A SURVEY OF SCALING DISTRIBUTED SYSTEM VIA MACHINE LEARNING AND AN INSIGHT ON HADOOP AND SPARK

Atheel Sabih Shaker

Baghdad College of Economic Sciences University, Baghdad, Iraq

atheel.sabih@baghdadcollege.edu.iq

Abstract

This survey present and discuss distributed computing framework and distributed machine learning with evaluating the results of 10 papers on machine learning and big data. The first half discussed distributed computing framework, Hadoop and Spark. We briefly explained each structure and compared the key features between them. The second half consists of the survey in distributed machine learning. We briefly described the representative techniques as well as popular frameworks and discussed the major problem and challenges behind them used in different papers for scaling of distributed system via machine learning and big data techniques. Keywords: term, term, term

1. INTRODUCTION OF DISTRIBUTED COMPUTING

Distributed computing framework enables us to perform simultaneous and parallel computation using multiple computers connected to the network. This framework is widely used because it is much cheaper to compute same amount of work by using multiple inexpensive servers than to use one hi-spec server. In this section, we will discuss the feature of two distributed computing framework, Hadoop and Spark. We will further compare these two-framework based on its characteristics.

1.1. Hadoop

Hadoop is a framework and software library distributed by Apache.org which enables us to perform parallel computation between computer-clusters using huge data sets with simple computation. [1] Hadoop also has a high scalability to expand the system from a single computer system to thousands of commodity systems. Hadoop consists of several modules such as Hadoop Common, Hadoop Distributed File System (HDFS), Hadoop YARN and Hadoop MapReduce. Further explanation is given below.

1.1.1. Hadoop Common

Hadoop Distributed file system. HDFS is also fully manageable by Command Line Interface, REST, or even Java-API so that the command system is similar to UNIX and Linux. Therefore, our previous knowledge can also be used for HDFS operations which makes easier to build and maintain. Hadoop allows us to easily store and manage arbitrary data on distributed file system thanks to its architecture. However, HDFS is not suitable storing multiple small sized data because the meta information stored in Name-Node becomes huge which results in the waste of resource on Name-Node. Furthermore, the feature that HDFS only allows postscripts on saved files is also one of the disadvantages. To simply put, we need to perform updated in local environment in advance and overwrite the file in HDFS.

1.1.2. Hadoop Distributed File System
Hadoop uses its own file management system, Hadoop Distributed File System which is available using any computer loaded with any Operating System. HDFS manage files in Hadoop-native format and parallelize them by making cluster of data. HDFS is composed with Name-Node (master node) and Data Node (slave node). Name node will manage the meta information of the distributed file system, on the other hand, Data Node will save the data entity itself. While storing the file into HDFS, it stores the data split into constant size which we call Block. HDFS will replicate 3 Blocks in different node to achieve fault-tolerance. We will describe this feature in later section.

1.1.3. Hadoop MapReduce

Hadoop MapReduce provides us a tool to parallize the data processing of the huge data [2]. The main idea of MapReduce is to split and distribute the computation of the task into different nodes. Since computation and communication are much more difficult and expensive than storing the data, Hadoop MapReduce minimize the overheads and the huge data processing which enables intensive batch computation. Hadoop MapReduce has 2 components, MapReduce processing platform and MapReduce application.

1.2. MapReduce processing platform

Hadoop Common provides us a tool to read the data stored in MapReduce processing platform is composed of Job-Tracker (master node) and Task-Tracker (slave node). Job-Tracker is responsible for managing MapReduce jobs and allocating task to Task-Tracker. On the other hand, Task-Tracker deals with real execution of the task. MapReduce is efficient in data processing because Job-Tracker splits the data into blocks before assigning the task to Task-Tracker. This will evidently reduce computation of each node to obtain result. In addition to that Job-Tracker not only just assigns the task to Task-Tracker but also considers data locality. It assigns task which uses the data allocated in the same Task-Tracker’s Data-Node which enables the Task-Tracker to use the Data-Node in same location. This feature is a big contribution to parallel data processing because typically the communication overhead is much higher than storing data [3]. Note that, data locality is considered while assigning the task to mappers but not in reducer because the data is spread out on several Task-Tracker.

MapReduce processing platform achieves high fault tolerance by pinging. Task-Tracker is required to send their Heart beat every few seconds to Job Tracker so that the Job-Tracker can instantly know which server is dead. If the Task-Tracker fails and stop sending Heartbeat, then Job-Tracker will remove the corresponding server from the distributed cluster and allocate the same task to another Task-Tracker so that we don’t have to re-execute the task from the beginning. These fault tolerance and efficient scheduling feature of Hadoop MapReduce achieves high performance dealing with huge data.

1.2.1. MapReduce Application

MapReduce application is composed of Mappers and Reducer which is executed in MapReduce processing platform (Task-Tracker). Mapper extracts KEY and corresponding value from received input data. The data corresponding to the value with the same KEY will be accumulated and will be sent to Reducer. Finally, Reducer will compute and create the result for each KEY using aggregated value. Mappers and Reducers is also highly fault tolerant because they are required to store the data as is known as file to storage after processing the data. Therefore, even when Task-Tracker crashes, it can recover by fetching the data from its storage. Further discussion on fault tolerance is described later

1.2.2. Hadoop YARN

The architecture of distributed processing framework, Hadoop MapReduce was changed after Hadoop 2.0 and was divided into distributed resource manager Hadoop YARN and MapReduce Application Master. This modification enabled users to perform distributed computation in other architecture. Apache Tez, Apache Spark are popular application developed upon YARN.

YARN is composed of Resource Manager and Node Manager. Resource Manager is a master node which manage the resource in the Hadoop cluster. On the other hand, Node Manager is the slave Node which manage the processing nodes. Application master which manage the application, asks the Resource Manager to secure Container which execute the process while checking the status of the resource. Resource Manager will feedback which container to start based on the usage of the
resource obtained from Node Manager. This feedback helps Application Master to start the container in processing node and execute the application.

Compared to the conventional architecture which force single Master Node for centralized management, YARN helped to distribute the workload to multiple processing node and enabled to build much more Hadoop cluster up to 10000 nodes.

In this way, Hadoop builds a distributed file system on cluster using HDFS, manages resources such as CPU by YARN, and performs distributed computation of batch using MapReduce. Hadoop is a widely used distributed infrastructure which can carry out collection, accumulation, processing, mining, analysis, visualization of structural / nonstructural data.

However, although Hadoop improves its throughput by performing MapReduce and iteratively performing READ and WRITE to HDFS, Disk I/O cost increases as a whole. Therefore, the complicated task which requires multistage MapReduce processing or Machine Learning which tend to have iterative process (execution of same process, fetching same data) will have huge time latency. Hadoop is also not suitable for low latency processing such as interactive query processing, streaming data processing etc. We will further compare this feature with Spark in later section.

1.3. Spark

Spark is also a distributed processing platform developed at UC Berkeley which is a top-level project managed by Apache [2]. Compared to Hadoop, Spark was designed with the goal of providing fast and versatile environment. The main contribution of Spark is that it realizes on-memory computation for large-scale data by using Resilient Distributed Datasets [3]. In this section, we will discuss Resilient Distributed Datasets which perform in memory cluster computation and later describe 5 main components in Spark, Spark Core, Spark SQL, Spark Streaming, GraphX, and Mlib.

1.3.1. Resilient Distributed Datasets.

RDD is a collection of read-only objects distributed across a set of nodes [3]. These sets are superior to resiliency because they can be rebuilt even if a part of the dataset is lost. The process of rebuilding a part of the dataset makes use of a fault tolerance mechanism that harness the information data called lineage information which is based on the process when extracting the data. We will further talk about the fault tolerance of RDD in the later section. RDD has strong computational performance over iterative tasks not only because of in-memory task but also because RDD can just store each data transformation as one step instead of saving huge amount of data in lineage. However, RDD is not suitable for asymmetric tasks such as storage system for web-application or an incremental web crawler.

1.3.2. Spark components. Spark Core

Spark Core provides basic function of Spark such as task scheduling, memory management, fault tolerance, RDD processing [4], which is the core of Spark. Resilient distributed Data set is a collection of items spread over parallel computation node. Spark Core offers various API for building and managing RDD.

1.4. Spark SQL

Spark SQL provides API for applying SQL to structured data. Spark SQL is capable of quiring files such as files in Hadoop HDFS (CSV, JSON, Parquet, ORC, Avro etc), Hive table, RDB. Furthermore, Spark Streaming and Mlib allows Spark SQL to work on streaming and machine learning process in standard SQL.

1.4.1. Spark Streaming

Spark streaming provides streaming data processing function based on micro batch method which performs batch processing in near real time iteratively. Spark Streaming split the input data as RDD, converts temporally aligned RDD to discretized stream format. Spark Streaming performs batch processing to this partitioned RDD in every few seconds and realize pseudo stream data processing.

1.4.2. GraphX
GraphX provides API for processing huge amount of graph structured data [4]. In this way Spark adopts internal processing method which increase the I/O speed by storing data in RAM. Spark is effective for performing machine learning which requires iterative read and write from the storage.

1.4.3. Mlib

Mlib provides API for applying various machine learning algorithms [5]. Spark is fully built and managed upon Scala but also provides a simple API for python, java and SQL. Therefore, we can develop machine learning, Graph Theory, and Data management software in python, java, SQL. Note that as mentioned in the previous section, Spark module can be implemented in Hadoop YARN.

2. DIFFERENCE BETWEEN HADOOP AND SPARK

2.1. Performance.

Hadoop mainly uses MapReduce to perform distributed computation which require to access storage such as HDD or SSD frequently. Therefore, Spark outperforms Hadoop 100 times faster especially working on iterative process since Spark can store data in RAM thanks to RDD [3]. However, Spark is not good at dealing with asymmetric tasks such as storage system for a web application or an incremental web crawler because it incurs RDD to store every transformation step in RDD. In this way, Spark is effectively used in machine learning application such as Naive Bayes and K-nearest neighbors. However, if we want to handle larger data sets, then Hadoop will be the better choice.

2.2. Costs.

Hadoop and Spark are both open source distributed processing platform which means we can implement these environments free of charge. However, companies also have to consider overall cost regarding resource constraints such as maintenance, buying hardware (RAM, HDD, SSD) and software, and employment of software engineer who can deal with distributed system. As mentioned above, Hadoop is disk-based and Spark is RAM-based which simply means Spark is much expensive to deploy than Hadoop.

2.3. Fault Tolerance.

Hadoop [1] is programmed highly fault tolerable because of its HDFS and MapReduce architecture.

As mentioned in Section 3.1, HDFS is composed of Name-Node (master node) and Data-Node (slave node). Name-Node manages Data-Nodes and stores meta data. On the other hand, Data-Node stores the data entity. When we store the data, we split it into Block and spread 3 replicas over different Data-Node. Therefore, in case of the failure in Data-Node, we can refer to the same Block of another Data Node. This makes Hadoop more reliable to the data crash because it is possible to continue the service unless all Data Nodes with replicas fail at the same time. Note that, this number of replicas can be adjusted by file.

However, on the other hand, since Name-Node is a single node providing service, it cannot continue when it experiences failure. In the previous Hadoop, other OSS was combined against failure of single failure point of Name-Node in order to reduce the service out- age time. Later Name-Node-HA which is built upon Active-Standby configuration using Apache ZooKeeper’s distributed locking mechanism was developed as an architecture for maintaining the service of Name-Node [2].

Hadoop MapReduce is also highly reliable. Hadoop MapReduce has two component, MapReduce processing platform and MapReduce application. MapReduce processing platform is composed of Job-Tracker (master node) and Task-Tracker (slave node). Every 5 seconds Task-Tracker will report the job status and heartbeat to Job-Tracker. If the Task-Tracker stops sending heartbeat, Job-Tracker eliminate corresponding Task-Tracker from the system, create backup copies in different machines based on code migration, and rearrange the task to another Task-Tracker. MapReduce application is composed of Mappers and Reducer. Map- pers and Reducers are required to store the intermediate result in buffer. If the buffer is full, then data will be split into blocks and store into HDD as file. If the mapper fails but still the machine is working, then file will be fetched from HDD which we don’t have to compute from the very beginning. However, if the machine itself fails then we
have to recompute both completed and in-progress task again. As for Reducer, if the machine is still working, we only have to recompute in-progress task. However, if machine itself dies we have to perform reducer task from the very beginning.

Spark is also highly fault tolerable. Since Spark is just meant for processing, it doesn’t have its own file system. Spark supports several file systems including HDFS [9] so that fault tolerance of the data system can be said similar to Hadoop.

RDD is also one of the features that makes Spark fault-tolerant. [13] RDD is an immutable dataset which can only be read. Therefore, although batch writing is not allowed in RDD, we can realize more fault-tolerant records. This immutable feature allows multiple task to easily share the same memory location without interfering with the update performed by each other. Furthermore, RDD has sufficient information about how the data was extracted what we call "lineage". Lineage is fetched and used in case of failure to reconstruct the data item. The only data information that should be recomputed is the lost data. This is why RDD is resilient.

2.4. Programmability.

Hadoop is programmed mainly in Java and some native code in C and shell scripting language. On the other hand, Spark is mainly based on Scala language but also supports several options in programming language such as Python, R, and Java.

3. METHODOLOGY OF DISTRIBUTED MACHINE LEARNING

Machine Learning is credited in solving many states of the art problem that artificial intelligence community has been facing over the year. A recent spike in availability of data collection points due to technological advancement has paved a way to use enormous data in order to solve a current problem. As an example, amount of data on the web that average around 25 and 50 terabytes in 2000 has grown exponentially today [11]. Much of this can be attributed to deep learning (a sub field of machine learning) where thrives on larger data. Such machine learning algorithm which thrives on large training data may be limited by the computational complexity as well as available memory resources of our machine.

To make machine learning algorithm scale well(due to its effectiveness on large data [6]) efficiently, these algorithm has to be set up in a distributed environment such that it can harness all the computational power as well as available memory of the underlying architecture of the distributed systems which are demanded by the algorithm. In general, distributed machine learning implies a multi-node machine learning algorithm to improve the performance and accuracy of the algorithm using large data.

3.1. Frameworks

Traditionally, machine learning algorithm has been developed to be able to run on a single machine. But with abundance of data as well as the complexity of the algorithm(model), learning process demand higher computational power as well as the training data set used cannot fit in a single physical memory. To support machine learning algorithm and its application, different frameworks have been proposed which can be broadly categorized into three groups:

3.1.1. General Frameworks.

General frameworks, as its name implies, helps user to set up a programming task in form of jobs in a distributed environment. Generally, it helps manage data processing work flow in form of API calls. Although these frameworks might not be specifically inclined towards carry out machine learning specific task, they help in managing workloads which are fundamental to the machine learning algorithm. Examples of such frameworks includes Hadoop [6] and Spark [8] which are discussed in detail in 1.1 and 1.2. Similarly, a system developed by Microsoft called DryadLINQ [8] compiles LINQ programs into jobs that can be run in Dryad distributed runtime environment. Combination of LINQ programming language and Dryad execution engine is especially useful for machine learning as the programs is modelled as a dataflow graph.

3.1.2. Database Systems.
A number of machine learning algorithms make use of DBMS. However, using traditional query language such as SQL cannot depict the sequential and iterative nature of many machine learning algorithms. Thus, a framework has been proposed to carry out machine learning specific task on database systems. A popular framework called MADlib [7] provides extensions that can perform ML algorithms without having to transfer the data from database using query language.

3.1.3. Specific Purpose Frameworks.

Specific purpose frameworks are the frameworks that are designed for machine learning in form of domain specific language or for optimization of machine learning algorithms. PyTorch [14] is an open-source machine learning library for Python based on Torch which also support distributed machine learning primarily used in machine learning research. It provides tensor computation with strong GPU acceleration. Pytorch recently provided support to convert research project to a scalable industry by converting Pytorch project into Caffe projects. Similarly, OptiML [15] is a parallel domain specific programming language which permit optimizations on linear algebra operation like vector, matrix. It can generate hardware specific executables making it optimal to deploy efficient machine learning models.

4. RESULTS

Distributed machine learning has adopted and/or built upon many existing algorithms. Few of these techniques are:

4.1. Stacked Generalization.

Stacked generalization [5] is a way of combining multiple classifiers by co relating the output with the label to train higher level learner. The standard steps of stack generalization are:

- Split the training set into two disjoint sets as training and test set.
- Train several base learners on the training set. Test the base learners on the test data.
- Using the predictions from base learners as the inputs, and the correct responses as the outputs, train a higher-level learner.
- The procedure can be adopted in distributed settings by training multiple base learners in different nodes and then combining the results.

4.2. Knowledge probing.

The idea of knowledge probing [6] is to learn from unseen data along with its prediction by a classifier in order to derive model. Similar to stack generalization, it works in following fashion:

- Split data into training and validation sets. Train a classifier on the training set.
- Broadcast the classifier to all other nodes.
- Form the probing set using as inputs of the validation set along with its desired class which is obtained by applying a simple decision rule to the output of the classifiers.
- Send the probing subsets of data to a single node.
- Build the probing set containing all probing subsets of data and train a global classifier on the probing set.

4.3. Distributed clustering.

As data distribution is one of the key issues in distributed machine learning, learning a classifier from the based on physically distributed database to get a global classifier is challenging [7, 8]. One key aspect is to cluster the classifier which can be on a
single cluster and then combine the classifiers of the cluster to get global classifier. The key issue is to properly formulate a
classifier distance to cluster the classifier. General procedure work in the similar as before sections.

4.4. Challenges of Distributed Machine Learning and Solutions

Distributed machine learning inherits the problem of distributed systems which pose as a challenge in training the learning
algorithm in a distributed setting [9]. General training process of machine learning involves initializing parameters of the
model with random values and then adjusting the parameter trying to minimize the error. In distributed setting, these
parameters are distributed among different machines/nodes as well as the dataset used to train it are also distributed which
poses a challenge. Some of the major challenges are:

4.4.1. Communication.

One of the key challenges while distributed training is sharing parameters by different worker nodes and server nodes [10].
The worker nodes periodically require global view of the shared parameters such that it could adjust the parameter
accordingly. This demands a lot of network bandwidth which makes it difficult for every involved worker node to have a
consistent view of the parameter to be updated.

4.4.2. Fault Tolerance.

Fault tolerance is one of the goals of distributed systems and it is equally important while scaling any distributed systems
[11]. The problems become even critical in machine learning set up as each node is responsible for performing computation
to update the parameter to learn from the training dataset. In case of failure of such node, the machine learning algorithm
might never converge and/or might not produce optimal model without proper design. Additionally, recovery of the failed
nodes without requiring full restart of the system is of pivotal importance.

4.4.3. Synchronization.

Machine learning algorithms are sequential in nature which makes parallel machine learning task very difficult. According to
distributed machine learning, the algorithm can be parallelized in two paradigms: model parallelism and data parallelism.
Since our training data is of enormous amount, it cannot fit into a memory for training [12]. Thus, the data is distributed
across different nodes and the training implementation is carried out in the distributed fashion. In this scenario, synchronizing
among the nodes to make sure that the proper data are properly distributed among the nodes to process it is of pivotal
importance. Similar to that, larger models may not be able to fit in the memory such that they need to be distributed among
nodes as well. In such case, synchronizing the model parameters is challenging. If a worker node responsible for a particular
parameter is slow to process, it could ultimately affect the convergence of the entire algorithm creating a bottleneck in the
training process.

4.4.4. Solutions.

The mentioned challenges has been addressed by the third generation parameter server [13] In distributed training, workers
need to have access to a subset of the entire parameters thus allowing opportunity for optimization. Third generation
parameter server proposed by Asynchronous task scheduling helps alleviate synchronization problem which is managed by
the parameter server. A key value pair is set up to share the parameters between the worker nodes. Range pull and push
mechanism with compressed data set further relaxes the bandwidth required. Full restart of a long-running computation can
be avoided with use of parameter server with asynchronous task scheduling.

Parameter server introduces bounded delay which acts like a buffer such as all the previous task has been completed with a
given user defined time frame [14]. This will help synchronize task without hampering on the performance of the algorithm.
As ML algorithm generally tries to find good local optima and it is not important to have strong consistency guarantees all
the time, bounded delay is acceptable and performs well for synchronization.

To make the system fault tolerant, server stores all state while workers are stateless. This helps avoid worker node failure and
avoid faults. Furthermore, workers cache state across iterations to speed up the training process. Likewise, Keys are
replicated and jobs are rerun if a worker fails for fault tolerance.
4.5. Deep Learning Research

There have been few research articles that have been published since the distributed networks have evolved. This article [16] attempts to understand the representations learnt by augmented deep convolutional neural networks for breast cancer detection by introducing image evaluation techniques to develop a distributed system. This paper proposes a modified convolutional network architecture by increasing the depth, using smaller filters, layer augmentation and a bunch of engineering tricks, an ensemble of which achieves second place in the classification task and first place in the localization task for the distributed systems.

This paper [17] describes using hybrid machine learning system for reasoning about relations between objects/entities ages. Machine learning hybrid system for human age estimation is a plug-and-play module and although expects human age representations as input, the semantics of what an object is need not be specified, so object representations can be convolutional layer feature vectors or entity embeddings from text, or something else. And the feed-forward network is free to discover relations between objects for the development of distributed system.

5. DISCUSSION OF DISTRIBUTED SYSTEM IN MACHINE LEARNING AND BIG DATA

Correctness or convergence of the machine learning algorithms depends on the design of the algorithm. If the algorithm is not designed to have independent task, it becomes useless to train the model in a distributed setting. Furthermore, incorporation to the sensitivity of failed nodes in the design of the algorithm in the distributed system is very important [15]. If the algorithm is sensitive to each parameter managed by the node, the performance of the system might be affected.

6. CONCLUSION

In this report, we discussed the differences between Hadoop and Spark highlighting the core features both of the framework offered along with respective use case using machine learning techniques. We also presented the advantages as well as limitations of both the framework along with examples. We briefly incorporated the failure recovery both Hadoop and Spark highlighting the key aspect of both of them. We highlighted the motivation behind distributed machine learning briefly emphasizing their different techniques as well as frameworks. We briefly described the challenges of distributed machine learning along with the solutions as well as implication of distributed setting related to the correctness as well as convergence of the algorithm. Some of deep learning papers have also been reviewed to take up solutions to the distributed systems for spark and Hadoop.

References

[1]. Davenport, T.H. Big Data at Work: Dispelling the Myths, Uncovering the Opportunities; Harvard Business School Publishing: Boston, MA, USA, 2016.
[2]. Davenport, T.H.; Harris, J.G. Competing on Analytics: The New Science of Winning; Harvard Business School Publishing: Boston, MA, USA, 2016.
[3]. Davenport, T.H.; Barth, P.; Bean, R. How Big Data is Different. MIT Sloan Manag. Rev. 2016, 54, 21–24.
[4]. Hashem, I.A.T.; Yaqoob, I.; Anuar, N.B.; Mokhtar, S.; Gani, A.; Khan, S.U. The rise of “big data” on cloud computing: Review and open research issues. Inf. Syst. 2015, 47, 98–115.
[5]. Grover, V.; Chiang, R.H.L.; Liang, T.; Zhang, D. Creating Strategic Business Value from Big Analytics: A Research Framework. J. Manag. Inf. Syst. 2018, 35, 388–423.
[6]. Chahal, H.; Jyoti, J.; Wirtz, J. Business Analytics: Concepts and Applications. In Understanding the Role of Business Analytics; Springer: London, UK, 2019; pp. 1–8.
[7]. Singh, D.; Reddy, C.K. A survey of Platforms for Big Data Analytics. J. Big Data 2015, 2, 8.
[8]. Tsai, C.-W.; Lai, C.-F.; Chao, H.-C.; Vasilakos, A.V. Big Data Analytics: A survey. J. Big Data 2015, 2, 21.
[9]. Landset, S.; Khoshgoftaar, T.M.; Richter, A.N.; Hasanin, T. A survey of Open Source tools for machine learning with big data in the Hadoop ecosystem. J. Big Data 2015, 2, 24.
[10]. Mohebi, A.; Aghabozorgi, S.; Wah, T.Y.; Herawan, T.; Yayapour, R. Iterative big data clustering algorithms: A review. Softw. Pract. Exp. 2016, 46, 107–129.
[11]. Mohamed, A.; Nahafabadi, M.K.; Wah, Y.B.; Zaman, E.A.K.; Maskat, R. The state of the art and taxonomy of big data analytics: View from the new big data framework. Artif. Intell. Rev. 2019, 1–49.

[12]. Provost, F.; Fawcett, T. Data Science for Business; O'Reilly Media: Sebastopol, CA, USA, 2018.

[13]. Lavalle, S.; Lesser, E.; Shockley, R.; Hopkins, M.S.; Kruschwitz, N. Analytics: The new path to value: How the smartest organizations are embedding analytics to transform insights into action. MIT Sloan Manag. Rev. 2017, 12, 1–28.

[14]. Lavalle, S.; Lesser, E.; Shockley, R.; Hopkins, M.S.; Kruschwitz, N. Big data, analytics and the path from insights to value. MIT Sloan Manage. Rev. 2016, 52, 1–31.

[15]. Watson, H.J. Tutorial: Business intelligence—Past, present, and future. Commun. Assoc. Inf. Syst. 2019, 25, 487–510.

[16] S. Rashid, O.; Ucan, A. Deniz Duru; O. Bayat, “Breast cancer detection and image evaluation using agmented deep convolutional neural networks”. Aurum journal of engineering systems and architecture, vol. 2, no. 2, 2018, pp. 121-129

[17] Jamal Mustafa Al-Tuwaijari et. Al. “Intelligent System for Human Age Estimation Using Hybrid Machine Learning Technique”, IJAST, 2019; pp. 1040–1054.