Scalable MapReduce-based Fuzzy Min-Max Neural Network for Pattern Classification

Shashikant Ilager
School of Computer and Information Sciences
University of Hyderabad
Hyderabad
Telangana, India
shashikant.ilager@gmail.com

Dr. P.S.V.S Sai Prasad
School of Computer and Information Sciences
University of Hyderabad
Hyderabad
Telangana, India
saics@uohyd.ernet.in

ABSTRACT

Fuzzy Min-Max Neural Network (FMNN) is a pattern classification algorithm which incorporates fuzzy sets and neural network. It is most suitable for online algorithms. Based on this, a MapReduce-based Fuzzy Min-Max Neural Network (MRFMNN) algorithm for pattern classification is proposed using Twister framework. MapReduce approach is used for scaling up the FMNN for massive large scale datasets. We used standard membership, expansion and the contraction functions of the traditional FMNN algorithm. The performance of the MRFMNN is tested by using several benchmark and synthetic datasets against the traditional FMNN. Results empirically established that MRFMNN achieves significant computational gains over FMNN without compromising classification accuracy.

Keywords

Neural Network, Fuzzy Sets, Classification, FMNN, MapReduce, Twister, MRFMNN

1. INTRODUCTION

The introduction of the MapReduce programming model [3], has changed the scenario of distributed computing by giving a framework for the development of scalable algorithms for big data problems. Converting the existing traditional algorithms to the MapReduce approach is the need of the hour to deliver quick results on large scale datasets. MapReduce framework incorporates data parallelism and utilizes the underlying distributed heterogeneous resources effectively. Several frameworks are designed to implement the MapReduce programming model. Google's Hadoop [3], Apache Hadoop [2], Apache Spark [14], [1] and Twister [4] are some of those. In this work, Twister is used to implement the FMNN for MapReduce approach.

The strength of the neural network, which is capable of simulating computers to work like a human brain and nervous system, has led to several innovative solutions in the field of artificial intelligence. After the introduction of Fuzzy sets [13], several computing models and algorithms have been proposed based on Fuzzy logic. The combination of fuzzy logic and the artificial neural network has resulted in hybrid intelligent system popularly known as Fuzzy Neural Network or Neuro-Fuzzy systems [11]. The combination of these systems makes use of the capability of each other and reduces their individual shortcomings. Fuzzy logic also helps neural networks to converge faster.

Fuzzy Min-Max Neural Network (FMNN) [10] was proposed by Simpson for pattern classification. It is a nonlinear, separable, online adaptive and one pass learning pattern classifier. MapReduce framework is more efficient for non-iterative algorithms. As FMNN is a single pass algorithm, a single MapReduce job is sufficient for constructing the classifier. Hence, FMNN is the best candidate for the construction of MapReduce-based classifier. Several improvements are proposed to the basic FMNN, and many refined solutions are given. Compensatory neuron based FMNN [9], center gravity data based FMNN [7], data core based FMNN [15] and enhanced FMNN [8] are some of those. All these different versions are suggested several changes to the basic FMNN to improve the performance and accuracy. In this work, we considered basic FMNN for conversion to the MapReduce approach.

The remaining paper is organized as follows: Section 2 gives an overview of fuzzy sets, traditional FMNN, and Twister. Section 3 describes proposed MapReduce approach to the FMNN. Results and analysis carried out are shown in Section 4. Finally, conclusions are drawn in Section 5.

2. BACKGROUND

2.1 Fuzzy Set

Theory of fuzzy sets was given by Zadeh [13]. Fuzzy sets represent the objects for the real-time nature where object precisely do not belong to a particular class. The membership function is proposed, which calculates membership value for all the objects. Unlike in traditional sets where objects are classified in a crisp manner where membership of an object to corresponding class is either 0 or 1, in fuzzy sets, objects have a partial membership for corresponding class ranging from [0,1]. The degree or probability of an object belonging to a particular class depends on the membership value. The membership value near to 1 indicates the object is more likely to belong to that class and 0 indicates that the object does not belong to the respective class.

ICDCN '17, January 04-07, 2017, Hyderabad, India

DOI: http://dx.doi.org/10.1145/3007748.3007776

2.2 Traditional FMNN

Simpson introduced pattern classification algorithm FMNN by using hyperbox fuzzy sets and neural networks [10]. It is a three layered neural network. FMNN algorithm is used for the construction of the classifier on datasets consisting of the numerical conditional attributes and categorical decision attribute.

Let C_1, C_2... C_m denote the decision classes and let n be the number of conditional attributes. As a preprocessing step, all attributes are normalized to the unit range $(0, 1)$, hence the entire space of the objects becomes an n dimensional unit cube I^n.

In the process of learning, hyperboxes are constructed such that all objects in a hyperbox belong to a unique decision class. A Hyperbox is a n dimensional cube represented by two n dimensional points V and W denoting the minimum corner and the maximum corner as shown in the Figure 1.

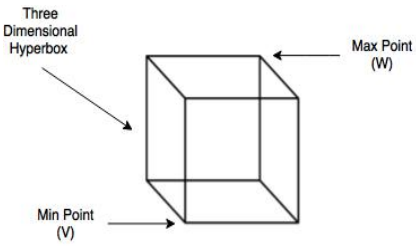

Figure 1: A Three Dimensional Hyperbox with Minimum Point V and Maximum Point W

The resulting FMNN, comprising the hyperboxes is represented as a three layer neural network as depicted in the Figure 2.

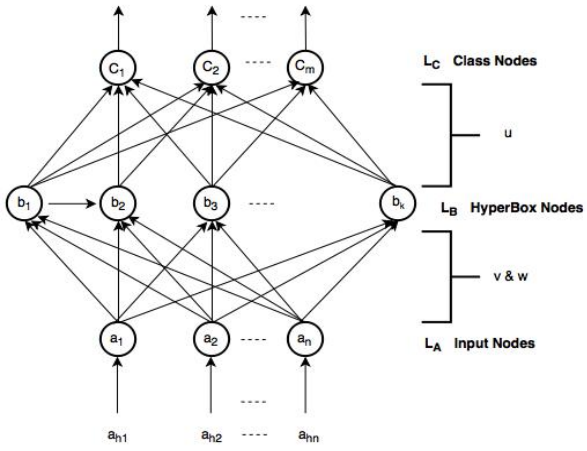

Figure 2: Traditional FMNN- A Three Layer Neural Network

The n dimensional input layer L_A corresponds to receiving the data for one training object with n conditional attributes. A node in a Hidden layer layer L_B represents an hyperbox and L_A and L_B are fully connected. The connecting edge from i^{th} input node to the j^{th} hidden layer node is associated with the two weight values v_{ij}, w_{ij} denoting i^{th} component of minimum (V) and maximum (W) points of the hyperbox b_j.

The output layer is of the size m corresponding to the m decision classes. As each hyperbox belongs to a single decision class, the weight vector u coming out of a hyperbox in a hidden layer is a boolean vector having all zeroes except in for the connection to its decision class.

The weights between the input layer to the hidden layer are represented by minimum point matrix (v) of size n/k and maximum point matrix (w) of size n/k. The boolean vector matrix between hidden layer and the output layer is represented by u of the size k/n, where k is the total number of hyperboxes constructed at the hidden layer.

2.2.1 FMNN Training Algorithm

FMNN neural network training comprises three steps. From the training dataset, each instance or object is passed to the hidden layer. The memebrship value of an object is caluclated using the membership function [10]. Let $A_h = (a_{h1}, a_{h2}, ..., a_{hn}) \in I^n$ is the h^{th} input training object, $V_j = (v_{j1}, v_{j2}..., v_{jn})$ is the minimum point for hyperbox B_j and $W_j = (w_{j1}, w_{j2}..., w_{jn})$ is the maximum point of the B_j. γ acts as sensitivity parameter that controls how fast membership value decrease as the distance between A_h and B_j increases. Let $b_j(A_h)$ is membership function for the j^{th} hyperbox $0 \leq b_j(A_h) \leq 1$, it is defined as the following equation 1.

$$b_j(A_h) = \frac{1}{2n}\sum_{i=1}^{n}[max(0, 1 - max(0, \gamma min(1, a_{hi} - w_{ji}))) + max(0, 1 - max(0, \gamma min(1, v_{ji} - a_{hi})))] \quad (1)$$

Initially, in case if no hyperbox of the corresponding class exists, a new point hyperbox is created with the same dimension as that of the current instance of the training object ($V_j = W_j = A_h$). Otherwise, fuzzy membership value is calculated for all the hyperboxes belonging to the same class.

For every training object, membership values are calculated for the corresponding hyperboxes (hyperboxes which belong to the same decision class). The training object is included to the hyperbox to which it is fully contained (membership value = 1). If the training object is not fully contained in any of the hyperbox, then the nearest hyperbox is identified which has the highest membership value. The expansion criteria are checked for the identified hyperbox. If the expansion criteria are satisfied, the hyperbox is expanded to contain the training object. If the expansion criteria are not satisfied, a new point hyperbox is included in the hidden layer such that $V_j = W_j = A_h$. After every expansion, overlap test is carried out to identify the overlapped region between two hyperboxes belonging to two different classes. If overlap exists, the contraction is done to remove the overlapped region. The detailed equations for expansion test, overlap test, and contraction process are present in [10].

2.2.2 FMNN Testing Algorithm

In FMNN testing algorithm, the test case or test object is passed to the all the neurons (hyperboxes) at the hidden layer. Fuzzy membership value is calculated for each hyperbox. At the output layer, the highest membership value

instance is chosen among multiple membership values. The corresponding class of the hyperbox is the class of the test case.

2.3 Twister: An Iterative MapReduce Framework

MapReduce is a programming model which facilitates to deal with the large size data. Twister[4] is a light-weight framework which implements MapReduce programming model. Unlike popular framework like Hadoop [12], Twister also supports in-memory computation for the iterative MapReduce algorithms. It effectively keeps all the initial loaded data in the main memory until all the iterations are over, hence reducing the cost of the I/O and computation time.

Twister framework has components like Driver, Mapper, Reducer, Collector, and Combiner. The driver acts as the main program. Twister categorizes the data into two types. First, the static data, which was loaded initially and kept in memory until all iterations are over. Second, dynamic data that changes over iteration. The static data which is huge in size is partitioned into several parts and kept in distributed locations in the cluster. All mappers load associated static data and produce intermediate $< key, value >$ pairs; these values are written to the collector and later passed to the reducers. Reducers collect the data from collector based on group by key operation. In the group by key operation, values are grouped based on the similar key and this $< key, listofvalues >$ will be input to the reducer. Reducer performs aggregation or summation of these list of values. Reducer output $< key, aggregatedvalue >$ is written to combiner where it acts as the global reducer for all the reducers output. The combiner produces the final result which need not be in the form of $< key, value >$. The driver gets the result from the combiner. Based on some condition, the driver decides about further iterations.

Even though in this work we are not using iterative MapReduce approach, we used Twister because it provides higher granularity for the *map* tasks. The configurable *map* tasks allows them to work with the large chunk of data and to produce the intermediate result. Basic Hadoop primarily focuses on fine-grained granularity for *map* tasks hence increases the size of intermediate data and causing more network latency. This is evident from the result obtained from [5], it is shown that even for non iterative (single MapReduce job) MapReduce application like CLARA, Twister outperformed Hadoop in terms of computation time. Other than this advantage, the proposed solution is generic in nature and implementable on all the frameworks which support MapReduce such as Hadoop [12], Spark [14], [1] and etc.

3. SCALABLE MAPREDUCE-BASED FMNN

MapReduce-based FMNN (MRFMNN) is the proposed algorithm for scaling FMNN using Twister's MapReduce framework. The input dataset is preprocessed before training the neural network.

3.1 Data Pre-Processing & Partitioning

The complexity of FMNN on training one object is directly proportional to the number of hyperboxes created thus far. But in the pattern space I^n, because of parameter γ, a hyperbox which is sufficiently away from current training object may not have any impact on the training, i.e., because of the inherent locality aspect of hyperboxes influencing each other in the training process. One can reduce the complexity of training by considering only the near by hyperboxes instead of the entire collection without compromising the classification accuracy of the system.

With this motivation and to incorporate parallelism through MapReduce, we are proposing a space division approach in which, based on the nature of training data the pattern space I^n is subdivided. The FMNN algorithm works parallel on each subspace by working on the only subset of training data falling into that space. When we used fixed cutpoint of 0.5 in a dimension to divide the space into subspaces, the result of the division was highly unbalanced due to the scattered training objects. Hence different mappers work with the varying sizes of data. To overcome this, we used the median of the dimension as the cutpoint.

The space division approach is given in the Algorithm 1. In our approach, the user needs to provide the number of mappers that are available in the cluster. Usually, this is constrained by the number of processing cores available including all the nodes in the cluster.

Algorithm 1 DataPartiton

DataPartiton
Input: Training dataset, N : Number of Mappers
Output: Partioned Files

1: Randomly select $\log_2 N$ number of dimension for dividing the space.
2: **for all** training dataset instances **do**
3: **for all** $i = 1 to \log_2 N$ number of dimensions **do**
4: $file_{id} =$ "";
5: **if** current training object in i^{th} dimenion $<$ median of i^{th} dimension of the dataset **then**
6: $file_{id} = Concatenate(file_{id}, 0)$;
7: **else**
8: $file_{id} = Concatenate(file_{id}, 1)$;
9: **end if**
10: **end for**
11: Write the training object into file associated with decimal of the binary $file_{id}$
12: **end for**

For the N number of mappers, an equal number of partitioned files are generated by dividing the pattern space I^n into N subspaces. The $file_{id's}$ are generated of the length of $\log_2 N$ binary digits, hence N number of such $file_{id's}$ are generated from $\log_2 N$ binary digits for each subspace. All the objects are written to the file of their respective $file_{id}$. During runtime, each mapper gets a single partitioned file as input. Hence, effectively all mappers work in parallel with different subspaces.

The effect of space division assures that there will not be any overlap between hyperboxes belonging to two different space partitions. This is because two points in two different space partitions satisfy to belong to on two different sides of the median cut point on at least one dimension from the set of dimensions used for space division. Hence on that particular dimension, no overlap can exist between hyperboxes constructed in these two space partitions. Hence, aggregating hyperboxes from different space partitions which are locally constructed will not affect the classification accuracy. But there is a chance for the larger number of hyperboxes being formed as there exist a possibility of the merger of

hyperboxes across the boundary of space partition in the expansion step. However, In this work, we do not consider the merging of hyperboxes across different subspaces in order to reduce the computational overhead and complexities with respect to the distributed environment. If the difference in cardinality of hyperboxes constructed through our proposed approach is similar to or slightly higher than the cardinality of hyperboxes constructed in the traditional approach, then the gains obtained in training time will compensate the disadvantage resulting from larger cardinality of hyperboxes. More importantly, this way of distribution helps in working with large datasets which can not fit into a single system's main memory.

3.2 MRFMNN Training

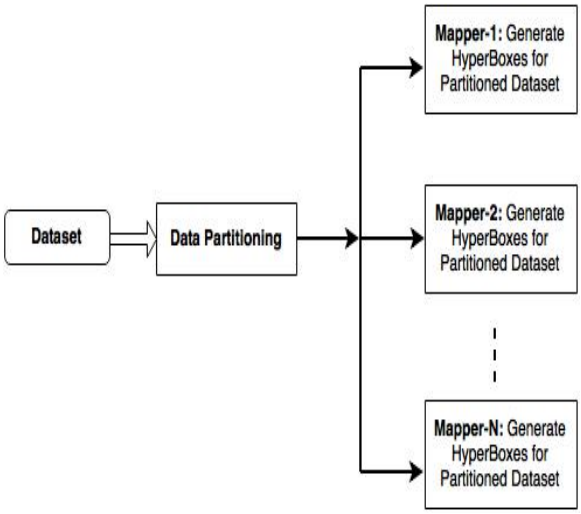

Figure 3: MRFMNN Training- A Map Only Approach

The flow of MRFMNN training algorithm is shown in Figure 3. The training part contains Map only approach. After generating hyperboxes locally at each mapper, the result is not aggregated at a single plcae since MRFMNN testing also follows distributed approach.

The partitioned training datasets will be loaded into all the mappers of Twister as the static data component, and all these mappers follow traditional FMNN training process. The hyperboxes are expanded and checked for overlap. If an overlap exists, the contraction is done to remove the overlap. All the mappers perform these processes locally and generate the hyperboxes. Once the complete network is trained, the final information of hyperboxes is written to the local disk of the respective machines.

In Algorithm 2 of MRFMNN training, we also propose a better performance logic for the FMNN. Instead of performing the overlap test on an inclusion of each training object to the network, we perform overlap and the contraction once the complete network is trained with all the training objects. In the case of the large size datasets, we observed that the total number of overlap tests resulting in contraction step is much smaller compared to the size of the training dataset. Hence, this introduces computational overhead in training process. This improved step will reduce significant computation time as it minimizes the number of overlap tests without compromising with the accuracy of the classification.

Algorithm 2 MRFMNN Training: A Map only Approach

Map$_i$ $\forall_i$ { i = 1 to Number of partitioned datasets}
Input: Training dataset from partiotioned dataset$_i$ obtained from dataset preprocessing.
Output: Set of hyperboxes, minimum point V, maximum point W, and class label C_x.

1: **for all** input dataset instances **do**
2: Calculate $Membership\_value$ for all the hyperboxes of same decision class
3: **if** $Membership\_Value == 0$ **then**
4: Create a new point hyperbox
5: **else**
6: Find a hyperbox of the same decision class which gives highest $Membership\_Value$
7: Expand the hyperbox if expansion criteria are satisfied
8: **end if**
9: **end for**
10: For a pair of hyperboxes belongs to two different classes, perform overlap test. If overlap exists, perform contraction.
11: Write all the hyperbox information to the local disk

3.3 MRFMNN Testing

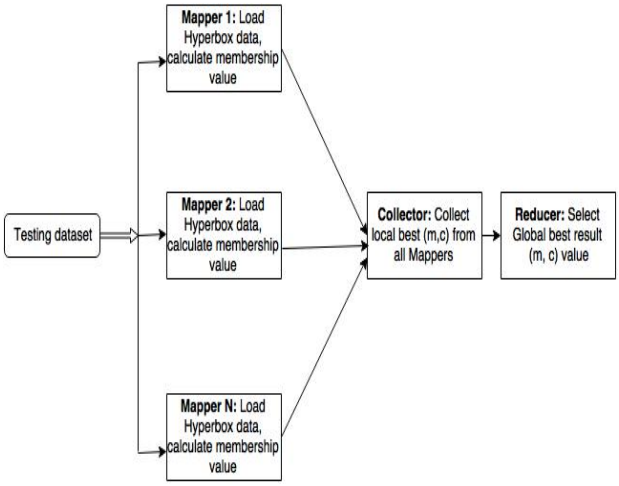

Figure 4: MRFMNN Testing: A MapReduce Approach

The flow of the MRFMNN Testing is shown in Figure 4. For the given large datasets, number of hyperboxes generated are usually large in number. The complexity of classifying a test case is directly proportional to the number of hyperboxes. Hence, the MRFMNN testing algorithm also follows distributed MapReduce approach to gain speed up by utilizing the distributed resources.

An equal number of mappers are used as that of training. All the test objects are broadcasted to all the mappers. As shown in Algorithm 3, Mappers load the respective hyperbox information which was generated during the training process. The Mapper process calculates membership value for all the test objects on all the hyperboxes. The local best

result, the one which has highest membership value, and corresponding class label of that hyperbox is the output of the Map function. All the mappers for every test object write the output, i.e., local best membership value and associated class label (m,c) pair to the collector process with some dummy key.

Algorithm 3 MRFMNN Testing: A MapReduce Approach

Map$_i$ $\forall_i$ { i = 1 to Number of partitioned hyperbox data}
Input: Testing dataset
Output: list of local best result, (m,c) for all test objects in dataset

1: Load the hyperbox information from local memory
2: **for all** test input data instance from testing dataset **do**
3: **for all** hyperboxes $1 : n$ **do**
4: calculate the membership value
5: **end for**
6: Find a local best result, which has highest membership value
7: **end for**
8: Write the local best result (m, c) for all objects to the Collector with some dummy key

Reduce
Input: List of (m,c) values for each test object of test dataset from all mappers.
Output: Class label for test objects in dataset.

1: **for all** test objects in test dataset **do**
2: Identify the global best result for all test objects, which has the highest membership value from the list
3: **end for**
4: Return classified class label for all the test objects to the Twister's Combiner.

Reducer gets the list of values (m,c) pairs for all test objects as the value object from the collector. The value object has the list of membership values for each test object. It filters out the value which has the highest membership value and it is considered as a global best result. Similarly, for all the test objects, the global best result is identified. Finally, Reducer writes the membership value and class label for each test object (m,c) pair as a list of values to the combiner, and the driver gets final output from the combiner.

4. EXPERIMENTS AND RESULTS

4.1 Experimental Setup

The experiments are carried out in a cluster. The experimental setup consists of 4 computing nodes. Each computing node has Intel i5 processor with clock speed of 3.2 GHz, 4 cores each, and have 4 GB of primary memory. One among the four nodes was configured as both master and worker node and remaining three were only worker nodes. These nodes were installed with OpenSuse 13.2 OS with Java 1.7.0_75 and cluster was set up using Twister 0.9 and ActiveMQ as message broker running in one of the nodes. In all the experiments with MRFMNN, we have used 16 Mappers in the training. In testing, the same number of mappers and a single reducer is used.

4.2 Datasets

To check the accuracy of pattern classification of MRFMNN, standard datasets are chosen from the UCI repository [6]. The dataset descriptions are shown in the Table 1.

Table 1: Dataset Information

Datasets	No of Dimensions	No of Attributes
Iris	150	4
Wine	178	13
Ionosphere	351	34
Segment	2310	19
Shuttle	58000	9
SatImage_100	600000	36
Synthetic	400000	100

To evaluate the scalability of MRFMNN on big datasets, we have extended the standard dataset Satellite Image (SatImage) from 6436 instances to 6 lakh (1 lakh = 100 thousand) instances by replicating into 100 times (SatImage$_{100}$). A Synthetic dataset is also generated which consist of 4 lakh instances and 100 attributes; this Synthetic dataset has two class labels, class 1 has randomly generated attributes ranging from [0,0.7] and class 2 has attributes ranging from [0.4,1.0]. The overlap is maintained between two classes which usually exists in real datasets also, this helps to evaluate the accuracy of classification in a more natural way.

4.3 Results and Analysis

For all the datasets, 70% of data is used for training the neural network and remaining 30% is used for testing the neural network. The stratified sampling is used to partition the data into training and test datasets. In both the FMNN and MRFMNN implementation, the sensitivity parameter γ of the membership function and the hyperbox expansion threshold parameter θ are set to 4 and 0.3 respectively, these are experimentally tuned parameters. The MRFMNN classification accuracy is compared with traditional FMNN (sequential) for all the datasets. The results are tabulated in Table 2, the number of hyperboxes created in both traditional FMNN and MRFMNN are tabulated in Table 3.

Table 2: Accuracy Comparison between Traditional FMNN and MRFMNN

Datasets	Traditional FMNN	MRFMNN
Iris	95.5%	100%
Wine	96.26%	100%
Ionosphere	93.39%	91.50%
Segment	92.06%	92.92
Shuttle	99.75%	99.81%
SatImage$_{100}$	90.43 %	89.56%
Synthetic	100%	100%

Table 3: Comparison of Number of Hyperboxes Created

Datasets	Traditional FMNN	MRFMNN
Iris	9	12
Wine	26	36
Ionosphere	59	67
Segment	23	49
Shuttle	26	50
SatImage$_{100}$	226	253
Synthetic	114276	114786

Results have shown that the classification accuracy of MRFMNN is comparable to FMNN. The observation can be made that the classification accuracy of MRFMNN for the datasets Iris and Wine is improved compared to the traditional FMNN. Even though hyperboxes constructed through MRFMNN and FMNN are based on the same training dataset, the resulting hyperboxes are not identical and hence raise the differences in classification accuracies.

It is important to note that due to the splitting of the space in multiple dimensions, the number of hyperboxes created is slightly increased compared to traditional FMNN. The division of pattern space results in the split of those hyperboxes that may form across the division axes between the subspaces. However, the difference in a number of hyperboxes is not huge, indicating MRFMNN does not significantly increase complexity at the hidden layer.

Changes resulting in hyperboxes because of space division in MRFMNN is not always resulting in better classification accuracy. For example, in the case of datasets Ionosphere and SatImage_{100} classification accuracy is slightly decreased compared to FMNN. The reasons for this will be investigated in the future.

4.4 Scalability Test with Large Datasets

For smaller datasets which can be computed in a single system easily may not get computational gain with our approach. The first four datasets in Table 1 are the very small size in nature and result in a small number of hyperboxes. So the communication in MapReduce approach causes more overhead compared to actual computation. In such cases, FMNN gives better computational gain compared to MRFMNN. Hence MRFMNN is relevant for large and very large decision systems.

The scalability test with large datasets has been conducted to compare the computational gain with respect to time while training the neural network. Experiments on MRFMNN for SatImage_{100} and Synthetic datasets are performed on single node and in the cluster of 4 node and resulting computational time for training are compared with traditional FMNN which runs sequentially on a single system. Results are summarized in Table 4.

Table 4: Computation Time Comparision between Traditional FMNN and MRFMNN

Datasets	Traditional FMNN (in sec's)	Space Division (in sec's)	MRFMNN 1 node (in sec's)	MRFMNN 4 nodes (in sec's)
SatImage_{100}	9.09	13.60	5.94	2.80
Synthetic	27717	44.63	1420	1189

Based on the results in Table 4, it can be observed that MRFMNN for the SatImage_{100} has achieved significant computational gains 34.65% over traditional FMNN in the single system alone. The computational gain has increased in the cluster environment (69.19%). Speed up of 2.12 on the single system and 3.24 on the cluster is observed. For the Synthetic dataset, speed up of 19.51 and computational gain of 94.16% is achieved between MRFMNN with the single system and traditional FMNN. But speedup for the synthetic dataset is not significantly scaled up on MRFMNN in the cluster compared to MRFMNN on a single node. Further analysis revealed that the space division in the synthetic dataset is more imbalanced across mappers compared to SatImage_{100} dataset. This is resulting in the delay for reducer invocation and increased communication overhead. In future, better space division approach will be investigated to overcome this problem.

The cardinality of hyperboxes resulted by FMNN is influenced by not just the cardinality of objects and attributes but by the purity (nonoverlapping of different class regions) of the dataset and so is the influence of the dataset on the training time. Owing to this reason, because SatImage_{100} is replicated dataset, the increase in the size of the dataset has not decreased the purity of the dataset and hence resulted in the fewer number of hyperboxes as equal to the single instance of SatImage dataset. However, the influence of replication on file I/O in space division algorithm is evident from Table 4. But in the case of dataset resulting in higher training complexity, the preprocessing time incurred for space division is negligible as evident in results obtained from the Synthetic dataset.

To analyze the computation time of testing, a comparative analysis is done for testing on MRFMNN in the cluster and traditional FMNN in a single system. A single test case of SatImage_{100} dataset incurred 0.106 seconds in traditional FMNN whereas it took 0.626 seconds on MRFMNN in the cluster. Similarly, a test case of the Synthetic dataset with traditional FMNN incurred 8.53 seconds whereas MRFMNN took 1.59 seconds in the cluster.

The result is expected as SatImage_{100} training results in only 253 hyperboxes whereas Synthetic dataset results in 114786 hyperboxes. Hence testing with MapReduce approach is recommended for the classification which results in a large number of hyperboxes. In the case of fewer hyperboxes, cluster communication overhead is overcompensated by actual computation.

5. CONCLUSION

A MapReduce-based distributed MRFMNN algorithm is proposed for the large size datasets which cannot fit in single system memory. Experiments and comparative analysis of the results have shown that our MRFMNN classification accuracy is comparable to the FMNN by achieving more computational gain. We can conclude that MRFMNN is most suitable and highly scalable classifier for massive scale huge datasets over traditional FMNN.

In future, we investigate other approaches for pattern space division to obtain the balanced subspaces such that all mappers get equal workload to obtain better speedup.

6. REFERENCES

[1] Documentation| apache spark. http://spark.apache.org/documentation.html. Accessed: 2016- 07- 16.

[2] M. A. Bhandarkar. Mapreduce programming with apache hadoop. In *IPDPS*, page 1. IEEE, 2010.

[3] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[4] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. C. Fox. Twister: a runtime for iterative mapreduce. In S. Hariri and K. Keahey, editors, *HPDC*, pages 810–818. ACM, 2010.

[5] P. Jakovits and S. N. Srirama. Evaluating mapreduce frameworks for iterative scientific computing

applications. In *HPCS*. IEEE, 2014.

[6] M. Lichman. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2013.

[7] D. Ma, J. Liu, and Z. Wang. The pattern classification based on fuzzy min-max neural network with new algorithm. In J. Wang, G. G. Yen, and M. M. Polycarpou, editors, *ISNN (2)*, volume 7368 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 2012.

[8] M. F. Mohammed and C. P. Lim. An enhanced fuzzy min-max neural network for pattern classification. *IEEE Trans. Neural Netw. Learning Syst.*, 26(3):417–429, 2015.

[9] A. V. Nandedkar and P. K. Biswas. A fuzzy min-max neural network classifier with compensatory neuron architecture. *IEEE Trans. Neural Networks*, 18(1):42–54, 2007.

[10] P. K. Simpson. Fuzzy min-max neural networks. i. classification. *IEEE Trans. Neural Networks*, 3(5):776–786, 1992.

[11] J. Vieira, F. M. Dias, and A. Mota. Neuro-fuzzy systems: a survey. In *5th WSEAS NNA International Conference*, 2004.

[12] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, 2. auflage. edition, 11 2010.

[13] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[14] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In E. M. Nahum and D. Xu, editors, *HotCloud*. USENIX Association, 2010.

[15] H. Zhang, J. Liu, D. Ma, and Z. Wang. Data-core-based fuzzy min-max neural network for pattern classification. *IEEE Trans. Neural Networks*, 22(12):2339–2352, 2011.