2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

A Google Approach for Computational Intelligence in Big Data

Andreas Antoniades
Department of Computing, University of Surrey
United Kingdom
Email: aa00542@surrey.ac.uk

Clive Cheong Took
Department of Computing, University of Surrey
United Kingdom
Email: c.cheongtook@surrey.ac.uk

***Abstract*—With the advent of the emerging field of big data, it is becoming increasingly important to equip machine learning algorithms to cope with volume, variety, and velocity of data. In this work, we employ the MapReduce paradigm to address these issues as an enabling technology for the well-known support vector machine to perform *distributed* classification of skin segmentation. An open source implementation of MapReduce called Hadoop offers a streaming facility, which allows us to focus on the computational intelligence problem at hand, instead of focusing on the implementation of the learning algorithm. This is the first time that support vector machine has been proposed to operate in a distributed fashion *as it is*, circumventing the need for long and tedious mathematical derivations. This highlights the main advantages of MapReduce - its generality and distributed computation for machine learning with minimum effort. Simulation results demonstrate the efficacy of MapReduce when distributed classification is performed even when only two machines are involved, and we highlight some of the intricacies of MapReduce in the context of big data.**

I. Introduction

Over 2.5 quintillion (2.5×10^{18}) bytes are produced everyday and 90% of the data that exists today were created over the last two years [1]. It is therefore not a surprise that *big data* has garnered the attention of the IEEE Computational Intelligence Society. In this year's World Congress of Computational Intelligence (WCCI), there are three special sessions dedicated to big data addressing themes such as evolutionary computation, handling uncertainties and its application in e-Health; this event is followed by another IEEE Symposium on Computational Intelligence on big data later in the year. Machine Learning algorithms have already been deployed on an open source scheme to address big data, see for instance [2]. However, many classical machine learning algorithms for big data are still sorely lacking such as the well-known classifier support vector machine (SVM), as it is only now that the community of IEEE Computational Intelligence is starting to realise the potential of this research area. Moreover, there is a tendency for researchers to adapt the problem (in this case big data) to their expertise, instead of adapting to the realities of problem. As a result, the impact of these research is likely to take longer. The main issue in big data remains, however, on the computation of the data or on handling the 'scalability' of the big data. A straightforward answer is distributed computing, yet it is hard to adapt machine learning algorithms to operate on distributed computing.

It is in this motivational context that we provide a comprehensive overview of a powerful computational framework proposed by Google to enhance the scalability ability of machine learning techniques - the Map Reduce approach. This Map Reduce approach has already been used in k-means clustering tasks [2] and its performance has been analysed in the context of intelligent systems [3]. Other algorithms that have exploited the Map Reduce paradigm for distributed learning include the random forest decision tree based classifier, the Naive Bayes classifier, amongst others, which are freely available in the project Apache Mahout [2]. However, this open source project has one major drawback: it is implemented in the programming language *Java*. This may restrict its use, since it may pose a problem to researchers who are used to other programming languages in particular MATLAB. In light of all these issues, we therefore introduce the Map Reduce paradigm (i) to demonstrate how it can be used to facilitate the deployment of a machine learning technique for distributed computing; (ii) to illustrate its generality, as Map Reduce offers a facility called streaming to overcome the barrier of programming language; (iii) to show its relevance for the well-known support vector machine (SVM) classifier[1].

[1]The main focus of this work is not on SVM per se, but to introduce the powerful and flexible Map Reduce paradigm which can be used to make any machine learning algorithm distributed. Distributed SVMs have already been proposed in [7][8].

978-1-4799-1484-5/14/$31.00 ©2014 IEEE

II. HADOOP

Hadoop is an open source computational framework that operates on the Map Reduce paradigm and is able to perform data intensive processing on commodity machines while being cost effective at any size and ensuring redundancy and load balance [4]. It makes use of two powerful technologies - the MapReduce programming model [5] and the Hadoop Distributed File System (HDFS), and runs on a Java Virtual Machine (JVM).

TABLE I
ILLUSTRATION OF MAP REDUCE IN A WORD COUNT TASK.

KEY	VALUE
This	1
room	1
is	1
rather	1
big	1
however	1
it	1
is	1
not	1
as	1
big	1
as	1
Tom's	1
room	1

(a) Map results

⇒

KEY	VALUE
This	1
room	2
is	2
rather	1
big	2
however	1
it	1
not	1
as	2
Tom's	1

(b) Reduce results

A. MapReduce

As its name suggests, the programming model consists mainly of two steps - Map and Reduce. Map enables users to specify a mapping function which creates key/value pairs from the input data, whereas the reduce function aggregates the results from the Mapper. For example, in a word count task the MapReduce will break down a sentence as shown in Table I.

The sentence "This room is rather big however it is not as big as Tom's room." is broken down by the Mapper as illustrated in the leftmost table, after which the Reduce component takes all of the intermediate key/value pairs created by the mapping function and produces a collective result according to the keys, shown in the rightmost table.

B. Hadoop Streaming

This component of Hadoop allows learning algorithms implemented in any programming language (such as MATLAB or C++) to be used by the mapper or reducer components of Hadoop. In other words, a MapReduce job is created in Hadoop as if the learning algorithms were implemented in Java programming, when they are actually implemented in another programming language. Java is the default implementation of Hadoop. This highlights the *main advantage* of Map Reduce Paradigm in the form of Hadoop, compared to any other distributed implementation such as in [8].

The Hadoop Distributed File System (HDFS). It is an architecture to store files on a distributed file system shown in Fig. 1. The files are broken by default in data size of 64MB blocks and are then distributed across the file system. Furthermore, the HDFS replicates those blocks to ensure duplication. This distribution is optimal for data intensive processing over a number of machines when the order of processing is not important. The architecture has a number of different components as described below.

- NameNode: Considered to be the master; it directs DataNodes to perform low level Input/Output tasks. It also keeps track of all the files on the HDFS and the health of each of the DataNodes. This is where the bottleneck of the system is likely to happen.
- DataNode: Slave daemon to perform low level Input/Output tasks on the HDFS; any files on the HDFS are distributed between the DataNodes. DataNodes must constantly report to the NameNode and request instructions after a task is completed.
- SecondaryNameNode: Assistant to the NameNode for monitoring the state of HDFS, it communicates with the NameNode to take snapshots at regular intervals. It mitigates downtime and data loss in case the NameNode goes offline.
- JobTracker: Usually runs on the same machine as the NameNode and it determines the execution plan of submitted code. Breaks the job into tasks and then divides them to the different DataNodes. Furthermore, it monitors the tasks during runtime.
- TaskTracker: Resides within every DataNode, it is responsible to receive and manage tasks by the JobTracker.

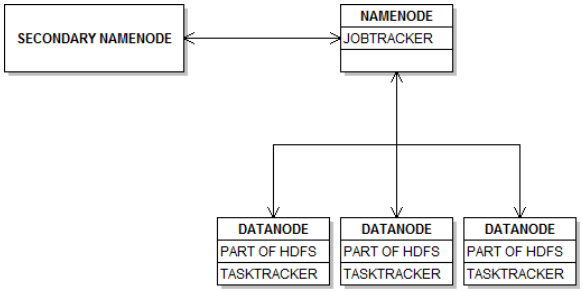

Fig. 1. HDFS model with three DataNodes

III. MapReduce for Support Vector Machines

Distributed support vector machine training mechanisms have already been proposed in [7][8]; this is not the main focus of this work. Our ultimate aim to provide an illustrative example of the Map Reduce paradigm by considering the support vector machine as the learning algorithm. In the context of Hadoop, Cascade SVM has been implemented [7], however, it is not clear how it was adapted to the Map Reduce computation and moreover, it did not make use of the streaming hadoop facility, which allows implementation of the machine learning algorithm in any programming language. Thanks to the generality of Map Reduce paradigm, any machine learning technique can operate in a distributed fashion, thereby addressing the problem of big data.

Support vector machine (SVM) remains, perhaps, one of the most popular classification algorithm [6]. It is a form of supervised learning used to analyze and match patterns. To train SVM, two sets of information are needed, raw data and class labels. By associating the data with the labels, SVM matches the patterns found in the data and classifies them accordingly by minimising the so-called 'structural' error. This model has proven to perform exquisitely in high dimensional problems.

During training, the data set is placed on a space map and the different classes are distinguished by (linear or nonlinear) decision boundaries (defined by the support vectors), as shown in Fig. 2. Similarly, during testing the data given are placed on the space map and according to the position of the data, its class is predicted by the SVM (illustrated in Fig. 3). The key feature of Hadoop is that it circumvents the need to break the classification problem into many sub-problems for distributed computing. This problem is addressed by Hadoop, which ensures the load of processing are equally distributed across all DataNodes.

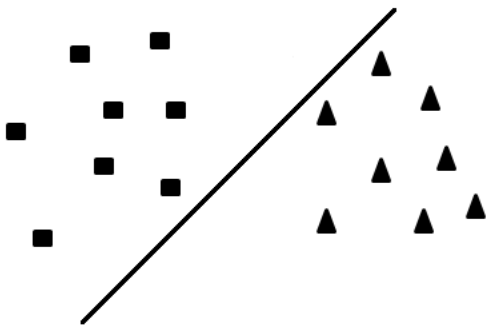

Fig. 2. Two class classification problem during training - rectangle and triangle classes.

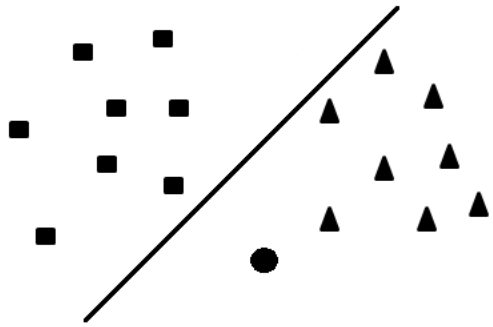

Fig. 3. Classification during testing - the circle (new data) is classified as a triangle.

A. Classification of Skin Segmentation

Skin segmentation was considered as the classification problem to demonstrate MapReduce paradigm for the SVM classifier. The dataset[2] is a collection of red-blue-green (RGB) images, which need to be categorised into two classes: Class 1 (skin) and Class 2 (non-skin). The attributes of the dataset are summarised in Table II.

TABLE II
ATTRIBUTES OF THE SKIN SEGMENTATION DATASET

Attribute	Range		Description
	Min	Max	
Blue	0	255	Colour intensity at pixel
Green	0	255	Colour intensity at pixel
Red	0	255	Colour intensity at pixel
Class	1	2	1 for skin pixel 2 for non-skin pixel

B. Distributed SVMs

Our Hadoop implementation invokes a traditional SVM algorithm written in Python and distributes its different processes across several DataNodes. Owing to the supervised nature of SVM, it requires a set of data and a set of labels associated with that data, and it is the NameNode that is responsible to distribute an instance of the SVM algorithm in each DataNode. Prior to that, the skin segmentation dataset must be uploaded on the HDFS. The large file is then be split into sub-blocks to be distributed evenly across all the DataNodes. An example of how the *labelled* RGB skin segmentation dataset is split into sub-blocks is illustrated in Fig. 4.

[2]Available at http://archive.ics.uci.edu/ml/datasets/Skin+Segmentation.

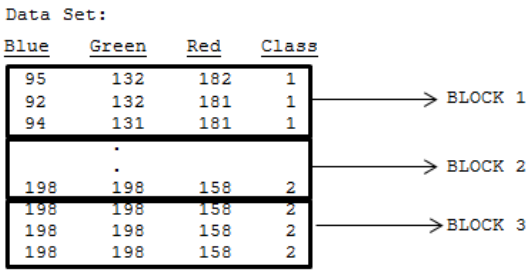

Fig. 4. The labelled skin segmentation dataset is split into sub-blocks by the NameNode.

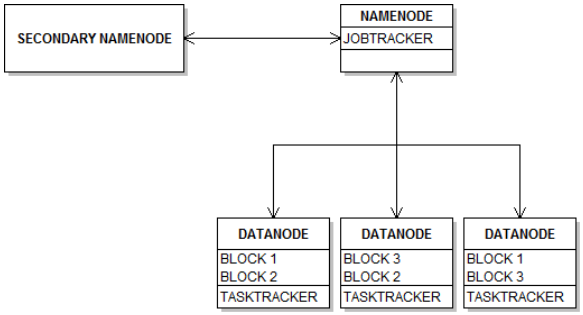

Fig. 5. The NameNode allocates the sub-blocks to the DataNodes, whereas the JobTracker monitors the classification task run on each DataNode.

C. Hadoop Setup

In our simulation, the number of DataNodes employed was varied from one to three. The NameNode and the JobTracker was set up on the same machine, and the NameNode acted as the server, issuing MapReduce jobs [5], which the JobTracker monitored.

Remark#1: For robustness against errors, Hadoop duplicates sub-blocks of data, e.g. in Fig. 5, Block 1 is forwarded to the leftmost and rightmost DataNodes. Redundancy is part and parcel of Hadoop; it is therefore likely that a Hadoop implementation running on one DataNode is likely to take more time than if it was implemented without MapReduce.

1) Training:

- Map: The primary task of the Mapper in each DataNode is to create Key/Value pairs - for the *Key*, the RGB values were separated by a comma, whereas the class was assigned as the *Value* as illustrated below.

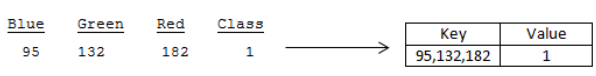

Fig. 6. Convertion of the data in Key/Value pairs

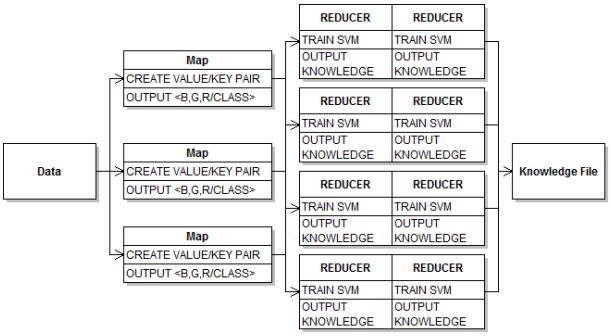

Fig. 7. Training process of the SVM on Hadoop

These Key/Value pairs were then distributed to the reducers for processing, as illustrated in Fig. 7.

- Reduce: This is where training of the SVM took place. Each reducer took a list of Key/Value pairs and first broke each Key to the original RGB values, storing them in variables. These variables were parsed as the data into the SVMs, and the Value was used as the label of the class. Note that the order that each data sample and label are provided to an SVM in each node does not affect the performance of the distributed SVMs.

Remark#2: Depending on the complexity of the classification task, there may be several Reduce steps in sequence for each DataNode. For simplicity, only two sequential Reduce steps are shown in Fig. 7, and only one Reduce step in Table I.

The accumulated knowledge from the training was written in a file for later use. Thread safety and multiple read/write operations are not an issue, for Hadoop handles all of the scheduling.

2) Testing:

- Map: The mapper for testing was similar to the one used in training, as only the Key/Value pairs were required. The only difference was that the class of the new data was not known. In terms of *Value*, it was assigned a value of 0.
- Reduce: As before each reducer received a list of Key/Value pairs and broke each key into the original RGB values. Then, the file with the accumulated knowledge from the training was loaded and parsed to an SVM instance. By parsing the RGB values the SVM was able to predict the class of these values. A new Key/Value pair was created holding the RGB values in the key and the class in the Value. The Key/Value pairs were outputted to a file. The distributed SVM for testing is summarised in Fig. 8.

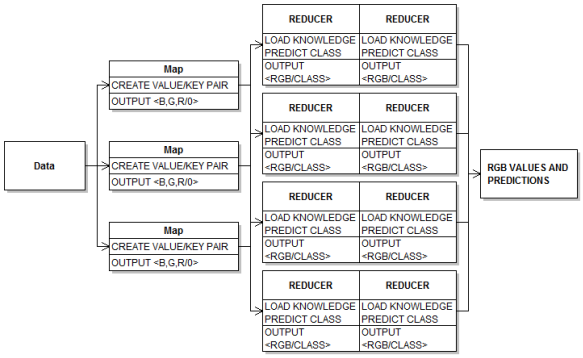

Fig. 8. Testing process of the SVM on Hadoop

Fig. 9. Run time of training and testing of SVM. 'Single' denotes nondistributed implementation, whereas '1', '2' and '3' denote the pseudodistributed SVM, and distributed SVM running over two and three machines respectively.

IV. Simulation

To show the efficacy of Hadoop, SVM was first run on a *single* machine *without* Hadoop, and then compared against a MapReduce-distributed SVM system with one, two and three DataNodes. For all four simulations, the accuracy was in the neighbourhood of 90% success rate. More importantly, the runtime shown in Fig. 9 decreased, as the number of DataNodes was increased.

Remark#3: Observe that the redundancy introduced by Hadoop in the classification task causes one DataNode implementation to be slower than if the SVM was implemented without MapReduce (labelled as *single* in Fig. 9). This observation conforms with Remark 1.

Remark#4: There was no major difference between the accuracy for each simulation ($\sim 90\%$ success rate). This suggests that running the SVM on a distributed framework does not hinder its accuracy, if implemented appropriately.

Remark#5: Owing to the 'knowledge file' learned from the training phase, the distributed SVM performed faster during the testing phase. However, the difference in running time between the training and testing was not that significant, as only the Reduce step benefited from the 'knowledge file'.

V. Conclusions

We have introduced the MapReduce paradigm proposed by Google in the context of support vector machines. In particular, we have firstly empowered the classifier support vector machine to operate in a distributed fashion *without* deriving long and tedious mathematical solutions, as in [7], [8]. Second, the generality of the MapReduce allows us to use any machine learning algorithms to operate in distributed fashion[3], whereas in [7], [8], these works were restricted to classification problems. Nonetheless, these work provided valuable initial insights into distributed SVMs. In the same spirit, it is hoped that our work has shed light on this emerging technology, which can be exploited in any machine learning endeavour for big data.

References

[1] IBM, *What is big data?*, http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html. Retrieved 26.08.2013.

[2] S. Owen, A. Robin, T. Dunning, E. Friedman, *Mahout in Action*, Manning Publications, 2011.

[3] X. Yang and J. Sun, *An analytical performace model of Map Reduce*, in Proceedings of the IEEE International Conference on Cloud Computing and Intelligence Systems, pp. 306 - 310, 2011.

[4] C. Lam, *Hadoop in Action*, Manning Publications, MEAP Edition, 2010.

[5] J. Dean, S. Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters*, In Proceedings of the 6th Symposium on Operating Systems Design and Implementation, Dec. 2004.

[6] C. Cortes and V. Vapnik, Machine Learning, *Support-Vector Networks*, Machine Learning, vol. 20, pp. 273-297, 1995.

[7] H. P. Graf, E. Cosatto, L. Bottou, I. Durdanovic, and V. Vapnik, *The cascade SVM*, in Proceedings of the Conference on Neural Information Processing Systems (NIPS), 2004.

[8] Y. Lu, V. Roychowdhury, and L. Vandenberghe, *Distributed Parallel Support Vector Machines in Strongly Connected Networks*, IEEE Transactions on Neural Networks, vol. 19, no. 7, pp. 1167-1178, 2008.

[3] An example implementation using the Hadoop Streaming API can be requested from the authors.