Advances in Computational Intelligence (2022) 2:20
https://doi.org/10.1007/s43674-022-00033-z

ORIGINAL ARTICLE

Instance selection for big data based on locally sensitive hashing and double-voting mechanism

Junhai Zhai[1] · **Yajie Huang**[1]

Received: 14 December 2021 / Revised: 1 February 2022 / Accepted: 28 February 2022 / Published online: 19 March 2022

Abstract
The increasing data volumes impose unprecedented challenges to traditional data mining in data preprocessing, learning, and analyzing, it has attracted much attention in designing efficient compressing, indexing and searching methods recently. Inspired by locally sensitive hashing (LSH), divide-and-conquer strategy, and double-voting mechanism, we proposed an iterative instance selection algorithm, which needs to run p rounds iteratively to reduce or eliminate the unwanted bias of the optimal solution by double-voting. In each iteration, the proposed algorithm partitions the big dataset into several subsets and distributes them to different computing nodes. In each node, the instances in local data subset are transformed into Hamming space by l hash function in parallel, and each instance is assigned to one of the l hash tables by the corresponding hash code, the instances with the same hash code are put into the same bucket. And then, a proportion of instances are randomly selected from each hash bucket in each hash table, and a subset is obtained. Thus, totally l subsets are obtained, which are used for voting to select the locally optimal instance subset. The process is repeated p times to obtain p subsets. Finally, the globally optimal instance subset is obtained by voting with the p subsets. The proposed algorithm is implemented with two open source big data platforms, Hadoop and Spark, and experimentally compared with three state-of-the-art methods on testing accuracy, compression ratio, and running time. The experimental results demonstrate that the proposed algorithm provides excellent performance and outperforms three baseline methods.

Keywords Big data · Instance selection · Locally sensitive hashing · Voting mechanism · Open source platforms

1 Introduction

With the rapid development of wireless sensing, data storage and internet network technologies, quintillion bytes of data are generated every day from social networks, business transactions, sensors, and many other domains. Big data era has arrived, big data has brought great challenges to traditional data mining. As a way of data preprocessing, instance selection is an effective method for big data. Historically, the instance selection algorithm Condensed Nearest Neighbor (CNN)Hart (1967) was proposed to improve the efficiency of K-NN algorithm. Since CNN was proposed, researchers have proposed many instance selection algorithms, which can be roughly divided into two categories: incremental and decremental Wilson and Martinez (2000).

The incremental algorithm starts with an empty set and iteratively selects important instances from the training set until the stop condition is met. Zhai et al. (2021) extended CNN to uncertain scenarios and proposed a fuzzy CNN algorithm. de Haro-García et al. (2019) presented a method that uses boosting to obtain a subset of instances that is able to improve the classification accuracy of the whole dataset with a significant reduction. The instances are incrementally added by selecting those that maximize the accuracy of the subset using the weighting of instances from the construction of ensembles of classifiers. Malhat et al. (2020) proposed two instance selection algorithms by balancing classification accuracy, reduction rate, and time complexity. The first algorithm selects most relevant instances using a global density and relevance function, the second algorithm maintains the effectiveness of the first algorithm while improving the reduction rate. Evolutionary algorithms based instance selection faces great challenges on search efficiency and com-

✉ Junhai Zhai
mczjh@126.com

[1] Hebei Key Laboratory of Machine Learning and Computational Intelligence, College of Mathematics and Information Science, Hebei University, Baoding 071002, Hebei, China

putational cost. To handle this issue, Cheng et al. (2021) proposed a multi-objective evolutionary algorithm for large-scale instance selection, where a length reduction strategy is adopted to recursively shorten the length of each individual in the population which improves the computational efficiency of the proposed algorithm greatly. García-Pedrajas et al. (2021) combined feature selection and instance selection together, and proposed an approach named SI(FS)^2 that simultaneously select important instances and features. SI(FS)^2 achieves better storage reduction and testing error than previous approaches, and it is scalable to datasets with millions of features. Based on locality-sensitive hashing, Arnaiz-González et al. (2016) proposed an instance selection algorithm with linear complexity for big data. Ma et al. (2020) found that most existing instance and feature selection methods overlook the input-output correlation. To address this issue, they presented a framework for multilabel learning from a topic view. The proposed framework can perform effective instance and feature selection in the latent topic space, as the relation between the input and output spaces is well captured in this space. Fu and Robles-Kelly (2009) investigated the problem of instance selection in multiple instance learning and proposed an instance selection framework for multiple instance learning, which is based on an alternative optimisation framework by iteratively repeating the steps of instance selection/updating and classifier learning. Carbonera and Abel (2020) proposed an attraction-based approach for instance selection, which adopts the notion of attraction for selecting the most representative instances of each class. The advantage of the approach is that it allows the user to define how many representative instances should be selected. Yuan et al. (2018) proposed an instance selection algorithm, which adopts two kinds of instance-selection criteria from two different views to select informative instances for multiple instance learning.

In recent years, due to the popularity of deep learning (LeCun et al. 2015; Goodfellow et al. 2017; Pouyanfar et al. 2019; Zhai et al. 2021, 2019, 2020), researchers have proposed many instance selection methods based on deep learning, especially in the field of computer vision, such as the active selection of informative images from image datasets or important frames from videos. Ding et al. (2021) propose a representative prototype selection algorithm embedding deep multi-instance representation learning. In the proposed approach, a self-expressive dictionary learning model based on sparse and low rank constraint is designed to select the representative prototypes from each subspace of instances. The key point of video summarization is to select the key frames to represent the effective contents of a video sequence. To this end, Huang and Wang (2020) proposed a novel framework for key-frames selection by introducing a self-attention model. Wang et al. (2019) presented an active sample mining framework based on a novel switchable selection criteria. The proposed framework can determine whether an unlabeled sample should be manually annotated via an expert or automatically pseudolabeled via a novel self-learning process. By carefully discriminating locally available training samples based on their relative importance, Jiang et al. (2017) proposed two metrics for prioritizing candidate training samples as functions of their test trial outcome: correctness and confidence.

The decremental method starts from the original training set and iteratively deletes unimportant instances from the training set until the stop condition is satisfied. Reduced Nearest Neighbor (RNN) Gates (1972), Minimal Consistent Set (MCS) Dasarathy (1994) and Decremental Reduction Optimization Procedure (DROP) Wilson and Martinez (2000) are three well-known early decremental algorithms. Cavalcanti and Soares (2020) presented an instance selection algorithm named ranking-based instance selection (RIS) that assigns a score to each instance depending on its relations with all other instances in the training set. Based on the definition of score, two concepts, safe region with higher score and indecision region with lower score are introduced, which is used in a selection process to remove instances from both safe and indecision regions that are considered irrelevant to represent their clusters in the feature space. Their experimental results show that RIS obtains promising accuracy and reduction rates. For the selection of medical image samples, Huang et al. (2021) introduced a divide-and-conquer based instance selection framework that aims to improve the performance of each specific instance selection algorithm. Two well-known decremental algorithms, i.e., DROP3 and IB3 Wilson and Martinez (2000), are used as the baseline, and various small and large scale medical datasets are used in the experiments. Aslani and Seipel (2020) adopted locality-sensitive hashing (LSH) to develop an instance selection method, which rests on rapidly finding similar and redundant training samples and excluding them from the original dataset. The proposed method has two advantages: (1) it has linear time complexity that makes it suitable for handling big datasets. (2) it can significantly reduce the number of instances and execution time. Ireneusz and Piotr (2019) proposed a cluster-based data reduction approach by selecting informative instances from clusters of majority to handle binary imbalanced data classification problems. Jensen et al. Jensen et al. (2019) proposed an effective instance selection method using the fuzzy-rough lower approximation. By utilizing the lower approximation information, instances that have a high similarity with more representative instances of a class can be removed. Orliński and Jankowski (2020) proposed an improved DROP algorithm, which uses random region hashing forests and jungle to keep the computational complexity as low as possible. The proposed algorithm reduces the computational time complexity from $O(m^3)$ to $O(m \log m)$, m is the number of instances.

Hashing techniques has become popular due to its promising performance in both efficiency and accuracy for indexing high-dimensional data, especially for searching similar data points. Hashing methods can be divided into two categories: data-independent and data-dependent. Four excellent surveys on hashing methods can be found in Wang et al. (2016), Chi and Zhu (2017), Wang et al. (2018), Cao et al. (2018). In data-independent hashing approaches, the locality sensitive hashing (LSH) Indyk and Motwani (1998) is the most commonly known data-independent method, its basic idea is to use a set of hash functions that map similar objects into the same hash bucket with a probability higher than non-similar objects. Inspired by the idea of LSH and divide-and-conquer strategy, we proposed an instance selection algorithm for big data. Since $\boldsymbol{a}$ and b are random variables in a hash function $h_{\boldsymbol{a},b}(\boldsymbol{x})$, randomness is inevitably introduced when generating a family of hash functions. In addition, random partition also has randomness. The randomness has a negative effect on the quality of the selected instances, but prior works do not consider the randomness effect on the quality of selected instances. To this end, we propose a double-voting mechanism to handle this issue. The main contributions of this article include the following three folds:

(1) Based on locality sensitive hashing (LSH), divide-and-conquer strategy, and double-voting mechanism, we proposed an instance selection algorithm for big data.

(2) We apply double-voting mechanism to reduce or eliminate unwanted bias introduced by the randomness of selecting hashing functions in LSH and local optimality of the instance subset selected in parallel at each node.

(3) We implemented the proposed algorithm using two big data open source platforms: MapReduce and Spark. We conducted experiments to demonstrate the effectiveness of the proposed algorithm by comparing the proposed algorithm with three state-of-the-art algorithms on three aspects: testing accuracy, compression ratio, and running time.

The rest of this paper is organized as follows. In Sect. 2, we briefly review the preliminaries used in this paper. In Sect. 3, we describe the details of the proposed algorithm. In Sect. 4, the experiments are carried out to verify the effectiveness of the proposed algorithm by comparing it with three state-of-the-art algorithms on three aspects: testing accuracy, compression ratio, and running time. At last, we conclude our work in the Sect. 5.

2 Preliminaries

In this section, we briefly review the preliminaries related to our work, including locality sensitive hashing, Hadoop, and Spark.

2.1 Locality sensitive hashing

Locality sensitive hashing (Indyk and Motwani 1998; Shakhnarovich et al. 2006; Datar et al. 2004; Slaney and Casey 2008; Wang et al. 2013) is a popular data-independent hash method, which is widely used in many fields, such as big data indexing (Bahmani et al. 2012), visual object retrieval (Joly and Buisson 2008), approximate nearest neighbor search (Lu et al. 2018), etc. Let $D = \{\mathbf{x}_i | \mathbf{x}_i \in \mathbb{R}^d, 1 \leq i \leq n\}$, and $\| \cdot \|_2$ represents the l_2 norm. Let $\mathcal{H} = \{\mathbf{h} : \mathbb{R}^d \rightarrow \mathbb{Z}^k\}$ be a set of hash functions. Formally, $\mathbf{h}$ can be expressed as:

$$\mathbf{h}(\cdot) = (h_1(\cdot), h_2(\cdot), \ldots, h_k(\cdot))$$

where the function $h_i(\cdot)(1 \leq i \leq k)$ are the elements of a locality sensitive hashing function set $H = \{h : \mathbb{R}^d \rightarrow \mathbb{Z}\}$. A hashing function set $H = \{h : \mathbb{R}^d \rightarrow \mathbb{Z}\}$ is called $(R, cR; P_1, P_2)$-sensitive for l_2, if for any $\mathbf{p}, \mathbf{q} \in \mathbb{R}^d$, we have

$$P\left(h(\mathbf{p}) = h(\mathbf{q})\right) \geq P_1, \text{when} \parallel \mathbf{p} - \mathbf{q} \parallel_2 \leq R \tag{1}$$

and

$$P\left(h(\mathbf{p}) = h(\mathbf{q})\right) \leq P_2, \text{when} \parallel \mathbf{p} - \mathbf{q} \parallel_2 \geq cR \tag{2}$$

where R is the distance threshold, c is the approximation ratio, and $c > 1$, and $P_1 > P_2$. Intuitively, the formulas (1) and (2) mean that nearby objects within distance R have a greater chance of being hashed to the same value than objects that the distance greater than cR Shakhnarovich et al. (2006).

Typically, the functions $h \in H$ are defined as:

$$h_{\mathbf{a},b}(\mathbf{x}) = \left\lfloor \frac{\mathbf{a} \cdot \mathbf{x} + b}{w} \right\rfloor \tag{3}$$

where $\mathbf{a} \in \mathbb{R}^d$ is a random vector with elements chosen independently from a Gaussian distribution and b is a real number chosen uniformly from the closed interval $[0, w]$, and the w is a user-defined parameter that controls the possibility of collision of two data points.

Let $u = \parallel \mathbf{p} - \mathbf{q} \parallel_2$, and $p(u)$ denote the probability that $\mathbf{p}, \mathbf{q}$ collide for a hash function $h_{\mathbf{a},b}(\cdot)$, $f(t)$ denotes the absolute value of the probability density function of the Gaussian distribution. The calculation of $p(u)$ is given by Eq. (4) Shakhnarovich et al. (2006).

$$\begin{aligned} p(u) &= P\left(h_{\mathbf{a},b}(\mathbf{p}) = h_{\mathbf{a},b}(\mathbf{q})\right) \\ &= \int_0^w \frac{1}{u} f\left(\frac{t}{u}\right)\left(1 - \frac{t}{w}\right) dt \end{aligned} \tag{4}$$

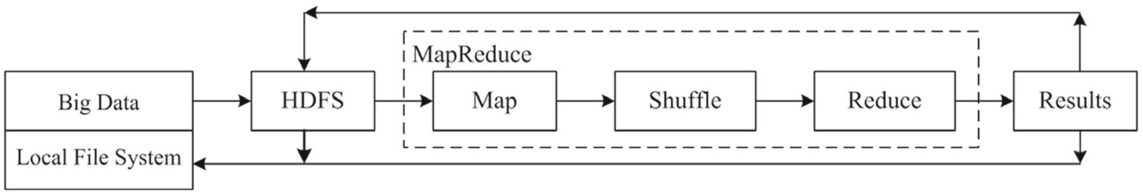
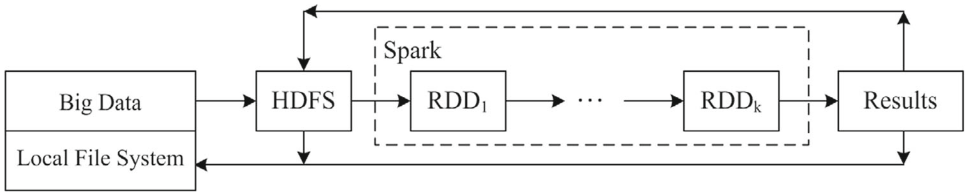


Fig. 1 The big data manipulation processes by Hadoop

Fig. 2 The big data manipulation processes by Spark



Obviously, for a fixed parameter w, the probability of collision of two data points $\mathbf{p}$ and $\mathbf{q}$ decreases monotonically with $u = \| \mathbf{p} - \mathbf{q} \|_2$.

2.2 Hadoop

Hadoop[1] is a framework that allows for the distributed processing of large datasets across clusters of computers using simple programming models. It deals with big data in a divide-and-conquer strategy. Hadoop uses HDFS (Hadoop Distributed File System) to organize and store big data, and employs MapReduce to handle big data, and fulfill the user's application logic. MapReduce processes big data in three stages which are map, shuffle, and reduce (see Fig. 1). The easy usage of Hadoop is because that it encapsulates the following processing details:

(1) Automatically partitions of computing tasks and deployments of computing sub-tasks;
(2) Automatically distributed storage of the processed big data;
(3) The synchronization of processing data and computing tasks;
(4) Aggregation and shuffle of the processed data;
(5) Communication between computing nodes;
(6) Load balance and performance optimization;
(7) Node failure detection and recovery.

2.3 Spark

Spark[2] is a unified analytics engine for large-scale data processing. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It introduces resident distributed dataset (RDD) to avoid excessive network and disk I/O overhead during computation. RDD is an abstract data structure that stores a large dataset in distributed memory on a cluster of servers. More specifically, RDD divides a large dataset into several data blocks, which are distributed across the nodes of the cluster, either in the memory of the nodes or in the disk of the nodes. Each data block has a ID, which is used to manage the data block by the metadata of ID. Logically, these data blocks are divided into splits. RDD manipulates the splits by operators, and the result of the operation is a new RDD (see Fig. 2).

Spark provides a set of operators for RDD. The operators can be roughly divided into the following two categories:

(1) Transformation operators: they are used to transform one RDD into another RDD;
(2) Action operators: they trigger the Spark jobs to be executed and saves the resulted RDD.

In addition, to deal with the problem of heavy loads due to start and schedule of jobs, Spark optimizes task scheduling based on directed acyclic graphs (DAG). As a result, there is no need to store intermediate results for each phase on HDFS, thus greatly improving the efficiency of task scheduling.

3 The proposed algorithm

A hash table is composed of buckets with each bucket indexed by a hash code. The conventional hashing algorithms attempt to avoid mapping two data points into the same bucket. Different from the conventional hashing algorithms, we aim at maximizing the probability of collision of similar instances and at the same time minimize the probability of collision of dissimilar instances. In other words, we attempt to map the similar instances into same hash bucket, and map the dissimilar instances into different hash buckets. Because the

[1] https://hadoop.apache.org/

[2] http://spark.apache.org/

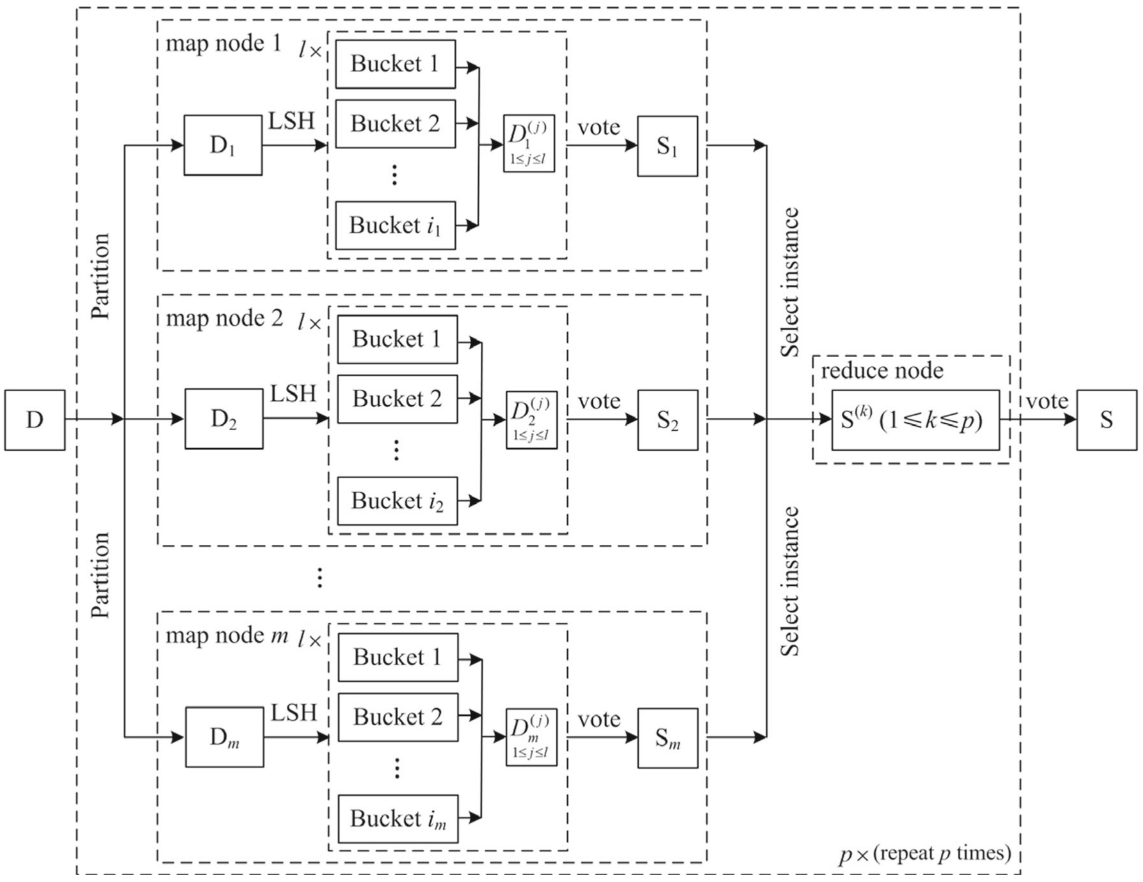


Fig. 3 The technical route of the proposed algorithm

instances in the same hash bucket are similar, we can randomly select a proportion of the instances from each hash bucket, this is the basic idea of the proposed algorithm. Obviously, this method can introduce unwanted bias to the solution (i.e., the selected instance subset) due to the randomness of hash functions and random selection mechanism. To this end, we use voting mechanism by multiple hash tables to reduce the bias. For big datasets, this instance selection process is performed in parallel on multiple computing nodes. Thus, the instance subset selected at each computing node is locally optimal, i.e., it is local optimal corresponding to local instance subset. To obtain the globally optimal instance subset or globally suboptimal instance subset, we use the second voting mechanism. The technical route of the proposed algorithm is shown in the Fig. 3.

It can be seen from Fig. 3 that the proposed algorithm includes 6 steps:

(1) Partition big dataset. Partition the big dataset D is divided into m subsets, $D_1, D_2, \ldots, D_m$, and the m subsets are distributed to m computing nodes.
(2) Construct hash functions and perform hash transforms. At each of the m nodes, l hash functions are constructed using Eq. (3), and l hash tables are obtained using the constructed l hash functions, and the instances in the local subset are inserted into each hash bucket of the l hash tables.
(3) Select instances by proportion. Instances are selected proportionally from each hash bucket of l hash tables, and l instance subsets, $D_i^{(1)}, D_i^{(2)}, \ldots, D_i^{(l)}$, are obtained.
(4) Select instances by voting. At each node of m nodes, a locally optimal instance subset corresponding to the local subset D_i is selected by voting using the l subsets, $D_i^{(1)}, D_i^{(2)}, \ldots, D_i^{(l)}$, and obtain S_i, $1 \le i \le m$.
(5) Merge m subsets. In reduce node, the m locally optimal instance subsets, $S_1, S_2, \ldots, S_l$, selected in m map nodes are merged, and obtain an instance subset $S^{(i)}$ corresponding to big dataset D.
(6) Repeat the above process p times to get p subsets, $S^{(1)}, S^{(2)}, \ldots, S^{(p)}$. Finally, an optimal or suboptimal subset S is obtained by double-voting.

At each of the m map computing nodes, LSH transform works as follows:

(1) Construct hash functions. l d-dimensional hash functions, $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_l$, are constructed independently and

randomly. Specifically, for each $\mathbf{h}_i(1 \le i \le l)$, we have to construct its q components $h_{ij}(1 \le j \le q)$, which are q 1-dimensional hash functions, the q is the length of hash code. The construction process consists of three steps: Firstly, determine the width w of the hash bucket. Secondly, generate a d-dimensional random vector $\mathbf{a}_i$ with a Gaussian distribution, and generate a random variable b_i that follows uniform distribution in the interval [0, w]. Finally, obtain a 1-dimensional hash function $h_{ij}(\mathbf{x}) = \left\lfloor \frac{\mathbf{a}_i \cdot \mathbf{x} + b_i}{w} \right\rfloor$.

(2) Construct hash tables. Each hash function is used to construct a hash table, as a result, l hash tables are obtained.

(3) Perform hash transform. Transform all instances $\mathbf{x} \in D$ into Hamming space, and assign all data points to each of the l hash tables by the corresponding l hash codes.

The pseudocode of the proposed algorithm denoted by LSHDV is given in Algorithm 1.

Algorithm 1: LSHDV Algorithm

```
Input: Data set $D$, parameters $l, p, w, \lambda_1, \lambda_2$.
Output: Instance subset $S$.
for ($k = 1; k \le p; k = k + 1$) do
    // Partition data set;
    Partition data set $D$ into $m$ subsets, $D_1, D_2, \ldots, D_m$, and distribute them to $m$ computing nodes;
    for ($i = 1; i \le m; i = i + 1$) do
        // Construct the hash functions;
        for ($j = 1; j \le l; j = j + 1$) do
            for ($s = 1; s \le q; s = s + 1$) do
                Generate a random vector $\mathbf{a}_j^{(s)}$ with Gaussian distribution;
                Generate a random variable $b_j^{(s)}$ with uniform distribution in [0, w];
                Let $h_{js}(\mathbf{x}) = \left\lfloor \frac{\mathbf{a}_j^{(s)} \cdot \mathbf{x} + b_j^{(s)}}{w} \right\rfloor$;
            end
            Generate a hash function $\mathbf{h}_j(\mathbf{x}) = (h_{j1}(\mathbf{x}), \ldots, h_{jq}(\mathbf{x}))$;
            // Perform hash transform;
            for ($\forall \mathbf{x} \in D_i$) do
                Perform hash transform $\mathbf{h}_j(\mathbf{x})$, and put $\mathbf{x}$ into different hash bucket;
            end
            // Select instances;
            for ($t = 1; t \le l_j; t = t + 1$) do
                Select instances in proportion from the $t^{th}$ hash bucket, and obtain subset $D_i^{(j)}$;
            end
            // Local voting;
            Let $S_i = \varnothing$; vote($\mathbf{x}$) = 0;
            if $\left(x \in D_i^{(j)}\right)$ then
                vote($\mathbf{x}$) = vote($\mathbf{x}$) + 1;
            end
            if ($vote(\mathbf{x}) \ge \lambda_1$) then
                $S_j = S_j \cup \{\mathbf{x}\}$;
            end
        end
    end
    // Merge selected subsets;
    $S^{(k)} = \bigcup_{i=1}^{m} S_i$;
    // Global voting;
    for ($k = 1; k \le p; k = k + 1$) do
        Let $S = \varnothing$; vote($\mathbf{x}$) = 0;
        if $\left(x \in S^{(k)}\right)$ then
            vote($\mathbf{x}$) = vote($\mathbf{x}$) + 1;
        end
        if ($vote(\mathbf{x}) \ge \lambda_2$) then
            $S = S \cup \{\mathbf{x}\}$;
        end
    end
end
Output $S$.
```

4 Experimental results and analysis

To verify the effectiveness of the proposed algorithm, we experimentally compared the proposed algorithm with three closely related methods on six datasets using two open-source big data platforms, MapReduce and Spark. In our experiments, the training set/test set mechanism is adopted, we randomly partition each data set into training set and testing set, 70% of instances are used for training, and 30% of instances are used for testing. The proposed algorithm implemented by MapReduce and Spark are denoted by LSHDV-MR and LSHDV-SP respectively. The three closely related methods are LSH, CNN, and VE based (voting entropy) instance selection algorithms for big data (Arnaiz-González et al. 2016; Hart 1967; Seung et al. 1992). The three methods implemented with MapReduce and Spark are denoted by LSH-MR, CNN-MR, VE-MR, and LSH-SP, CNN-SP, VE-SP respectively. The six datasets include two artificial datasets and 4 UCI datasets (Dua and Graf 2019). The two artificial datasets follows Gaussian distribution, and their parameters are given in Tables 1 and 2 respectively. The basic information of the six datasets is given in Table 3. In Table 3, ♯ Instances, ♯ Attributes, and ♯ Classes represent the number of instances, the number of attributes, and the number of classes, respectively.

4.1 Determination of the hyper-parameters

The proposed algorithm LSHDV include seven parameters: three parameters for LSH and four parameters for the proposed algorithm. The three parameters in LSH are the quantization width w, the number of hash tables l, and the number of projections k (i.e., the length of hash code), we use the method in Shakhnarovich et al. (2006) to determine the three parameters, the optimization of the three parameters in LSH can be found in Slaney et al. (2012). The four parameters in the proposed algorithm are the parameter δ

Table 1 The mean vectors and covariance matrices of the first artificial dataset Gaussian1

i	¯$_i$	$\mathbf{6}_i$
1	$(1.0, 1.0)^T$	$\begin{bmatrix} 0.6 & -0.2 \\ -0.2 & 0.6 \end{bmatrix}$
2	$(2.5, 2.5)^T$	$\begin{bmatrix} 0.2 & -0.1 \\ -0.1 & 0.2 \end{bmatrix}$

Table 2 The mean vectors and covariance matrices of the second artificial dataset Gaussian2

i	¯$_i$	$\mathbf{6}_i$
1	$(0.0, 0.0, 0.0)^T$	$\begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$
2	$(0.0, 1.0, 0.0)^T$	$\begin{bmatrix} 1.0 & 0.0 & 1.0 \\ 0.0 & 2.0 & 2.0 \\ 1.0 & 2.0 & 5.0 \end{bmatrix}$
3	$(-1.0, 0.0, 1.0)^T$	$\begin{bmatrix} 2.0 & 0.0 & 0.0 \\ 0.0 & 6.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$
3	$(0.0, 0.5, 1.0)^T$	$\begin{bmatrix} 2.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 3.0 \end{bmatrix}$

Table 3 The basic information of the six datasets

Datasets	♯ Instances	♯ Attributes	♯ Classes
Gaussian1	1000000	2	2
Gaussian2	1000000	3	4
Shuttle	58000	9	7
Poker	1000000	10	10
CovType	581012	54	7
Skin	245057	3	2

that determines the proportion of instances to be selected from each hash bucket, the local voting threshold λ_1, the global voting threshold λ_2, and the parameter p, the number of times the algorithm is repeated. It is easy to determine the values of the four parameters, δ, λ_1, λ_2, and p.

4.2 The experiments compared with the three related methods

We experimentally compared LSHDV with LSH, CNN, and VE based methods on two open-source big data platforms, MapReduce and Spark on three aspects: testing accuracy, compression ratio, and running time, the classifier used in our experiments is extreme learning machine Huang et al. (2006), the experimental results compared with three related methods with big data platform MapReduce are given in Tables 4, 5, 6, and the experimental results compared with three related methods with big data platform Spark are given in Tables 7, 8, 9, respectively. In Tables 4, 5, 7, and 8, the bold values are the corresponding maximum, and in Tables 6 and 9, the bold values are the corresponding minimum.

The experimental results listed in Tables 4, 5, 6 and Tables 7, 8, 9 demonstrate that the proposed algorithm LSHDV is superior to the other three algorithms in test accuracy and compression ratio except for CovType data set. The reason that LSHDV is superior to the other three algorithms

Table 4 The experimental results compared with three related methods with MapReduce on test accuracy

Datasets	LSH-MR	CNN-MR	VE-MR	LSHDV-MR
Gaussian1	0.979	0.971	0.980	**0.993**
Gaussian2	0.499	0.450	0.515	**0.538**
CovType	0.920	0.913	**0.933**	0.930
Poker	0.853	0.878	0.904	**0.911**
Shuttle	0.987	0.983	0.981	**0.989**
Skin	0.962	0.906	0.970	**0.983**

Table 5 The experimental results compared with three related methods with MapReduce on compression ratio

Datasets	LSH-MR	CNN-MR	VE-MR	LSHDV-MR
Gaussian1	9.33	3.53	11.25	**12.44**
Gaussian2	5.52	2.77	11.55	**12.31**
CovType	2.23	1.05	**7.24**	6.98
Poker	8.64	6.53	10.64	**11.54**
Shuttle	1.53	1.05	2.87	**3.19**
Skin	3.88	1.00	5.21	**5.49**

Table 6 The experimental results compared with three related methods with MapReduce on running time (s)

Datasets	LSH-MR	CNN-MR	VE-MR	LSHDV-MR
Gaussian1	66045	73709	723662	**60114**
Gaussian2	136023	87112	1302360	**51771**
CovType	**62336**	579003	936665	65531
Poker	421098	632171	3884616	**300114**
Shuttle	213094	342111	367825	**96771**
Skin	239113	591030	660351	**208514**

Table 7 The experimental results compared with three related methods with Spark on test accuracy

Datasets	LSH-SP	CNN-SP	VE-SP	LSHDV-SP
Gaussian1	0.980	0.973	0.987	**0.994**
Gaussian2	0.499	0.459	0.511	**0.548**
CovType	0.923	0.915	**0.930**	0.928
Poker	0.851	0.874	0.902	**0.909**
Shuttle	0.988	0.985	0.990	**0.993**
Skin	0.960	0.905	0.974	**0.979**

Table 8 The experimental results compared with three related methods with Spark on compression ratio

Datasets	LSH-SP	CNN-SP	VE-SP	LSHDV-SP
Gaussian1	9.30	3.55	11.19	**12.21**
Gaussian2	5.49	2.73	11.74	**12.00**
CovType	2.28	1.04	**6.77**	6.69
Poker	8.68	6.49	11.44	**11.50**
Shuttle	1.57	1.03	2.80	**3.33**
Skin	3.90	1.07	5.24	**5.50**

Table 9 The experimental results compared with three related methods with Spark on running time (s)

Datasets	LSH-SP	CNN-SP	VE-SP	LSHDV-SP
Gaussian1	34121	41280	67288	**32528**
Gaussian2	97563	30114	147015	**24889**
CovType	**50109**	157426	184774	53159
Poker	269344	357483	422381	**211602**
Shuttle	100494	110610	231865	**47355**
Skin	108442	317734	431078	**135145**

in test accuracy is that it adopts double-voting mechanism, one is local voting, the other is global voting. The combination of local and global voting can significantly improve the quality of the selected instances and improve the test accuracy of the classifier. The reason that LSHDV is superior to the other three algorithms in compression ratio is due to the convenient parameter control mechanism. The compression ratio of the proposed algorithm LSHDV is affected by three parameters: (1) δ, the proportion of instances selected from each hash bucket at each calculation node, (2) the local voting threshold parameter λ_1, and (3) the global voting threshold parameter λ_1. It is easy to select appropriate values of the three parameters using good prior knowledge. In addition, the high computational efficiency of locally sensitive hashing also provides support for the good performance of the proposed algorithm. On the CovType data set, the performance of the proposed algorithm LSHDV is inferior to other algorithms, we think that's because the size of the data set is small and it is difficult to give full play to the advantages of the big data platform and the algorithm LSHDV itself.

From the experimental results listed in Tables 4, 5, 6 and Table 7, 8, 9, it can be seen that there is no significant difference in test accuracy and compression ratio between the two implementations with the two big data open source platforms, MapReduce and Spark. It is easy to understand why there is no significant difference in test accuracy and compression ratio, because the algorithms select instances by the same mechanisms, but the two implementation mechanisms are different. However, there are significant differences in

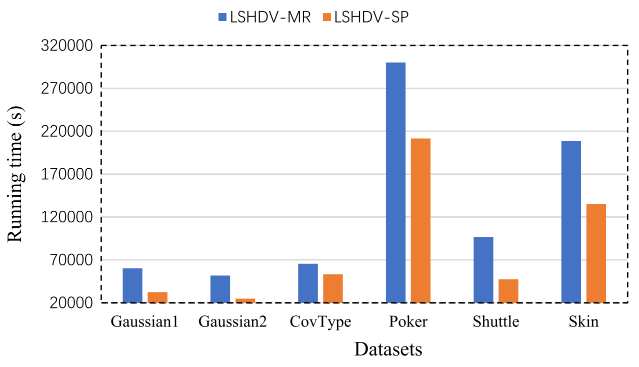


Fig. 4 Visualization of running time comparisons of LSHDV-MR and LSHDV-SP

running time between the four algorithms implemented on the two big data platforms. Taking LSHDV-MR and LSHDV-SP as examples, Fig. 4 shows the significant difference in the running time of the algorithm LSHDV implemented by the two big data platforms, MapReduce and Spark.

Why are there such a big difference in running time? We think the reasons are as follows:

(1) The reason comes from algorithm itself. Compared with the other three methods, the algorithm proposed in this paper has no heuristic calculation, which selects some representatives from similar instances, the drift of data distribution is negligible. After the hash transformation, the data points are transformed from the original space to the Hamming space. In Hamming space, no extra computation cost is needed to determine whether two data points are similar. This is the main reason for the high efficiency of the algorithm in this paper.
(2) Reasons from big data platform. This is because Spark is an in-memory computation-based platform, where the intermediate results of computation are buffered in memory, and the intermediate results are cached into external memory only when the cache's capacity reaches a certain threshold (e.g., 0.8), which results in I/O operations. MapReduce, on the other hand, is a batch processing platform where the intermediate results are cached into external memory, resulting in a large number of I/O operations.

In the following, we present a theoretical analysis from the perspective of computational time complexity. The running time of the proposed algorithm LSHDV with the two platforms includes four parts: T_r which is the time of reading data files, T_s which is the time of sorting intermediate data, T_t which is the time of transferring intermediate data, and T_w which is the time of writing files. T_r and T_w are same for the two big data platforms, MapReduce and Spark. On MapReduce, sorting the intermediate data is indispensable. Sorting realizes the merge of intermediate data, which can ensure that a map task corresponds to an ordered intermediate file. When the data files are transferred from the map node to the reduce node, the network load and transmission are largely reduced. Suppose there are m map tasks in MapReduce, and each map task is responsible for processing q pieces of data. Because MapReduce uses quick-sort algorithm to sort intermediate data, the sort time of MapReduce for intermediate files is $T_{\mathrm{s-mr}} = m \log q$. On Spark, because there is no sort process for intermediate data, the sort time for intermediate files is $T_{\mathrm{s-spark}} = 0$. Obviously, $T_{\mathrm{s-mr}} > T_{\mathrm{s-spark}}$.

Because the MapReduce sorts and merges intermediate data, the scale of the data decreases as it transforms from the map node to the reduce node. When MapReduce and Spark process the same data set, the intermediate data on the MapReduce platform will be smaller than the intermediate data on Spark. Namely, $N_{\text{t-mr-number of files}} < N_{\text{t-spark-number of files}}$.

When MapReduce processes a large data set, it first reads the data files to be processed from HDFS into the map node. The data set is processed by the map node and the intermediate files generated are stored in HDFS. Then, the reduce node pulls the intermediate files from HDFS according to the key values. Spark is a big data platform based on in-memory computing, the intermediate results of the data set calculated by Spark will be directly stored in memory. When the memory cannot provide enough capacity for the intermediate results, they will be overwritten to HDFS, which greatly reduces network I/O and disk I/O. Therefore, the transfer time of the intermediate files based on MapReduce is much greater than the transfer time based on Spark, i.e., we have $T_{\mathrm{t-q-mr}} >> T_{\mathrm{t-q-spark}}$.

Since, $T_t = N_{\mathrm{t-numberoffiles}} \times T_{t-q}$. It is easy to obtain that $T_{\mathrm{t-mr}} >> T_{\mathrm{t-spark}}$ is hold for large scale data sets.

5 Conclusion

Inspired by the idea of locality sensitive hashing, divide-and-conquer strategy, and double-voting mechanism, we proposed an instance selection algorithm for big data. The proposed algorithm has three advantages: (1) The idea of the proposed algorithm is simple, but it is very effective and efficient; (2) The selected instance set is not only of high quality but also of high compression ratio; (3) For different data sets, it is easy for the proposed algorithm to choose appropriate hyperparameters, and the algorithm has strong autonomous controllability. In the future works, (1) we will study how to use data mining method to learn the hash function more suitable for different data sets according to the characteristics of data sets themselves; (2) we will conduct researches of applications of the proposed algorithm, investigating how to apply the proposed algorithm LSHDV to select key frames from video and how to select important samples from image big data ImageNet.

Acknowledgements Finding This research is supported by the key R&D program of science and technology foundation of Hebei Province (19210310D), and by the natural science foundation of Hebei Province (F2021201020).

Availability of data and materials Not applicable.

Code availability Not applicable.

Compliance with ethical standards

Conflict of interest Authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

Arnaiz-González A, Díez-Pastor JF, Rodríguez JJ et al (2016) Instance selection of linear complexity for big data. Knowl Based Syst 107:83–95

Aslani M, Seipel S (2020) A fast instance selection method for support vector machines in building extraction. Appl Soft Comput 97(Part B):106716

Bahmani B, Goel A, Shinde R (2012) Efficient distributed locality sensitive hashing. In: Proceedings of the 21st ACM international conference on Information and knowledge management, pp 2174–2178

Cao Y, Qi H, Zhou W et al (2018) Binary hashing for approximate nearest neighbor search on big data: a survey. IEEE Access 6:2039–2054

Carbonera JL, Abel M (2020) An attraction-based approach for instance selection. In: 2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI), 2020, pp. 1053–1058. https://doi.org/10.1109/ICTAI50040.2020.00161

Cavalcanti GDC, Soares RJO (2020) Ranking-based instance selection for pattern classification. Expert Syst Appl 150:113269

Cheng F, Chu F, Zhang L (2021) A Multi-Objective Evolutionary Algorithm based on Length Reduction for Large-Scale Instance Selection. Inform Sci. https://doi.org/10.1016/j.ins.2021.06.052

Chi L, Zhu C (2017) Hashing techniques: a survey and taxonomy. ACM Comput Surv 50(1):1–36

Dasarathy BV (1994) Minimal consistent set identification for optimal nearest neighbor decision systems design. IEEE Trans Syst Man Cybern 24(1):511–517

Datar M, Immorlica N, Indyk P, et al. (2004) Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of Symposium on Computational geometry, pp 253–262

de Haro-García A, Cerruela-García G, García-Pedrajas N (2019) Instance selection based on boosting for instance-based learners. Pattern Recogn 96:106959

Ding X, Li B, Li Y et al (2021) Web objectionable video recognition based on deep multi-instance learning with representative prototypes selection. IEEE Trans Circ Syst Video Technol 31(3):1222–1233. https://doi.org/10.1109/TCSVT.2020.2992276

Dua D, Graf C (2019) UCI machine learning repository. University of California, School of Information and Computer Science, Irvine. http://archive.ics.uci.edu/ml

Fu Z, Robles-Kelly A (2009) An instance selection approach to Multiple instance Learning. IEEE Conf Comput Vis Pattern Recog 2009:911–918. https://doi.org/10.1109/CVPR.2009.5206655

García-Pedrajas N, del Castillo JAR, Cerruela-García G (2021) SI(FS)^2: fast simultaneous instance and feature selection for datasets with many features. Pattern Recogn 111:107723

Gates GW (1972) The reduced nearest neighbor rule. IEEE Trans Inform Theory 18(3):431–433

Goodfellow I, Bengio Y, Courville A (2017) Deep learning. Mit Press, Cambridge

Hart P (1967) The condensed nearest neighbor rule. IEEE Trans Inform Theory 14(5):515–516

Huang C, Wang H (2020) A novel key-frames selection framework for comprehensive video summarization. IEEE Trans Circ Syst Video Technol 30(2):577–589. https://doi.org/10.1109/TCSVT.2019.2890899

Huang GB, Zhu QY, Siew CK (2006) Extreme learning machine: Theory and applications. Neurocomputing 70:489–501

Huang MW, Tsai CF, Lin WC (2021) Instance selection in medical datasets: a divide-and-conquer framework. Comput Elect Eng 90:106957

Indyk P, Motwani R (1998) Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the 30th Annual ACM Symposium on Theory of Computing, pp 604–613

Ireneusz C, Piotr J (2019) Data reduction and stacking for imbalanced data classification. J Intell Fuzzy Syst 37(6):7239–7249

Jensen R, Amiri M, Parthaláin NM (2019) Effective instance selection using the fuzzy-rough lower approximation. In: 2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), pp. 1-6, https://doi.org/10.1109/FUZZ-IEEE.2019.8858799

Jiang Z, Zhu X, Tan W et al (2017) Training sample selection for deep learning of distributed data. IEEE Int Conf Image Process (ICIP) 2017:2189–2193. https://doi.org/10.1109/ICIP.2017.8296670

Joly A, Buisson O (2008) A posteriori multi-probe locality sensitive hashing. In: Proceedings of the 16th ACM international conference on Multimedia, October 26-31, 2008, Vancouver, British Columbia, Canada, pp 209–218

Le Cun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521:436–444

Lu K, Wang H, Xiao Y et al (2018) Why locality sensitive hashing works: a practical perspective. Inform. Process. Lett. 136:49–58

Ma J, Chow TWS (2020) Topic-based instance and feature selection in multilabel classification. In: IEEE Transactions on Neural Networks and Learning Systems, Early Access Article. https://doi.org/10.1109/TNNLS.2020.3027745

Malhat M, Menshawy ME, Mousa H et al (2020) A new approach for instance selection: algorithms, evaluation, and comparisons. Expert Syst Appl 149:113297

Orliński M, Jankowski N (2020) $O(m \log m)$ instance selection algorithms–RR-DROPs. Int Joint Conf Neural Netw (IJCNN) 2020:1–8. https://doi.org/10.1109/IJCNN48605.2020.9207158

Pouyanfar S, Sadiq S, Yan Y et al (2019) A survey on deep learning: algorithms, techniques, and applications. ACM Comput Surv 51(5):9.21-92.36

Seung HS, Opper M, Sompolinsky H (1992) Query by committee. In: Proceedings of the fifth annual workshop on Computational learning theory, pp 287–294

Shakhnarovich G, Darrell T, Indyk P (2006) Locality-sensitive hashing using stable distributions. In: Nearest-neighbor methods in learning and vision: theory and practice. MIT Press, pp 61–72

Slaney M, Casey M (2008) Locality-sensitive hashing for finding nearest neighbors. IEEE Signal Process Mag 25(2):128–131

Slaney M, Lifshits Y, He J (2012) Optimal parameters for locality-sensitive hashing. Proc IEEE 100(9):2604–2623

Wang H, Cal J, Shu L, et al. (2013) Locality sensitive hashing revisited: filling the gap between theory and algorithm analysis. In: Proceedings of the 22nd ACM international conference on Information and Knowledge Management, pp 1969–1978

Wang J, Liu W, Kumar S et al (2016) Learning to hash for indexing big data—a survey. Proc IEEE 104(1):34–57

Wang J, Zhang T, Song J et al (2018) A Survey on Learning to Hash. IEEE Trans Pattern Anal Mach Intell 40(4):769–790

Wang K, Lin L, Yan X et al (2019) Cost-effective object detection: active sample mining with switchable selection criteria. IEEE Trans Neural Netw Learn Syst 30(3):834–850. https://doi.org/10.1109/TNNLS.2018.2852783

Wilson DR, Martinez TR (2000) Reduction techniques for instance-based learning algorithms. Mach Learn 38(3):257–286

Yuan L, Wen X, Xu H, et al (2018) Multiple-instance learning with empirical estimation guided instance selection. In: 018 24th International Conference on Pattern Recognition (ICPR), 2018, pp. 770–775. https://doi.org/10.1109/ICPR.2018.8546304

Zhai M, Chen L, Tung F et al (2019) Lifelong GAN: continual learning for conditional image generation. IEEE/CVF Int Conf Comput Vis (ICCV) 2019:2759–2768. https://doi.org/10.1109/ICCV.2019.00285

Zhai M, Chen L, He J et al (2020) Piggyback GAN: efficient lifelong learning for image conditioned generation. In: Vedaldi A, Bischof H, Brox T, Frahm J (eds) Computer vision-ECCV 2020. ECCV 2020. Lecture notes in computer science, vol 12366. Springer, Cham. https://doi.org/10.1007/978-3-030-58589-1_24

Zhai MY, Chen L, Mori G (2021) Hyper-LifelongGAN: scalable lifelong learning for image conditioned generation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR2021), pp. 2246-2255

Zhai JH, Qi JX, Zhang SF (2021) An instance selection algorithm for fuzzy K-nearest neighbor. J Intell Fuzzy Syst 40(1):521–533