Journal of Artificial Intelligence and Data Mining (JAIDM), Vol. 13, No. 4, 2025, 441-454.

Shahrood University of Technology

Journal of Artificial Intelligence and Data Mining (JAIDM)
Journal homepage: http://jad.shahroodut.ac.ir

Research paper

A New Framework for Data Reduction in Large-scale Data Using Mapreduce

Zeinab Abbasi*

Faculty of Engineering, Mahallat Institute of Higher Education, Mahallat, Iran.

Article Info

Article History:
Received 04 May 2025
Revised 08 June 2025
Accepted 10 July 2025

DOI:10.22044/jadm.2025.16130.2732

Keywords:
Large-scale Data, MapReduce, Feature Selection, Instance Selection.

**Corresponding author: z.abbasi@mahallat.ac.ir (Z.Abbasi).*

Abstract

Storing and processing large volume datasets presents one of the most critical challenges in large-scale processing. Therefore, reducing their size before further processing is essential. This paper proposes a framework for data reduction in large-scale datasets, based on the MapReduce algorithm. The framework comprises three steps. Firstly, reservoir sampling is used to select instances from the dataset. In the second step, the features of these selected instances are weighted using the ReliefF algorithm. Subsequently, the weights for each feature are averaged, and features with the highest average weights are selected. Finally, these selected features are used in the classification process. Implementation results demonstrate that the proposed framework effectively reduces processing time and, when a large amount of data is removed by eliminating irrelevant features, either increases or maintains classification accuracy.

1. Introduction

The extensive use of the internet, particularly social networks, has led to a rapid increase in the volume of information. Furthermore, sensors in the Internet of Things (IoT) and other tools generate vast amounts of data, requiring specialized storage and processing capabilities. Researchers refer to this data as large-scale data or massive data, characterized by its volume, variety, and velocity [1].

One challenge in storing and processing large-scale data is compressing and extracting useful information. Numerous methods have been developed to address this problem, including data reduction techniques. These methods aim to remove redundant and noisy data, thereby increasing the accuracy and efficiency of results while reducing processing time and costs [2]. They are often applied in the preprocessing phase of machine learning and data mining algorithms.

When data is represented as a table (with features as columns and instances as rows), three primary types of data reduction methods can be employed:

Feature reduction: This category encompasses feature selection and feature extraction methods. Feature selection involves selecting a subset of features and discarding the rest, while feature extraction combines existing features to create new ones.

Instance reduction: This type includes instance (or record) reduction methods, such as sampling (or instance selection) and information compression. Sampling involves randomly selecting a subset of instances [3] or using specialized methods [4]. Information compression, similar to feature extraction, combines and/or compresses instances into a smaller form [5, 6].

Discretizing feature values: These methods convert real-valued features into discrete values, reducing the domain of values for each feature [7].

Large-scale data often exhibits a high volume and may be distributed across multiple machines in a network. Consequently, processing such data using conventional systems can be costly, time-consuming, and sometimes infeasible. Parallel and distributed data reduction techniques are essential for tackling this challenge.

Numerous papers explore data reduction techniques for large-scale datasets and their

associated challenges. Papers [8, 9, 10] propose feature selection algorithms for this purpose. Researchers in [11, 12, 13, 14, 15] present various methods for feature reduction in large-scale datasets. Additionally, [16, 17, 12] investigate feature reduction techniques implemented in a distributed manner, particularly using MapReduce. Instance selection using MapReduce for imbalanced datasets [1], instance selection based on stratification strategies [7], and ensemble instance selection [18, 19] are other proposed approaches for data reduction based on instance selection. Studies like [20], [21], and [22] focus on instance selection for application in large datasets.

Efforts have been made to study instance and feature selection concurrently. Addressing both selection problems simultaneously is reasonable as they can be executed independently. This approach facilitates further data reduction and improves classification accuracy [2]. Studies such as [23, 24, 25, 26, 27, 28, 29] fall into this category. However, none of these studies have implemented the combination of feature and instance selection on large-scale datasets in a distributed manner. Moreover, most datasets used in reviewed papers do not have a large volume. Consequently, methods like evolutionary algorithms or ensembles of classifiers, which are employed as data reduction techniques, are feasible for these smaller datasets. Implementing these algorithms on large-scale and distributed data, however, is impractical.

This paper proposes a framework that combines feature selection and instance selection algorithms. The proposed framework is based on MapReduce [1], a programming model for data processing designed to handle large datasets [30, 31]. MapReduce has gained widespread adoption and become the de facto standard for large-scale data analysis. Hadoop [32] provides an implementation of MapReduce. Apache Hadoop is a framework that utilizes simple programming models for the distributed processing of large-scale datasets. To leverage Hadoop's parallel processing capabilities, our query must be expressed as a MapReduce job [30]. MapReduce enables the processing of large-scale data through distributed computing, eliminating concerns regarding concurrency, robustness, scalability, and other common challenges [33].

The proposed framework combines an instance selection algorithm called Reservoir Sampling [34] with enhancements and a feature selection algorithm called ReliefF [35] in a parallel and distributed manner.

Reservoir Sampling is a technique for selecting instances without replacement. It is particularly useful for datasets with an undefined number of instances or those that grow continuously (such as data streams). Reservoir Sampling operates in optimal time and requires constant memory to execute [34]. To select k items uniformly from a dataset containing n items (where $n \geq k$), the original reservoir sampling algorithm functions as Pseudo-code 1.

This algorithm, guarantees that each item in the dataset has an equal probability (k/n) of being included in the final selected set. The procedure maintains this uniform probability distribution throughout the sampling process.

1. the first k items from the dataset are selected as the initial set.
2. for each subsequent item i where $i > k$:
   - generate a uniformly distributed random integer r in the interval [1, i].
   - if $r \leq k$, replace the r-th item in the selected set with the i-th item.

Pseudo-code 1. Reservoir Sampling Algorithm.

Relief [36] is a filter method [37, 15] inspired by instance-based learning. It is one of the most successful preprocessing algorithms and a general feature estimator, widely used in various problems [38]. Relief is particularly effective in ranking features based on their quality and can be applied to both nominal and numerical features [35]. The core idea behind Relief is to estimate the relevance of features by measuring their separation between instances that are close to one another [38]. Despite its advantages, Relief has some limitations. It cannot handle incomplete data and is restricted to problems with only two classes [35]. To address these limitations, ReliefF was introduced. ReliefF can operate on datasets with incomplete and multiclass data [35].

The ReliefF algorithm functions as shown in Pseudo-code 2. First, the algorithm calculates the prior probability (pc) for each class in the dataset (line 5). It then randomly selects an instance from the dataset (line 6.a) and identifies its B nearest neighbors from the same class (hits) as well as B nearest neighbors from each of other classes (misses) (lines 6.b to 6.d).

The algorithm proceeds to compute feature weights using a difference function (δ) that measures the distance between the selected instance and its nearest hits, as well as the distance between it and its nearest misses (line 6.e). The term $pc / (1 - p(class(x_t))$ in the feature weight formula

is used for making the facility of using ReliefF in multi-class problem [39]. Finally, the algorithm returns a list of feature weights that can be used for feature selection, where features with weights below a predefined threshold may be removed [38].

Algorithm ReliefF (input:X, L, B)
1./*X=the set of training instances */
2./*L=the number of random instances to draw/*
3./*B=the number of nearest neighbours to compute*/
4. For each feature k{w_k=0.0}
5. For each class c{pc:=the fraction of X belonging to class c}
6. For l:=1 to L do
   a. Randomly select an instance (x_t,y_t)
   b. Let hits be the set of B instances (x_i,y_i) nearest to x_t such that y_i=y_t.
   c. For each class c≠y_t
      c-1. Let misses be the set of B instances (x_j,y_j) Nearest to x_t such that y_j=c.
   d. End for
   e. For each feature k

$$w_k = w_k - \frac{1}{LB}\sum_{(x_i,y_i)\in Hit}\delta(x_{t,k},x_{i,k}) + \sum_{c\neq y_t}\frac{pc}{(1-p(class(x_t)))LB}\sum_{(x_i,y_i)}\delta(x_{t,k},x_{i,k})$$

   f. End for
7. End for
8. Return the set of weights w_k
9. End ReliefF

Pseudo-code 2. ReliefF algorithm.

The rest of this paper is organized as follows: Section 2 discusses the structure of the proposed framework. Section 3 presents the implementation results and further discussions. Finally, the paper concludes in Section 4.

2. Methods

This section describes our proposed data reduction framework. A server, as used in this paper, can refer to a separate computer, a dedicated processor, or even a separate core within a processor capable of independent function. The framework comprises three steps:

Initially, the input dataset is horizontally partitioned into subsets. Each partition contains a complete set of features but a subset of instances. Each server receives one partition. If data resides on different servers in a network or is collected from sensors or other sources near a server, partitioning is unnecessary. However, all servers must store the same features in their datasets to enable independent execution of subsequent steps.

In the first step, each server runs parallel reservoir sampling to reduce the number of instances under the MapReduce model. Each mapper performing reservoir sampling creates a min-heap [40] for instances of each class. A min-heap is a data structure where each node's key (value) is less than or equal to the keys of its children, and the node with the lowest key is at the root. We assume each instance has a unique ID, used as the key to create the min-heap node.

The use of min-heaps is an innovative approach introduced in this study to enhance reservoir sampling implementation. This modification simplifies the distributed algorithm's execution and ensures balanced sampling across all data classes, effectively addressing imbalanced datasets (as elaborated below). The MapReduce-based reservoir sampling process is shown in Pseudo-code 3.

Mapper Phase:

1. In each mapper, for each class, select the first k records and insert them into their corresponding class-labeled min-heap in the appropriate position.
2. Repeat until end of the data partition (or for a predefined number of)
   a) For each subsequent record (at position i), generate a random instance ID (r) uniformly from [1, i].
   b) Insert the record into its class-specific min-heap based on r.

Reducer Phase:

1. Aggregate all min-heap key-value pairs to generate the final output (Each server produces a list of selected instances).

Pseudo-code 3. MapReduce-based reservoir sampling.

As previously mentioned, the proposed sampling algorithm distinguishes itself from conventional approaches through its distributed implementation and effectiveness in handling imbalanced datasets. In imbalanced datasets, such as KDDCup99, some classes have significantly more instances than others. Random or blind sampling in these datasets may eliminate instances from certain classes. However, by assigning a min-heap to each class in the proposed algorithm, at least some instances are selected from each class, ensuring representation in the reduced dataset.

In the second step, servers act as mappers, each independently running the ReliefF algorithm. As described previously, ReliefF calculates a weight

for each feature by selecting a random instance and finding nearest samples within the same class or from other classes. After calculating weights for all features, they are sent to a reducer function, which calculates the average weight for each feature. These average values become the final feature weights, similar to the final decision-making process in ensemble classifiers. In ensemble classifiers, each classifier independently generates an output, and the final value is obtained through majority voting or averaging the outputs. The reducer then retains a predefined ratio of features with the highest weights, removing the rest.

Users define the reduction ratio. Once the final selected features are determined, they can be used for classification in the third step. Classification can be performed non-distributed on a single server or locally on each server. As this paper focuses on data reduction, it does not detail the classification process. However, the proposed distributed framework significantly reduces inter-server communication and classification runtime.

Figure 1 illustrates the framework's application to a dataset with 18 instances and 8 features, using three servers. In the first step, the dataset is partitioned into three parts, with each server receiving one part containing six instances. Applying the reservoir algorithm with a 50% reduction ratio to the partitions across different servers, each server selects only three instances. These selected instances are then passed to the second step. In the second step, the ReliefF algorithm assigns weights to features based on the selected instances. The reducer calculates the average weight for each feature based on the votes received from the mappers. After applying a 50% reduction ratio, only the four features with the highest weight values remain. Finally, the classification algorithm is applied to the dataset, now containing only the four features and nine selected instances.

3. Implementation Results

3.1. Specifications and implementation parameters

This section presents the experimental results to evaluate the proposed framework's performance. First, we describe key aspects, including hardware and software support, dataset characteristics, algorithm and MapReduce framework parameters, and the performance measures used for evaluation. Next, the framework's performance is evaluated on two datasets, KDDCup99 and MNIST, by applying a classification algorithm to the reduced data. Finally, the obtained results are presented and discussed.

Two datasets were used to test and analyze the proposed framework. Table 1 presents the characteristics of these datasets. "# Instances" shows the number of instances, while "# Features" and "# Classes" represent the number of features and classes, respectively. Additionally, the original dataset was classified using the C4.5 algorithm [41]. The table provides the runtime required for classification and the classification accuracy for comparison with the experimental results.

Table 1. Summary description of datasets.

Datasets	# instances	# features	#classes	Accuracy (percent)	Runtime for classification (sec)
KDDcup99	4,898,431	41	23	99.9915	13767
Mnist (test part)	50,000	784	10	86.5824	470

The KDDCup dataset was used for detecting network intrusions and distinguishing between attacks and normal traffic. It contains a standard set of audit data, including a variety of simulated intrusions within a military network environment. As previously mentioned, this dataset is imbalanced, with some classes containing only a few instances. The MNIST (Mixed National Institute of Standards and Technology) database is a large dataset of handwritten digits, commonly used for training image processing systems. It is also widely used for training and testing machine learning algorithms. Notably, this paper uses only the test portion of the MNIST dataset. These datasets were chosen due to their large number of instances and features, enabling a thorough evaluation of the proposed framework.

To compare the proposed framework with the original method and measure the efficiency of the MapReduce model, the experiments were conducted in two ways. One approach involves reducing the data using the MapReduce framework, which runs on a 3-node cluster. This cluster consists of a master node and two slave nodes. The features of each node are shown in **ERROR! REFERENCE SOURCE NOT FOUND.** .

Table 2. The software specifications for Map-reduce implementation.

	CPU	RAM	Hard disk
Master	2Core from a cpu with below specification (intel core i7 6700hq)	4Gb	100Gb
slaves	2Core from a cpu with below specification (intel core i7 6700hq)	4Gb	100Gb

The software specifications are as follows:

- JAVA version: 1.8.0
- Hadoop version: 1.2.1

- Operating system: CENTOs 6.4
- MapReduce version:1.2.1
- Map slot: 11
- Reduce slot: 1

In Hadoop 1.0, the user must specify the number of mappers and reducers that the cluster can run simultaneously. Typically, 2 slots (jobs) can be defined for each CPU core. The cluster used for the experiments consists of 3 machines, each with two cores. Therefore, to achieve better performance, up to 12 jobs can run simultaneously. Since the proposed framework has one reducer in each phase, we define 11 map slots and one reducer slot.

The other method for reducing data involves running the original reservoir sampling and the original ReliefF algorithm on a conventional, standard system. It should be noted that in this approach, none of the proposed modifications have been implemented in these algorithms. The original algorithms were only executed to compare their results with those obtained using the MapReduce framework. It is worth mentioning that, due to the high volume of data, implementing the algorithms on a standard system requires a very fast computer. For example, running the sampling algorithm on the KDDCup dataset required about 20 GB of memory. Therefore, the standard system used for running the algorithms is much more powerful than the system on which Hadoop runs. If the original algorithms were implemented in a non-parallel manner on a system with the same specifications as the one used for Hadoop, a memory error would likely occur during algorithm execution.

The software and hardware specifications of the computer used to run the algorithms in the second method (standard system) are as follows:

- CPU: Intel® Xeon® CPU E5-2620 v3 @ 2.40GHz (2 processors)
- RAM: 40 GB
- Operating system: Windows 7
- The used software for ReliefF and Reservoir sampling: Weka 3.7.

In each approach, three models are applied to reduced data: 1) The reservoir sampling algorithm alone, referred to as SO (Sampling Only) in this paper. 2) Feature reduction using only the ReliefF algorithm, referred to as FO (Feature Reduction Only) in this paper. 3) A combination of reservoir sampling followed by the ReliefF algorithm applied to the reduced data, referred to as S+F (Sampling + Feature Reduction) in this paper.

To evaluate the efficiency of the proposed methods compared to other distributed approaches, datasets are reduced using the distributed methods introduced in [18] and [16], too and the results are show in the related tables along with our proposed method. Since no existing study was found that simultaneously performs distributed feature and *instance* reduction (F+S) on the mentioned datasets, the results in Tables 7 and 8 could not be directly compared with those of another work. In [18], a voting-based method for instance selection is proposed, and its results are demonstrated in the SO (Sampling Only) implementation. Additionally, paper [16] proposes a distributed feature selection approach based on horizontal or vertical partitioning of the dataset, and its results are presented in the FO (Feature Reduction Only) reduction. To compare the outcomes of data reduction using MapReduce and a standard system, the Weka software is used to classify the obtained results, with the C4.5 algorithm employed for classification. It generates a decision tree and prunes it after creation. C4.5 can handle attributes with missing values and supports both continuous and discrete attributes [42]. The algorithm uses the concept of information entropy to construct the tree. At each node, C4.5 selects the feature that most effectively splits the instances into subsets enriched in one class or another [43]. In all experiments, 66% of the dataset is used for training the classifier, while the remaining portion is used for testing. The classification algorithm is also executed on the same machine that performs the reduction algorithm in a non-parallel manner (standard system).

The measures used to evaluate the quality of the proposed framework are: Accuracy, Reduction Rate [19] (defined as (1)), Runtime, and Root-Mean-Square Error (RMSE)[44].

$$Reduction\_Rate = 1 - \frac{size(decreased\_dataset)}{size(original\_dataset)} \quad (1)$$

This measure is shown in percent in this paper.

3.2. Execution results

In this subsection, we present the results obtained from the experiments. Following each table, the parameters of the corresponding execution are explained. The results of running the sampling (SO) algorithm on the KDDcup and Mnist datasets are shown in Tables 3 and 4, respectively. These tables display the outcomes for each sampling state when executed on either the standard system or the MapReduce framework.

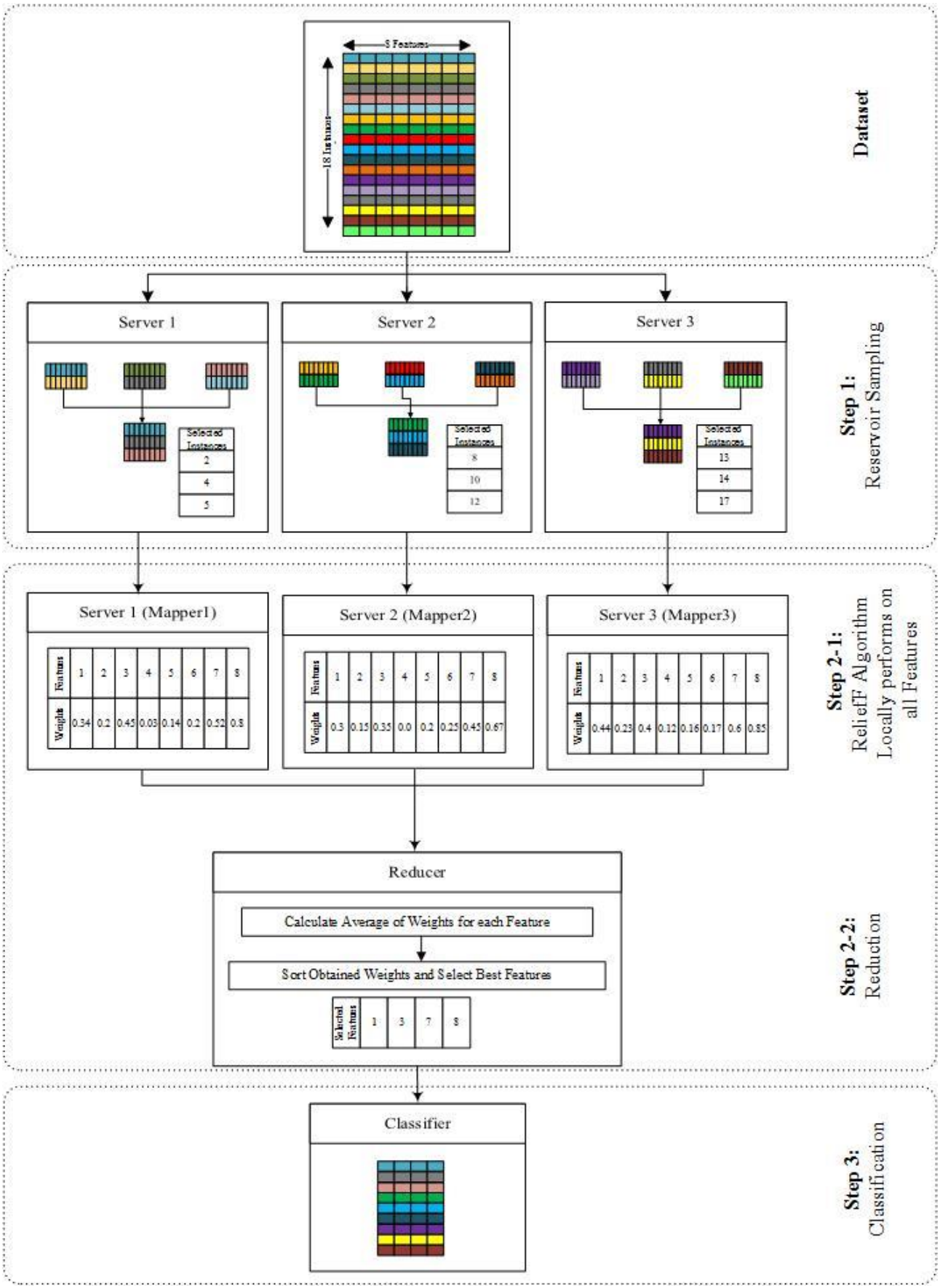

Figure 1. Example of how to apply the proposed framework on a dataset.

In the MapReduce implementation, the number of map functions used for sampling the KDDcup dataset is 11, while for the Mnist dataset, it is 9. The number of map functions indicates the number of parallel tasks processing the data. Additionally, since the proposed framework employs a single reducer to combine the results, all data reductions using MapReduce utilize only one reduce function. Furthermore, the results of the method presented in [18], referred to as Democratic Selection, are provided in Table 4. In [18], multiple voters perform instance selection on the same dataset in a distributed manner, and the instances receiving the most votes are selected. Due to the large size of the KDDcup dataset, the Democratic Selection method was not implemented on this dataset.

As the sampling results demonstrate, the runtime of the proposed algorithm is significantly lower than that of the standard algorithm and the Democratic Selection method. This is because, in Democratic Selection, each voter processes all instances, and additional time is spent calculating the instances with the highest number of votes, which increases the runtime compared to the standard method.

Additionally, the prediction accuracy and RMSE of the data classification do not differ significantly across the three methods and are nearly identical. Only in some cases do the RMSE and accuracy for the proposed method show slight improvements. As the results indicate, for the KDDcup dataset, despite the large volume of data removed, the accuracy of data classification remains relatively unchanged. It is worth noting that the precision of Democratic Selection depends on the sampling algorithm used by the voters. Since the reservoir sampling algorithm is used in this simulation, the accuracy achieved does not differ significantly from the other two methods. Furthermore, this method does not provide solutions for handling imbalanced datasets.

Table 3. The results of sampling (SO) for KDDcup99 dataset.

Reduction rate	# Instances	# Attributes	Accuracy (percent)		RMSE		Runtime (sec)	
			Standard System	Map Reduce	Standard System	Map Reduce	Standard System	Map Reduce
90 %	489835	42	99.9694	99.9802	0.0057	0.0041	465	30
70 %	1469517	42	99.9826	99.985	0.0041	0.0036	519	49
50%	2449210	42	99.9912	99.988	0.0027	0.0032	575	65

Table 4. The results of sampling (SO) for Mnist (test part) dataset.

Reduction rate	# Instances	# Attributes	Accuracy (percent)			RMSE			Runtime (sec)		
			Standard system	Democratic selection	Map Reduce	Standard system	Democratic selection	Map Reduce	Standard system	Democratic selection	Map Reduce
90 %	4997	784	74.279	74.6471	78.340	0.216	0.213	0.198	26	38	19
70 %	14996	784	83.012	82.0588	82.290	0.176	0.182	0.179	30	74	21
50%	24998	784	84.693	84.4706	84.013	0.168	0.17	0.172	42	100	22

The results of feature reduction using the ReliefF algorithm (FO) for the KDDcup and MNIST datasets are presented in Tables 5 and 6, respectively. In this execution model, the number of map functions used to run ReliefF is 100 for the KDDcup dataset and 10 for the MNIST dataset.

The parameters employed for ReliefF in these implementations are L and B, where L represents the number of selected instances to execute the algorithm, and B denotes the number of neighbors used to identify the closest instances. For the experiments conducted in this study, the values of L are set to 1,000,000 and 2,000,000 for the KDDcup dataset, and 25,000 and 50,000 for the MNIST dataset.

The value of B used for the experiments in this paper is 10. Additionally, as shown in the tables, the time required to select and rank features in the ReliefF algorithm is significant, rather than the reduction rate. This is because, after ranking all the

features, only a subset of the top-ranked features is selected, which does not impact the runtime. The time differences observed in the table for the 66% and 50% reduction rates are attributed to external factors influencing the system's speed.

Table 6 also presents the results of feature selection using the proposed method described in [16]. The reason this method was not implemented for the KDDcup dataset is that the complexity measures introduced in the article are applicable only to numerical data (not nominal data). We employed the vertical partitioning technique from [16], with the parameters set to 5 rounds and an alpha value of 0.25. Additionally, we utilized the ReliefF algorithm to select features in each round. In the method proposed in this paper, the feature reduction rate is not directly specified. Instead, the number of features to be removed is determined by a formula. The number of selected features can vary significantly depending on parameters such as the alpha value and the type of features. As a result, implementing this method with the mentioned parameters produces a dataset with 192 features. The classification results for this reduced dataset are presented in the table. As shown in Table 6, although this method did not improve accuracy or RMSE, its runtime is even longer than that of the standard method.

Because there are a few rounds in this way. In each round, the dataset is divided into several parts, and feature selection is performed in parallel. However, the execution of the rounds themselves is non-parallel, meaning that as the number of rounds increases, the algorithm's runtime also increases. Additionally, after completing this step, a significant amount of time is required to calculate the number of features to be eliminated.

Table 5. The results of ReliefF Algorithm(FO) for KDDcup99 dataset.

Reduction rate	ReliefF execution parameters L	# Instances	# Attribute	Accuracy (percent)		RMSE		Runtime (sec)	
				Standard system	Map Reduce	Standard system	Map Reduce	Standard system	Map Reduce
66%	1000000	4,898,431	14	99.9256	99.9502	0.0079	0.0062	58484	23935
66%	2000000	4,898,431	14	99.9237	99.9500	0.0081	0.0061	128156	47733
50%	1000000	4,898,431	21	99.9629	99.9653	0.0053	0.0050	58492	23927
50%	2000000	4,898,431	21	99.9628	99.9652	0.0052	0.0050	128167	47720

Table 6. The results of ReliefF Algorithm(FO) for Mnist dataset.

Reduction rate	ReliefF execution parameters L	# Instances	# Attribute	Accuracy (percent)			RMSE			Runtime (sec)		
				Standard system	Distributed FS	Map Reduce	Standard system	Distributed FS	Map Reduce	Standard system	Distributed FS	Map Reduce
66%	25000	50000	262	86.747	86.105	86.770	0.156	0.16	0.156	12355	39113	5019
66%	50000	50000	262	86.588		86.652	0.157		0.157	27985		10366
50%	25000	50000	392	87.047		87.011	0.155		0.155	12364		5033
50%	50000	50000	392	87.123		87.305	0.154		0.154	27997		10354

The results of running both instance and feature reduction (F+S) are presented in Tables 7 and 8. Additionally, in the tables, the "# mapfunctions" column indicates the number of map functions used to execute the (F+S) algorithm on the datasets. As can be seen in the tables, although the data volume has been significantly reduced, the classification accuracy has remained relatively unchanged. As evident from the accuracy values in Tables 4 and 8, the MNIST dataset's relatively small sample size compared to KDDCup and the fact that most features in this dataset contain irrelevant data, means that applying either (SO) or (F+S) results in significant loss of valuable data. This is particularly noticeable when the data reduction rate is 90%. In contrast, (FS) reduction method does not decrease the accuracy parameter because only irrelevant features are removed.

This outcome occurs due to the elimination of a significant number of valuable instances, and thus, even the type of reduction algorithm does not have a substantial impact on this issue. We note that a substantially larger initial dataset volume would likely mitigate this accuracy reduction when removing instances. More generally, both the quantity and quality of collected data in a dataset can significantly influence the selection of an appropriate data reduction methodology.

Table 7. The results of running both sampling and ReliefF Algorithm(F+S) for KDDcup99 dataset.

Reduction rate of sampling	Reduction rate Of ReliefF	ReliefF execution parameters L	# Instances	# Attribute	# Map functions	Accuracy (percent)		RMSE		Run time (sec)	
						Standard system	Map Reduce	Standard system	Map Reduce	Standard system	Map Reduce
90%	66%	1000000	489,835	14	20	99.8943	99.8727	0.0106	0.0113	6189	2408
	50%			21		99.9412	99.9293	0.0077	0.0088	6172	2391
90%	66%	2000000	489,835	14		99.8895	99.8726	0.0109	0.0114	13079	4734
	50%			21		99.9364	99.9297	0.0081	0.0087	13062	4717
70%	66%	1000000	1,469,517	14	60	99.9211	99.8943	0.0087	0.0088	18067	7234
	50%			21		99.962	99.9556	0.0058	0.0066	18054	7220
70%	66%	2000000	1,469,517	14		99.9211	99.9051	0.0087	0.0083	38438	14060
	50%			21		99.9532	99.9556	0.0065	0.0063	38423	14042
50%	66%	1000000	2,449,210	14	100	99.9233	99.9015	0.008	0.0081	30029	12216
	50%			21		99.9605	99.9562	0.0058	0.0057	30012	12200
50%	66%	2000000	2,449,210	14		99.9245	99.9064	0.008	0.0078	62664	23443
	50%			21		99.960	99.956	0.0056	0.0058	62659	23428

Table 8. The results of running both sampling and ReliefF Algorithm(F+S) for Mnist dataset.

Reduction rate of sampling	Reduction rate Of ReliefF	ReliefF execution parameters L	# Instances	# Attribute	# Map functions	Accuracy (percent)		RMSE		Run time (sec)	
						Standard system	Map Reduce	Standard system	Map Reduce	Standard system	Map Reduce
90%	66%	25000	4997	262	1	75.044	76.633	0.214	0.207	3683	1461
	50%			392		74.926	74.867	0.213	0.214	3670	1448
90%	66%	50000	4997	262		75.632	75.809	0.208	0.215	7721	2866
	50%			392		75.279	75.279	0.212	0.212	7709	2854
70%	66%	25000	14996	262	3	79.678	81.447	0.192	0.186	4264	1736
	50%			392		79.952	82.526	0.191	0.179	4251	1723
70%	66%	50000	14996	262		81.267	81.780	0.186	0.183	9746	3602
	50%			392		81.306	82.427	18.693	0.181	9734	3589
50%	66%	10000	24998	262	5	82.24	84.527	0.179	0.169	6898	2851
	50%			392		82.689	84.574	0.176	0.168	6882	2837
50%	66%	20000	24998	262		82.191	84.339	0.181	0.170	15254	5672
	50%			392		82.555	84.598	0.177	0.168	15240	5672

3.3. Discussions

According to the results, we evaluated the reduced data in two aspects: classification accuracy and the time required for data reduction. Results from the KDDcup dataset indicate that whether the algorithms are implemented in parallel or non-parallel has little effect on accuracy. Additionally, the data reduction model—whether it is instance reduction (SO), feature reduction (FO), or both (F+S)—does not significantly impact the results. Even the reduction rate has minimal influence on the accuracy of the predicted outcomes. In the worst-case scenario, the number of misclassified instances may reach up to 7,000 out of 5 million instances (0.14%). This is likely due to the large volume of data and the presence of irrelevant information.

A good data reduction algorithm should achieve optimal results using the minimum necessary data. It must eliminate data whose removal does not significantly affect the classifier's precision. However, if a dataset is small or contains essential and useful information, data reduction may lead to a decrease in classification accuracy.

The results for the MNIST dataset are slightly different. In this dataset, the number of instances is much smaller compared to the KDDcup dataset, and a large number of features have zero values. These features are essentially redundant and do not contribute to classification. As shown in Table 4, the fewer instances removed, the more accurate the classification results. The reasons for this are the smaller number of instances per class, the relative balance among classes, and the fewer repeated or overlapping instances.

As shown in Table 6, the accuracy of data reduced by feature selection is higher than that of data reduced by sampling. This is because the dataset

contains many unnecessary features; removing them does not significantly affect accuracy.
Table 6 also indicates that the accuracy levels are similar across all three methods.
Table 8 demonstrates that both instance and feature reduction at a 50% reduction ratio can effectively reduce data while maintaining accurate classification. The results reveal that the MapReduce method yields slightly higher accuracy. This improvement occurs because multiple ReliefF algorithms (one per mapper) are executed, and the final result is derived from their average. Additionally, prior sampling removed certain instances, further enhancing the accuracy of ReliefF.
To evaluate the efficiency of the proposed framework from a runtime perspective, we first analyze the computational complexity of the proposed algorithms and then compare the results.

The original Reservoir Sampling algorithm has a time complexity of O(n), as it must traverse all n elements and perform constant-time operations for each element. However, when we used the min-heap structure as proposed in this paper—where a heap tree is used for each class and insertion/deletion operations take O(log k) time (k is the size of the selected list))—two cases may occur.
The worst case occurs when the dataset is highly imbalanced (e.g., nearly all data belongs to a special class). In this case, since every potential element requires an insertion or deletion operation in a heap tree, the overall time complexity becomes $O(n\log k)$. Conversely, if the data is uniformly distributed among classes, the average time complexity of Reservoir Sampling using the min-heap approach approximates $O(n\log(k/c))$, where c represents the number of classes.
If MapReduce is used to execute the modified Reservoir Sampling algorithm with a min-heap in a distributed manner—assuming n instances and m mappers—each mapper processes an average of $\frac{n}{m}$ instances when the data is uniformly distributed. In this situation, in the worst-case (highly imbalanced data distribution), each mapper takes $O((\frac{n}{m})\log k)$ time. And in the average-case (uniform data distribution), each mapper takes $O((\frac{n}{m})\log(\frac{k}{c}))$ time.
Additionally, the overhead from data transfer between mappers and the reducer is negligible due to the small size of the transmitted data. Also, the reducer performs no significant computations; it merely merges the candidate lists from the mappers into a final list. Thus, its runtime contribution can be disregarded.
So, by using MapReduce and increasing the number of mappers (m), the computational load per mapper decreases significantly, leading to a substantial reduction in overall execution time.
As shown in Tables 3 and 4, sampling in the standard system takes longer than in the MapReduce model. Although processing a single instance is not time-consuming, the sampling algorithm must be applied to every instance. Thus, execution time increases proportionally with the number of instances. But in the MapReduce model, as the number of instances increases, the number of mappers will increase (as far as system power allows). In this case, each mapper processes a subset of instances, distributing the computational load and reducing overall runtime.
The democratic selection method is even slower than the standard method. While it performs sampling in parallel across multiple stages, additional time is required to partition the data into disjoint sets with different permutations. Moreover, after sampling, further time is needed to tally votes for each sample.
The time complexity of the original ReliefF algorithm primarily depends on four key parameters: the number of training instances (n), features (m), nearest neighbors (k), and classes (c). Additional factors influencing execution time include the specific difference measurement function employed (such as Manhattan or Euclidean distance) and the number of iterations performed in the main loop of the algorithm (line 6 in Pseudo-code 2).
By using a simple difference metric and executing the main loop over all n training instances, the algorithm demonstrates a time complexity of O(n·m·k·c). However, in practical implementations where both k (typically 5-10 neighbors) and c represent relatively small constants, the effective complexity simplifies to O(n·m).
When executing the ReliefF algorithm in the proposed MapReduce framework, the time complexity becomes as follows :

- Each Mapper operates on a subset of the data (e.g., $\frac{n}{p}$ of instances, where p is the number of Mappers). Thus, the computational complexity per Mapper is: $O(\frac{n.m}{p})$.
- The Reducer aggregates and averages the weights obtained from all Mappers. For m

features, the Reducer computes an average of the weights received from p Mappers for each feature. Calculating this average for p values, takes O(p) time, so performing this operation for all features requires O(m·p) time.

- Data transfer to the Reducer depends on the number of features, as each Mapper sends an m-sized list. Since this operation is typically performed in parallel by the Mappers, its overhead is negligible unless the number of features is extremely large.

Considering these factors, the total computational complexity of ReliefF with MapReduce can be approximated as: $O(\frac{n.m}{p} + m.p)$.

Since typically $(m.p) < \frac{n.m}{p}$ (because p is usually small compared to n and m), the execution speed improves by approximately a factor of p.

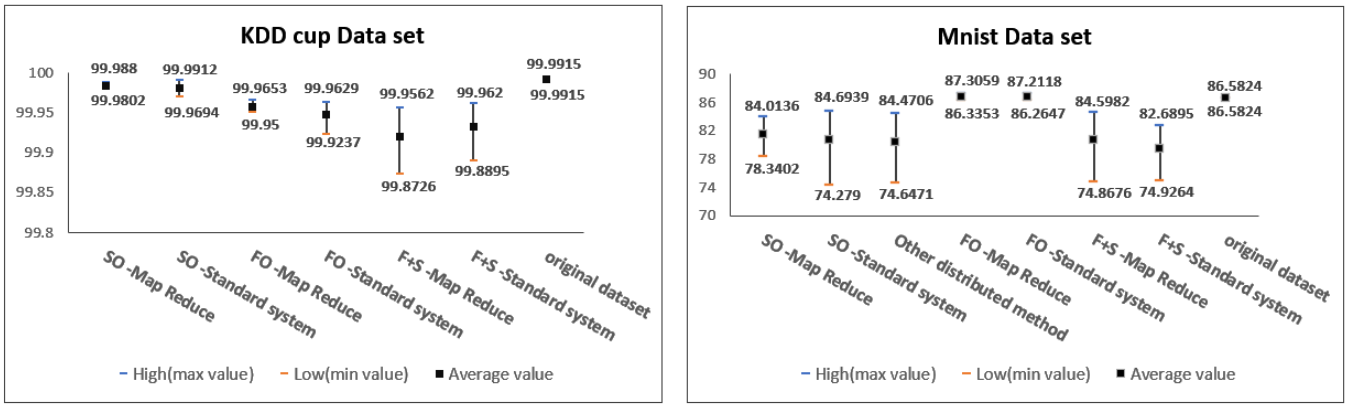

Figure 1. The average value of accuracy.

Figure 2 presents a comparative analysis of average classification accuracy across distinct data reduction methods, visualized using a stock-style chart. Each method's performance is represented by its mean accuracy (central marker), bounded by vertical error bars indicating the minimum-to-maximum range observed across execution trials. As shown in the chart, the MapReduce-based data reduction achieves equal or better results in most cases—except when applying the combined feature selection and sampling (F+S) method to the KDDcup dataset. The chart also highlights the importance of selecting an appropriate reduction model for each dataset. For instance:

- The sampling model (SO) performs better on the KDDcup dataset, where the number of instances significantly exceeds the number of features.
- In contrast, the feature selection model (FO) is more effective for the Mnist dataset, which contains numerous unnecessary and irrelevant features.
- The combined (F+S) model is beneficial for datasets with both a high number of instances and redundant features.

Figure 3 illustrates the mean execution time across various evaluated data reduction models. As the execution time of SO was very lower than other reduction techniques and thus not visible on the plot, the data are presented using a logarithmic scale was employed on the Y-axis to enable clear comparison across all measurement points. The results demonstrate that the MapReduce implementation requires significantly less time than other approaches. Additionally, the sampling algorithm (SO) has a much shorter runtime compared to FO or F+S, as it merely scans the dataset once without performing complex calculations per record.

4. Conclusion

Due to the importance of reducing data volume and enabling faster processing, this paper proposes a framework for reducing both instances and features while preserving useful information. The proposed framework is based on MapReduce, a parallel and distributed processing model.

In the proposed method, the dataset is first partitioned into multiple segments, each assigned to a separate server. Reservoir sampling is applied on each server under MapReduce to reduce the number of samples. The sampled data is then processed by another MapReduce program, where each mapper independently executes the Relief algorithm across all features.

Next, the mapper outputs a list of feature weights, which are sent to the reducer. The reducer computes the average weight of each feature and

selects the highest-weighted features based on the specified reduction rate. Finally, the selected instances (containing the chosen features) are sent to the server for classification.

Implementation results demonstrate that the proposed framework achieves comparable—and in some cases, superior—results to standard system implementations while significantly reducing processing time. This framework can be utilized for data reduction in the preprocessing phase of machine learning algorithms.

Further research is needed to develop methods for effectively combining feature reduction and instance reduction algorithms. Additionally, more studies should focus on identifying algorithms that can be efficiently parallelized using MapReduce.

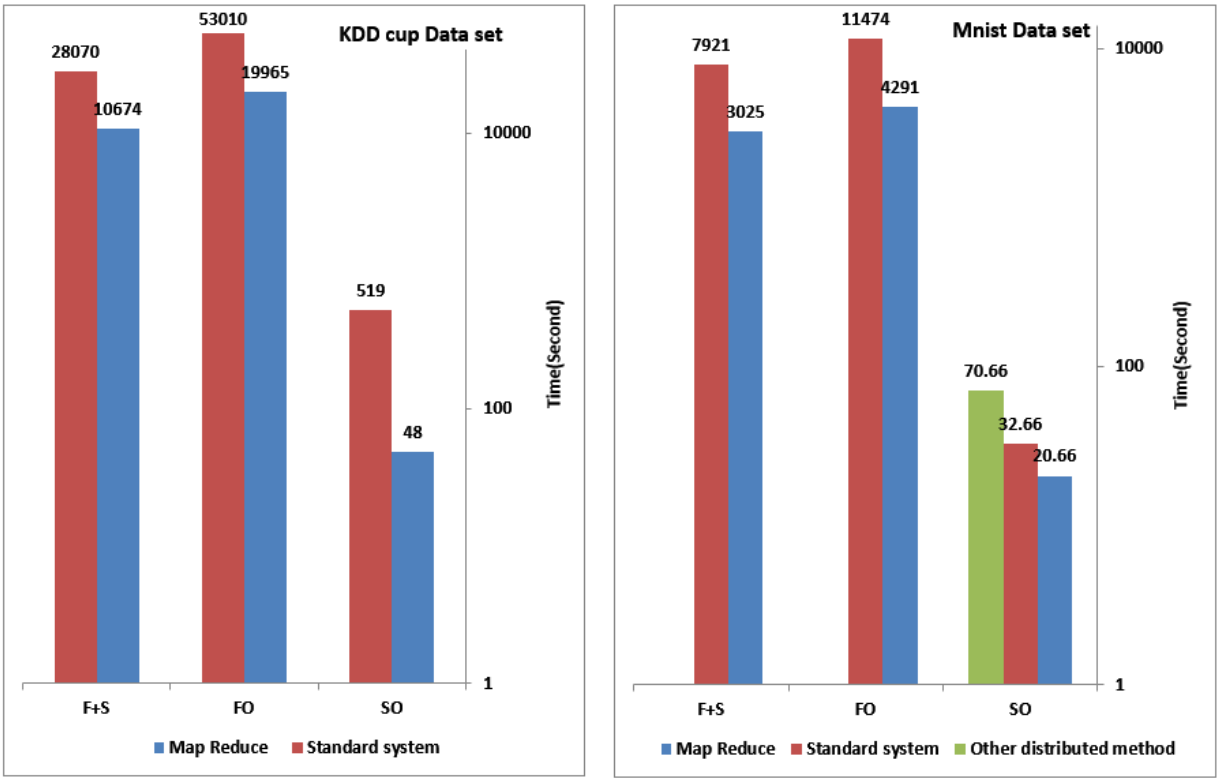

Figure 2. The average of time.

References

Journal Article in Print: Full titles

[1] S. d. Río, V. Lopez, J. M. Benítez and F. Herrera, "On the use of MapReduce for imbalanced big data using Random Forest," *Information Sciences*. vol. 285, pp. 112-137, 2014.

[2] J. Derrac, S. Garcia and F. Herrera, "IFS-CoCo: Instance and feature selection based on cooperative coevolution with nearest neighbor rule," *Pattern Recognition*, vol. 43, no. 6, pp. 2082-2105, 2010.

[3] P. Bradley, U. Fayyad and C. Reina, " Clustering very large databases using EM mixture models," In *Proceedings 15th International Conference on Pattern Recognition, Barcelona*, ICPR-2000, 2000, pp. 76-80.

[4] H. Liu, H. Motoda and L. Yu, "A selective sampling approach to active feature selection," *Artificial Intelligence*, vol. 159, pp. 49-74, 2004.

[5] W. G. Cochran, Sampling Techniques,1st ed., New York: Wiley, 1977, [E-book] Available: www.cambridge.org.

[6] H. Liu and H. Motoda, Instance Selection and Construction for Data Mining, 1st ed., Boston: Kluwer Academic, 2001, [E-book] Available: https://books.google.com/.

[7] J. R. Cano, F. Herrera and M. Lozano, "On the combination of evolutionary algorithms and stratified strategies for training set selection in data mining," *Applied Soft Computing*, vol. 6, p. 323–332, 2006.

[8] M. Rashid, J. Kamruzzaman, T. Imam, S. Wibowo and S. Gordon, "A tree-based stacking ensemble technique with feature selection for network intrusion detection," *Applied Intelligence*, vol. 52, no. 9, pp. 9768-9781, 2022.

[9] A. V. Turukmane and R. Devendiran, "M-MultiSVM: An efficient feature selection assisted network intrusion detection system using machine

learning," *Computers & Security*, vol. 137, p. 103587, 2024.

[10]F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel , P.Prettenhofer, R. Weiss, V. Dubourg and j. Vanderplas, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, pp. 2825-2830, 2011.

[11] F. Li, Z. Zhang and C. Jin, "Feature selection with partition differentiation entropy for large-scale data sets," *Information Sciences* , vol. 329, pp. 690-700, 2016.

[12] J. Qian, P. Lv, X. Yue, C. Liu and Z. Jing, "Hierarchical attribute reduction algorithms for big data using MapReduce," *Knowledge-Based Systems*, vol. 73, p. 18–31, 2015.

[13] X. Yu and X. Cai, "A multi-objective evolutionary algorithm with interval based initialization and self-adaptive crossover operator for large-scale feature selection in classification," *Applied Soft Computing*, vol.127, p. 109420, 2022.

[14] Y. Lv, P. Liu, J. Wang, Y. Zhang, A. Slowik and J. Lv, "GA-based feature selection method for oversized data analysis in digital economy," *Expert Systems*, vol.41, no. 1, p. 13477, 2024.

[15] H. Liu and L. Yu, "Feature selection for high-dimensional data: A fast correlation-based filter solution," in *Proceedings of the 20th international conference on machine learning*, *Washington*, ICML-03, 2003, pp. 856-863.

[16] L. MoránF., V. B. Canedo and A. A. Betanzos, "Centralized vs. distributed feature selection methods based on data complexity measures," *Knowledge-Based Systems*, vol. 117, pp. 27-45, 2017.

[17] C. Kai, W. W. qiang and L. Yun, "Differentially private feature selection under MapReduce framework," *The Journal of China Universities of Posts and Telecommunications*, vol. 20, no.5, pp. 85-103, 2013.

[18] C. García-Osorio, A. d. Haro-García and N. G. Pedrajas, "Democratic instance selection: A linear complexity instance selection algorithm based on classifier ensemble concepts," *Artificial Intelligence*, vol. 174, no. 5-6, pp. 410-441, 2010.

[19]D. S. F. Isaac Triguero, "MRPR: A MapReduce solution for prototype reduction in big data classification," *Neurocomputing*, vol.150, p. 331–345, 2015.

[20] G. E. Melo-Acosta, F. Duitama-Muñoz and J. D. Arias-Londoño, "An Instance Selection Algorithm for Big Data in High imbalanced datasets based on LSH," preprint arXiv:2210. 04310, 2022.

[21] C. Gong, Z.-g. Su, P.-h. Wang, Q. Wang and Y. You, "Evidential instance selection for K-nearest neighbor classification of big data," *International Journal of Approximate Reasoning*, vol. 138, pp. 123-144, 2021.

[22] L. Qin, X. Wang and Z. Jiang, "A distributed evolutionary based instance selection algorithm for big data using Apache Spark," *Applied Soft Computing*, vol.159, p. 111638, 2024.

[23] D. Fragoudis, D. Meretakis and S. Likothanassis, "Integrating feature and instance selection for text classification," in *8th ACM SIGKDD international conference on Knowledge discovery and data mining*, *New York*, KDD '02, 2002, pp. 501-506.

[24] K. Yu, X. Xu, M. Ester and H.-P. Kriegel, "Feature weighting and instance selection for collaborative filtering: An information-theoretic approach," *Knowledge and Information Systems*, vol. 5, no. 2, pp. 201-224, 2003.

[25] H. Ahn and K.-j. Kim, "Bankruptcy prediction modeling with hybrid case-based reasoning and genetic algorithms approach," *Applied Soft Computing*, vol.9, no.2, pp. 599-607, 2009.

[26] C.-F. Tsai, W. Eberle and C.-Y. Chu, "Genetic algorithms in feature and instance selection," *Knowledge-Based Systems*, vol. 39, p. 240–247, 2013.

[27] T. Chen, X. Zhang, S. Jin and O. Kim, "Efficient classification using parallel and scalable compressed model and its application on intrusion detection," *Expert Systems with Applications*, vol. 41, no.13, pp. 5972-5983, 2014.

[28] Z.-H. You, Y.-H. Hu, C.-F. Tsai and Y.-M. Kuo, "Integrating feature and instance selection techniques in opinion mining," *Research Anthology on Implementing Sentiment Analysis Across Multiple Disciplines- IGI Global*, vol.1, pp. 800-815, 2022.

[29] C. F. Tsai, K.-L. Sue, Y.-H. Hu and A. Chiu. , "Combining feature selection, instance selection, and ensemble classification techniques for improved financial distress prediction," *Journal of Business Research*, vol. 130, pp. 200-209, 2021.

[30] T. White, Hadoop, The Definitive Guide, 3rd ed., USA: O'Reilly Media, 2012, [E-book] Available: books.google.com.

[31] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no.1, pp. 107-113, 2008.

[32] Apache Software Foundation, "Apache Hadoop Project," 2013. [Online]. Available: <http://hadoop.apache.org/>. [Accessed December 2013].

[33] D. Miner and A. Shook, MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems,1st ed., USA: O'Reilly Media, 2012, [E-book] Available:books.google.com.

[34] J. S. vitter, "Random sampling with a reservoir," *ACM Transactions on Mathematical Software (TOMS)* vol.11, no.1, pp. 37-57, 1985.

[35] E. Š. M. R.-Š. Igor Kononenko, "Overcoming the myopia of inductive learning algorithms with RELIEFF," *Applied Intelligence*, vol. 7, no.1, pp. 39-55, 1997.

[36] L. A. R. Kenji Kira, "The feature selection problem: Traditional methods and a new algorithm," *AAAI*, vol. 2, pp. 129-134, 1992.

[37] S. Das, "Filters, wrappers and a boosting-based hybrid for feature selection," in *Proceedings of the 18th international conference on machine learning, San Francisco*, ICML-01, 2001, pp. 74-81.

[38] I. K. Marko Robnik-Šikonja, "Theoretical and empirical analysis of ReliefF and RReliefF," *Machine learning*, vol. 53, no.1-2, pp. 23-69, 2003.

[39] G. Frederickson, "An Optimal Algorithm for Selection in a Min-Heap," *Information and Computation*, vol. 104, no. 2, pp. 197-214, 1993.

[40] J. R. Quinlan, C4.5: Programs for Machine Learning, 1st ed., USA: Morgan Kaufmann Publishers, 1993, [E-book] Available: books.google.com.

[41] J. R. Quinlan, "Improved use of continuous attributes in c4.5," *Journal of Artificial Intelligence Research*, vol.4, pp. 77-90, 1996.

[42] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22, no. 4, p. 679–688, 2006.

[43] S. Mii Rostami, and M. Ahmadzadeh, "Extracting predictor variables to construct breast cancer survivability model with class imbalance problem, "*Journal of AI and Data Mining*, vol.6, no. 2, 263-276, 2018.

[44] R. J. Hyndman, and A. B. Koehler," Another look at measures of forecast accuracy," *International journal of forecasting*, vol.22, no.4, 679-688, 2006.

(یک چارچوب جدید برای کاهش حجم داده‌ها با مقیاس بزرگ با استفاده از MapReduce)

زینب عباسی*

دانشکده مهندسی، مرکز آموزش عالی محلات، محلات، ایران.

ارسال ۲۰۲۵/۰۵/۰۴ بازنگری ۲۰۲۵/۰۶/۰۸؛ پذیرش ۲۰۲۵/۰۷/۱۰

چکیده:

ذخیره و پردازش مجموعه داده‌های حجیم یکی از مهم‌ترین مشکلات در پردازش در مقیاس بزرگ است. بنابراین، قبل از پردازش بیشتر، نیاز به کاهش اندازه آنها وجود دارد. این مقاله چارچوبی برای کاهش داده‌ها در مجموعه داده‌های حجیم ارائه می‌دهد. چارچوب پیشنهادی مبتنی بر الگوریتم MapReduce است. این چارچوب سه مرحله دارد. ابتدا با نمونه‌برداری از مخزن، نمونه‌هایی از یک مجموعه داده انتخاب می‌شوند. در مرحله دوم، ویژگی‌های این نمونه‌های انتخاب شده با استفاده از الگوریتم ReliefF وزن‌دهی می‌شوند. سپس، تمام وزن‌ها برای هر ویژگی میانگین‌گیری می‌شوند و ویژگی‌هایی با بالاترین مقادیر وزن انتخاب می‌شوند. در نهایت، ویژگی‌های انتخاب شده در طبقه‌بندی استفاده شده‌اند. نتایج پیاده‌سازی چارچوب پیشنهادی، کاهش خوبی در زمان را نشان می‌دهد. همچنین با حذف ویژگی‌های نامربوط در الگوریتم‌های طبقه‌بندی، دقت را افزایش می‌دهد یا آن را حفظ می‌کند حتی زمانی که مقدار زیادی از داده‌ها حذف می‌شوند.

کلمات کلیدی: داده‌های با مقیاس بزرگ، MapReduce، انتخاب ویژگی، انتخاب نمونه.