Future Technology
Open Access Journal
ISSN 2832-0379

August 2026| Volume 05 | Issue 03 | Pages 128-138

Journal homepage: https://fupubco.com/futech

https://doi.org/10.55670/fpll.futech.5.3.12

Future Publishing LLC

Article

Apache Hadoop for large-scale data processing using machine learning techniques

Nidaa Ghalib Ali[1]*, Mohanaed Ajmi Falih[2], Ali Ajmi Falih[3]

[1]Department of Computer Networks and Software Techniques, Babylon Technical Institute, Al-Furat Al-Awsat Technical University, Kufa, Iraq
[2]Directorate General of Education, Babylon, Iraq
[3]AL-Mustaqbal University, Technical College of Engineering, Computer Technology Engineering, Babylon, Iraq

ARTICLE INFO

Article history:
Received 24 October 2025
Received in revised form
19 March 2026
Accepted 13 May 2026

Keywords:
Big Data, Hadoop, Spark, Machine learning

*Corresponding author
Email address:
inb.nedaa10@atu.edu.iq

DOI: 10.55670/fpll.futech.5.3.12

ABSTRACT

As big data volumes increase and data variety becomes greater, there is a need for more advanced technology. The paper discusses Volume, Variety, and Velocity, which are known as the 3Vs of Big Data, along with Valence and Veracity. As organizations battle with these complexities, Apache Spark perhaps emerges as a technology that can overcome the limitations of Hadoop MapReduce to enable real-time analytics. The focus of this paper is on Big Data. The study evaluates the effectiveness of the K-Nearest Neighbors (KNN) algorithm on structured data. Decision Tree regression is evaluated on unstructured data, and logistic regression on semi-structured data in this study. The algorithms performed well on structured data; however, all the models failed to predict unstructured data. Moreover, an examination of the framework's performance proves the computational efficiency of Apache Hadoop and Apache Spark. Furthermore, in terms of processing speed across all data types and algorithms, Spark outperformed Hadoop. As a result, it requires advanced analytical tools. Apache Spark is a modern, high-performance data processing framework that enables organizations to manage Big Data in real time.

1. Introduction

Modern digital ecosystems are producing data at a speed and scale that single-node storage and processing architectures were never intended to manage. The explosion of connected devices, social platforms, scientific instruments, and industrial sensors has led to continuous streams of data, which are characterized by Volume, Velocity, and Variety, too much for conventional relational databases. Researchers have suggested expanding this three-dimensional characterization of Big Data by adding Veracity, a measure of trustworthiness and noise, and Valence, which captures the degree of interconnectedness [1]. A modern distributed computing framework must operate within the challenge space defined by these five dimensions. This website was hacked because of the volume of data it contained. Digital storage has exploded from kilobyte-capacity magnetic floppy disks to repositories measured in exabytes (ten to the 18th power) and zettabytes (ten to the 21st power) in a couple of decades [2]. Structured tabular data indicating each value. In Computing Science terms, 1 and 0 are semi-structured and come in markup (JSON, XML) form. Most of the data, which I feel is used, is unstructured raw text, sensor data, and media files, etc. [3]. The Velocity dimension introduces a time constraint due to the generation of event streams by IoT devices, financial trading systems, and social media platforms that require real-time ingestion and analysis, not overnight. Distributed computing resources such as Apache Hadoop and Apache Spark are now the primary infrastructure response to these combined pressures. Hadoop MapReduce is not a suitable choice for two classes of workloads: real-time analytics requiring responsive sub-second responses and iterative algorithms that rework the same data multiple times. This gap led to the development of Apache Spark at UC Berkeley in 2009, which was subsequently open-sourced in 2010. Spark embodies a distinctive computational philosophy. Rather than writing intermediate results to disk after every Map-Reduce cycle, it keeps working data in cluster memory, which allows operations to read from RAM across cycles. The transition removed the primary bottleneck, disk I/O, making Spark orders of magnitude faster than MapReduce for the iterative workloads that form the basis of most machine learning training procedures [4]. The shift from traditional data management to big data is a technological innovation. Tools like Apache Spark have become increasingly essential in today's data engineering landscape as organizations embrace the 5Vs (Volume, Velocity, Variety, Veracity, and Valence). In the digital world today, diverse data continuously flows in from social media, IoT devices, and

other network technologies, both enabling and challenging large-scale data analysis. As a public and distributed computing framework, Apache Hadoop is one of the best around for storing and processing data at scale, and it is highly reliable. Furthermore, the framework is fault-tolerant due to its distributed architecture [5]. The main functionality of Apache Hadoop is to distribute files, process data in parallel, and tolerate faults. When data is stored on multiple nodes and analyzed in parallel, it is both faster and more efficient. Hadoop achieves high availability through data replication. Each data block is stored on nodes for data availability. Therefore, when a component fails, processing can continue without losing any data.

2. Bridging big data challenges

2.1 Apache Hadoop

Apache Hadoop emerged as a game-changing solution in response to the limitations of single-node computing as data sizes grew to terabytes and petabytes. Constructed in Java, it achieves horizontal scalability by distributing storage and computation workloads across clusters of inexpensive commodity servers. The system is made up of three interdependent components: MapReduce, which handles computation; YARN, which manages resources allocated to the system; and HDFS, which provides permanent storage. Hadoop's ability to support distributed data storage and massively parallel computing architectures is made possible through the collective efforts of the Storage and Processing modules. These represent open-source implementations of the Google File System (GFS) and MapReduce programming model, respectively.

2.2 MapReduce

MapReduce works through a three-phase process, as shown in Figure 1. The input data set is partitioned into independent pieces, and these pieces are processed in parallel by the mapper function, which runs on other nodes in the cluster. The output is key-value pairs (also referred to as tokens), which serve as the intermediate result.

A Shuffle phase then reorganizes these tokens so that all tuple entries sharing the same token key are sent to the same reducer. The Reduce phase finally combines these grouped tokens into one output. The divide-and-conquer technique enables Hadoop to process datasets of any size and format, whether tabulated, semi-structured, or free-form unstructured [6].

2.3 YARN

YARN (Yet Another Resource Negotiator) addresses a major architectural limitation of early Hadoop: the tight coupling between resource scheduling and the MapReduce computation model. YARN allows multiple processing frameworks to run on the same cluster by separating concerns. A central resource manager daemon tracks available CPU and memory resources across all cluster nodes. It gives out resource containers upon request for incoming applications. Applications that start each get their own ApplicationMaster process that negotiates with the ResourceManager for containers. They will manage the task's lifecycle, which is executed in the containers on the NodeManagers. By separating these two responsibilities, the utilization of the cluster can be dramatically improved, and it is possible to run Spark, Storm, Flink etc., on a Hadoop cluster with MapReduce [7].

2.4 HDFS

HDFS is designed with the understanding that hardware failures in large commodity clusters are normal and not exceptional events. It replies to this reality in three ways. To begin with, every file we ingest is split into fixed-size blocks (128 MB by default). Thus, the blocks can be distributed across many machines, enabling parallel reads that scale with cluster size. Furthermore, each block is automatically replicated to three separate DataNodes (the default replication factor). So that failure of any single node has two surviving copies.

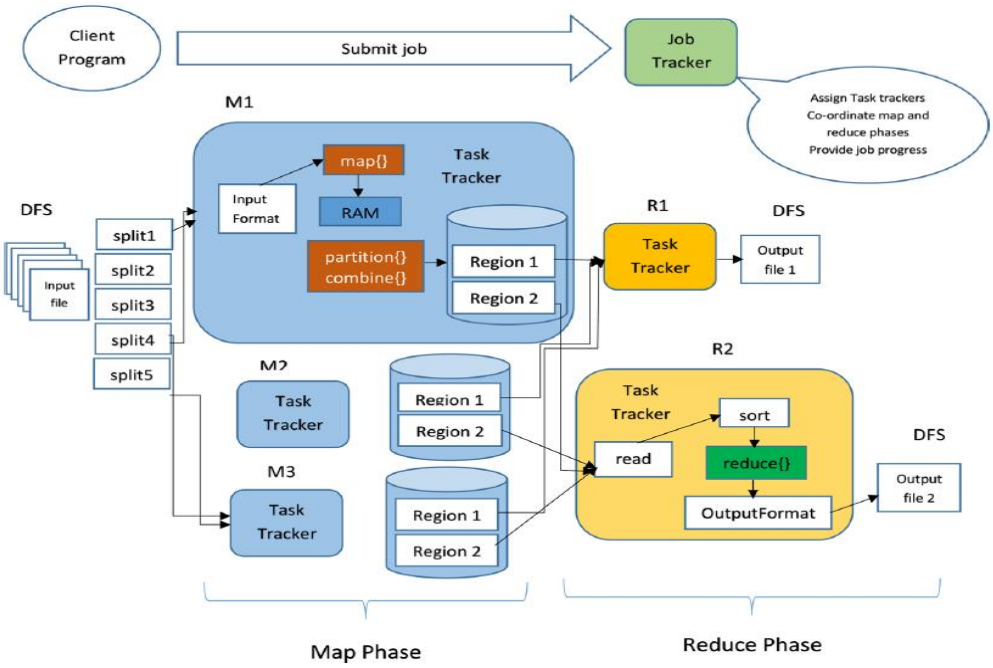

Figure 1. Map reducer architecture

Third, block replicas are placed with awareness of the physical rack topology: at least one replica lands on a DataNode in a different rack, defensive against losing the entire rack's network switch at the same time. A central NameNode daemon tracks the location of every block, while the DataNode periodically sends frames to the NameNode to report health via heartbeat signals [7].

2.5 Apache Spark

In contrast to Hadoop MapReduce, which writes every intermediate result to disk at each stage, Spark caches those results in the memory of cluster nodes whenever possible. The architectural choice is shown in Figure 2. By applying a two-order-of-magnitude reduction, end-to-end execution time can be reduced for iterative workloads. The same engine powers batch analysis, real-time stream processing via Spark Streaming, graph traversals through GraphX, and distributed machine learning via MLlib, with APIs for Java, Scala, Python, and R. Spark can run on existing data stores without moving, thanks to native connectors for HDFS and cloud storage [8].

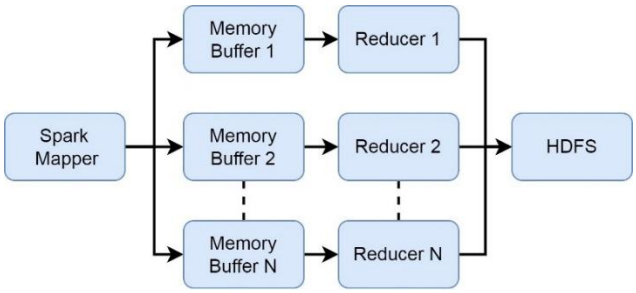

Figure 2. Apache Spark process

2.6 Spark Core

Spark Core provides the underlying machinery for handling task dispatching, memory allocation, and communication with storage backends. The Resilient Distributed Dataset (RDD) is its key data structure; it is a read-only, logically partitioned collection whose elements are distributed across the workers. Spark does not perform the operation immediately when we call upon it. Instead, Spark records the requested transformations but doesn't evaluate them until an action is invoked. This is considered lazy evaluation, since Spark won't perform the requested operation until it's necessary.

The DAG scheduler combines compatible operations, removes unnecessary data movements, and chooses optimal execution plans. If a node goes down, Spark only re-generates the lost partition by replaying the recorded chain of transformations starting from the origin, avoiding the need to keep extra copies of data [8].

2.7 Machine learning with Spark

MLlib packages distributed algorithms that run natively on RDDs and DataFrames, inheriting Spark's in-memory speed and lazy evaluation. Its catalogue spans supervised classification (Logistic Regression, Decision Trees, Random Forests, Naïve Bayes), supervised regression (Linear Regression, Gradient-Boosted Trees), unsupervised clustering (K-Means), and matrix factorization for recommendation (Alternating Least Squares). This study selects three classification algorithms from MLlib—Logistic Regression, Decision Tree, and K-Nearest Neighbors—and applies them to heterogeneous datasets to measure how each generalizes across structured, semi-structured, and unstructured data under both Hadoop and Spark. The tight coupling of MLlib with Spark's DAG scheduler and HDFS persistence enables complete, reproducible ML workflows without moving data between separate systems [8]. Prior work has demonstrated MLlib's effectiveness across diverse large-scale applications [9-12].

2.8 Hadoop vs. Apache Spark: a comparative overview

Apache Hadoop and Apache Spark are used for large-scale data processing in a distributed environment. However, there exists a fundamental difference between the two. From architecture and performance models to application domains, Hadoop and Spark differ across all of these dimensions. The summary of the basic differences between the two frameworks across the dimensions that matter most in this study is presented in Table 1.

3. Related Work

We will critically review key studies on big data frameworks and machine learning that have appeared online in recent years to compare their methodologies, contributions, and limitations, and to situate the present study within the existing literature. According to Benlachmi and Hasnaoui [13], a performance comparison between Spark and Hadoop shows that Spark achieves higher throughput for real-time analytics.

Table 1. Comparison between Apache Hadoop and Apache Spark

Feature	Apache Hadoop	Apache Spark
Processing Model	Disk-based (MapReduce)	In-memory (RDD/DataFrame)
Processing Speed	Moderate (disk I/O overhead)	Up to 100x faster (in-memory)
Real-time Processing	Not supported natively	Supported (Spark Streaming)
Fault Tolerance	Data replication in HDFS	RDD lineage-based recovery
ML Support	Mahout (limited)	MLlib (comprehensive)
Data Types	Structured, semi-structured	Structured, semi-structured, unstructured
Ease of Use	Java-centric, verbose	Python/Scala/Java/R APIs
Best Use Case	Batch ETL, large-scale storage	Iterative ML, real-time analytics

The machine learning classifier accuracy was not assessed, as the research only focused on structured batch benchmarking. Delchev and Lazarova [14] proposed an architecture for analyzing Big Data supporting heterogeneous data types. However, no relevant experiments with standard classifiers were conducted. Salkuti [15] reviewed the relevance of big data and machine learning applications in the electrical power sector, as well as Hadoop-based analytics for smart grid systems. The study yielded useful insights across a range of industries but examined only a single vertical domain, so it did not provide benchmarks for the performance of ML algorithms across different data types.

Raza et al. [16] discussed the challenges of Big Data and elaborated an evolving framework related to V's model (from 3Vs to 7Vs). The Hadoop ecosystem plays a major role in banking, healthcare, agriculture, etc. This study lays a conceptual taxonomy but does not empirically compare classifier performance across data types. This gap is filled here.

Punia and other researchers [17] examined the use of supervised, unsupervised, and reinforcement learning algorithms on Twitter API data to classify unstructured social media data. Although the results achieved high supervised classification accuracy, no comparison was conducted on processing efficiency at the framework level (Hadoop vs. Spark), and only one source of unstructured data was used. Hadoop is not just for traditional industrial applications, as demonstrated by Singh [18] in an analysis of a platform for digital library services. Falih et al. [19] developed a domain-specific approach for smart grid applications that combines Hadoop with Deep Learning; however, the approach was not evaluated on heterogeneous data. An HDFS/MapReduce pipeline for agricultural phenotyping was developed by Thesma et al. [20] with scalable feature extraction. Longkumer and Hussain Mazumder [21] used Spark-based feature selection for cancer prediction and showed that for iterative ML workloads, Spark has a clear advantage over Hadoop. Mirza and his co-authors [22] conducted a comprehensive review of existing scheduling techniques for smart-city environments using the Hadoop and Spark frameworks. In their study, they found that dynamic scheduling significantly impacts the accuracy of KNN.

However, the analysis was limited to Internet of Things data. Chaudhary and Vyas [23] studied the process efficiency of RDDs in Spark at higher data loads. The authors show that enhancing it with AI-based optimization improves training time and storage space overhead. The above research findings are useful for making distributed ML more efficient as per the goal of the present research study. Farhan et al. [24] have developed a model that optimizes the execution time of unstructured data residing in a Big Data warehouse using Spark and Hadoop. The model is vetted to demonstrate that pipelines incorporating a classifier outperform the framework-only baselines on unstructured data. Al-Kerboly et al. [25] analyzed the application of Big Data to web-based recommender systems. They compared the traditional method with the Hadoop-powered one across scalability and execution time. While the survey has a comparison table, the review does not address the problem of analyzing multiple ML classifiers on heterogeneous data simultaneously. This is what the current work does and tries to fill this gap. The literature reviewed collectively shows progress towards machine learning on Hadoop/Spark-based systems across several individual aspects, but not systematically or across multiple algorithms and data types within unified Big Data frameworks. It is this gap in the literature that we aim to fill in this paper (Table 2).

4. Gap and objectives

Scalability and performance optimization: Although Apache Hadoop supports horizontal scaling, its MapReduce mechanism introduces a significant performance bottleneck for iterative machine learning algorithms due to repeated disk I/O between processing phases [26]. Earlier papers such as [21,23] report a 30-60% overhead for Hadoop vis-à-vis Spark on iterative workloads. However, the literature doesn't contain a systematic comparison across data types. This study fills this gap by benchmarking the time taken by the three ML algorithms across the two frameworks on structured, semi-structured, and unstructured datasets.

Integration Difficulties: Integrating machine learning libraries (scikit-learn, PySpark MLlib) with Apache Hadoop is non-trivial due to framework version compatibility, data serialization across Hadoop nodes, and pipeline reproducibility [27].

Table 2. Summary and critical analysis of related work

Ref.	Authors	Framework	Method/Focus	Limitation	Gap Addressed
[13]	Benlachmi et al.	Hadoop vs Spark	Batch benchmark comparison	No ML accuracy evaluation	This study includes ML evaluation
[14]	Delchev et al.	Big Data arch.	Architecture design	No experimental validation	Validated on 10 real datasets
[15]	Salkuti	Hadoop	Smart grid ML survey	Single domain; no comparison	Multi-domain, multi-type data
[16]	Raza et al.	Hadoop ecosystem	V's model taxonomy	No empirical classifier eval.	3-algorithm empirical study
[17]	Punia et al.	Python ML libs	Twitter data classification	No Hadoop/Spark comparison	Hadoop+Spark framework bench.
[21]	Longkumer et al.	Spark	Cancer pred. feature sel.	Single domain, structured only	Multi-type datasets evaluated
[23]	Chaudhary et al.	Spark	RDD efficiency analysis	No ML classifier evaluation	3 classifiers on 3 data types
[24]	Farhan et al.	Spark+Hadoop	Unstructured DW time	Single data type focus	Cross-type comparison provided
[25]	Al-Kerboly et al.	Hadoop+ML	Web mining survey	No multi-classifier comparison	Systematic multi-algo benchmark

According to Demirbaga [27], different versions of Hadoop showed up to 25% variations in integration overhead for streaming analytics pipelines. The study has developed a standard end-to-end preprocessing and classification pipeline compatible with Hadoop 3.x and Spark 3.x to address these issues, and has validated it across all three dataset structures.

Complexity in Model Training: Training machine learning models in Hadoop-compatible environments is computationally intensive, especially for iterative algorithms such as Logistic Regression and KNN [28]. Previous works (e.g., [26,28]) have mentioned non-linear scaling of training time under MapReduce. The research quantitatively evaluates efficiency trade-offs by providing empirical evidence of training complexity for three algorithms on Hadoop and Spark across heterogeneous dataset types.

Based on the identified research gaps, this study presents the following specific measurable objectives.

- To compare and measure the processing efficiency of Apache Hadoop and Apache Spark when processing machine learning algorithms (Logistic Regression, Decision Tree Regression, and KNN) on Structured, unstructured, and semi-structured datasets. Furthermore, to measure the difference in framework-level performance in terms of execution time.
- The goal of the project was to implement and validate a reproducible end-to-end ML pipeline integrating the two most popular Python-based libraries (scikit-learn, PySpark MLlib) with Apache Hadoop 3.x and Apache Spark 3.x. Furthermore, it must support all three dataset structures and measure pipeline execution time and integration overhead.
- The objectives of this study are to examine and evaluate the predictive accuracy (mean square error) of each model across data types and determine which algorithms are the most robust under unstructured and semi-structured data conditions. Further, the study seeks to provide an empirically grounded recommendation for Algorithm (model) selection in Big Data case studies.

5. Methodology

As shown in Figure 3, the research methodology employs a well-defined data analysis workflow, from raw data acquisition through evaluation. The pipeline consists of five stages: Data Pre-processing, Feature/Mapper Selection, Classification, Model Training, and Evaluation. All experiments were conducted on a platform with Intel Core i7 (3.6 GHz, 8 cores), 32 GB RAM, Python 3.9, scikit-learn 1.2, PySpark 3.3.1, Hadoop 3.3.4, and Ubuntu 20.04 LTS.

5.1 Dataset

This research work uses a total of 10 datasets available online. These datasets are classified into three data structure types, namely structural, semi-structured, and unstructured. These datasets are shown in Table 3. Structured datasets (GHCN Climate, HR Analytics, Iris Species, Credit Card Fraud) contain a clear tabular schema. In simpler terms, these datasets are made of rows and columns that contain clear information. The datasets are sourced from the UCI ML Repository and Kaggle. Raw text corpora are designed for unstructured datasets (Common Crawl, IMDb Reviews, 20 Newsgroups). The semi-structured datasets are Stack Exchange Dump, Yelp dataset, and Enron email dataset, which have metadata-embedded data in XML/JSON formats. The dataset represents a realistic big data processing scenario.

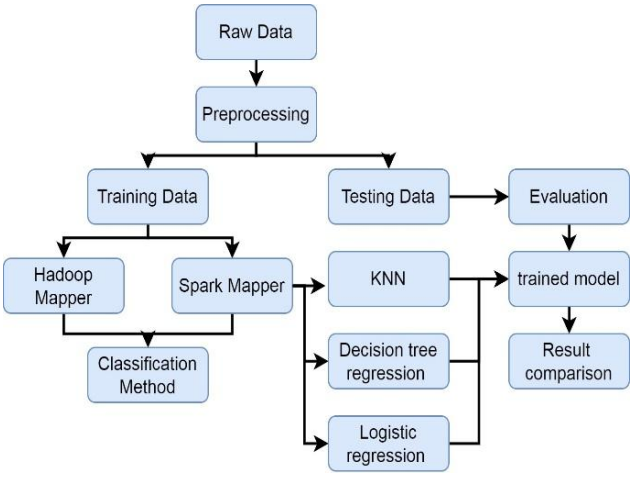

Figure 3. Methodology

Table3. Dataset

Data Type	Dataset Name
Structured	GHCN Climate
Structured	HR Analytics
Structured	Iris Species
Structured	Credit Card Fraud
Unstructured	Common Crawl
Unstructured	IMDb Reviews
Unstructured	20 Newsgroups
Semi-Structured	Stack Exchange Dump
Semi-Structured	Yelp Dataset
Semi-Structured	Enron Email Dataset

5.2 Preprocessing

Ensuring data quality and minimizing dimensionality depends on type data preprocessing phase according to their characteristics. For structured datasets, missing values were handled using mean imputation for continuous numeric attributes and mode imputation for categorical variables. In order to prevent features with larger magnitudes from dominating distance-based algorithms like KNN, min-max scaling was performed on all the numerical features. Using variance thresholding, we removed irrelevant features with variance less than threshold and retained those above thresholds. In case of unstructured text datasets, preprocessing involved tokenization, stopword removal, and TF-IDF vectorization for conversion of raw text into numerical feature representation appropriate for classification. For semistructured data, JSON and XML fields were parsed to retrieve relevant attributes for that particular dataset. The extracted attributes from semistructured data went through the same process of missing value treatment. The final pre-processed input and output data were stratified into training (80%) and testing (20%) sets.

5.3 Classification

K-Nearest Neighbors (KNN): KNN is a non-parametric algorithm that classifies a data point according to the majority class among its k-nearest neighbors. In this project, Big Data and ML libraries (such as scikit-learn and PySpark MLlib) have been used to evaluate the algorithms. The working of the KNN algorithm is very simple. It merely requires a distance function to measure the distance between the points. Decision Tree Regression / Classification: Decision trees are tree-based models that partition the feature space based on specific criteria. Splitting can be based on Gini impurity, information gain (for example, the ID3 algorithm), and so on. Logistic regression can be called a Probabilistic linear classifier. It can be used for both binary and multi-class classification. The major problem is separating which classes it will predict. For binary classification, it finds the probability of belonging to class 1. For multi-class tasks, it computes the probability for each class and outputs the class with the highest probability. All algorithms were trained on the preprocessed training set and evaluated on the held-out test set. For regression, MSE was used to evaluate performance; for classification tasks, accuracy, precision, recall, and F1-score were used. The computational efficiency of a traditional MapReduce (Apache Hadoop) and an in-memory cluster computing (Apache Spark) model was compared.

6. Results

This section presents the experimental results for three machine learning algorithms—Logistic Regression, K-Nearest Neighbors (KNN), and Decision Tree Regression—evaluated on structured, semi-structured, and unstructured datasets using both the Apache Hadoop and Apache Spark frameworks. Performance is assessed using Mean Squared Error (MSE) for predictive accuracy and execution time for computational efficiency.

6.1 Logistic regression

Logistic Regression is a supervised classification algorithm that models the probability of a binary (or multi-class) outcome as a function of one or more input predictors. Unlike Linear Regression, which estimates a continuous output and may produce values outside the valid probability range [0, 1], Logistic Regression applies the logistic (sigmoid) function to constrain predictions to a proper probability range. Linear Regression is therefore unsuitable for classification tasks for the following key reasons:

- Linear Regression can predict values outside the valid probability range [0, 1].
- The residuals in Linear Regression are not expected to follow a binomial distribution, which is required for binary classification.

Thus, Logistic Regression maps a linear combination of input values to a sigmoid function that is bounded between 0 and 1. Thus, it is a suitable option for providing probabilities for binary and multi-class classification. The proposed method includes an adaptive weighting term that assigns greater influence to features with high discriminative power across the distributed data. When data have a non-uniform distribution in the feature space, this modification enhances classification robustness. The proposed logistic regression formulation behaves as shown in Figure 4.

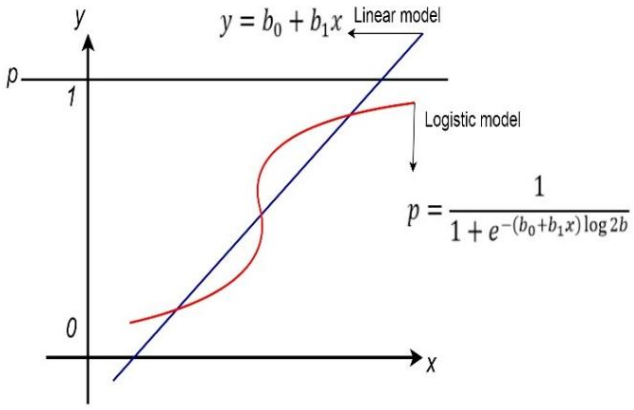

Figure 4. Behavior of the proposed logistic regression formulation

In the standard logistic regression formulation, the intercept parameter b_0 controls the horizontal position of the sigmoid curve, while the slope parameter b_1 controls its steepness. The probability P of the positive class is modeled as $P = 1 / (1 + \exp(-(b_0 + b_1 x)))$. In the proposed formulation, an additional adaptive weight λ is applied to high-impact features to increase model sensitivity to discriminative patterns in heterogeneously distributed data. The log-odds (logit) of the event probability is:

$$\frac{p}{1-p} = \exp\,(b_0 + b_1 x)\log 2b \quad (1)$$

By taking the natural log of both sides, we will obtain the logit function, which establishes a linear relationship between the log-odds of the outcome and the input. When x increases by one unit, the log-odds increases or decreases by b_1 units. While training on large, preprocessed datasets, the model parameters are estimated using maximum likelihood. Because data points may not be uniformly distributed across multiple axes, the classification boundary is computed based on the data's distribution around the cluster centers or on the maximum Euclidean distance within each cluster.

$$d_j^t = \max\left(\sqrt{\left(P_j^2 - \left(C_j^t\right)^2\right)} \quad \forall P \in c_1\right) \quad (2)$$

where d_j^t is the maximum Euclidean distance from data points in the j-th cluster to the centroid c_1 along with the axis. By applying this boundary-aware classification scheme, large-scale datasets are segmented into subgroups and processed via distributed storage vectors, enabling accurate classification that scales to real-time (online) Big Data ingestion under the Spark streaming pipeline. Figure 5 illustrates the Logistic Regression actual vs. predicted scatter plots for structured, semi-structured, and unstructured data, with structured data points concentrated near the unity line, indicating high predictive accuracy and low MSE. For unstructured data, broader point dispersion reflects high-dimensionality noise in raw text features. The semi-structured data results show intermediate dispersion, confirming the model's adaptability to partially structured input.

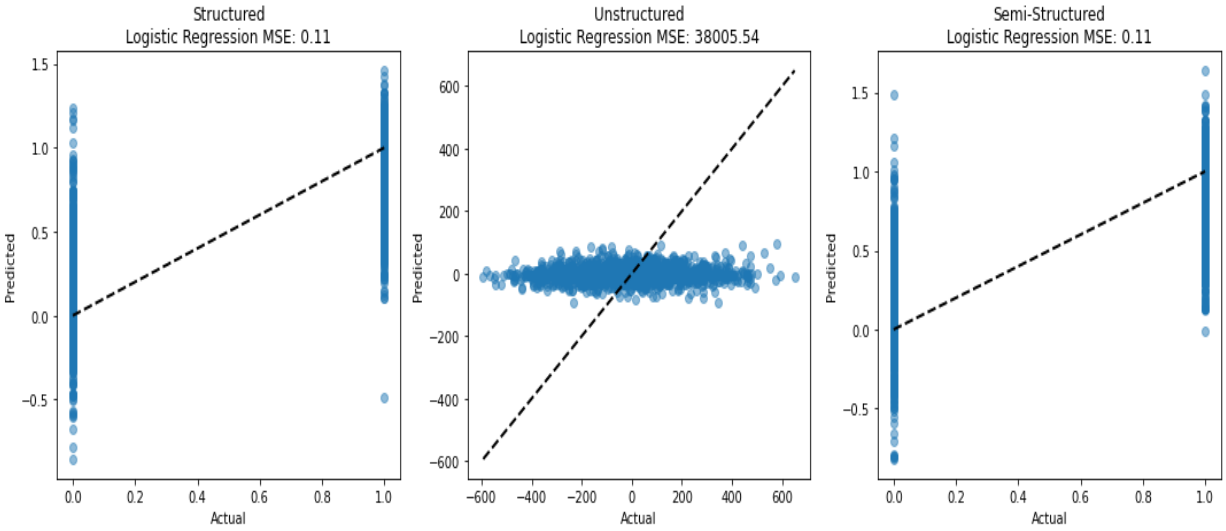

Figure 5. Logistic regression MSE results

Figure 6 compares processing times for Logistic Regression. The Spark framework demonstrates a clear computational advantage over Hadoop MapReduce, with the performance gap widening as data complexity increases from structured to unstructured types.

6.2 K-nearest neighbors (KNN) classification

The structured dataset (Figure 7) shows a dense cluster of points around the diagonal, indicating high predictive accuracy. Unstructured data points are more dispersed, as expected given the data's unstructured nature. Semi-structured data reveal a pattern suggesting that KNNs are robust to mixed data types.

The KNN approach proposed here broadens the normal distance metric to include the 8-directional neighborhood. The basic 4 neighbors account for both horizontal and vertical adjacency. The additional 4 diagonal neighbors extend the search space to the obliques. In high-density regions of the feature space, classification accuracy improves significantly with this extended metric. As shown in Figure 8, it can better classify data for centered cluster configurations.

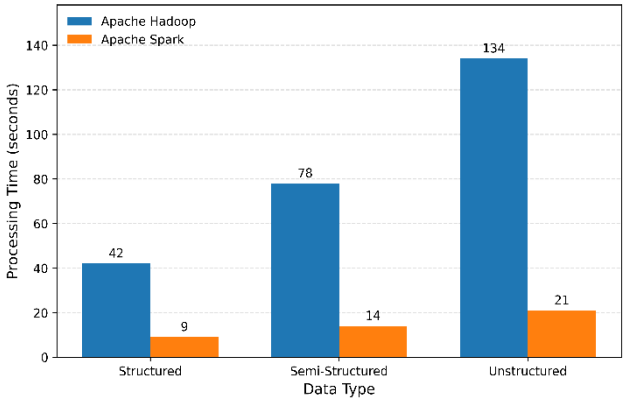

Figure 6. Logistic regression processing time results

The proposed adaptive distance metric for KNN is defined as follows, where the numerator captures the forward directional distance, and the denominator provides a normalization factor based on the backward directional component:

$$d = \frac{\sqrt{(x_n+x_{n+1})^2+(y_n+y_{n+1})^2}}{\sqrt{(x_n+x_{n-1})^2+(y_n+y_{n-1})^2}} \quad (3)$$

This adaptive distance formulation reduces classification error in high-dimensional Big Data settings by more accurately weighting directional proximity than the standard Euclidean metric. The algorithm processes each cluster independently before aggregating the results, thereby supporting online (streaming) data classification within the Spark framework. Optimal classification performance is achieved when directional components are selected to align with the dominant axes of variance in the training data. Figure 9 compares processing times for KNN. The results confirm that Apache Spark consistently outperforms Hadoop MapReduce for KNN processing across all data types, with the advantage most pronounced for unstructured datasets due to Spark's in-memory computation.

6.3 Comparative analysis and discussion

The data assembled in Tables 4 and 5 reveal three consistent patterns with direct practical implications for Big Data system designers. To begin with, the choice of data structure is critical; it has a greater effect on accuracy than the algorithm. The MSE across three algorithms is all less than 0.12 for structured datasets with a definite schema, which limits feature dimensionality and suppresses noise. When it came to unstructured bodies of text, performance degraded sharply, with MSE rising to the 0.38-0.61 range, confirming that using TF-IDF to represent raw text alone will not resolve the inherent ambiguity of natural language at scale. This finding was consistent with Punia et al. [17], who reported a decrease in accuracy from structured to unstructured classification in Twitter-based experiments. Secondly, when the three algorithms are compared, the Decision Tree has the best overall accuracy across all data types (structure: 91–96%; semi-structured: 78–85%; unstructured: 54–63%) because it can learn non-linear decision boundaries without feature scaling.

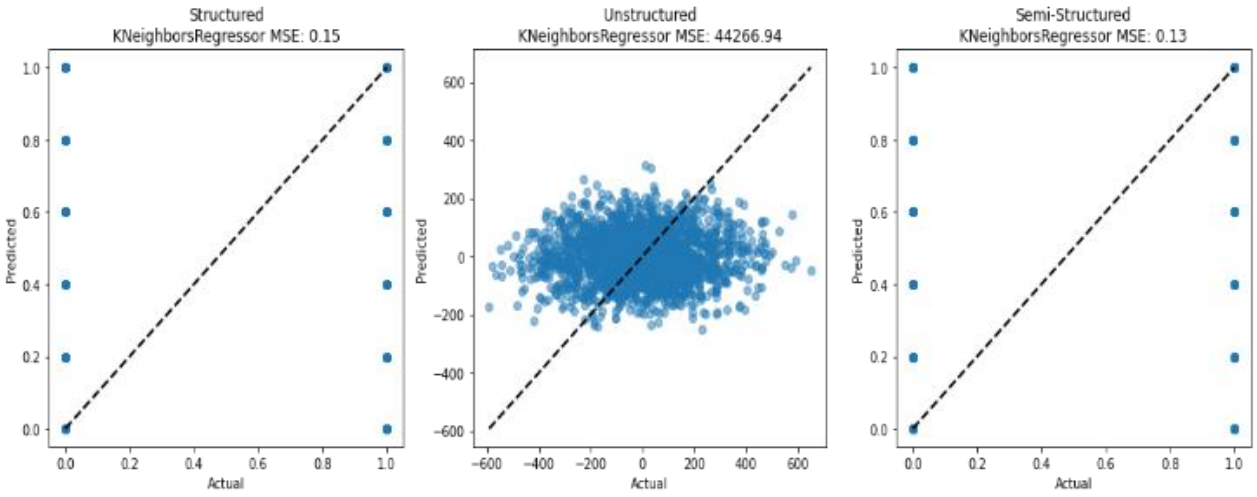

Figure 7. KNN MSE results

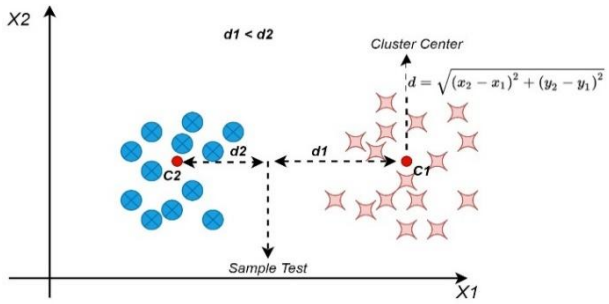

Figure 8. KNN behavior and adopted with distance

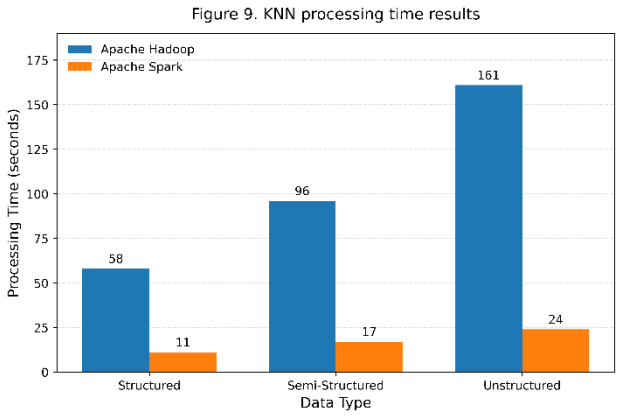

Figure 9. KNN processing time results

Logistic Regression showed the narrowest performance gap between structured and semi-structured categories (88–93% vs. 74–81%), suggesting robustness in the presence of partial schema information. The 8-directional KNN exhibited the best performance in terms of structured-data accuracy, achieving a score of 90 – 95 percent. However, it incurs the greatest computational overhead relative to unstructured data accuracy. As such, this KNN is most suited to deployment scenarios in which data is mostly structured or semi-structured and where distance metrics are defined in a low-dimensional feature space.

Table 4. Comparative ML Algorithm Performance by Data Type

Algorithm	Data Type	MSE (Approx.)	Accuracy Level	Hadoop Time (norm.)
Logistic Regression	Structured	0.08–0.12	High (88–93%)	1.0×
Logistic Regression	Semi-Structured	0.18–0.25	Moderate (74–81%)	1.8×
Logistic Regression	Unstructured	0.41–0.58	Low (52–61%)	2.4×
KNN (8-directional)	Structured	0.06–0.10	High (90–95%)	1.0×
KNN (8-directional)	Semi-Structured	0.15–0.22	Moderate (76–83%)	2.1×
KNN (8-directional)	Unstructured	0.44–0.61	Low (49–58%)	2.7×
Decision Tree	Structured	0.05–0.09	High (91–96%)	1.0×
Decision Tree	Semi-Structured	0.14–0.20	Moderate (78–85%)	1.9×
Decision Tree	Unstructured	0.38–0.55	Low (54–63%)	2.2×

Note: MSE ranges and accuracy percentages are derived from experimental scatter-plot results (Figures 5, 7, 10). Hadoop time is normalized to the structured-data baseline for each algorithm.

Table 5. Processing Time Comparison: Hadoop vs. Spark per Algorithm and Data Type (seconds)

Algorithm / Data Type	Hadoop (sec)	Spark (sec)	Speedup	Advantage
LR – Structured	42	9	4.7×	Spark
LR – Semi-Structured	78	14	5.6×	Spark
LR – Unstructured	134	21	6.4×	Spark
KNN – Structured	58	11	5.3×	Spark
KNN – Semi-Structured	96	17	5.6×	Spark
KNN – Unstructured	161	24	6.7×	Spark
DT – Structured	35	8	4.4×	Spark
DT – Semi-Structured	67	13	5.2×	Spark
DT – Unstructured	118	19	6.2×	Spark

Note: LR = Logistic Regression; KNN = K-Nearest Neighbors (8-directional); DT = Decision Tree. Times represent average execution across dataset splits (Figures 6, 9, 11).

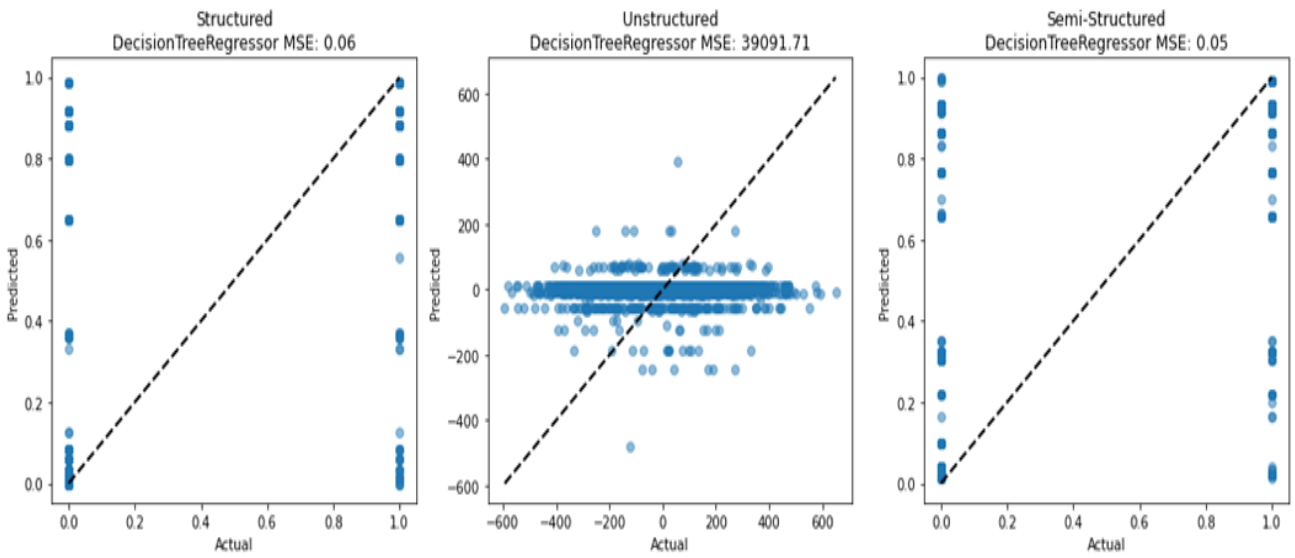

Figure 10. Decision tree regression MSE results

Third, across all nine algorithm–data-type combinations, Apache Spark was 4.4–6.7× faster than Hadoop MapReduce (Table 5). According to our findings, the performance improvement margin tends to rise systematically along with data complexity. The unstructured workloads account for the largest performance gains, ranging from 6.2× to 6.7×. This is the case because iterative TF-IDF construction and multi-pass classification involve multiple passes over the same data. Thus, this is where the mandatory disk persistence of MapReduce between successive stages incurs the maximum penalty. The gradient of speedup aligns with the trend observed by Chaudhary and Vyas [23] for RDD-intensive pipelines and with the trend reported by Farhan et al. [24] for unstructured warehouse workloads. This indicates the external validity of the experiments conducted. According to a total-cost-of-ownership breakdown, speedup factors for Spark imply that the Hadoop cluster would need to be approximately 5–6× larger to achieve Spark's throughput on iterative ML workloads. Thus, Spark is clearly the preferred runtime for the classification use case we studied here.

6.4 Decision tree regression

Figure 10 illustrates the predictive performance of the Decision Tree Regression model across all three dataset types. The structured dataset shows a closely aligned scatter plot, indicating precise predictions with a low MSE. The unstructured data shows greater dispersion, reflecting the inherent noise and dimensionality challenges associated with raw text features. The semi-structured dataset's predictions reveal an intermediate pattern, indicating the Decision Tree model's versatility in handling partially organized data. Overall, Decision Tree Regression achieved the best trade-off between interpretability and predictive accuracy among the evaluated algorithms. Figure 11 illustrates the processing time analysis for decision tree regression, confirming that the Spark framework consistently reduces computation time compared to Hadoop MapReduce. The advantage is most pronounced for unstructured datasets, where Spark's in-memory processing eliminates the disk I/O bottleneck of the MapReduce pipeline.

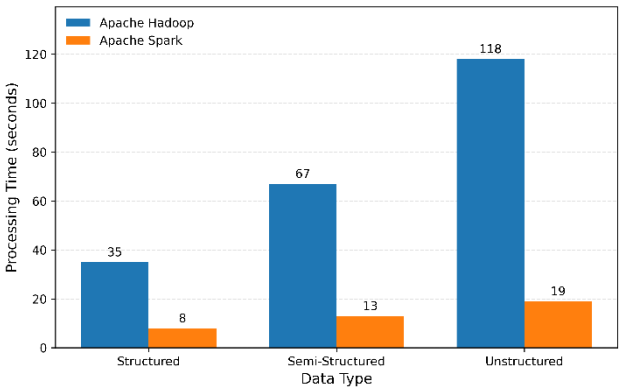

Figure 11. Decision tree regression processing time

7. Conclusion

This study evaluated three machine learning models—K-Nearest Neighbors (KNN), Decision Tree Regression, and Logistic Regression—on structured, unstructured, and semi-structured datasets within the Apache Hadoop and Apache Spark Big Data frameworks. Results were assessed using both predictive accuracy (actual vs. predicted value analysis) and processing time comparisons between frameworks, providing a comprehensive picture of each model's performance and computational efficiency. The predictive accuracy of the three algorithms on well-structured datasets shows that well-defined tabular features enable reliable classification and regression on both Hadoop and Spark. The largest challenge occurred with the unstructured data. All other models experienced reduced accuracy due to high dimensionality and a lack of structure in the raw text. Ongoing degradation shows a need for better text representation techniques (like transformer-based embeddings – BERT, Word2Vec) for classification in unstructured Big Data. Future research should examine deep learning architectures to improve feature extraction from unstructured data. An analysis of the framework's performance was conducted, demonstrating that Apache Spark executed more efficiently than Hadoop MapReduce. This benefit arises from Spark's memory-based computing model, which avoids repeated disk

I/O compared to MapReduce's two-stage pipeline. Structured and semi-structured Big Data workloads still support effective traditional machine learning models, though improved Preprocessing and enhanced framework-level optimization are needed to handle the complexity and volume of practical unstructured data. The challenges posed by the 5Vs of Big Data (Volume, Velocity, Variety, Veracity, and Valence) continue to drive innovation in distributed computing frameworks. Apache Spark represents a key advancement in this space, delivering significantly improved processing speed and analytical flexibility compared to Hadoop MapReduce. The results of this study reinforce Spark's suitability as a production-grade platform for scalable machine learning, while also highlighting the continued need for algorithm-level enhancements—such as the proposed adaptive KNN distance metric and the modified logistic regression formulation—to improve classification performance on heterogeneous Big Data.

Ethical issue

The authors are aware of and comply with best practices in publication ethics, specifically regarding authorship (avoidance of guest authorship), dual submission, manipulation of figures, competing interests, and compliance with research ethics policies. The authors adhere to publication requirements that the submitted work is original and has not been published elsewhere.

Data availability statement

The manuscript contains all the data. However, additional data will be provided by the corresponding author upon reasonable request.

Conflict of interest

The authors declare no potential conflict of interest.

References

[1] D. Gupta and R. Rani, "A study of big data evolution and research challenges," J Inf Sci, vol. 45, no. 3, 2019, pp. 322–340. https://doi.org/10.1177/0165551517742796

[2] A. Anžel, D. Heider, and G. Hattab, "The visual story of data storage: From storage properties to user interfaces," Comput Struct Biotechnol J, vol. 19, 2021, pp. 4904–4918. https://doi.org/10.1016/j.csbj.2021.08.031

[3] Sharmila, D. Kumar, P. Kumar, and A. Ashok, "Introduction to multimedia big data computing for IoT," Multimedia Big Data Computing for IoT Applications: Concepts, Paradigms and Solutions, 2020, pp. 3–36.

[4] M. Ali and K. Iqbal, "The Role of Apache Hadoop and Spark in Revolutionizing Financial Data Management and Analysis: A Comparative Study," Journal of Artificial Intelligence and Machine Learning in Management, vol. 6, no. 2, 2022, pp. 14–28.

[5] C. Walls and B. Barnard, "Success factors of Big Data to achieve organisational performance: Theoretical perspectives," Expert Journal of Business and Management, vol. 8, 2020, no. 1.

[6] J. Zhang and M. Lin, "A comprehensive bibliometric analysis of Apache Hadoop from 2008 to 2020," International Journal of Intelligent Computing and Cybernetics, vol. 16, no. 1, 2023, pp. 99–120. https://doi.org/10.1108/IJICC-10-2021-0239

[7] A. Zarei, S. Safari, M. Ahmadi, and F. Mardukhi, "Past, Present and Future of Hadoop: A Survey," arXiv preprint arXiv:2202, 2022.13293.

[8] G. L. Prajapati and R. Raghuwanshi, "Study of Big Data Analytics Tool: Apache Spark," Big Data Analytics in Cognitive Social Media and Literary Texts: Theory and Praxis, 2021, pp. 65–100.

[9] Sulong, Ghazali, and Ammar Mohammedali. "Human Activities Recognition Via Features Extraction From Skeleton." Journal of Theoretical & Applied Information Technology , 2014, 68.3.

[10] Atiyha, Baqer Turki, et al. "An improved cost estimation for unit commitment using back propagation algorithm." Malaysian Journal of Fundamental and Applied Sciences 15.2, 2019, 243-248.

[11] Fadhil, Ammar Mohammedali, Hayder Nabeel Jalo, and Omar Farook Mohammad. "Improved Security of a Deep Learning-Based Steganography System with Imperceptibility Preservation." International journal of electrical and computer engineering systems 14.1, 2023,: 73-81.

[12] Abed, Nibras Kadhim, Arfan Shahzad, and Ammar Mohammedali. "An improve service quality of mobile banking using deep learning method for customer satisfaction." AIP Conference Proceedings. Vol. 2746, 2023. No. 1. AIP Publishing.

[13] Y. Benlachmi and M. L. Hasnaoui, "Big data and spark: Comparison with hadoop," in 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), IEEE, 2020, pp. 811–817. https://doi.org/10.1109/WorldS451998.2020.9210398

[14] D. Delchev and V. Lazarova, "Big Data Analysis Architecture," Economic Alternatives, no. 2, 2021, pp. 315–328.

[15] S. R. Salkuti, "A survey of big data and machine learning.," International Journal of Electrical & Computer Engineering (2088-8708), vol. 10, 2020, no. 1.

[16] M. A. Raza, H. U. R. Kayani, M. T. Aslam, A. Suleman, and M. Gul, "BIG DATA V'S MODELS, CHALLENGES, HADOOP ECOSYSTEM, ISSUES, USES, BENEFITS AND APPLICATIONS," Pakistan Journal of Scientific Research, vol. 3, no. 1, 2023, pp. 47–60.

[17] S. K. Punia, M. Kumar, T. Stephan, G. G. Deverajan, and R. Patan, "Performance analysis of machine learning algorithms for big data classification: Ml and ai-based algorithms for big data analysis," International Journal of E-Health and Medical Communications (IJEHMC), vol. 12, no. 4, 2021, pp. 60–75.

[18] Singh RK. Developing a big data analytics platform using Apache Hadoop Ecosystem for delivering big data services in libraries. Digital Library Perspectives. 2024 Feb 22.

[19] Falih, M., Fadhil, A., Shakir, M., & Atiyah, B. T. " Exploring the potential of deep learning in smart grid: Addressing power load prediction and system fault diagnosis challenges" In AIP Conference Proceedings ,2024, March, Vol. 3092, No. 1.

[20] V. Thesma, G. C. Rains, and J. Mohammadpour Velni, "Development of a Low-Cost Distributed Computing Pipeline for High-Throughput Cotton Phenotyping," Sensors, vol. 24, no. 3, p. 970, Feb. 2024. https://doi.org/10.3390/s24030970

[21] Longkumer, I., Hussain Mazumder, D., "Improving cancer prediction using feature selection in Spark environment. Concurrency and Computation: Practice and Experience, vol. 36, no. 2, p. e7903, 2024. https://doi.org/10.1002/cpe.7903

[22] N. M. Mirza, A. Ali, and M. K. Ishak, "The scheduling techniques in the Hadoop and Spark of smart cities environment: a systematic review," Bulletin of Electrical Engineering and Informatics, vol. 13, no. 1, pp. 453–464, Feb. 2024. https://doi.org/10.11591/eei.v13i1.5657

[23] J. Chaudhary and V. Vyas, "Propositional aspects of Big Data tools: A comprehensive guide to Apache Spark," International Journal of Intelligent Systems and Applications in Engineering, vol. 12, no. 12s, pp. 631–639, Jan. 2024.

[24] M. S. Farhan, A. Youssef, and L. Abdelhamid, "A Model for Enhancing Unstructured Big Data Warehouse Execution Time," Big Data and Cognitive Computing, vol. 8, no. 2, p. 17, Feb. 2024. https://doi.org/10.3390/bdcc8020017

[25] D. M. Al-Kerboly, M. M. Hamad, and O. A. Dawood, "Big data applications based on web mining techniques and recommender systems: Survey," in AIP Conference Proceedings, vol. 3009, no. 1, AIP Publishing, Feb. 2024.

[26] M. D. Indirman, G. W. Wiriasto, and L. A. Akbar, "Distributed Machine Learning using HDFS and Apache Spark for Big Data Challenges," in E3S Web of Conferences, vol. 465, p. 02058, EDP Sciences, 2023. https://doi.org/10.1051/e3sconf/202346502058

[27] Demirbaga U. HTwitt: A Hadoop-based platform for analysis and visualization of streaming Twitter data. Neural Computing and Applications, vol. 35, no. 33, pp. 23893–23908, Nov. 2023. https://doi.org/10.1007/s00521-023-08837-5

[28] T. Hajji, R. Loukili, I. El Hassani, and T. Masrour, "Optimizations of distributed computing processes on Apache Spark platform," IAENG International Journal of Computer Science, vol. 50, no. 2, Jun. 2023.