing. *ACM SIGMOD international conference on Management of data*. ACM. 2008, p. 1099-110.
9. Thushoo A, Sharma J, Jain N, Shao Z, Chakka P, Anthony S. Hive: a warehousing solution over a map-reduce framework. 2009; 2(2):1626-29.
10. Zaharia M, Chowdhury M, Franklin M, Shenker S, Stoica I. Spark: cluster computing with working sets. *USENIX Conference on Hot Topics in Cloud Computing*. 2010, p. 10–10.
11. Toshniwal A, Taneja S, Shukla A, Patel J, Kulkarni S, Jackson J. Storm @Twitter. *ACM SIGMOD International Conference on Management of Data*. 2014, p. 147–56.
12. Neumeyer L. S4: Distributed Stream Computing Platform. *IEEE International Conference on Data Mining Workshops (ICDMW)*. 2010, p. 170–77.
13. Nickolls J, Dally W. The GPU Computing Era. 2010; 30(2):56–9.
14. Hartigan J. Clustering Algorithms. Wiley, 1975.
15. Langley P, Thompson K. An analysis of Bayesian classifiers. *Tenth National Conference on Artificial Intelligence*. 1992, p. 223–28.
16. Sajana T, Rani CMS, Narayana KV. A Survey on Clustering Techniques for Big Data Mining. *Indian Journal of Science and Technology*, 2016 Jan; 9(3). Doi: 10.17485/ijst/2016/v9i3/75971
17. Widmer G, Kubat M. Learning in the presence of concept-drift and hidden context. *Journal of Machine Learning*. 1996; 23(1):69–101.
18. Vitter J. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*. 1985; 11(1):37–57.
19. Hulten G, Domingos P. VFML - A toolkit for mining high-speed time-changing data stream. http://www.cs.washington.edu/dm/vfml/. 2003.
20. Giraud-Carrier C. A note on the utility of incremental learning. *AI Communications*. 2000; 13(4):215–23.
21. Shindler M, Wong A, Meyerson A. Fast and Accurate k-means for Large Data sets. *Advances in Neural Information Processing Systems*. 2011, p. 2375–83.
22. Ester M, Kreigel H, Sander J, Wimmer M, Xu X. Incremental Clustering for Mining in a Data Warehousing Environment. *24th International Conference on Very Large Data Bases*. 1998, p. 323–33.
23. Ester M, Kreigel H, Sander J, Xu X. A density-based algorithm for discovering clusters in large spatial databases with noise. *Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, Portland, 1996, p. 226–31.
24. Fisher D. Knowledge Acquisition *via* Incremental Conceptual Clustering. *Journal of Machine Learning*. 1987; 2(2):139–72.
25. Zhang T, Ramakrishnan R, Livn M. BIRCH: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*. 1997; 1(2):141–82.
26. Aggarwal C, Han J, Wang J, Yu P. A framework for clustering evolving data streams. *Very Large Databases*. 2003, p. 81–92.
27. Cao F, Ester M, Qian W, Zhou A, Density-based clustering over an evolving data stream with noise. *SIAM International Conference on Data Mining*, US, 2006, p. 326–37.
28. Williams R, Rumelhart D, Hinton G, Learning representation by back-propagation errors. *Nature*. 1986 Oct 9; 323:533–36.
29. Altman N, An introduction to kernel and nearest-neighbor nonparametric regression. 1992Aug; 46(3):175–85.
30. Platt J, Fast training of support vector machines using sequential minimal optimization. *Advances in Kernel Methods - Suport Vector Learning*, 1999, p. 185–208.
31. Breiman L. Random Forests. *Machine Learning*. 2001; 45(1):5–32.