FUDMA Journal of Sciences (FJS)
ISSN online: 2616-1370
ISSN print: 2645 - 2944
Vol. 7 No. 5, October, 2023, pp 156 - 164
DOI: https://doi.org/10.33003/fjs-2023-0705-1691

DEVELOPMENT OF AN ENHANCED C4.5 DECISION TREE ALGORITHM USING A MEMOIZED MAPREDUCE MODEL

***[1]Paul, O. Florence, [1]Obiniyi, A. Afolayan, [1]Donfack-Kana, F. Armand and [2]Paul, D. Elaoyi**

[1]Department of Computer Science, Faculty of Physical Sciences, Ahmadu Bello University, Zaria,
[2]Department of Chemistry, Faculty of Physical Sciences, Ahmadu Bello University, Zaria,

*Corresponding authors' email: bimione2004@gmail.com

ABSTRACT

Classification technique in data mining concentrates on the prediction of categorical or discrete target variables which is designed to be handled by the classical C4.5 decision tree algorithm, an algorithm whose aim is to produce a tree which accurately predicts the target variable for a new unseen data. However its recursive nature poses a limitation when huge volume of dataset is involved; making computation more complex and resulting in an inefficient implementation of the algorithm in terms of computing time, memory utilization and data complexity. Meanwhile, several researches have been done to control these limitations. One of such improvements is the parallelizing of the algorithm using the MapReduce model. This involves dividing the large dataset into smaller units and sharing them on multiple computers for parallel processing, but the recursive nature of the algorithm makes the cost of computing large number of repeated calculations quite high, which is our concern in this work. . This research is aimed at reducing computation time further, by using a memoized MapReduce model, which involves the saving of the result of previous calculations in a cache; hence, when same calculations are encountered again, the cached result is returned, thus re-computation is avoided. The cached result is considered a reduced cost compared to the computational cost of re-computation.

Keywords: Data mining, C4.5 Algorithm, Memoization, MapReduce, Hadoop

INTRODUCTION

The emergence of powerful computing and data storage technologies has led to the explosion of massive data in the area of medicine, agriculture, education, business and in several other fields. It is important to note that the generation of this huge amount of data can become a great challenge if not properly harnessed (Jaseena *et al*., 2014). In the bid to solve this problem, Gregory Piatetsky-Shapiro (a Data Mining Expert) saw the need to discover useful knowledge from these databases and this led to the first workshop on Artificial Intelligence which took place in Detroit in 1989 where he coined the phrase "Knowledge Discovery in Databases" (KDD) (Piatetsky-Shapiro, 1991). A phrase which deals with the overall process of obtaining useful knowledge from data (Fayyad *et al.,* 1996). Data mining is a method in KDD process which involves the application of specific algorithms for developing patterns from data. Majority of these algorithms are limited in their capacity to deal with large volume of data, therefore, the need to improve them has become of great importance, so that their efficiency can be enhanced.

Research has shown that most of these algorithms have undergone series of improvements in order to enhance their performance with regards to accuracy, execution time as well as memory management. These algorithms perform well when the data is small but does otherwise when the data is large (Han and Kamber, 2006). C4.5 decision tree algorithm is an example of such algorithms, it is an algorithm which is widely used in classification due to its ability to perform well in the prediction of unknown variables from data that is already known. It is easy to use and the cost of implementing it is less expensive compared to other classification algorithms. The algorithm has undergone series of improvements in order to enhance its efficiency in terms of time and storage capability. However, the current performance of the algorithm shows that it suffers from high computational time and memory due to storage requirement.

To address this, this study propose to apply a novel method to the high memory problem through memorized MapReduce model. The remaining part of the paper is organized as follows: The review of related work and concepts is presented in section 2, while a detailed presented of methodology used in the study are discussed in section 3. In sections 4 and 5, experimentation, results and discussions are outlined, while conclusion is drawn in section 6 to emphasize the findings from the study.

Related Works

The work of Dai and Ji (2014) improved the C4.5 algorithm through the introduction of the MapReduce programming model,which involved the parallelization of the algorithm through distributed computing, which resulted in time efficiency and scalability of the algorithm. Data structures were also designed to minimise communication cost which was generated by the movement of some computations to the external storage due to memory limitation. Wang *et al*., (2016) proposed the classical SPRINT algorithm which was a scalable and parallel method of the C4.5 decision tree. The authors came up with a new method for improving the calculation process by looking for a better candidate segmentation point for the discrete and continuous attributes. Muslim *et al*., (2017) also proposed an improvement of the C4.5 algorithm for breast cancer diagnosis. In order to enhance the accuracy of the algorithm, they combined the Particle Swarm Optimization (PSO) algorithm with C4.5 algorithm. The combination of the two algorithms was used to enhance attribute selection in the C4.5 algorithm, for a better performance of the algorithm. Rajeshinigo and Jebamaler, (2017) also improved on the accuracy in prediction of the C4.5 algorithm by using the K-means clustering algorithm to change continuous values into categorical values. Cherfi *et al*., (2018) proposed a novel method named VFC4.5, implying a Very Fast C4.5 algorithm. It was meant to speed up and increase the performance of the algorithm. VFC4.5 was able to handle the problem of the C4.5

algorithm in managing continuous values through the introduction of some of the statistical tools (for example the mean and the median), as an ancillary to the original process of finding the threshold in the C4.5 algorithm. Badgujar and Sawant, (2017) improved the rule of C4.5 through the use of L'Hospital Rule, which involved the use of an approximation method to simplify the calculation process by removing the logarithmic calculation involved in calculating the Gain-Ratio in the C4.5 algorithm and approximating them with less complex operations. Mu *et al*., (2017) proposed a parallelized C4.5 decision tree algorithm built on MapReduce (MR-C4.5-Tree) in order to solve the problem of memory restriction and time complexity owing to large dataset. This research work recommends an enhanced C4.5 decision tree algorithm using a Memoized MapReduce model, which integrates a memoization technique with the parallelized C4.5 decision tree algorithm. This integration is meant to improve the existing algorithm by further reducing the execution time, thereby improving the performance of the algorithm. The perceived trade-off as observed in the study is the cache occupancy memory cost.

Memoization

Memoization is a technique from the concept of dynamic programming, it used in computer programming to optimize the performance of a function by caching its result for later use. In other words the function stores the results of its calculations in the memory so that when the same inputs occur again, the cached result is returned. The term "Memoization", which means to memorise or remember was first introduced by Donald Michie in the 60's. His work was the first to show the benefit of storing the result of a function in a table and re-using it as the need arises (Suresh *et al*., 2016). He proposed an effective Memoization in generalizing the context of subproblems by using interpolants. This was meant to tackle, the difficulty of effective reuse in subproblem solutions within the field of dynamic programming (Jaffar *et al*., 2008). A summary of the generalizations and its optimal solutions was stored in a memo table to be reused when it encounters a subproblem with a similar context in the search tree. It is an approach that proffers hands-on problem-solving tools in several application areas such as Artificial Intelligence, for instance in Machine Learning Applications, it can be used to speed up the training of models. It can also be used in Network optimization, decision analysis, inventory problems, computer science, agriculture, finance and medicine (Sniedovich, 2004). A major disadvantage of this technique is the overhead incurred on function calls. It trades memory for speed, it uses more memory by storing the results of previous computations in order to remove some code. execution. We anticipate a slight memory swell during computation when results are kept in cache for avoidance of re-computation.

The Classical C4.5 Decision Tree Algorithm

The traditional C4.5 algorithm which is used in building decision trees and generating classification rules used in decision making, is an improvement to a previous algorithm created in 1986 by Quinlan Ross, namely the ID3 algorithm (Iterative Dichotomiser 3) (Quilan, 1993). The C4.5 algorithm was rated one of the top ten most influential data mining algorithms in the research community (Wu, X., Kumar, V., Ross Quinlan, J. *et al*., 2008). It is a standard algorithm that employs a divide-and-conquer method to grow decision trees, this process is based on information gain ratio which is evaluated by entropy. The measure of information gain ratio, which is also referred to as feature selection measure is used to select the test attribute at each node in the tree, where the attribute with the maximum information gain ratio is chosen as the best feature for the current node (Seema *et al*., 2013).

Hadoop

Hadoop is an open source software framework, which enables data storage and the parallel distribution and processing of large dataset on a group of interconnected computers, using the Hadoop Distribution File System (HDFS) and MapReduce programming model respectively. Its ability to store data and run applications for data computation on the same computer within the network gives it the capability to process large volumes of data efficiently. Hadoop provides an uncomplicated easy to use tool for the purpose of storage and analysis of huge datasets as well as the management of large number of computers. In addition to this, the components within the Hadoop framework are built to handle hardware failures. Its fault-tolerant capability enables the system to keep working uninterruptedly despite the failure of one or more machines within the network, this is made possible by what is known as data replication or duplication in Hadoop.

MapReduce

The MapReduce programming paradigm is used in Hadoop for the parallel and distributed processing of large datasets on a network of computers (Dean and Gemawat, 2008; Zhu *et al*., 2018). Users are able to manage a large amount of information because of its flexibility and capacity to tolerate faults while processing these data (Wang *et al*., 2014). Its computation has two sides, namely Map phase and Reduce phase. The former plays a crucial role in the processing and analyzing of large datasets, at this phase a user defined function known as mapper function is applied on the input data from the HDFS which are stored as input files. The InputFormat which an important component in the map phase specifies how these input files are used for input by creating InputSplits, each split is divided into records which are meant to be processed by the mapper i.e. a record per mapper. The RecordReader which is also a built-in component of this phase converts the input data into a <key, value> pair format which is suitable for the mapper to process. The mapper processes each record in parallel and generates a new <key, value> pair completely different from the input pair. The new <key, value> pair is referred to as an intermediate output and it is written to the local hard disk because it is a temporary data. In between the mapper phase and the reducer phase is the partitioning stage which partitions the mapper output based on the key. The partitioner does this, by ensuring that all the values for a particular key are grouped together and sent to the same reducer such that an even distribution of the map output is allowed over the reducer. The combiner like the partitioner also acts on the mapper output, although it is optional, its main function is to sum up the map output according to their key values. The purpose of this function, is to assist in saving as much bandwidth as possible by reducing the amount of data that needs to be transferred over the network to the reducer phase. The shuffled and sorted output is then passed to the reducer phase as input, the reducer in turn applies a user defined reduced function on the input and performs a series of operations such as addition, filtration, aggregation and finding the minimum or maximum value. Finally the output of the reduce function is written to the output file or sent to another MapReduce job for processing.

Hadoop Distributed File System (HDFS)

The Hadoop File System is designed for storing very large number of files with streaming data access, which implies that applications or instructions are executed directly using the

MapReduce processing model. The HDFS has a master-slave architecture, with cluster consisting of a single namenode known as the master and several datanodes known as the slaves. The namenode is so critical to the HDFS such that when it fails or it is down, the Hadoop cluster becomes inaccessible. One of its functions is that it stores information about the actual data, that is, the real content which is stored in form of blocks, in the datanode. These information includes the location where the data resides in the datanode, the number of blocks, block id and so on. As the masternode, it communicates regularly with the slavenodes by keeping track of the various activities that goes on within them. It supervises the different tasks assigned to them and receives feedback on how each task is being carried out. The namenode is also responsible for replicating the data on a datanode to other datanodes to help prevent any failure that may occur on any node, thus ensuring that the network is not affected at all. Its ability to handle these challenges makes it highly fault tolerant and reliable. The datanodes on the other hand are responsible for storing and retrieving data in the form of blocks. They maintain data integrity by ensuring that the stored data are accurate, complete and consistent.

MATERIALS AND METHODS

Proposed Architecture

The proposed system architecture in figure 1 below shows the integration of the memoization technique at the map phase. This is the point where the gain-ratio is calculated for each attribute, in order to determine the attribute with the highest gain-ratio that would be selected for splitting the dataset.

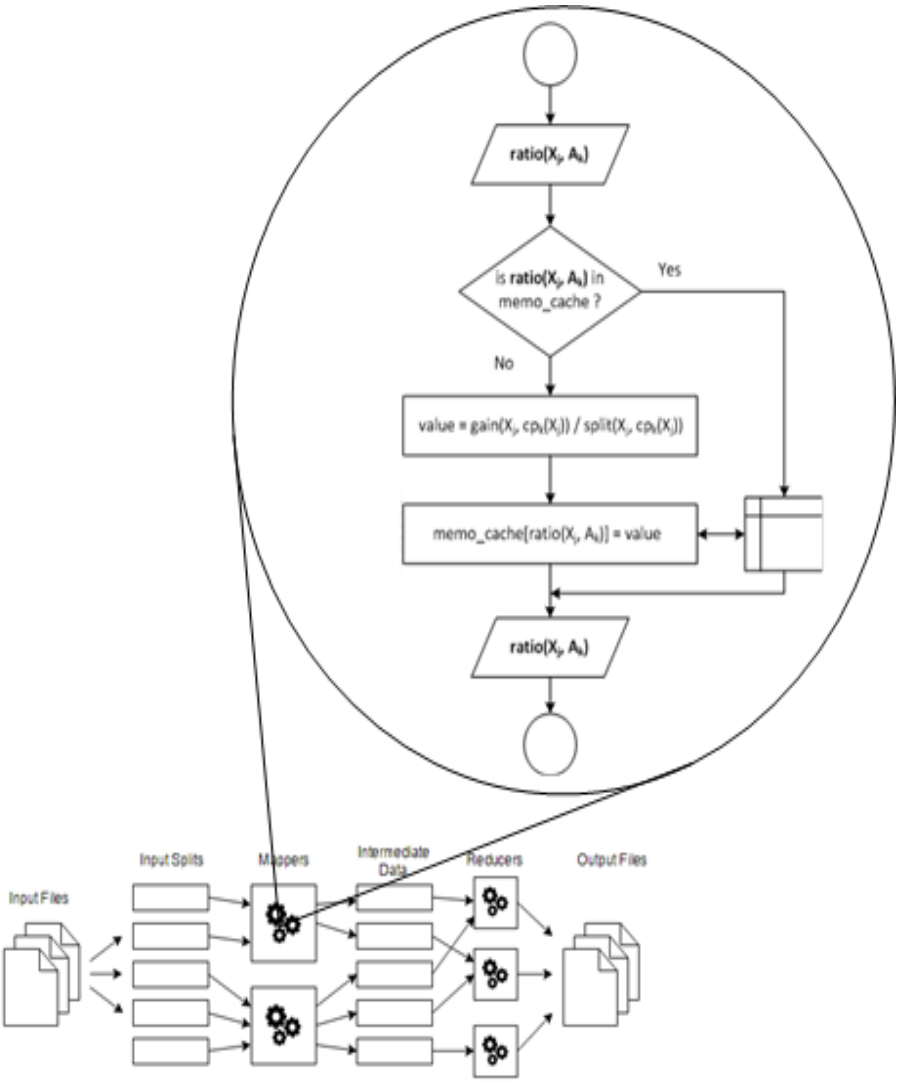

Figure 1: Memoized MapReduce Architecture

Memoized MapReduce Flowchart

In Figure 2 below, the flowchart depicts the workflow in the proposed method. The input to the system is the attribute A_k (where k = 1,2, …….. n) in the m subsets. The algorithm checks the cache to ensure if the input and corresponding result of the calculation exists in the hash table [line 26]. If the result exists in the cache, it is returned and used [line 27], but if it does not exist [line 28], the gain-ratio is evaluated by taking the ratio of the quantity of information gained to the intrinsic information for each value of an attribute and storing the result into the variable 'value'. The variable 'value' is then stored in the memoization cache to be used when the need for it arises [line 30 & 31].

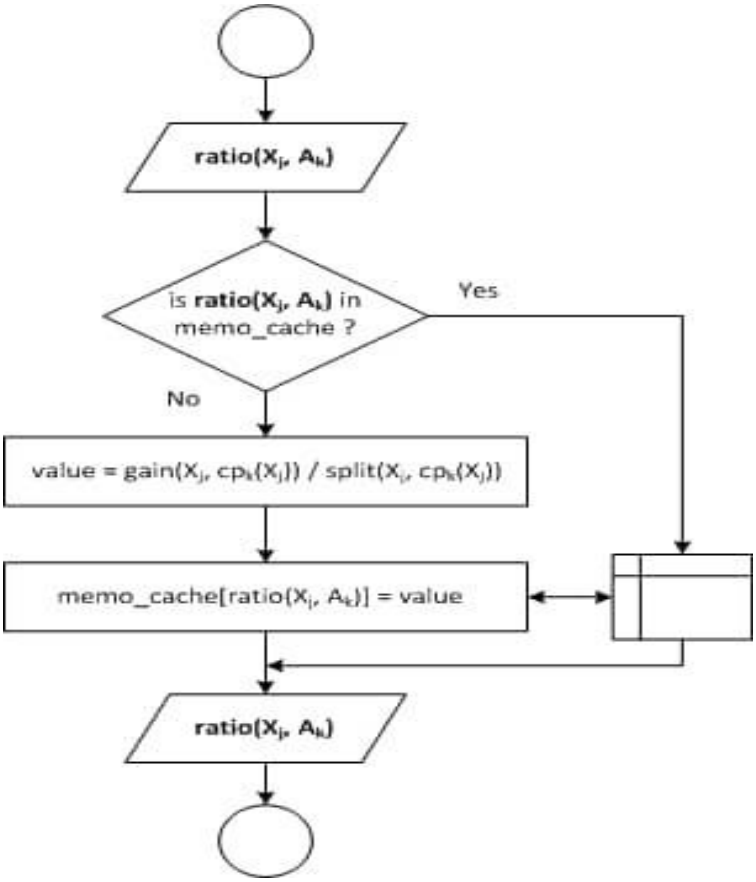

Figure 2: Memoized MapReduce flowchart

Proposed Algorithm

Input: A training data X:
Output: A selected attribute A_k and the cut points $cp_{(k^*)}(X)$.
1: Initialise a Hadoop Job SELECTJOB:
2: Set SelectTaskMapper as the Mapper class
3: Set SelectTaskReducer as the Reducer class
4: Adjust the block size of HDFS until the dataset X can be divided into m splits $\{X_j\}_{(j=1)}^m$
5: In the j-th SelectTaskMapper:
Input: $X_j = (x_i)_{(i=1)}^N$, where x_i is the i-th occurrence with n attributes $(A_k)_{(k=1)}^n$
Output: $\langle key,value\rangle = \langle A_k,[Ratio_k(X_j,A_k),cp_k(X_j)]\rangle$
6: Memo_Cache ={ }
7: for each attribute A_k, k =1,2,…..,n do
8: if A_k is numerical then
9: Sort its values $x_{1k},\ldots\ldots,x_{Nk}$ and record as $x_{1k}^*,\ldots..,x_{Nk}^*$
10: Find all cut points $cp_{ik} = (x_{ik}^* + x_{(i+1,k)}^*)/2$,i=1,….,N-1
11: for each cut points cp_{ik} do
12: Calculate the Information Gain:
13: $Gain(X_j,cp_{ik}) = Info(X_j) - [|x_{ij}^1|/|x_j| Info(X_{ij}^1) + |x_{ij}^2 |/|x_j| Info(X_{ij}^2)]$ where $X_{ij}^1=\{x_i \in X_j \mid x_{ik} \le cp_{ik}\}, X_{ij}^2 =\{x_i \in X_j \mid x_{ik} > cp_{ik}\}$ and symbol |x| is size of x
14: end
15: Select the optimal cut point $cp_k(X_j) = cp_{(i^*k)}$ of A_k:
16: $i^* = argmax\{Gain(X_j,cp_{ik})\}_{(i=1)}^{(N-1)}$
17: Calculate the split information of $cp_k(X_j)$:
18: $Split(X_j,cp_k(X_j)) = -[|x_{(i^*k)}^1|/|x_j| \log_2|x_{(i^*k)}^1|/|x_j| + |x_{(i^*k)}^2|/|x_j| \log_2|x_{(i^*k)}^2|/|x_j|]$
19: else
20: Cut points: $cp_k = \cup_{(i=1)}^C\{x_{ik}'\}$, where $x_{ik}' \in \{x_{1k},\ldots,x_{Nk}\}$, C is the number of attribute values
21: Calculate the Information Gain:
22: $Gain(X_j) = Info(X_j) - \sum_{(i=1)}^C |x_j^i|/|x_j| Info(X_j^i)$ where $X_j^i =\{x_i \in X \mid x_{ik} = x_{ik}'\}$
23: Calculate the split information of cp_k
24: $Split(X_j,cp_k(X_j)) = -\sum_{(i=1)}^C |x_j^i|/|x_j| \log_2|x_j^i|/|x_j|$
25: end
26: If Memo_Cache[A_k]not null
27: $Ratio(X_j,A_k)$ = Memo_Cache[A_k]
28: else
29: Calculate the gain ratio of A_k: $Ratio(X_j,A_k) = Gain(X_j,cp_k(X_j))/(Split(X_j,cp_k(X_j)))$
30: Memo_Cache[A_k] = $Ratio(X_j,A_k)$
31: end
32: Mapper Output: $\langle key,value\rangle = \langle A_k,[Ratio_k(X_j,A_k),cp_k(X_j)]\rangle$
33: end
34: In the SelectTaskReducer:
Input: $\langle key,value\rangle = \langle A_k, List[Ratio_k(X_j,A_k),cp_k(X_j)]\rangle, j = 1,2,\ldots,m$
Output: $\langle key,value\rangle = \langle A_{(k^*)},[Ratio_{(k^*)}(X),cp_{(k^*)}(X)]\rangle$
35: for each attribute A_k do
36: $Ratio_k(X) = \sum_{(j=1)}^m Ratio_k(X_j,A_k)$
37: $cp_k(X) = (\sum_{(j=1)}^m cp_k(X_j))/m$ (A_k is Numerical) or $cp_k(X) = U_{(j=1)}^{(j=m)} cp_k(X_j)$ (A_k is not Numerical)
38: end
39: Select the optimal index, where $k^* = argmax\{Ratio_k(X)\}_{(k=1)}^n$
40: Reducer Output: $\langle key,value\rangle = \langle A_{(k^*)},[Ratio_{(k^*)}(X), cp_{(k^*)}(X)]\rangle$
41: return $A_{(k^*)}$ and $cp_{(k^*)}(X)$.

Algorithm 1: Modified Attribute Selection Algorithm (MR-A-S)

The **Input** at the beginning of the algorithm is the data saved into the Hadoop Distributed File System (HDFS), at this point the file is broken down into blocks of size 128MB each and dispersed across datanodes in the hadoop distributed file system. However, **Output** is the expected output after the best attribute, which has the maximum gain ratio has been selected for splitting the dataset at each node for the building of the decision tree. **In lines 1-4**: During the job initialization, the job tracker creates an object to track the tasks and their progress. At this stage the map tasks for each InputSplit are created and the number of reduce tasks is specified by the configuration mapred.reduce.tasks which is set by the setNumReduceTasks method. The Input in **Line 5** is the input to each individual mapper. When Hadoop submits a job, it splits the input data logically, this is also known as Input splits, and these splits are processed by each mapper. The InputFormat.getSplits() method is responsible for generating the InputSplits which are converted by the RecordReader into a <key, value> pair format that can be read by each mapper for processing. For each InputSplit, a map task is created, hence the number of mappers required depends on the amount of InputSplits generated. The relationship between the input at the beginning and the input in **line 5** is that the latter is a subset of the previous, which is m subsets as stated in **line 4**. The previous data is broken down into smaller chunks or input splits, and assigned to individual mappers for parallel

processing. The output in **line 5** is the expected output from the map phase known as the intermediate output which is in a key/value pair format. It serves as Input to the reduce phase. **In line 6** the Memo_Cache is created for storing the results from gain ratio computation. **In Line 7**, the **For** loop iterates on each of the attributes in the m subsets of the dataset, and line 8 checks if the values are numerical or nominal. If it is numerical (continuous values need to be converted to nominal values) then **Line 9** sorts out the values in ascending order that is, from the smallest to the highest.
In Line 11 A loop is created where an iteration is carried out on all values and the dataset is divides into two parts that is instances less than or equal to the current value, and instances greater than the current value. **Lines 12 and 13** calculates the gain for every step while **line 14** ends the loop. In **Lines 15 and 16**, the value with the highest gain is selected as the threshold value. The Split Information for the optimal cut point is evaluated in **Lines 17 and 18**, this information determines how broadly and uniformly the attribute separates the data.

Experimentation

Computational environment

Host System

i. Windows 10 Pro Operating system
ii. Intel Core i5-4210U CPU
iii. 8.00 GB RAM
iv. A 64 bit processor laptop (minimum operating frequency of 2.4GHz)

Virtual System

i. 6 GB RAM
ii. 20.13 GB Hard Disk
iii. Ubuntu 16.04 Long-Term Support (64-bit) Operating System
iv. Netbeans IDE 8.1
v. Hadoop Framework Installation setup

Experimental Setup

A single node Hadoop cluster experiment was carried out on a dual core personal computer with the CPU running at a clock rate 1.70GHz to 2.40GHz. Hadoop version 2.6 and JDK version 8 were installed on Linux Ubuntu with SSH properly configured to manage the nodes on the cluster.

Data Collection

The datasets used for the experiment were sourced from online open sourced repositories. This include the University of California Irvine (UCI) Machine Learning *Repository, which is* a pool of databases, domain theories, and data generators. Three data sets were downloaded from the repository namely diabetes, breast cancer and dermatology (Khan, n.d; Zwitter &Soklic 1988; llter & Guvenir, 1998) respectively, they are commonly used datasets in the machine learning community. *The* Fifa Worldcup dataset (Becklas, 2018) and NBA Historical Stats and Betting Data (Hallmark, 2018), were downloaded from Kaggle, which is a virtual community of data scientists while the Labour statistics dataset was retrieved from Data.gov. Repository.

RESULTS AND DISCUSSION

The proposed model was compared with the existing system with the execution time used as the performance metric for evaluating the result on different sizes of datasets.
An experiment was carried out to assess the performance of the proposed algorithm using different sizes of datasets to measure the time it takes to run the proposed algorithm and that of the existing algorithm. Figure 3.1a, b and c shows the MapReduce log file statistics at different stages of processing

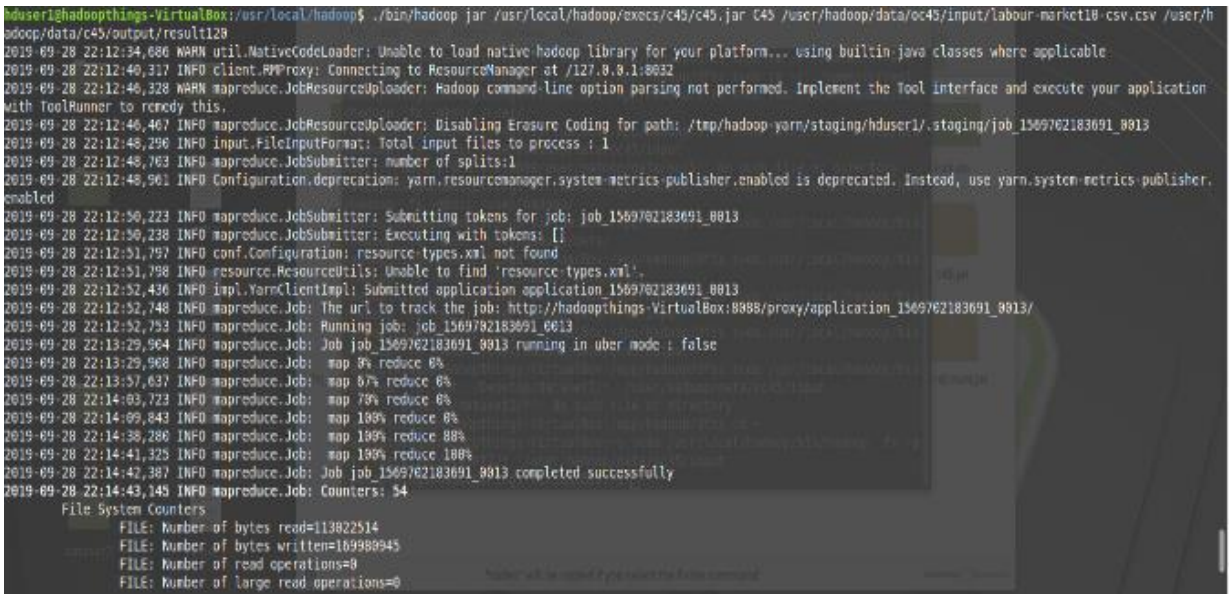

Figure 3a: Image of a Memoized MapReduce Process

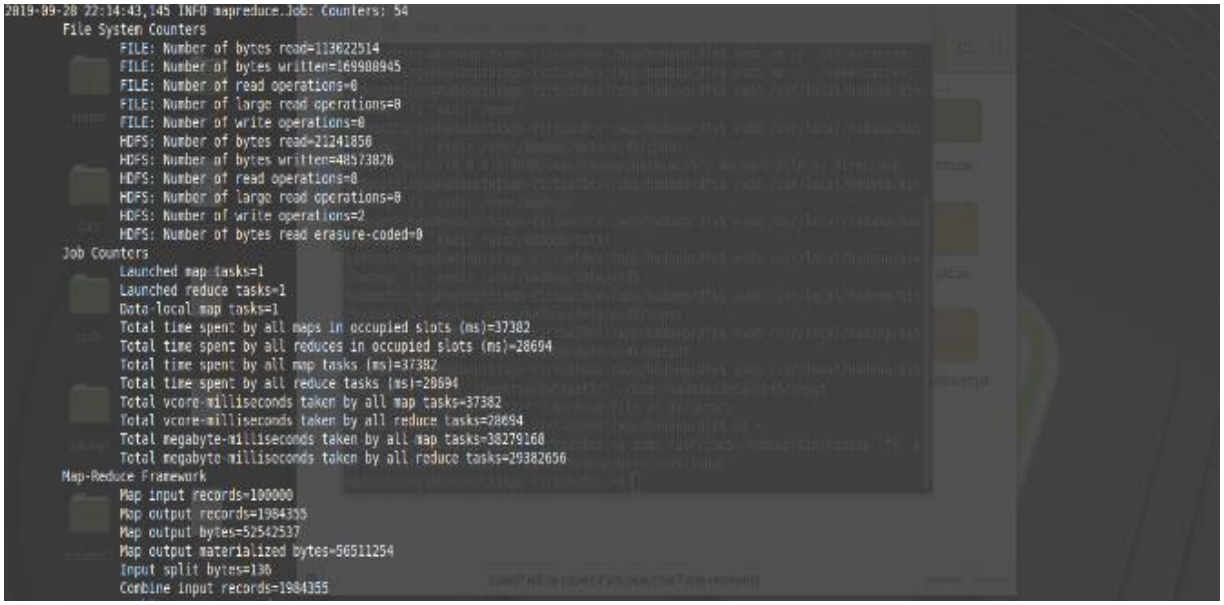

Figure 3b: Image of Memoized MapReduce Process

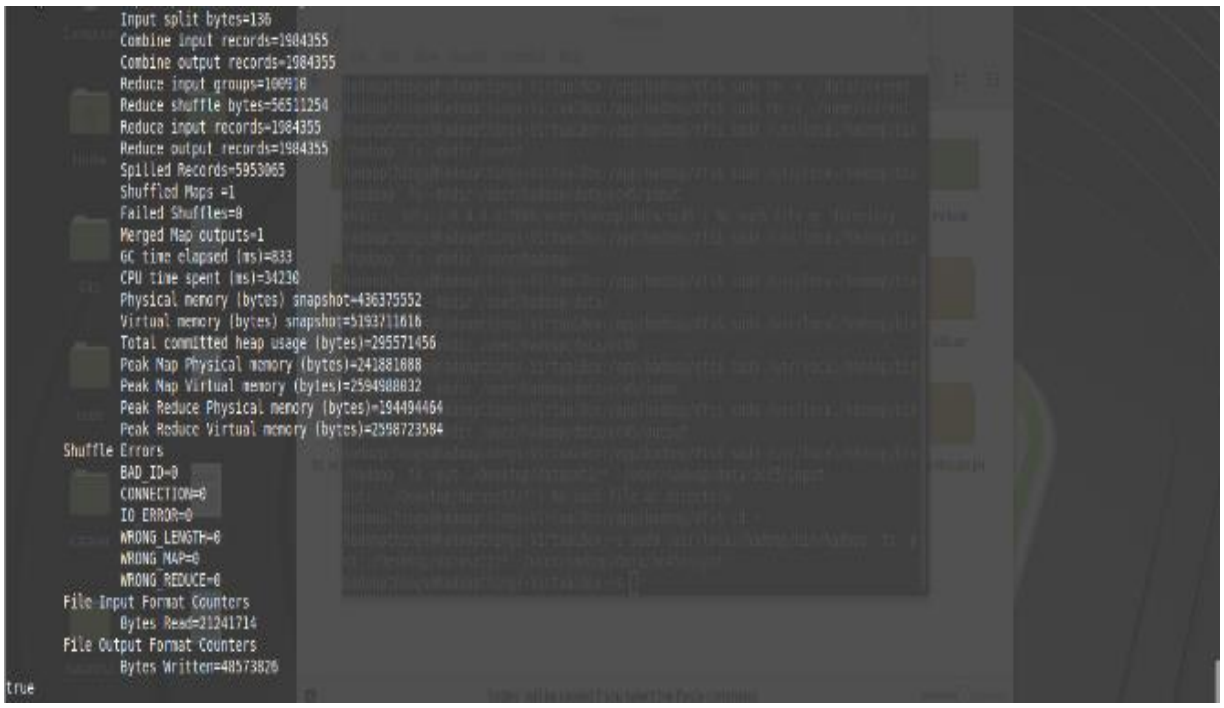

Figure 3c: Image of Memoized MapReduce Process

Table 1: Comparing execution time between the proposed algorithm and the existing system

S/No	Datasets	Time taken without Memoization (sec)	Time taken with Memoization (sec)	Percentage decrease in time
1.	Diabetes	4.02	4.22	- 4.7%
2.	Worldcups	3.31	1.6	51.66%
3.	Dermatology.csv	4.27	4.06	4.92%
4.	Breast cancer	3.84	3.82	0.52%
5.	Nba_players_all	5.66	5.26	7.07%
6.	Nba_betting_Totals	13.44	13.66	- 1.61%
7.	Nba_betting_money_line	13.79	12.4	10.08%
8.	Nba_betting_spread	13.87	14.24	- 2.66%
9.	Worldcupplayers	11.41	5.16	54.78%
10.	Ici_dec_18qtr	9.94	8.67	12.7%
11.	Ques-dec 18qtrs	19.99	19.26	3.65%
	Total	103.54	92.35	10.80%

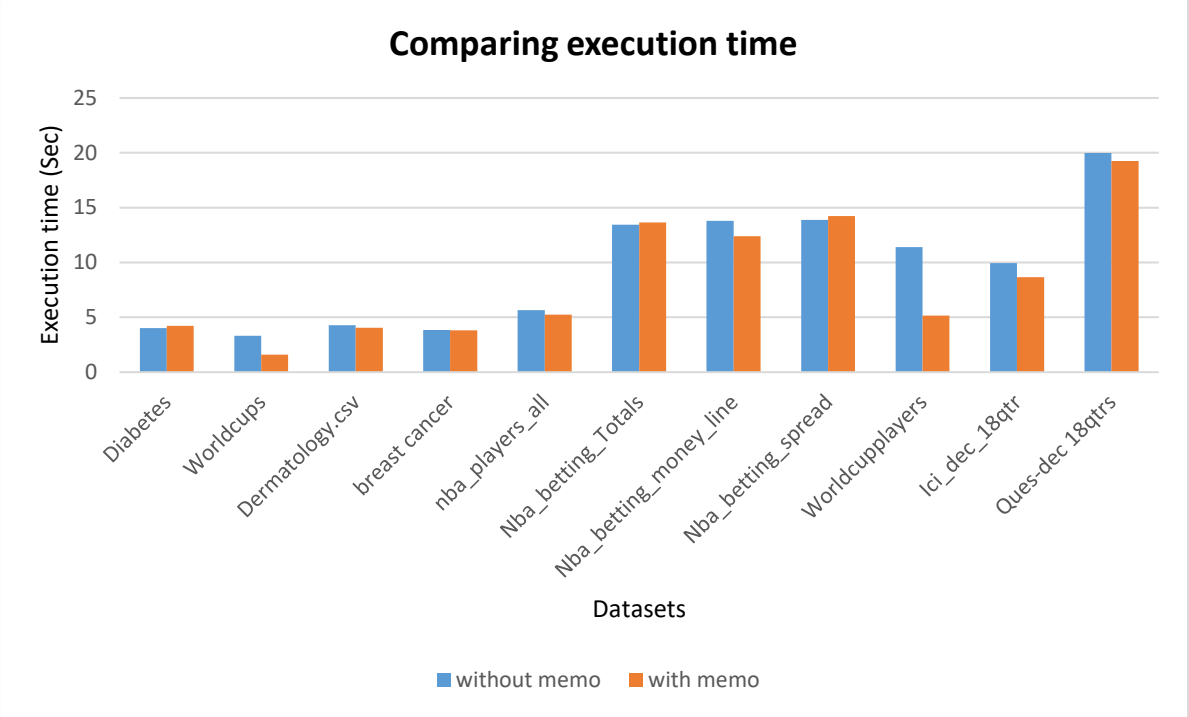

Figure 4: Comparing execution time

The result in Table 1 above shows the execution time (in seconds) of the old system and the new system when applied on 11 datasets using the Mapreduce framework on a single node cluster environment. When compared with each other it was observed that the recommended system did better than the existing system except on three datasets where the results were negative - 4.7 %, - 1.66 % and - 2.6 % indicating a percentage increase in the execution time of the proposed system. The result is shown on the graph in Figure 4, with y axis showing the execution time and x axis the different datasets.

The red colour indicates the time taken with memoization, while the blue colour depicts the time taken without memoization. The comparison of the results showed that the proposed algorithm performed better in some cases though not all. The ability of the memoization technique to store results of computation and make them available for reuse when the need arises without wasting time to re-compute them, has helped to enhance the C4.5 decision tree algorithm.

CONCLUSION

The C4.5 decision tree algorithm is an algorithm which is commonly used for classication and prediction in different applications, but its limitation in handling large datasets which results in high execution time has been one of the bottlenecks. Parallelising the C4.5 algorithm through the use of a MapReduce programming model has improved the algorithm by reducing the execution time. This research further enhanced the existing system by reducing the execution time through the application of memoization. The enhancement exhibited an average of 10.80 % decrease in execution time when compared with the time taken by the existing system.

REFERENCES

Badgujar, G., & Sawant, K. (2017). Improved C4.5 Decision Tree Classifier Algorithm for Analysis of Data Mining Application. *International Journal for Research in Engineering Application & Management, 2*(10), 18-24.

Becklas, A. (2018). FIFA World Cup. *Kaggle Repository*. Kaggle Inc. Retrieved October 17, 2019, from https://www.kaggle.com/abecklas/fifa-world-cup

Cherfi, A., Nouira, K., & Ferchichi, A. (2018). Very Fast C4.5 Decision Tree Algorithm. *Applied Artificial Intelligence, 32*(2), 119–137. doi:10.1080/08839514.2018.1447479

Dai, W., & Ji, W. (2014). A MapReduce Implementation of C4.5 Decision Tree Algorithm. *International Journal of Database Theory Application, 7*(1), 49-60.

Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified data processing on large clusters. *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation -, 6*, pp. 137-150. San Francisco, CA.

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). From Data mining to Knowledge Discovery in Databases. *AI Magazine, 17*(3), 37-54.

Frawley, W. J., Piatetsky-Shapiro, G., & Matheus, C. J. (1992). Knowledge discovery in databases: an overview. *AI Magazine, 13*(3), 57-70.

Hallmark, E. (2018). NBA Historical Stats and Betting Data. *Kaggle Repository*. Kaggle Inc. Retrieved August 13, 2019, from https://www.kaggle.com/ehallmar/nba-historical-stats-and-betting-data

Han, J., & Kamber, M. (2006). *Data Mining: Concepts and Techniques* (2nd ed.). San Francisco, CA, USA: Morgan Kaufmann.

Jaffar, J., Santosa, A. E., & Voicu, R. (2008). Efficient Memoization for Dynamic Programming with Ad-Hoc Constraints. *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, (pp. 297-303). Chicago, USA.

Jaseena , K. U., & David, J. M. (2014). Issues, Challenges and Solutions : Big Data Mining. *Sixth International Conference on Networks & Communications*, (pp. 131–140). doi:10.5121/csit.2014.41311

Kahn, M. (n.d.). Diabetes Data Set. *UCL Machine Learning Repository*. St. Louis: Center for Machine Learning and Intelligent Systems. Retrieved August 13, 2019, from https://archive.ics.uci.edu/ml/datasets/Diabetes

Ilter, N., & Guvenir, H. A. (1998, January 1). Dermatology Data Set. UCI Machine Learning Repository. Ankara, Turkey: Center for Machine Learning and Intelligent Systems. Retrieved August 13, 2019, from https://archive.ics.uci.edu/ml/datasets/Dermatology

Mu, Y., Liu, X., Yang, Z., & Liu, X. (2017). A parallel C4.5 decision tree algorithm based on MapReduce. Concurrency and Computation. *Concurrency and Computation: Practice and Experience, 29*(8), 1-12. doi:10.1002/cpe.4015

Muslim, M. A., Rukmana, S. H., Sugiharti, E., Prasetiyo, B., & Alimah, S. (2018). Optimization of C4.5 algorithm-based particle swarm optimization for breast cancer diagnosis. *Journal of Physics: Conf. Series*, *983*(2018), 1-8. doi:10.1088/1742- 6596/983/1/012063.

Piatetsky-Shapiro, G. (1991). Knowledge Discovery in Real Databases. *AI Magazine, 11*(5), 68-70.
Quinlan, J. (1993). *C4.5: Programs for machine learning* (Vol. 16). San-Mateo, CA: Morgan Kaufman.

Rajeshinigo, D., & Jebamalar, J. P. (2017). Accuracy Improvement of C4.5 using K Means Clustering. *International Journal of Science and Research, 6*(6), 2755-2758.

Seema, S., Agrawal, J., & Sharma, S. (2013). Classification through Machine Learning. *International Journal of Computer Applications, 82*(16), 20- 27

Sniedovich, M., & Lew, A. (2006). Dynamic Programming : an overview. *Control and Cybernatics, 35*(3), 513-533.

Suresh, A., Swamy, B. N., Rohou, E., & Seznec, A. (2015). Intercepting Functions for Memoization: A Case Study Using Transcendental Functions. *ACM Transsactions on Architecture and Code Optimization, 12*(2), 18:1-18:23.

Wang, B., Huang, S., Qiu, J., Liu, Y., & Wang, G. (2014). Parallel online sequential extreme learning machine based on MapReduce. *Neurocomputing, 149*, 224-232.

Wang, Z., Wang, J. Huo, Y. Tuo, Y., & Yang, Y. (2016) "A Searching Method of Candidate Segmentation Point in SPRINT Classification," *Journal of Electrical and Computer Engineering*, vol. 2016, pp. 1-5. doi:10.1155/2016/2168478

Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H.. Geoffery, J.M., Ng, A., Liu, B., Yu, P.S., Zhou, Z., Steinbach, M., Hand, D.J., & Steinberg, D. (2008) "Top 10 algorithms in data mining," *Knowledge Information System*, 14, pp. 1–37.

Zhu, F., Tang, M., Xie, L., & Zhu, H. (2018). A Classification Algorithm of CART Decision Tree based on MapReduce Attribute Weights. *International Journal of Performability Engineering, 14*(1), 17-25. doi:10.23940/ijpe.18.01.p3.1725

Zwitter, M., & Soklic, M. (1988, July 11). Breast Cancer Data Set. *UCI Machine Learning Repository*. Ljubljana, Yugoslavia: Center for Machine Learning and Intelligent Systems. Retrieved August 13, 2019, from http://mlr.cs.umass.edu/ml/datasets/Breast+Cancer