Information Sciences 577 (2021) 579–598

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Efficient and decision boundary aware instance selection for support vector machines

Mohammad Aslani [a,*], Stefan Seipel [a,b]

[a] Department of Computer and Geo-spatial Sciences, University of Gävle, Gävle, Sweden
[b] Division of Visual Information and Interaction, Department of Information Technology, Uppsala University, Uppsala, Sweden

ARTICLE INFO

Article history:
Received 11 October 2020
Received in revised form 29 June 2021
Accepted 3 July 2021
Available online 6 July 2021

Keywords:
Instance selection
Data reduction
Big data
Support vector machines
Machine learning

ABSTRACT

Support vector machines (SVMs) are powerful classifiers that have high computational complexity in the training phase, which can limit their applicability to large datasets. An effective approach to address this limitation is to select a small subset of the most representative training samples such that desirable results can be obtained. In this study, a novel instance selection method called border point extraction based on locality-sensitive hashing (*BPLSH*) is designed. *BPLSH* preserves instances that are near the decision boundaries and eliminates nonessential ones. The performance of *BPLSH* is benchmarked against four approaches on different classification problems. The experimental results indicate that *BPLSH* outperforms the other methods in terms of classification accuracy, preservation rate, and execution time. The source code of *BPLSH* can be found in https://github.com/mohaslani/BPLSH.

1. Introduction

Support vector machines (SVMs) are effective classifiers with a definite theoretical foundation and have been extensively used in various applications in different fields, such as data mining [42], remote sensing [32], and geoscience [40]. SVMs come with a minimal structural risk because they search for a separating hyperplane that represents the maximum margin between classes. This feature makes SVMs more effective than other classifiers. However, training an SVM, in which support vectors are obtained, requires solving a quadratic programming optimization problem, which poses a computational complexity of $O(n^3)$, where n is the number of training samples. This computational cost inhibits the applicability of SVMs to tasks involving large datasets, such as feature extraction from high-resolution aerial images.

Within such a context, researchers have developed different approaches to address the mentioned drawback of SVMs. Among others, instance selection has shown high potential for circumventing the computational burden of the training phase in handling big datasets [33]. The notion of instance selection rests on the fact that training data are not used to describe classes, but to assist in accurate separation. Instance selection methods speed up training by selecting a small subset of instances that is representative of the original set and can retain the original classification power of the SVM [46]. In other words, the instances with great potential to contribute to the classification and construction of demarcation hyperplanes are preserved. These patterns, called support vector candidates, lie close to the border of classes, and thus they have also been

* Corresponding author.
E-mail addresses: Mohammad.Aslani@hig.se, Moh.Aslani@gmail.com (M. Aslani), Stefan.Seipel@hig.se (S. Seipel).

https://doi.org/10.1016/j.ins.2021.07.015
0020-0255/

named border instances [26]. The remaining instances (i.e., interior instances) that do not significantly contribute to the classification can be discarded to accelerate the training phase of SVMs.

Many instance selection methods have been proposed owing to the increasing number of records in datasets. An ideal instance selection method maintains the original classification accuracy while significantly eliminating superfluous instances within the shortest possible execution time. However, designing an ideal instance selection method is difficult, and most existing methods with high reduction rates and classification accuracies suffer from long execution times and are consequently inapplicable to large datasets.

Contribution of this study. In the present study, a new instance selection method that is applicable to large datasets and effectively balances reduction rate and classification accuracy is designed. Specifically, the concept of locality-sensitive hashing (LSH) [15,13] is adopted to design a novel instance selection method. The proposed method, named border point extraction based on LSH (*BPLSH*), preserves border patterns and a few interior data points to accelerate the SVM training. In *BPLSH*, the nearest patterns belonging to opposite classes of a given instance are regarded as border samples and are preserved, whereas instances that are far from opposite classes are considered as interior ones and are removed. The closeness of instances to a given sample is measured by the similarity index, which is defined based on partitioning the feature space.

Outline of this study. The remainder of this paper is structured as follows. Section 2 presents a brief review of some existing instance selection methods. The concepts of LSH are introduced in Section 3. Section 4 explains the proposed method (*BPLSH*) in detail. The performance of *BPLSH* is analyzed and benchmarked on nine datasets in Section 5. Section 6 evaluates the performance of *BPLSH* in automatic building extraction from high-resolution aerial images, where handling a large dataset is necessary. Finally, in Section 7 we present our conclusions and propose directions for future work.

2. Related work

The main objective of instance selection methods for SVMs is to identify instances that are close to the decision boundaries. To do so, various strategies have been proposed, and a comprehensive survey of which can be found in [19,33]. In this section, some of the most relevant methods are briefly explained. It has to be underlined that evolutionary methods are not reviewed because of their high computational complexity, which makes them inapplicable to big datasets.

One of the most common strategies is clustering to partition the feature space, accompanied with some mechanisms to identify border instances. In this context, Lyhyaoui et al. [30] proposed a standard yet effective procedure for instance selection that utilizes clustering. Their proposed procedure, which has since been widely used by other researchers, consists of three main steps: 1- clustering instances, 2- identifying critical clusters that are likely to contain support vectors, and 3- selecting a representative subset that encompasses border instances and a small number of internal samples. They used frequency sensitive competitive learning [2] to cluster data and the closeness of instances to the opposite-class cluster centroids as a heuristic to find effective instances.

Another clustering-based method, referred to as SVMKM, was proposed by Almeida et al. [3]. This method used the k-means algorithm to partition instances. The heuristic used for finding border instances is the heterogeneity of each cluster, which indicates whether the cluster is mixed-class. More specifically, all instances of heterogeneous clusters (i.e., clusters formed by instances from different classes) are preserved as they represent border samples, whereas all homogeneous clusters (i.e., clusters that include instances from one class) are discarded, and only one point from each of them (the centroid or a point near the centroid) is retained. To improve the performance of SVMKM, Koggalage & Halgamuge [22] proposed an algorithm in which homogeneous clusters are not automatically discarded; instead, exterior instances positioned near the boundary of homogeneous clusters are preserved together with heterogeneous clusters. The concept of the safety region was defined for extracting scattered instances. Birzhandi & Youn [7] developed the clustering-based convex hull (CBCH) algorithm, which follows the same strategy but uses the convex hull algorithm to extract the exterior instances of homogeneous clusters. To increase the reduction rate of SVMKM, Olvera-López et al. [34] proposed an algorithm called prototype selection by clustering (PSC), in which only few instances that define the decision boundaries are preserved in heterogeneous clusters instead of all instances. Simply put, a set of nearest instances to each instance of opposite classes is retained. In [39], critical clusters are selected based on their closeness to approximate decision boundaries. Furthermore, the data distribution in clusters is considered to find effective instances. In the method proposed by Chen et al. [11], instances of each class are clustered separately, and the distance of patterns to opposite-class cluster centers is used to extract informative points.

Other clustering techniques, apart from k-means, have also been used to develop instance selection methods. Cervantes et al. [8] developed a method by using a fuzzy clustering algorithm. In their method, the homogeneous clusters that directly contribute to the construction of approximate hyperplanes along with all the heterogeneous clusters are preserved. Cervantes et al. [9] introduced the concept of minimum enclosing ball for clustering and employed a strategy similar to SVMKM for selecting instances from clusters. Wang & Shi [43] developed a method, named sample reduction by data structure analysis (SR-DSA), by employing the Ward-linkage algorithm for clustering instances of different classes independently (i.e., each class is processed separately). In SR-DSA, after clustering, the interior instances of each cluster along with those that are far from opposite-class clusters are discarded as superfluous instances. The performance of clustering-based methods highly depends on the clustering parameters (e.g., the number of clusters and stopping criteria).

To avoid the need to predefine the number of clusters and the high computational complexity of clustering, López-Chau et al. [28] suggested using a decision tree to make partitions and guided random sampling to select instances from each

partition. Chang et al. [10] also proposed using a decision tree to decompose the original classification problem into several smaller problems. In fact, the decision tree decomposes the original feature space into homogeneous and heterogeneous areas, and thus, an independent SVM is trained on each heterogeneous region. In this manner, the computational complexity of the SVMs reduces as they are trained on smaller datasets. However, it has been noted in [26] that performance may not be satisfactory when dealing with non-convex data.

In some pattern selection techniques, called distance-based methods, border points are extracted by calculating the distance of each pattern to opposite-class instances. Euclidean, Mahalanobis, and Hausdorff are different metrics employed to measure distances [1,27,44]. The main drawback of distance-based methods is their high computational and memory complexity arising from the need to calculate distances between all samples.

Inspecting the neighborhood property of instances to identify support vector candidates is another strategy that has been explored in different studies. Neighborhood-based methods rely on the idea that border points tend to have heterogeneous neighbors, whereas interior ones do not. In accordance with this idea, Shin & Cho [41] proposed a neighborhood-based method called NPPS for selecting correctly labeled border instances. NPPS consists of three steps. First, the k-nearest neighbors (using the Euclidean distance) of the instances are identified. Second, the entropy value of each instance showing the homogeneity of its neighbors is computed. A positive entropy value indicates that the instance has heterogeneous neighbors and can thus be a support vector candidate. In the final step, noisy instances are excluded by considering the consistency of border instances with their neighbors. In crude terms, if the class of a border instance differs from that of its neighbors, it is likely to be noisy and is consequently removed. Li et al. [23] argued that only preserving border instances obtained by NPPS may lead to overfitting, which is why it is necessary to preserve some interior patterns to describe the extent of each class. They suggested preserving the cluster centroids selected by k-means as interior points in addition to the border instances selected by NPPS. In [45], the concept of entropy was used to develop a new active learning method for sample selection for SVMs. Along the same line of research, Li & Maguire [25] proposed a method called border-edge pattern selection (BEPS) that preserves edge instances along with border patterns. Edge instances lie on the extremes of a class region and define its extent. Edge and border patterns were extracted by using the concepts of approximated tangent hyperplanes and Bayes posterior probability, respectively. The experimental results show that BEPS is effective for different classifiers. Malhat et al. [31] proposed two algorithms that are generally similar to NPPS in terms of selecting border instances. However, they also preserve a few inner instances located in dense areas. Rico-Juan et al. [36] developed some neighborhood-based heuristics that allow sorting instances of a dataset according to their expected relevance in the classification task, which is then used to decimate the dataset to a specific number of the most relevant samples. Zhu et al. [49] proposed a new neighborhood-based heuristic named *cited count* for identifying border instances. The cited count is the number of times an instance is selected as the nearest neighbor of other-class instances. A cited count close to zero indicates that the instance is far from the instances of other classes. Thus, the instance is considered as an interior sample and is removed. The number of neighbors is a critical parameter in neighborhood-based methods.

Most of the above-mentioned methods are computationally expensive and face many difficulties in processing large datasets. To address this issue, Arnaiz-González et al. [4] used LSH to develop two methods that can handle large datasets. In these methods, one instance of each class is selected in each of the buckets constructed by LSH. More specifically, instances are hashed into buckets in such a way that closer samples have a higher probability of falling into the same bucket, and then one instance from each class in each bucket is randomly preserved. In this manner, similar instances are removed, while preserving the general structural properties of the original dataset. Aslani & Seipel [6] also proposed a new instance selection method by adopting LSH. Their method rapidly finds similar instances with a specific degree of similarity and removes them. Although the design of the methods in [4,6] is optimal and suitable for filtering instances in large datasets, they are not aimed at preserving border instances, making them susceptible to missing support vector candidates.

In this study, the ideas of LSH are employed in an effective manner to design a new method, *BPLSH*, aimed at quickly identifying border instances. It eliminates unnecessary patterns that do not contribute to classification to accelerate the training process of SVM without significantly degrading the original classification accuracy. In contrast to the methods in [4] that target similar instances, *BPLSH* removes interior instances and retains border ones to achieve high reduction rates. Moreover, unlike clustering and distance-based methods that use distance or density measures, *BPLSH* does not need to compute the distance between instances and is therefore suitable for handling large datasets.

The performance of *BPLSH* is benchmarked against those of NPPS, SVMKM, CBCH, and NoIS (in NoIS, all instances are used in the classification process) on eight publicly available datasets and one synthetically generated dataset. In addition, the instance selection methods are incorporated into the procedure of automatic building extraction to compare their performance and effectiveness in handling a big dataset. It is worth noting that Pareto optimality [17] is used to ensure that the comparisons are fair.

3. Locality-sensitive hashing: LSH

In this section, we briefly explain LSH, which we use in the study to develop our novel instance selection method. LSH is an effective and fast approach for approximate similarity search in massive datasets. It accelerates finding similar instances by hashing data into buckets to avoid comparing all pairs of instances.

A family of hash functions $g(x)$, which is the concatenation of multiple locality-preserving hash functions $(h_1(x), h_2(x), \ldots, h_M(x))$, is used to hash instances into buckets such that similar (close) instances fall into the same buckets with a high probability. In other words, g partitions the input space into buckets, and the id of each bucket is the concatenation of multiple locality-preserving hash functions. In this study, the following hash function $\left\{h : R^d \rightarrow N\right\}$, where N is the set of natural numbers to characterize the bucket ids, is used to hash each instance x [13]:

$$h_{a,b}(x) = \left\lfloor \frac{a \cdot x + b}{w} \right\rfloor \tag{1}$$

Eq. 1 maps a d-dimensional vector x onto an integer number. In this equation, a is a d-dimensional random vector with elements independently drawn from a Gaussian distribution with mean 0 and standard deviation 1, b is a real number drawn from $[0, w]$, and w controls the width of the buckets. A small value of w leads to a large number of buckets with a small width. For the sake of simplicity, all instances are normalized into $[0, 1]$, and the parameter w is set to 1. To increase the collision probability of neighbors and improve the hashing discriminative power, a set of hash function families G, rather than one hash function family, is constructed. Therefore, similar instances can be identified by simply examining buckets created by different hash function families (G) without needing to perform time-consuming pairwise distance calculations [15].

4. The proposed method

In this section, first, the steps of the proposed method and its pseudocode are elaborated. Then, the time complexity of the proposed method is explained. Finally, to understand how the input parameters affect the performance of *BPLSH*, a detailed analysis is performed using a synthetically generated dataset.

4.1. BPLSH

Let $T = \left\{(x_i, y_i) \,|\, x_i \in R^d, y_i \in \{1, 2, \ldots, p\}\right\}, i = 1, 2, \ldots, t$ be a training dataset, where x_i is a d-dimensional input feature vector, y_i is the corresponding class, t is the number of instances, and p is the number of classes. It is also assumed that each instance has only one class. *BPLSH* preserves border patterns as support vector candidates and a small number of interior instances as representatives of the extent of classes. *BPLSH* searches for border instances in heterogeneous areas (areas that contain instances from different classes). It relies on the idea that a border instance has heterogeneous *neighbors* and is the *nearest neighbor* of a pattern from another class. Generally, *BPLSH* consists of two major phases: 1- identifying the buckets of each sample, and 2- finding border samples and eliminating dispensable instances.

In the first phase, each instance is assigned to a set of buckets by using a group of hash function families $G = \{g_1, g_2, \ldots, g_L\}$, where L is the number of hash function families. Each hash function family $g(x)$ is the concatenation of M hash functions h. Each family of hash functions can be perceived as a layer of random hyperplanes that divides the input feature space into buckets. In other words, g projects instances from a d-dimensional real-valued feature space into an integer number that is a bucket id. Furthermore, the set of hash function families (G) can be perceived as a set of layers of hyperplanes that map an instance to an array of bucket ids. The projection is performed such that close instances have a high probability of being assigned into the same bucket.

Fig. 1a depicts an example of partitioning a 2-dimensional feature space by using three families of hash functions (three layers). In this example, each hash function family (g_i) contains 10 random hash functions (h). Instances a and b are assigned into the same buckets in all the layers, as shown in the figure.

To describe the second phase explicitly, a few definitions are presented.

Definition 1. The *similarity index* between two instances (*SimIndex*) is defined as the number of common buckets between them in all layers.

As depicted in Fig. 1c, the *similarity index* between instances a and b is three because they share the same buckets in all three layers.

Definition 2. *Neighbors* of a given instance (x_i) are the instances that share at least one bucket with x_i. In other words, an instance is a *neighbor* of x_i if its *similarity index* with x_i is greater than 0.

As shown in Fig. 1b, the *neighbors* of a are b, c, and d.

Definition 3. Two instances (x_i and x_j) are *quite close* if their *similarity index* is equal to L, where L is the number of layers.

In Fig. 1, instances a and b are *quite close* as their *similarity index* is equal to three.

Definition 4. The *nearest neighbors* of an instance (x_i) are the *neighbors* that have the maximum *similarity index* with x_i.

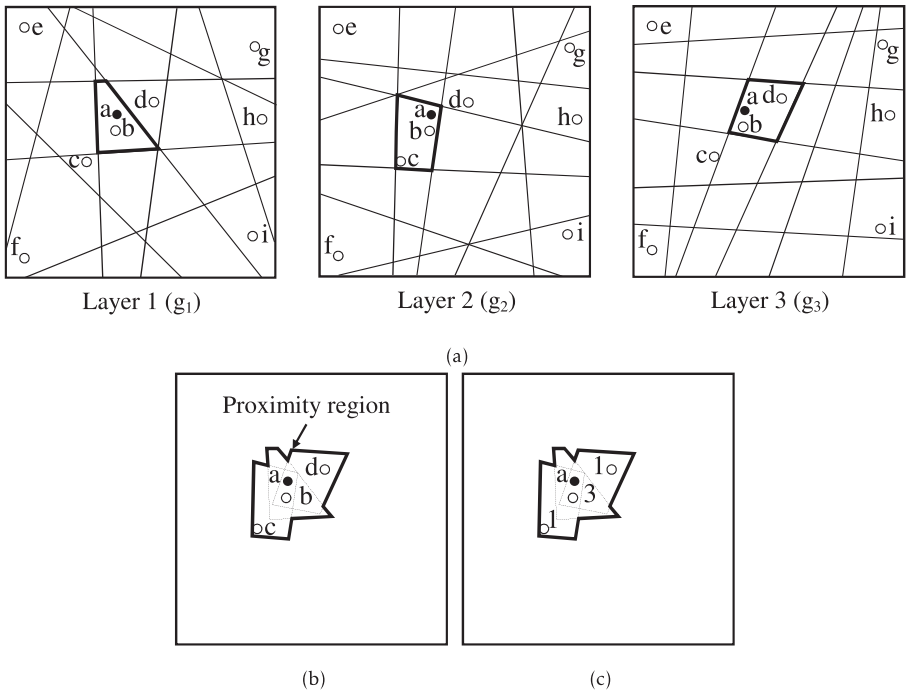

Fig. 1. (a) Partitioning a 2-dimensional feature space by using three hash function families (L = 3). Each hash function family consists of 10 hash functions (M = 10). (b) *Neighbors* of instance a and its proximity region. (c) *Similarity index* of instances b, c, and d with a.

In Fig. 1, b is the *nearest neighbor* of a, as it has the highest *similarity index* in comparison to c and d.
The second phase is composed of six minor steps summarized as follows:

I Extract the *neighbors* of sample x_i.
II Calculate the *similarity index* of the *neighbors*.
III If x_i and its *neighbors* are homogeneous (they belong to the same class), save x_i, remove the *neighbors* with *SimIndex* >=2 from the training dataset (T), and go to step VI.
IV If x_i and its *neighbors* are heterogeneous and the *nearest neighbors* of x_i from opposite classes are *quite close* to it, save x_i and the *nearest neighbors* from each opposite class and go to step VI.
V If x_i and its *neighbors* are heterogeneous and the *nearest neighbors* of x_i from opposite classes are not *quite close* to it, save the *nearest neighbors* from each opposite class, remove the *quite close neighbors* to x_i with class y_i from the training dataset (T), and go to step VI.
VI Repeat steps I to V until all instances in the training dataset (T) are either investigated or deleted.

In step III, *BPLSH* preserves only one representative instance from a homogeneous region because these regions are far from the decision boundaries and do not provide useful information for the classification. Furthermore, retaining many instances from these regions simply increases the size of the dataset without significantly improving the classification accuracy. In this step, *neighbors* of x_i are also eliminated from the training dataset to prevent *BPLSH* from investigating them in subsequent iterations and accordingly accelerate the acquisition of border instances. However, removing all the *neighbors* with *SimIndex* >= 1 might be problematic because some of the outer *neighbors* might be border instances (Fig. 2). Thus, the *neighbors* with *SimIndex* >= 2 are removed.

Fig. 2 illustrates how instance selection for x_i is performed in step III (in only one iteration). This figure shows x_i and its surrounding instances along with their *similarity index* values. As the classes of x_i and the *neighbors* are identical (class 1), this is a homogeneous region. Therefore, x_i is the only instance preserved, and the *neighbors* with *SimIndex* >= 2 are removed. The *neighbors* with *SimIndex* = 1 are left and will be investigated in subsequent iterations in case they contain some border instances.

Algorithm 1: BPLSH, the proposed instance selection algorithm

Input: A training dataset $T = \left\{(x_i, y_i) \,|\, x_i \in R^d, y_i \in \{1, 2, \ldots, p\}\right\}, i = 1, 2, \ldots, t$
A set of hash function families $G = \{g_1, g_2, \ldots, g_L\}$, such that
$g(x \in T) = (h_1(x), h_2(x), \ldots, h_M(x))$ and h_i is a hash function $\left(h : R^d \rightarrow N\right)$
Output: The set of selected instances $(SI \subseteq T)$

```
1: SI, b_x^g, b_bid^g ← ∅
2: for each (x, y) ∈ T do
3:    i ← index of (x, y)
4:    for each function family g ∈ G do
5:       bid ← bucket id assigned to x by g
6:       Add bid to b_i^g
7:       Add (x, y) to b_bid^g
8:    end for
9: end for
10: for each (x, y) ∈ T do
11:    i ← index of (x, y)
12:    Neighbors ← ∅
13:    for each function family g ∈ G do
14:       bid ← b_i^g
15:       I ← all instances in b_bid^g except x
16:       Add I to Neighbors
17:    end for
18:    UNeighbors ← unique elements of Neighbors
19:    SimIndex ← similarity index of UNeighbors to x
20:    C_N ← Classes of UNeighbors
21:    C_A ← {y, C_N}
22:    if C_A is homogeneous then
23:       Remove UNeighbors with SimIndex >=2 from T
24:       Add (x, y) to SI
25:    else if C_A is heterogeneous then
26:       MS ←0
27:       for each C ∈ C_N except y do
28:          N_C ← UNeighbors whose class = C
29:          SimIndex_C ← SimIndex of N_C
30:          M ← max(SimIndex_c)
31:          SN ← N_C with SimIndex_C = M
32:          Add SN to SI
33:          MS ← max(MS, M)
34:       end for
35:       if MS is equal to L then
36:          Add (x, y) to SI
37:       else
38:          SR ← UNeighbors whose class = y & SimIndex = L
39:          Remove SR from T
40:       end if
41:    end if
42: end for
```

Steps IV and V aim to identify border instances that exist in heterogeneous proximity regions. In the case of heterogeneity, both steps IV and V save the *nearest neighbors* from each opposite class ($y \neq y_i$) as border instances (Figs. 3 and 4). The only difference between steps IV and V is the closeness of x_i to the border instances. If x_i is *quite close* to the border instances, it is preserved as well (Step IV). Otherwise, *quite close neighbors* are regarded as interior-redundant instances and are removed (Step V). Figs. 3 and 4 illustrate the difference between steps IV and V.

The pseudocode of *BPLSH* is presented in Algorithm 1. The inputs of the algorithm are a training dataset (T) and a set of hash function families (G). The output of the algorithm is a subset of the selected instances. Lines 2–9 correspond to the first phase previously mentioned, that is, the bucket id of each instance in each layer is independently determined and stored. The

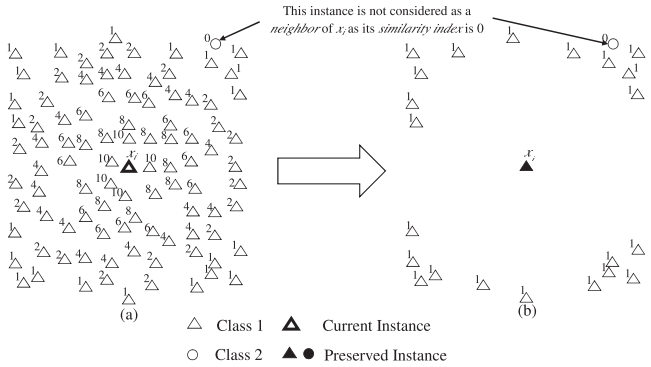

Fig. 2. Example of a homogeneous case for step III. In this example, L is 10. (a) x_i, its surrounding instances, and their *similarity index* values. x_i and its *neighbors* belong to the same class. (b) Because of the homogeneity of x_i and its *neighbors*, x_i is retained and the *neighbors* with *SimIndex* >= 2 are considered as interior-redundant instances and removed from the dataset.

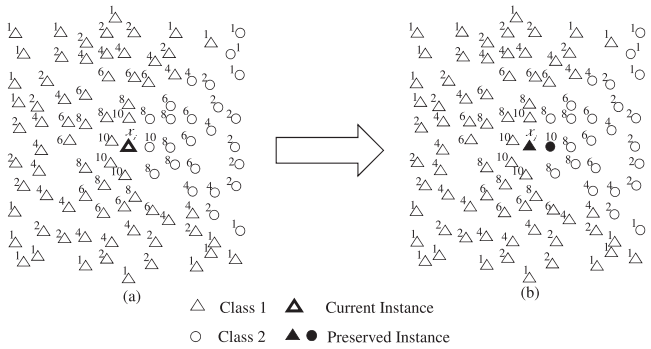

Fig. 3. Example of a heterogeneous case for step IV (L is 10). (a) The instances are heterogeneous, and thus, there are some instances that may form the decision boundary. (b) The *nearest neighbors* that belong to the opposite class are preserved as border instances. As x_i is *quite close* to the *nearest neighbor* from the opposite class (*SimIndex* = L), it is also preserved as a border instance.

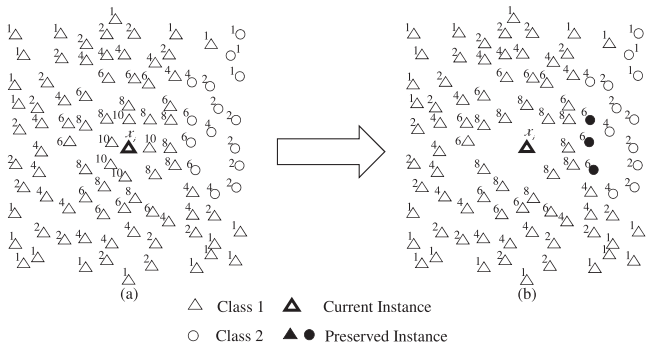

Fig. 4. Example of a heterogeneous case for step V (L is 10). (a) Heterogeneous instances. (b) The *nearest neighbors* belonging to the opposite classes are preserved. As x_i is not *quite close* to the opposite class (its *similarity index* is less than L), almost identical instances to the current instance (instances that belong to class y_i and are *quite close* to x_i) are considered as interior-redundant ones and are removed from the training dataset.

time complexity of the first phase is $O(n \times d \times M \times L)$, where n is the number of samples, d is the number of features of the dataset, M is the number of hash functions (hyperplanes) in each layer, and L is the number of layers.

The second loop of the algorithm (lines 10–42) corresponds to the second phase. Unlike the first phase, all variables have an integer data type. The time complexity of this step depends on the distribution of instances and their class diversity, in

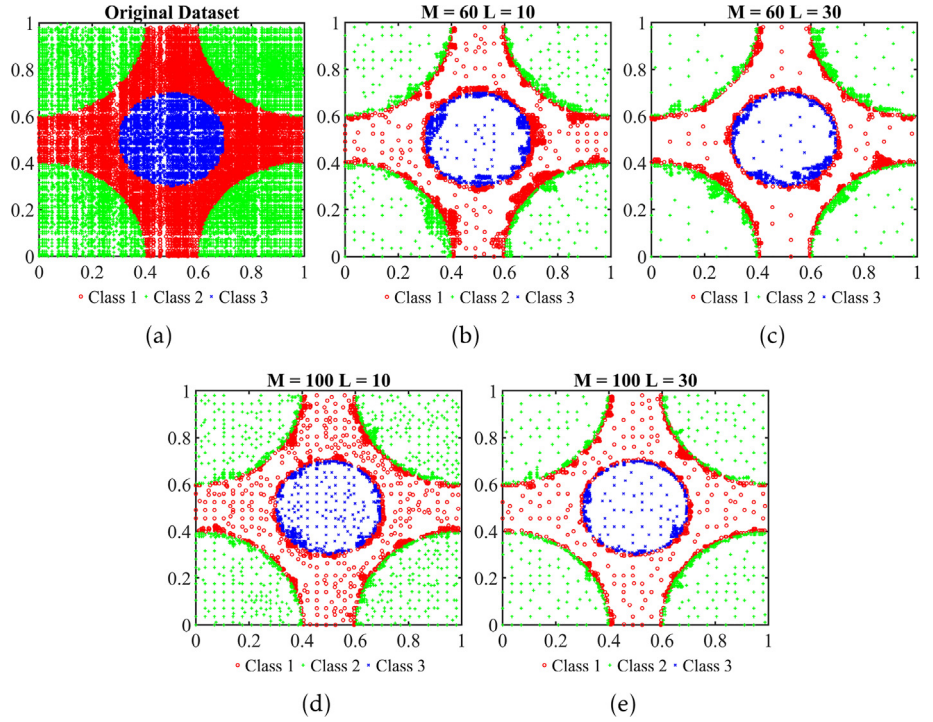

Fig. 5. Synthetic dataset and refined subsets for different values of *M* and *L*. BPLSH first hashes all the instances into buckets regardless of their labels and is based on only their closeness. In the second phase, border instances are identified by considering both the *similarity index* and the labels of instances. The number of buckets is a function of *M* and *L*.

addition to the number of instances. If all instances are identified as interior ones, *BPLSH* will not need to check all of them, significantly reducing the time complexity. In contrast, if all instances are recognized as border instances and no instance is removed, the time complexity will increase. The lower bound of the time complexity is $O(n)$, when *neighbors* of all instances are homogeneous, and the upper bound is $O(n \times \overline{N})$, when all instances lie on the boundaries of classes. $\overline{N}$ is the average number of *neighbors* for each instance.

4.2. Parameter analyses of BPLSH

To provide a better understanding of the behavior of the proposed method, this section illustrates how its input parameters, namely the number of hash functions (M) and hash function families (L), affect the resulting subsets. To do so, a synthetic two-dimensional dataset that contains 30,141 instances with three classes is produced (Fig. 5a), and *BPLSH* is run for different values of *M* and *L*. The implementation was run in MATLAB 2019[1] on a computer with an Intel(R) CoreTM i7-8700 CPU @ 3.6 GHz and 32 GB RAM.

Figs. 5b-5e show the refined subsets for $M \in \{60, 100\}$ and $L \in \{10, 30\}$. It is axiomatic that *BPLSH* has mainly eliminated the interior-redundant instances and preserved the patterns located near the decision boundaries. The buffer of the border instances narrows and the number of internal samples rises as *M* increases from 60 to 100. This is because bucket sizes and proximity regions decrease as *M* grows. Furthermore, the number of internal instances decreases when *L* increases from 10 to 30 because homogeneous proximity regions that filter out interior instances become bigger and the similarity index of instances in the homogeneous proximity regions increase as *L* grows. It is noteworthy that if *M* is set to a very large value, the proximity region's size and the average number of *neighbors* of instances significantly diminishes; thus, *BPLSH* will obtain a microscopic view (very local view) over patterns, and its ability to distinguish border instances from internal ones substantially reduces.

Tables 1 and 2 indicate how the input parameters (*M* and *L*) influence the preservation rate (i.e., the fraction of the select instances to the whole dataset) and execution time. There is a visible trend toward a lower preservation rate for an increase in *M* and *L*. Additionally, the execution time of *BPLSH* on the dataset decreases as *M* grows. This is because when *M* increases, bucket sizes decrease, the number of instances with the homogeneous proximity region rises, and thus fewer instances need to be evaluated to find border samples.

[1] The source code of *BPLSH* can be found in https://github.com/mohaslani/BPLSH.

Table 1
Preservation rate (%).

Number of hash functions families (L)	Number of hash functions (M)		
	20	60	100
10	40.52	18.38	13.61
20	37.16	17.04	11.65
30	33.56	16.13	10.78

Table 2
Execution time (sec.).

Number of hash functions families (L)	Number of hash functions (M)		
	20	60	100
10	2.12	1.64	1.56
20	5.52	3.20	2.71
30	9.48	4.76	3.91

5. Experimental study 1: publicly available datasets

In this section, the effectiveness of *BPLSH* is verified by comparing it with other methods on different datasets. To this end, the characteristics of the utilized datasets are first explained. Then, the performance of *BPLSH* for different values of M and L is presented. Finally, *BPLSH* is benchmarked against three representative instance selection methods, namely NPPS [41], SVMKM [3], and CBCH [7], in terms of preservation rate, classification error, and execution time. It must be highlighted that the performance of the SVM on the original training sets (NoIS, no instance selection method is used) is also presented to provide a better comparison.

As the number of selected instances is a dependent variable in *BPLSH*, NPPS, SVMKM, and CBCH, it must be assessed together with classification accuracy at the same size. Therefore, the comparative analysis is performed using the Pareto optimality concept [17] and the normalized Euclidean distance to the ideal point [29]. The experiments are conducted on a computer with an Intel(R) CoreTM i7-8700 CPU @ 3.7 GHz and 32 GB RAM in MATLAB 2019.

5.1. Datasets

The experiments are performed on some benchmark datasets from the UCI[2] and KEEL[3] repositories, as well as on a simulated dataset. Table 3 lists the number of instances of the datasets along with the name of the resource that each has obtained. All the datasets contain more than 1,000 instances; the last one (Skin Segmentation) is substantial as it has over 200,000 samples. We believe that these datasets are representative of a range of classification problems with regard to scale and provide a comprehensive evaluation of the methods.

5.2. Results and discussion

For an unbiased and reliable evaluation of the instance selection methods, a repeated stratified q-fold cross-validation scheme is used. In this scheme, a given dataset is partitioned into q exclusive folds, and each time, q-1 folds (training set) are utilized to train the SVM after an instance selection method is applied to them. The remaining fold is used as a test set to estimate the predictive performance of the SVM. Each fold is employed once as a test set, and the entire process is repeated r times to reduce the influence of partitioning the datasets. The average over folds and repetitions is reported as the final values of the aforementioned measures. Because of the r repetitions, this scheme yields lower variance than the non-repeated version. In the experiments, q and r are set to 10 and 5, respectively.

Two measures for estimating the performance of the instance selection methods, the classification error on the test set and preservation rate, are calculated according to Eqs. 2 and 3, respectively. In these equations, N_{TE} is the number of instances in the test set, N_{CTE} is the number of correctly classified instances of the test set by the SVM, N_{TR} is the total number of training instances, and N_{STR} is the number of training instances that are selected.

[2] https://archive.ics.uci.edu/.
[3] http://keel.es/.

Table 3
Datasets.

Datasets	# Instances	Repository
CMC	1,473	UCI
Banana	5,300	Keel
Indoor Movement Prediction	13,197	UCI
Synthetic Dataset	14,000	–
HTRU2	17,898	UCI
Magic Gamma Telescope	19,020	Keel
Credit Card	30,000	UCI
Electricity Board	45,781	UCI
Skin Segmentation	245,057	UCI

Table 4
Optimized values of hyperparameters obtained by Bayesian optimization.

Datasets	Kernel Function	Kernel Scale	Box Constraint
CMC	Gaussian	1.6	24.1
Banana	Gaussian	0.2	4.3
Indoor Movement Prediction	Gaussian	0.2	1.1
Synthetic Dataset	Gaussian	1.8	292.4
HTRU2	Gaussian	0.3	21.3
Magic Gamma Telescope	Gaussian	0.6	925.1
Credit Card	Gaussian	1.0	10.6
Electricity Board	Gaussian	0.05	8.3
Skin Segmentation	Gaussian	0.04	15.2

$$Classification\ error = 100 - \frac{N_{CTE}}{N_{TE}} \times 100 \tag{2}$$

$$Preservation\ rate = \frac{N_{STR}}{N_{TR}} \times 100 \tag{3}$$

To achieve the best performance of SVMs on the datasets, their hyperparameters, namely the kernel function (Gaussian, polynomial, and linear), kernel parameter, and box constraint, are optimized via Bayesian optimization [38]. Table 4 lists the optimized values of the hyperparameters. Gaussian $k(x_i, x_j) = exp\left(-\frac{||x_i - x_j||^2}{\gamma^2}\right)$ is the best kernel function in all datasets, where x_i and x_j are the i^{th} and j^{th} training samples and γ is a kernel scale. It must be underlined that the same optimal hyperparameters are used in all the instance selection methods to ensure that final results are not affected by the hyperparameter values.

As *BPLSH* performance is a function of both the input parameters (M and L) and the dataset, they have no unique best values in all the datasets. Therefore, 18 different configurations for M and L, where $M \in \{10, 30, 50, 70, 90, 110\}$ and $L \in \{10, 20, 30\}$ are evaluated to present the sensitivity of *BPLSH* to the input parameters and to find their optimal values in each dataset. These configurations are chosen by considering the limited available computational resources and to prevent creating excessive buckets. The average preservation rate and classification error for the configurations are shown in Figs. 6 and 7, respectively. The original classification error of SVM without using any instance selection method (NoIS) is presented in Fig. 7.

The following points can be made based on Figs. 6 and 7. In terms of maintaining the original classification accuracy, *BPLSH* can achieve data reduction without substantial loss of the discriminatory power of the SVMs. In most datasets (CMC, Banana, Indoor Movement Prediction, HTRU2, Magic Gamma Telescope, Credit Card, and Electricity Board), the preservation rate decreases as L increases. Moreover, the classification error arises from the increase in L for small values of M. This is because the buffer of border instances narrows and the number of preserved internal instances significantly decreases when L grows, which is visible when comparing Figs. 5b-5c or Figs. 5d-5e. Therefore, the chance of losing border instances and damaging the extent of each class increases. This is particularly applicable for $M = 10$ in Skin Segmentation. Moreover, in all the datasets except Banana, the classification error is almost constant over a wide range of M, i.e., setting the parameter M is fairly easy.

We conduct a comparative analysis of *BPLSH* with NPPS, SVMKM, CBCH, and NoIS. NPPS, which is based on k-nearest neighbors (using Euclidean distance), has an input parameter of the number of neighbors (k). SVMKM and CBCH, which select informative instances via k-means clustering, have the number of clusters (c in SVMKM and p in CBCH) as their input parameters. The parameters k, c, and p significantly affect the performance of their corresponding methods, and there is no unique optimal configuration for them in all the datasets. Therefore, to conduct a fair comparison and ensure that (near) optimal performance of NPPS, SVMKM, and CBCH are achieved, they are run using different values of $k \in \{10, 20, 50, 100, 200, 500, 1000\}, c \in \{5, 10, 15, 20, 30, 50, 100, 200, 300, 500, 1000\}$, and $p \in \{5, 10, 20, 30, 50, 100\}$. Fur-

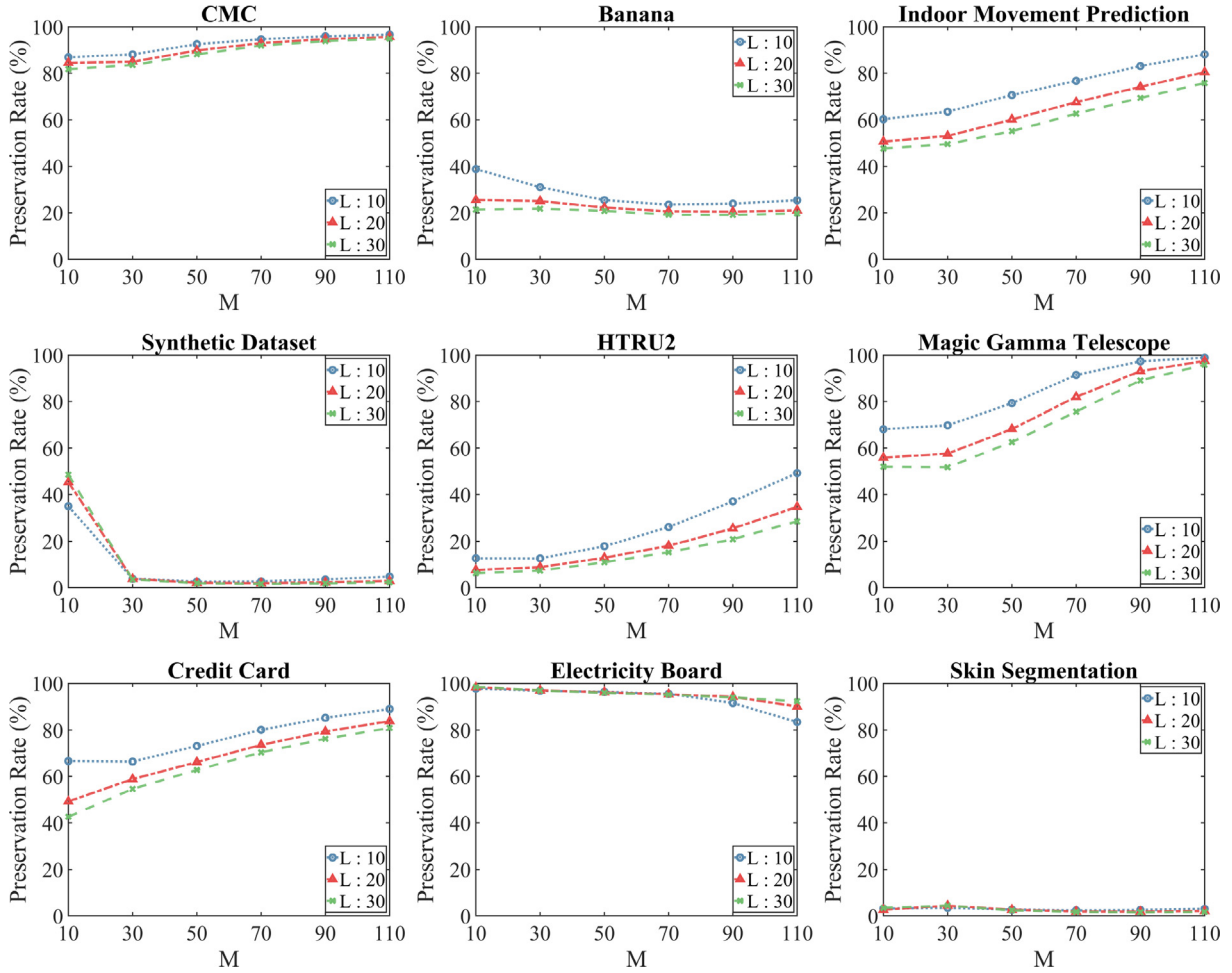

Fig. 6. Comparison of preservation rate for different configurations of BPLSH over the datasets.

thermore, the comparison is made by simultaneously considering the classification error and preservation rate; this is done because an ideal instance selection method should yield the lowest error while retaining the lowest number of instances. It should be noted that k-means++ [5], as one of the most optimal variants of the k-means algorithm, is employed in SVMKM and CBCH. Fig. 8 shows a comparison between *BPLSH*, NPPS, SVMKM, CBCH, and NoIS for different values of the input parameters over the datasets.

The figure indicates that *BPLSH* obtains the lowest preservation rate in comparison with NPPS, SVMKM, and CBCH in most of the datasets (CMC, Banana, Indoor Movement Prediction, HTRU2, Magic Gamma Telescope, Credit Card, and Electricity Board). NPPS does not yield convincing classification errors in the Synthetic and Skin Segmentation datasets, although it results in the lowest preservation rate on average. In addition, none of the methods induce a low preservation rate (less than 50%) in the CMC, Magic Gamma Telescope, and Electricity Board datasets. This is because many instances in these datasets are located near the decision boundaries. Overall, the classification error is clearly less sensitive to the input parameters than the preservation rate.

The main issue that arises in Fig. 8 is identifying which method (and with what configuration) obtains the best performance in each dataset in terms of both the preservation rate and classification error. Selecting the most effective method and configuration can be regarded as a multi-objective problem because there are two competing objectives (classification error and preservation rate) that should be minimized simultaneously [17]. Multi-objective problems usually involve a set of optimal solutions, rather than a single one, that reflect various trade-offs among competing objectives. Such a set (best-performing methods and configurations), which is referred to as the Pareto-optimal set, is determined via dominance relations. h^* is a nondominated solution if and only if there does not exist another solution h such that $objective_i(h) \leqslant objective_i(h^*)$ for all i and $objective_i(h) < objective_i(h^*)$ for at least one i. In other words, nondominated (Pareto-optimal) solutions cannot be improved in one objective without being worsened in another.

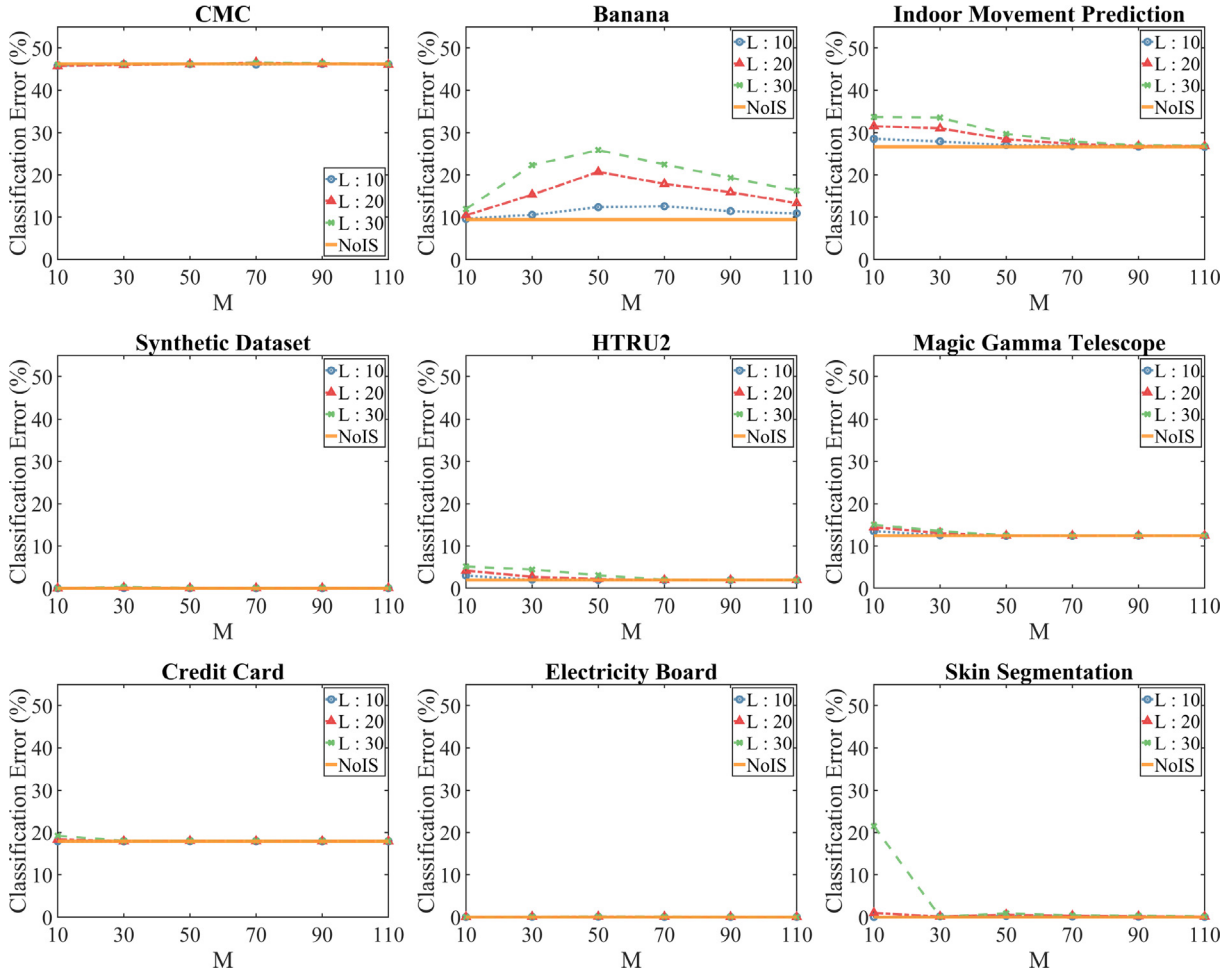

Fig. 7. Comparison of classification errors for different configurations of BPLSH over the datasets.

The Pareto-optimal methods and configurations are shown in Fig. 9. From a comparison of Figs. 8 and 9, more than 50% of the alternatives are filtered out as non-optimal ones by using the Pareto optimality concept. This simplifies the evaluation of the methods and configurations. The following points can be observed in Fig. 9:

- *BPLSH* significantly contributes to the Pareto-optimal sets in all the datasets. This shows its ability to adequately balance between the preservation rate and classification error.
- *BPLSH* has outstanding performance in HTRU2 as it completely dominates other methods.
- CBCH is dominated by other methods in most datasets (CMC, Banana, Indoor Movement Prediction, HTRU2, Magic Gamma Telescope, Credit Card, and Electricity Board), showing its inferior performance in comparison to other methods.
- *BPLSH* dominates NoIS in HTRU2, Synthetic, and Magic Gamma Telescope.

Although using the Pareto optimality concept has eliminated many non-optimal alternatives and simplified comparisons, it cannot indicate the best single method and configuration. To perform a comprehensive comparison of the optimal alternatives and identify a final preferred selection for each dataset, the Euclidean distance between each solution of the Pareto-optimal set and the ideal case is employed as a measure [29]. The ideal case has a preservation rate and classification error of almost zero. The closer the point is to the ideal case, the more preferred it is. The performances of the three best points (nearest to the ideal point) for each dataset are presented in Table 5. It is clear that *BPLSH* is the leader for all the datasets except for Synthetic, obtaining the best trade-off between classification error and preservation rate. In Banana, Indoor Movement Prediction, HTRU2, Credit Card, Electricity Board, and Skin Segmentation, the three best points are obtained by *BPLSH*, proving its remarkable performance.

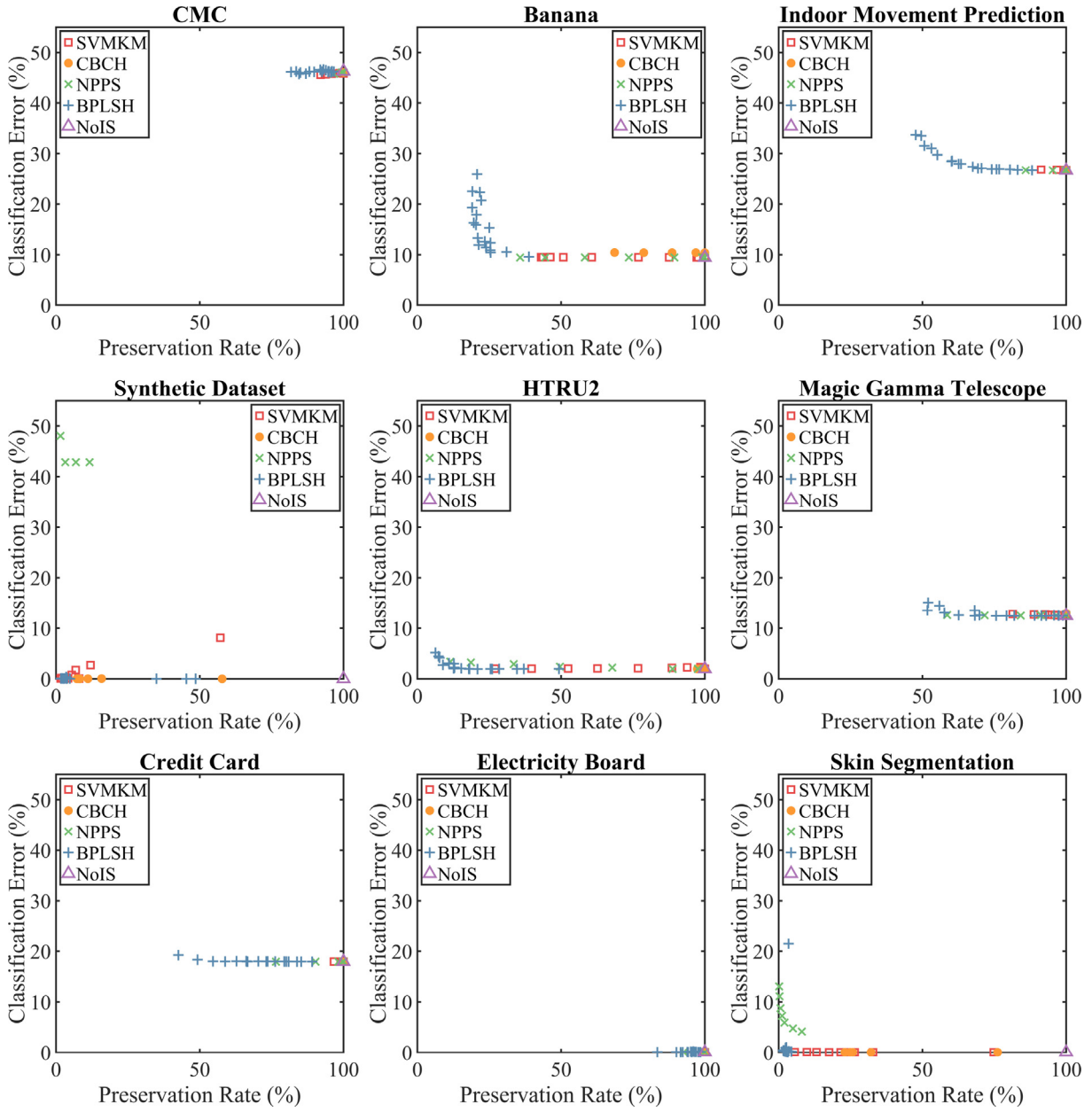

Fig. 8. Performances of SVMKM, CBCH, NPPS, BPLSH, and NoIS in terms of the classification error and preservation rate.

Table 6 compares the total execution times of the methods for the best input values (near the ideal case). The total execution time is the sum of the time required for instance selection and training the SVM. For NoIS, the total execution time encompasses only the training runtime, as it does not contain any instance selection method and all the training instances are contributed in the training phase. As is evident from the table, for most of the datasets, *BPLSH* demonstrates superiority over the other methods. The results in Tables 5 and 6 support the claim that *BPLSH* is more effective than NPPS, SVMKM, CBCH, and NoIS on the datasets.

6. Experimental study 2: a dataset of building extraction from the fusion of high-resolution aerial images and point clouds

This experiment aims to examine the performance of the proposed method in handling a large dataset. To do so, the proposed method is incorporated into the procedure of automatic building extraction from high-resolution aerial images. Pre-

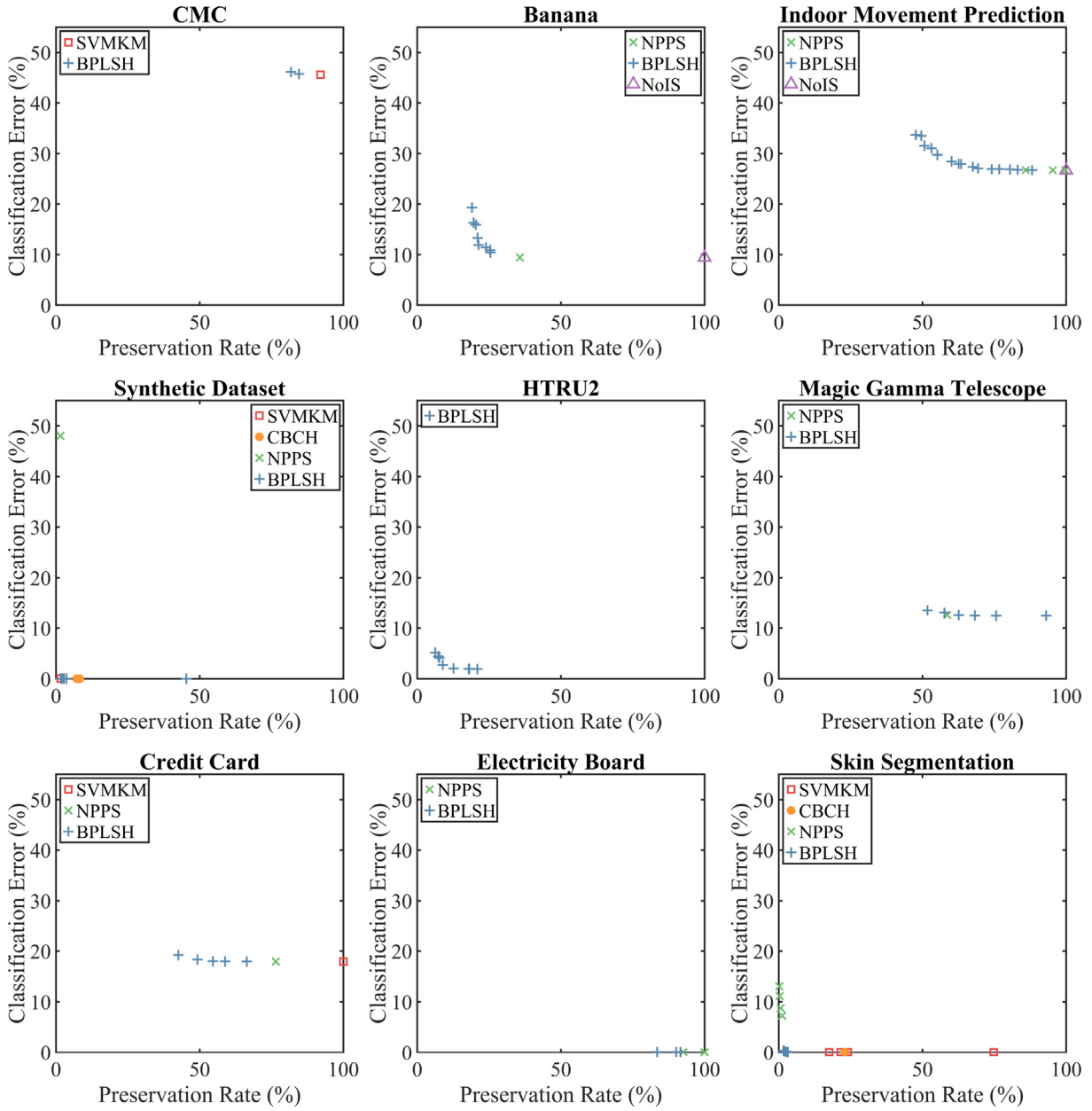

Fig. 9. Pareto-optimal sets obtained by regarding both the preservation rate and classification error.

viously, significant effort was devoted to building extraction by using different data resources in photogrammetry and computer vision communities owing to the importance of building maps for a variety of geospatial applications [24,21].

The methods developed for building extraction can be categorized into three classes in terms of the data resources: 1- optical imagery data, e.g., high-resolution aerial images [14]; 2- active imagery data, e.g., LiDAR [16]; and 3- fusion of optical and active imagery data [20]. In this experiment, the fusion of high-resolution aerial images and LiDAR is used to compensate for the weakness of each data source with the strength of the other [47]. Among data fusion-based methods, pixel-based image classification by using supervised classifiers, particularly SVMs, is a common approach for building extraction [32]. The accuracy of pixel-based building extraction derived by SVMs is attributed to a multitude of factors, and instance selection is of particular importance. Indeed, the generalization ability and accuracy of SVMs for predicting unseen pixels in test areas are attributable to the training instances used.

Table 5
Comparing the performances of the three best points.

Datasets	Rank	Method and its configuration	Classification error (%)	Preservation rate (%)
CMC	1th	BPLSH (M = 10, L = 30)	46.111	81.762
	2th	BPLSH (M = 10, L = 20)	45.712	84.534
	3th	SVMKM (C = 500)	45.565	92.032
Banana	1th	BPLSH (M = 10, L = 30)	11.917	21.326
	2th	BPLSH (M = 110, L = 20)	13.304	21.012
	3th	BPLSH (M = 110, L = 30)	16.285	19.613
Indoor Movement Prediction	1th	BPLSH (M = 10, L = 30)	33.695	47.667
	2th	BPLSH (M = 10, L = 20)	31.549	50.669
	3th	BPLSH (M = 30, L = 30)	33.531	49.587
Synthetic Dataset	1th	SVMKM (C = 100)	0.086	1.61
	2th	BPLSH (M = 70, L = 30)	0.061	1.694
	3th	BPLSH (M = 70, L = 20)	0.049	1.96
HTRU2	1th	BPLSH (M = 10, L = 30)	5.164	6.292
	2th	BPLSH (M = 30, L = 30)	4.455	7.359
	3th	BPLSH (M = 10, L = 20)	4.157	7.679
Magic Gamma Telescope	1th	BPLSH (M = 30, L = 30)	13.536	51.750
	2th	BPLSH (M = 30, L = 20)	13.079	57.639
	3th	NPPS (K = 10)	12.593	58.602
Credit Card	1th	BPLSH (M = 10, L = 30)	19.232	42.568
	2th	BPLSH (M = 10, L = 20)	18.329	49.218
	3th	BPLSH (M = 30, L = 30)	18.004	54.597
Electricity Board	1th	BPLSH (M = 110, L = 10)	0.077	83.485
	2th	BPLSH (M = 110, L = 20)	0.071	90.115
	3th	BPLSH (M = 90, L = 10)	0.067	91.602
Skin Segmentation	1th	BPLSH (M = 90, L = 30)	0.374	1.615
	2th	BPLSH (M = 110, L = 30)	0.226	1.769
	3th	BPLSH (M = 110, L = 20)	0.163	2.089

Table 6
Total execution time (instance selection + training) in seconds.

Datasets	NPPS	SVMKM	CBCH	NoIS	BPLSH
CMC	0.33	0.66	0.26	**0.25**	0.39
Banana	0.35	0.34	0.36	**0.32**	0.58
Indoor Movement Prediction	4.29	4.13	4.14	4.25	**4.07**
Synthetic Dataset	0.74	0.85	0.40	0.69	**0.38**
HTRU2	6.66	6.18	6.30	6.38	**3.17**
Magic Gamma Telescope	240	258	261	263	**172**
Credit Card	41	98	98	99	**30**
Electricity Board	279	278	284	283	**179**
Skin Segmentation	64.0	14.07	**8.18**	86.32	28.57

In this context, including all the pixels in the training parts of aerial images can reduce the risk of misclassification and more appropriately characterize the spectral responses of classes. However, this results in a huge number of unnecessary and redundant training samples, which increases the computational complexity of the training phase of SVMs. This is particularly true for high-resolution aerial images due to the large number of pixels and spectral correlations between neighboring pixels.

Many studies on remote sensing have investigated the effect of instance selection on the performance of different classifiers. However, most of them have only considered the number of training instances and their spatial diversity in the image space as the key factors for instance selection. For example, in [12], instances were selected by considering the area and/or distribution of each class in the image space. In [35], a new filter was proposed to ensure the spatial diversity of instances. The proposed filter eliminates pixels between classes as they may be incorrectly labeled and retains at least one pixel in each group of neighboring pixels. Although the spatial diversity of instances in the image space can describe the variability of spectral signatures of classes, the resultant samples may be ineffective for classification as the hyperplanes of demarcation between classes will be formed in the feature space, rather than the image space, by using border instances [18]. Therefore, in this experiment, *BPLSH* is used to identify the informative instances (pixels) in the feature space of a large training dataset obtained from the fusion of high-resolution aerial images and point clouds.

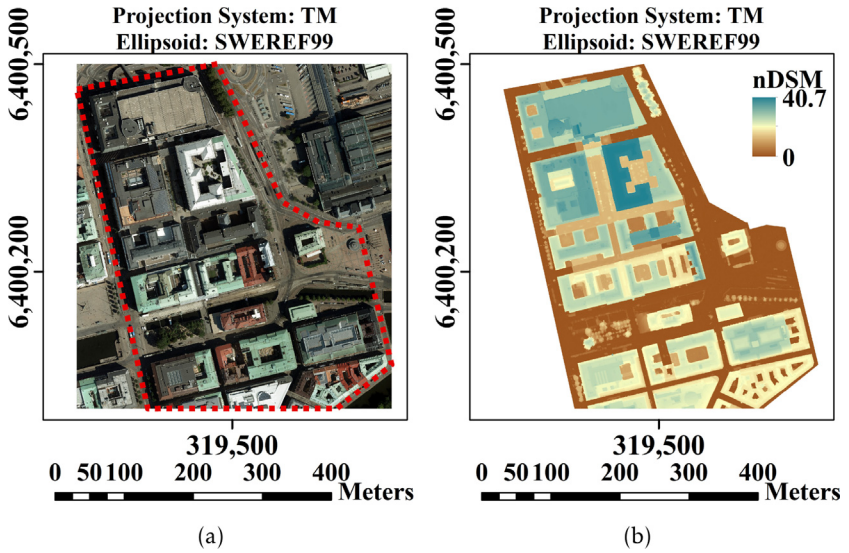

Fig. 10. (a) True-Orthophoto and boundary of the study area, (b) nDSM.

6.1. Dataset preparation

The data employed include the True-Orthophoto (in four bands: near-infrared, red, green, and blue), an image-derived digital surface model (DSM) cloud, LiDAR data, and the footprint of buildings of a small part of downtown Gothenburg. Fig. 10a shows the True-Orthophoto and the boundary of the study area. All the data are georeferenced in SWEREF99TM and provided by the Swedish mapping, cadastral, and land registration authority[4]. The sampling density of the LiDAR data is 1 point per square meter, and the spatial resolution of both the True-Orthophoto and image-derived DSM is 10 cm. The image-derived DSM is enhanced by LiDAR data to ensure that it does not contain any gaps.

Several features have been proposed for training SVMs for building extraction. In this experiment, the five features used are the normalized digital surface model (nDSM), gradient magnitude of the DSM, second spatial derivative (Laplacian) of the DSM, terrain roughness, and normalized difference vegetation index (NDVI). The nDSM (Fig. 10b), which indicates the height of each pixel from the ground, is obtained by subtracting the DSM from the digital terrain model (DTM). The DTM itself is generated by utilizing a progressive morphological filter on the DSM [48]. The gradient magnitude of the DSM, which measures local height changes, is calculated using Eq. 4, where $\frac{\partial F}{\partial x}$ and $\frac{\partial F}{\partial y}$ are gradients in the x and y directions, respectively. The Sobel operator, as a discrete differential operator, is used to approximate the gradients in the x and y directions.

$$Gradient\ Magnitude = \sqrt{\left(\frac{\partial F}{\partial x}\right)^2 + \left(\frac{\partial F}{\partial y}\right)^2} \tag{4}$$

The second derivative of the elevation model, which enhances sharp changes of height values, is estimated by using a Laplacian filter with a kernel size of 3×3 over the DSM. Terrain roughness, as a quantitative index that characterizes the heterogeneity in the elevation model, is measured by the method proposed in [37]. In this method, the terrain roughness index is calculated by averaging the change in elevation between a cell and its neighbors. The NDVI is a widely used remote sensing index to estimate vegetation growth and biomass. It is calculated using Eq. 5, in which NIR and R denote the reflectance of the near-infrared and red bands, respectively. A high positive value of the NDVI indicates a high amount of green vegetation.

$$NDVI = \frac{NIR - R}{NIR + R} \tag{5}$$

The normalized values of the aforementioned features of the pixels along with the pixel labels extracted from the reference data (footprint of the buildings) form the training dataset. The size of the produced training dataset is approximately 1,100,000 samples, with five input features and one output feature (building/non-building).

[4] https://www.lantmateriet.se/.

6.2. Results and discussion

Analogous to experimental study 1 in Section 5, all the results are achieved through the repeated stratified k-fold cross-validation scheme in MATLAB 2019 on a computer with an Intel(R) CoreTM i7-8700 CPU @ 3.7 GHz and 32 GB RAM. Furthermore, the optimized hyperparameters of the SVM obtained by the Bayesian optimization method are used. The optimized values for the kernel function, kernel scale, and box constraint, respectively, are Gaussian, 0.6, and 612. In this experiment, *BPLSH* is run for $M \in \{10, 30, 50, 70, 90, 110, 130, 150\}$ and $L \in \{10, 20, 30, 40, 50\}$.

Fig. 11 shows the effects of M and L on the preservation rate and classification error. It is obvious that *BPLSH* can significantly reduce the data by over 85%. The minimum preservation rate is achieved for $M = 90$ or 110. The classification error is almost constant over a wide range of M and L; setting input parameters is not difficult. Fig. 12a compares the performances of *BPLSH*, SVMKM, CBCH, and NPPS for different values of input parameters by simultaneously considering the preservation rate and classification error. Parameter k of NPPS is set to 10, 20, 50, 100, 200, 500, and 1000, parameter c of SVMKM is set to 5, 10, 15, 20, 30, 50, 100, 200, 300, 500, and 1000, and parameter p of CBCH is set to 5, 10, 20, 30, 50, and 100. As shown in the figure, NPPS and SVMKM yield the highest classification error and preservation rate, respectively. In contrast, *BPLSH* appropriately balances the two measures. Fig. 12b shows the Pareto-optimal sets. It is seen that *BPLSH* can dominate SVMKM, CBCH, and NoIS, exhibiting its good performance and its ability to reduce the size of the dataset without affecting the original classification accuracy (NoIS). In addition, it reaches the minimum classification error for the selection ratio of 2.8%. Table 7 compares the performance of the Pareto-optimal points with one another and NoIS. Obviously, *BPLSH* (M = 110, L = 50) can maintain the original classification error while significantly reducing instances.

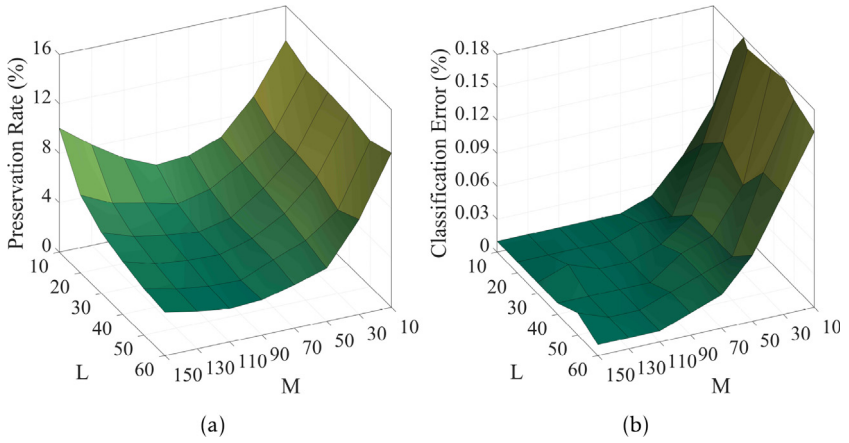

Fig. 11. Performance of BPLSH in terms of (a) Preservation rate and (b) Classification error.

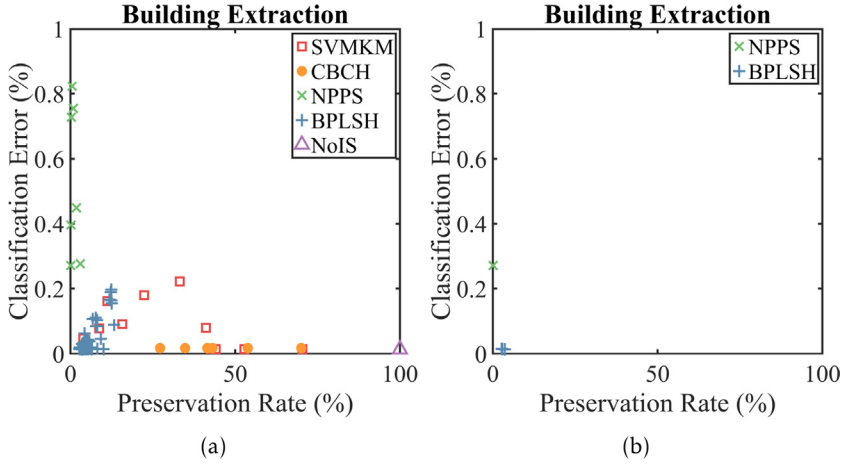

Fig. 12. (a) Comparing SVMKM, CBCH, NPPS, BPLSH, and NoIS for different values of input parameters, (b) The Pareto-optimal sets.

Table 7
Comparing the performances of the three best points with one another and NoIS.

Method and its configuration	Classification error (%)	Preservation rate (%)
BPLSH (M = 110, L = 60)	0.015	2.677
BPLSH (M = 110, L = 50)	**0.014**	2.834
NPPS (k = 10)	0.272	0.087
NoIS	0.014	100

Table 8
Total execution time (instance selection + training) in seconds.

Method	NPPS	SVMKM	CBCH	NoIS	BPLSH
Total execution time	1174	2283	725	910	**509**

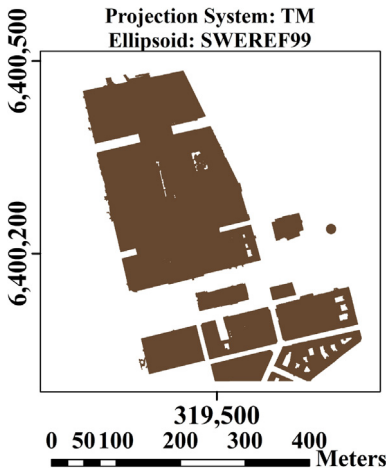

Fig. 13. Building extraction result.

Another aspect to be considered is the total execution time. Table 8 compares the total execution times of different methods. *BPLSH* has a significant advantage over the other methods in terms of the speed of handling the large-scale dataset. By considering the results presented in Fig. 12b and Tables 7 and 8, it is concluded that *BPLSH* is suitable for integration into the building extraction procedure and is preferable to other instance selection methods for producing the final map of buildings in this experiment.

The final building map is produced by applying the SVM, which is trained on the instances selected by *BPLSH* (M = 110, L = 50), to all the pixels of the study area. Furthermore, a morphological opening filter with a circular shaped kernel as the structuring element is applied to the output of the SVM to remove the salt and pepper effects of the pixel-based classification. Fig. 13 depicts the extracted binary map of the buildings.

7. Conclusion and future work

In this paper, a novel instance selection method, named *BPLSH*, was designed by adopting some notions of LSH in a unique and effective manner. *BPLSH* decimates a dataset by eliminating nonessential instances and preserving border patterns to speed up the training process of SVMs. It is based on the idea that a border instance has heterogeneous neighbors and is the closest neighbor to a sample from another class. More specifically, border instances are identified by analyzing the similarity index and sample labels. The concept of similarity index was incorporated into *BPLSH* to measure the closeness of instances. Using the property of approximate distance-preserving mapping makes *BPLSH* an effective method for big data analysis.

The performance of *BPLSH* was compared with that of four approaches on ten classification problems. Pareto optimality theory was used in the evaluation procedure to ensure fair comparison. The comprehensive experimental results indicated that the proposed method has high computation efficiency and strong viability on different datasets. Furthermore, it was

shown that *BPLSH* can retain the original classification accuracy while removing unnecessary instances. Considering the results of the comparative experiments, it can be concluded that *BPLSH* allows users to obtain small subsets that best balance between the size of the training set and the discriminatory power of SVMs.

Although *BPLSH* has shown remarkable performance, there is still room for improvement. The input parameters, *M* and *L*, need to be empirically adjusted depending on the dataset at hand, which is a limitation. In this context, one further development is to develop an optimization method for *BPLSH* that automatically tunes the input parameters. The ability to handle noisy datasets is another potentially important area for improvement. Noisy instances disrupt the training dataset and negatively impact the classifiers' performance. *BPLSH* could be developed such that it improves the quality of the dataset while also reducing its size. Indeed, some steps can be added to make it robust to the presence of noise in the dataset. Another potential direction is to evaluate or develop *BPLSH* for other instance-based learning methods with high memory and computational complexity, such as the k-nearest neighbors classifier.

CRediT authorship contribution statement

Mohammad Aslani: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing - original draft, Visualization. **Stefan Seipel:** Conceptualization, Writing - review & editing, Visualization, Supervision, Funding acquisition.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was partly funded by the European Regional Development Fund (ERDF), contract ID 20201871. The authors would like to thank Lantmäteriet, the Swedish mapping, cadastral, and land registration authority, for providing the data for this study.

References

[1] S. Abe, T. Inoue, Fast training of support vector machines by extracting boundary data, in: G. Dorffner, H. Bischof, K. Hornik (Eds.), International Conference on Artificial Neural Networks, Springer, Berlin, Heidelberg, 2001, pp. 308–313.
[2] S.C. Ahalt, A.K. Krishnamurthy, P. Chen, D.E. Melton, Competitive learning algorithms for vector quantization, Neural Netw. 3 (1990) 277–290.
[3] M.B. de Almeida, A.d.P. Braga, J.P. Braga, SVM-KM: speeding SVMs learning with a priori cluster selection and k-means, in: Proceedings, vol. 1. Sixth Brazilian Symposium on Neural Networks, RJ, Brazil, 2000, pp. 162–167..
[4] Á. Arnaiz-González, J.-F. Díez-Pastor, J.J. Rodríguez, C. García-Osorio, Instance selection of linear complexity for big data, Knowl-Based Syst. 107 (2016) 83–95.
[5] D. Arthur, S. Vassilvitskii, K-means++: The Advantages of Careful Seeding. In Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, New Orleans, Louisiana, 2007, pp. 1027–1035.
[6] M. Aslani, S. Seipel, A fast instance selection method for support vector machines in building extraction, Soft Comput. Appl. (2020) 106716.
[7] P. Birzhandi, H.Y. Youn, CBCH (clustering-based convex hull) for reducing training time of support vector machine, J. Supercomput. 75 (2019) 5261–5279.
[8] J. Cervantes, X. Li, W. Yu, Support vector machine classification based on fuzzy clustering for large data sets, in: A. Gelbukh, C.A. Reyes-Garcia (Eds.), MICAI 2006: Advances in Artificial Intelligence, Springer, Berlin, Heidelberg, 2006, pp. 572–582.
[9] J. Cervantes, X. Li, W. Yu, K. Li, Support vector machine classification for large data sets via minimum enclosing ball clustering, Neurocomputing 71 (2008) 611–619.
[10] F. Chang, C. Guo, X. Lin, C. Liu, C. Lu, Tree decomposition for large-scale SVM problems, in: 2010 International Conference on Technologies and Applications of Artificial Intelligence, Hsinchu, Taiwan, IEEE, 2010, pp. 233–240.
[11] J. Chen, C. Zhang, X. Xue, C.-L. Liu, Fast instance selection for speeding up support vector machines, Knowl-Based Syst. 45 (2013) 1–7.
[12] R.R. Colditz, An evaluation of different training sample allocation schemes for discrete and continuous land cover classification using decision tree-based algorithms, Remote Sens. 7 (2015) 9655–9681.
[13] M. Datar, N. Immorlica, P. Indyk, V. Mirrokni, Locality-sensitive hashing scheme based on p-stable distributions, in: Proceedings of the Twentieth Annual Symposium on Computational Geometry, ACM, New York, USA, 2004, pp. 253–262.
[14] L.-J. Deng, M. Feng, X.-C. Tai, The fusion of panchromatic and multispectral remote sensing images via tensor-based sparse modeling and hyper-Laplacian prior, Inform. Fusion 52 (2019) 76–89.
[15] K. Ding, C. Huo, B. Fan, S. Xiang, C. Pan, In defense of locality-sensitive hashing, IEEE Trans. Neural. Netw. Learn. Syst. 29 (2018) 87–103.
[16] S. Du, Y. Zhang, Z. Zou, S. Xu, X. He, S. Chen, Automatic building extraction from LiDAR data fusion of point and grid-based features, ISPRS J. Photogramm. Remote Sens. 130 (2017) 294–307.
[17] M. Ehrgott, Multicriteria Optimization, Springer, Berlin Heidelberg, 2005.
[18] G.M. Foody, A. Mathur, C. Sanchez-Hernandez, D.S. Boyd, Training set size requirements for the classification of a specific class, Remote Sens. Environ. 104 (2006) 1–14.
[19] S. Garcia, J. Derrac, J. Cano, F. Herrera, Prototype selection for nearest neighbor classification: Taxonomy and empirical study, IEEE Trans. Pattern Anal. Mach. Intell. 34 (2012) 417–435.
[20] H. Ghassemian, A review of remote sensing image fusion methods, Inform. Fusion 32 (2016) 75–89.
[21] Q. Hu, L. Zhen, Y. Mao, X. Zhou, G. Zhou, Automated building extraction using satellite remote sensing imagery, Automat. Constr. 123 (2021) 103509.
[22] R. Koggalage, S. Halgamuge, Reducing the number of training samples for Fast Support Vector Machine Classification, Neural Inform. Process. Lett. Rev. 2 (2004) 57–65.
[23] B. Li, Q. Wang, J. Hu, A fast SVM training method for very large datasets, in: International Joint Conference on Neural Networks, Atlanta, GA, USA, IEEE, 2009, pp. 1784–1789.

[24] W. Li, C. He, J. Fang, J. Zheng, H. Fu, L. Yu, Semantic Segmentation-Based Building Footprint Extraction Using Very High-Resolution Satellite Images and Multi-Source GIS Data, Remote Sens. 11 (2019) 403.
[25] Y. Li, L. Maguire, Selecting critical patterns based on local geometrical and statistical information, IEEE Trans. Pattern Anal. Mach. Intell. 33 (2011) 1189–1201.
[26] C. Liu, W. Wang, M. Wang, F. Lv, M. Konan, An efficient instance selection algorithm to reconstruct training set for support vector machine, Knowl-Based Syst. 116 (2017) 58–73.
[27] Y.-G. Liu, Q. Chen, R.-Z. Yu, Extract candidates of support vector from training set, Proceedings of the 2003 International Conference on Machine Learning and Cybernetics, vol. 5, IEEE, Xi'an, China, 2003, pp. 3199–3202.
[28] A. López-Chau, L.L. García, J. Cervantes, X. Li, W. Yu, Data Selection Using Decision Tree for SVM Classification, in: 2012 IEEE 24th International Conference on Tools with Artificial Intelligence, vol. 1, 2012, pp. 742–749..
[29] M.T. Lozano, J.S. Sánchez, F. Pla, Using the geometrical distribution of prototypes for training set condensing, in: R. Conejo, M. Urretavizcaya, J.-L. Pérez-de-la Cruz (Eds.), Current Topics in Artificial Intelligence. TTIA 2003. Lecture Notes in Computer Science, Springer, Berlin Heidelberg, Berlin, Heidelberg, pp. 618–627..
[30] A. Lyhyaoui, M. Martinez, I. Mora, M. Vaquez, J.-L. Sancho, A.R. Figueiras-Vidal, Sample selection via clustering to construct support vector-like classifiers, IEEE Trans. Neural. Netw. 10 (1999) 1474–1481.
[31] M. Malhat, M.E. Menshawy, H. Mousa, A.E. Sisi, A new approach for instance selection: algorithms, evaluation, and comparisons, Expert Syst. Appl. 149 (2020) 113297.
[32] U. Maulik, D. Chakraborty, Remote Sensing Image Classification: a survey of support-vector-machine-based advanced techniques, IEEE Geosci. Remote. Sens. Mag. 5 (2017) 33–52.
[33] J. Nalepa, M. Kawulok, Selecting training sets for support vector machines: a review, Artif. Intell. Rev. 52 (2018) 857–900.
[34] J.A. Olvera-López, J.A. Carrasco-Ochoa, J.F. Martínez-Trinidad, A new fast prototype selection method based on clustering, Pattern Anal. Appl. 13 (2010) 131–141.
[35] J. Radoux, C. Lamarche, E. Bogaert, S. Bontemps, C. Brockmann, P. Defourny, Automated training sample extraction for global land cover mapping, Remote Sens. 6 (2014) 3965–3987.
[36] J.R. Rico-Juan, J.J. Valero-Mas, J. Calvo-Zaragoza, Extensions to rank-based prototype selection in k-Nearest Neighbour classification, Appl. Soft Comput. 85 (2019) 105803.
[37] S. Riley, S. Degloria, S.D. Elliot, A terrain ruggedness index that quantifies topographic heterogeneity, Int. J. Sci. 5 (1999) 23–27.
[38] B. Shahriari, K. Swersky, Z. Wang, R.P. Adams, N. de Freitas, Taking the human out of the loop: a review of bayesian optimization, Proc. IEEE 104 (2016) 148–175.
[39] X.-J. Shen, L. Mu, Z. Li, H.-X. Wu, J.-P. Gou, X. Chen, Large-scale support vector machine classification with redundant data reduction, Neurocomputing 172 (2016) 189–197.
[40] G. Shi, Data Mining and Knowledge Discovery for Geoscientists, Elsevier, Amsterdam, 2013.
[41] H. Shin, S. Cho, Neighborhood property-based pattern selection for support vector machines, Neural Comput. 19 (2007) 816–855.
[42] P.-N. Tan, M. Steinbach, A. Karpatne, V. Kumar, Introduction to Data Mining, second ed., Pearson, New York, USA, 2019.
[43] D. Wang, L. Shi, Selecting valuable training samples for SVMs via data structure analysis, Neurocomputing 71 (2008) 2772–2781.
[44] J. Wang, P. Neskovic, L.N. Cooper, Selecting data for fast support vector machines training, in: K. Chen, L. Wang (Eds.), Trends in Neural Computation, Springer, Berlin, Heidelberg, 2007, pp. 61–84.
[45] R. Wang, S. Kwong, Sample selection based on maximum entropy for support vector machines, 2010 International Conference on Machine Learning and Cybernetics, Vol. 3, IEEE, Qingdao, China, 2010, pp. 1390–1395.
[46] J. Zhai, X. Wang, X. Pang, Voting-based instance selection from large data sets with mapreduce and random weight networks, Inf. Sci. 367–368 (2016) 1066–1077.
[47] J. Zhang, X. Lin, Advances in fusion of optical imagery and LiDAR point cloud applied to photogrammetry and remote sensing, Int. J. Imag. Data Fusion 8 (2017) 1–31.
[48] K. Zhang, S.-C. Chen, D. Whitman, M.-L. Shyu, J. Yan, C. Zhang, A progressive morphological filter for removing nonground measurements from airborne LiDAR data, IEEE Trans. Geosci. Remote. Sens. 41 (2003) 872–882.
[49] Z. Zhu, Z. Wang, D. Li, W. Du, Nearcount: Selecting critical instances based on the cited counts of nearest neighbors, Knowl-Based Syst. 190 (2020) 105196.