Received: 12 August 2020 | Revised: 13 May 2021 | Accepted: 31 May 2021

DOI: 10.1002/int.22546

RESEARCH ARTICLE

WILEY

Scalable feature selection using ReliefF aided by locality-sensitive hashing

Carlos Eiras-Franco[1] | **Bertha Guijarro-Berdiñas**[1] | **Amparo Alonso-Betanzos**[1] | **Antonio Bahamonde**[2]

[1]Research Center on Information and Communication Technologies (CITIC), Universidade da Coruña, A Coruña, Spain

[2]Department of Computer Science, Universidad de Oviedo, Asturias, Spain

Correspondence
Carlos Eiras-Franco, Research Center on Information and Communication Technologies (CITIC), Universidade da Coruña, Edificio Área Científica D1.02, Campus de Elviña, 15008 A Coruña, Spain.
Email: carlos.eiras.franco@udc.es

Funding information
Ministerio de Economía y Competitividad, Grant/Award Numbers: PID2019-109238GB-C2, TIN 2015-65069-C2-1-R and TIN 2015-65069-C2-2-R; Xunta de Galicia, Grant/Award Numbers: ED431C 2018/34, Centro singular de investigación de Galicia, accreditation (2016–2019); European Union, Grant/Award Number: FEDER funds

Abstract

Feature selection algorithms, such as ReliefF, are very important for processing high-dimensionality data sets. However, widespread use of popular and effective such algorithms is limited by their computational cost. We describe an adaptation of the ReliefF algorithm that simplifies the costliest of its step by approximating the nearest neighbor graph using locality-sensitive hashing (LSH). The resulting ReliefF-LSH algorithm can process data sets that are too large for the original ReliefF, a capability further enhanced by distributed implementation in Apache Spark. Furthermore, ReliefF-LSH obtains better results and is more generally applicable than currently available alternatives to the original ReliefF, as it can handle regression and multiclass data sets. The fact that it does not require any additional hyperparameters with respect to ReliefF also avoids costly tuning. A set of experiments demonstrates the validity of this new approach and confirms its good scalability.

KEYWORDS
big data, feature selection, locality-sensitive hashing, ReliefF, scalability

1 | INTRODUCTION

The greatly increased popularity of data science in recent years is partly due to the accumulation of enormous volumes of data and that continue to be produced daily. So-called *big data*[1] is, nonetheless, a double-edged sword, since the promise of greater insight is counteracted by the inherent difficulty of processing vast amounts of data, rendering many machine learning algorithms unusable. Data sets may contain many samples, many variables per sample, or both. For data sets with many variables, dimensionality reduction is advisable to improve the performance of learning methods.[2] Dimensionality reduction techniques can be classified as feature extraction or feature selection techniques. In feature extraction, the number of features in a data set is reduced by creating new features from original features that are then discarded, while in feature selection, irrelevant or redundant variables are discarded to obtain a reduced subset of input variables that still accurately describes the problem.[2] Reducing the number of variables enhances the comprehensibility of the data set and the fit of data by learning methods.

Another important way to tackle big data is distributed computing. Developed as a software solution are distributed processing platforms based on using clusters of several desktop computers to process enormous quantities of data. This approach started gaining popularity with Google's unveiling of the MapReduce programming paradigm in 2008.[3] Since then several open-source implementations have been described that follow this paradigm, the most popular being Apache Hadoop[4] and, launched a few years later, Apache Spark.[5] Their success has given rise to machine learning libraries, such as Mahout[6] for Hadoop and MLLib[7] for Spark, not to mention the re-implementation of popular algorithms—including some feature selection algorithms—to leverage distributed computing.

Nonetheless, sometimes the computational complexity of an algorithm cannot be reduced or the volume of data is such that even distributed implementation requires lengthy execution times. A possible solution is for data scientists to use techniques that obtain an approximation of the exact (but computationally more costly) model, which, despite being approximate, may be comparable to the exact model in terms of accuracy.

We describe an approximation of the ReliefF feature selection algorithm that addresses its main limitation—a high computational cost. While the effectiveness of ReliefF has been amply demonstrated, as will be shown in Section 2, its use for large-scale data sets has been hindered by the great computational demand. Accounting for most of ReliefF's computational load is its use of the nearest neighbor graph. To address this crucial limitation, we used a locality-sensitive hashing (LSH) algorithm to build an approximate, rather than an exact, graph. This approach, which has yielded encouraging results in preliminary tests,[8] is now fully developed and compared with the state of the art. The contributions of this article are as follows:

1. Our algorithm can accurately approximate the work done by ReliefF for a fraction of the computational cost.
2. Our implementation in Apache Spark leverages distributed computing to process data sets beyond the reach of ReliefF. Distributed computation can also be used to speed up the processing of data sets of any size.

3. Like the original ReliefF, and unlike current alternatives for approximating the nearest neighbor graph, our algorithm can handle binary, multiclass, and regression data sets.
4. Since no new hyperparameters are introduced, our method does not require manual hyperparameter tuning.

2 | RELATED WORK

The original Relief algorithm led to the development of a family of Relief-based feature selection algorithms, including ReliefF. Relief is a supervised feature ranking algorithm that estimates the relevance of features and ranks them for classification purposes[9] by establishing a relevance threshold above which only important features are retained. Weights are assigned to each attribute that help discerns elements that are very close together. In the case of classification data sets, for each example, Relief searches for the nearest elements within the same class (nearest *hit*) and within a different class (nearest *miss*) and updates the weight of each attribute A in proportion to the difference between the example value and the nearest hit and nearest miss values. The resulting weight W of attribute A can be interpreted in terms of probability. W approximates the following difference of conditional probabilities[10]:

$$\begin{aligned} W[A] = {} & P(\text{different value of } A \,|\, \text{different class}) \\ & - P(\text{different value of } A \,|\, \text{same class}). \end{aligned} \tag{1}$$

The good results obtained using Relief encouraged further research that yielded numerous extensions capable of dealing with multiclass problems and incomplete or noisy examples.[10] ReliefF is one such extension that has become even more popular than the original algorithm. Implementations of ReliefF are present in many machine learning software libraries. Further specializations of the algorithm were later introduced[11] that enabled it to tackle regression problems[12] and multilabel data sets,[13,14] and that factored in the cost of obtaining attributes.[15] While ReliefF has been thoroughly tested[16,17] and has been shown to be useful in a great variety of problems,[18–23] its great computational complexity makes its use on large-scale data sets prohibitively costly.[24]

While modifications to increase ReliefF's ability to handle large data sets have been developed that use distributed implementations,[25,26] sampling,[27] or random *kd*-trees to approximate the nearest neighbor graph,[28] their effectiveness in handling very large-dimensional data sets is limited. Of the ReliefF alternatives, the most efficient is DiReliefF,[26] which approximates ReliefF's attribute ranking by using a sampling scheme that greatly reduces the number of calculated neighbors. DiReliefF is also parallelizable, with its distributed implementation in Apache Spark simplifying the handling of large data sets.

Like many other feature selection, information retrieval, data mining, and machine learning methods, ReliefF relies on the analysis of similarity graphs. The k-nearest neighbor graph (*kNN*-graph) is the most popular such graph used in machine learning. A *kNN*-graph is a directed graph constructed over a set with n elements $\mathcal{D} = \{x_1, x_2, ..., x_n\}$, such that edges (x_i, x_j) indicate that x_j is among the k more similar elements to x_i according to some similarity measure $\sigma(x_i, x_j)$.

The resulting graph is very useful, but is computationally very costly to build because $n(n-1)/2$ pairwise comparisons are required; the result is a computational complexity value

of $\mathcal{O}(n^2)$. Some variations of the algorithm speed up calculations in specific conditions, for example, small dimensionality in the input space[29] or use of certain similarity measures.[30] However, only approximate solutions to dealing with high-dimensionality data in a reasonable time have been developed. One approach is to use generic similarity measures to replicate the exact graph as closely as possible while maintaining a low computational cost. Algorithms that obtain approximate graphs—as opposed to exact graphs—have been based on divide-and-conquer approaches[31] and local search.[32] However, LSH[33,34] is the approach that has shown most success to date.

Although the optimal way to exploit this technique remains an open problem, LSH essentially builds data structures that enable an element in a data set to be searched for in sublinear time. LSH maps data of any size into a smaller space using a hashing function that maximizes the possibility of assigning the same value to elements that are similar. It has been used, for instance, selection[35] and also has been successfully employed for *kNN*-graph construction[33,34] and for the related problem of *kNN* search.[36] Solutions are based on isolating small clusters of similar elements, used to calculate small isolated subgraphs that are then merged to obtain the complete approximate graph. Each element is only compared to other elements that LSH deems to be similar, that is, comparisons are avoided with elements that are very distant in the input space and so not among the nearest neighbors. This reduction in the number of unnecessary pairwise comparisons is what brings the computational cost down.

2.1 | Variable resolution LSH

Variable resolution LSH (VRLSH)[34,37] is the most recent algorithm that uses the LSH approach. As shown in Algorithm 1 and as described in detail below, it takes iterative LSH steps to obtain an approximate *kNN*-graph.

To compute the approximate *kNN*-graph, VRLSH first assigns one or several hash keys to each element x_i in the data set $\mathcal{D}$, using the selected LSH function (line 4). This function gives the same hash key to elements that are similar according to a specific similarity measure. A given, initially large, resolution level for similarity is used to assign the same hash key to very similar elements that are then grouped (line 5). Thanks to the properties of the hash function, these groups—or buckets—will contain elements that are very similar according to the similarity measure selected. An exact subgraph is constructed for each of the groups using the brute-force procedure, that is, computing each possible pairwise similarity. Since each element x_i is assigned several keys, it can appear in various groups and, therefore, in various subgraphs; those elements can be used to merge the initially isolated subgraphs, yielding larger subgraphs (loop in line 6). The data set $\mathcal{D}$ is then simplified by removing elements that have been involved in a prespecified $C_{MAX} > k$ number of pairwise comparisons. Those steps are repeated using the simplified $\mathcal{D}$, generating new subgraphs that are merged with the existing subgraphs until the result is a single graph containing all points and links to the nearest neighbors. After each iteration the search resolution is reduced, forcing LSH to create groups of elements that are slightly more dissimilar than in the previous iteration. As a final step (line 10 on), if the simplified data set has fewer than k elements, the process is stopped and, for every $x_i \in \mathcal{D}$, the nearest neighbors will be sought among the neighbors of the neighbors, or, if there are no outgoing links in the graph, among the neighbors of elements selected at random.

Algorithm 1: VRLSH algorithm.

Input: $\mathcal{D}$,k ←Dataset, Desired number of neighbors of each class

Output: G ← Graph linking each point to its k nearest neighbors

```
1  R_0,C_MAX ← estimateHyperparameters(D)
2  G ← ∅ , D' ← D, R ← R_0
3  while |D'| > k and |buckets| > 1 do
4      hashElems ← LSH(D', R)
5      buckets ← hashElems.groupByHash()
6      foreach b in buckets do
7          if (b.size > 1) then
               G ← G ∪ exactKNN(b.elems, k) end
       end
8      D' ← D' −
         G.getNodesWithAtLeastComparisons(C_MAX)
9      decrease R
   end
10 D' ←
     D' ∪ G.getNodesWithFewerNeighborsThan(k)
11 if |D'| > 1 then
12     foreach p in D' do
13         if |p.neighbors| = 0 then
14             p.neighbors ← randomSample(D, k)
           end
           else
15             p.neighbors ← topK(k, p.neighbors ∪
                 neighborDescent(p, G))
           end
       end
   end
```

An additional advantage of VRLSH is that the estimation procedure finds suitable values for the hyperparameters of the method. It does this by efficiently searching for resolution values and hash function hyperparameters that yield adequately sized buckets to start computing the graph (a process described in detail in Reference [34]). With this automated procedure, the user only needs to provide the data set and indicate the desired number of neighbors to be obtained in the graph. In avoiding costly test runs of the complete algorithm with different hyperparameter values, the total computational effort of using this method is effectively reduced. Moreover, although VRLSH automatically finds a suitable value for C_{MAX} that offers a reasonable trade-off between execution time and accuracy, this value can be overridden by users to obtain, depending on their needs, either a faster or a more accurate approximation.

2.2 | Limitations of *kNN*-graph computing algorithms for ReliefF

The above-described algorithms are designed to obtain a single *kNN*-graph for the whole data set, while ReliefF requires a graph that links each element to its k nearest neighbors in each class in the data. Needed to approximate the graph required by ReliefF are nontrivial

modifications, which depend on the structure of the graph-building algorithm. Those modifications can have a great impact on the performance and scalability of the resulting method. In the worst-case scenario, the building process needs to be repeated for the whole data set as many times as there are classes in the data, which, for multiclass data sets, results in a great computational overhead. To enhance VRLSH suitability for tackling this type of problem, we propose a profound modification that obtains the desired graph in a single pass over the data.

3 | PROPOSED ALGORITHM

Our aim was to obtain an algorithm that is capable of handling all the data set types that can be processed with the most popular implementations of ReliefF—including multiclass and regression data sets—and that yields an approximation of the ReliefF ranking output while requiring less computational effort.[12] The computational cost of this algorithm predominantly comes from the process that obtains the nearest hits and misses for each element, which entails obtaining the kNN-graph. Our proposal to reduce the computational cost is, therefore, to substitute the exact kNN-graph required for ReliefF with an approximate kNN-graph computed with a new VRLSH-based method.

For regression data sets, VRLSH does not need to be changed for ReliefF to use the resulting approximate kNN-graph, as no distinction needs to be made between hits and misses for regression data. For classification data sets, however, VRLSH has to be vastly changed for ReliefF to be able to rank attributes by weight. While the approximate graph calculated by VRLSH has a single list of neighbors for each element in the data set, for classification data sets ReliefF needs to distinguish between hits and misses. VRLSH therefore has to be extended to maintain, for each element in the data set, a list of neighbors for each class.

VRLSH also simplifies the initial data set $\mathcal{D}$ after each iteration (see line 8 in Algorithm 1), a process that needs to be modified to accommodate classification problems. In particular, given element x_i, VRLSH keeps $count\_x_i$, which is a count of pairwise comparisons, to determine if x_i needs to be removed from the data set (when $count\_x_i > C_{MAX}$). The $count\_x_i$ scalar count has to be transformed to a vector of counts $count\_x_i^j$ of pairwise comparisons with elements of each class c_j. In this new scenario, x_i is only removed from the data set when it has been involved in at least C_{MAX} comparisons for every possible class, that is, $count\_x_i^j > C_{MAX}\ \forall\ c_j \in \mathcal{C}$ where $\mathcal{C}$ is the set of possible classes. Moreover, depending on the distribution of the classes in the input space, some points may be distant from a given class c_k and, for these to be involved in C_{MAX}, pairwise comparison requires maintaining them in the data set for a large number of iterations. Those points will therefore accumulate a much larger number of pairwise comparisons than C_{MAX} for some classes, that is, $count\_x_i^j \gg C_{MAX}$ for some $j \neq k$. This challenges the ability of the method to perform a low total number of pairwise calculations.

We propose tackling this problem by modifying the bucketing step so that we avoid comparing points x_i with points of class c_j if $count\_x_i^j > C_{MAX}$. To simplify the notation, we will say that point x_i *requests* class c_j if $count\_x_i^j < C_{MAX}$, that is, it still needs to be compared to elements of class c_j before it is removed. This new bucketing step requires adding an additional component h_r for each hash. Considering the requested classes, h_r effectively splits the buckets of similar elements originated by the LSH function. Hence, point x_i of class c_j given hash h by the LSH function will have an updated set of hashes H' computed as described in Algorithm 2.

Algorithm 2: Hashing procedure modification for multiclass problems

Input: $h \leftarrow$ Hash given to $\boldsymbol{x_i}$ by the LSH function
Input: $c_j \leftarrow$ Class of $\boldsymbol{x_i}$
Input: $count\_x_i \leftarrow$ Vector containing the number of pairwise comparisons of $\boldsymbol{x_i}$ to elements of each class.
Output: H' $\leftarrow$ Set of modified hashes for $\boldsymbol{x_i}$

```
1 H' ← (h, c_j)
2 if count_x_i^k < C_MAX ∀c_k ∈ C then
3 |  H'.append((h, ∅))
  end
4 else
5 |  foreach c_k in C do
6 |  |  if count_x_i^k < C_MAX then
7 |  |  |  H'.append((h, c_k))
  |  |  end
  |  end
  end
```

The first element of H' is always the hash (h, c_j), as described in line 1, intended to keep x_i available for other elements that request class c_j. H' is completed by one of two alternatives: if x_i requests all possible classes, line 3 forces x_i to be compared to any element in the same situation; otherwise line 7 adds a hash for every requested class to H'. This process originates two types of buckets that have to be treated differently according to their characteristics. Since this has the effect of invalidating the bucketing step in VRLSH (corresponding to the loop in line 6 of Algorithm 1), an updated version (detailed in Algorithm 3) is required.

Algorithm 3: Updated bucket processing procedure

Input: $G \leftarrow$ Graph containing the computed nearest neighbors for each point
Input: $buckets \leftarrow$ Buckets of points that received the same hash value
Output: $G \leftarrow$ Updated graph

```
1 foreach ((h, h_r), points) in buckets do
2 |  if h_r = ∅ then
3 |  |  G ← G ∪ exactKNN(points, k)
  |  end
  |  else
4 |  |  targets ← ∅ , requesters ← ∅
5 |  |  foreach p in points do
6 |  |  |  if (h_r = p.class) then targets.append(p)
  |  |  |  end
7 |  |  |  if (h_r ≠ p.class or
  |  |  |  p.counts^{p.class} < C_MAX) then
  |  |  |  requesters.append(p) end
  |  |  end
8 |  |  G ← G ∪ pairKNN(requesters, targets, k)
  |  end
  end
```

Figure 1 summarizes the modified bucketing procedure, in which buckets containing elements given the same hash are handled according to h_r, the modifier component added to their hash. The first option, shown in line 3 of Algorithm 2, processes buckets of elements requesting all classes by computing all possible pairwise comparisons between elements in the bucket, as in the original Algorithm 1. This guarantees that those elements are compared to all points deemed similar by the LSH function, quickly retrieving neighbors from every nearby class. The second option deals with buckets containing elements that either request a class or belong to a requested class. Lines 6 and 7 divide each bucket with hash (h, h_r) into two sets: *requesters*, that is, points requesting h_r, and *targets*, that is, points with class h_r. The *pairKNN* function then measures the similarity of each target to each requester and builds the corresponding subgraph, as described in line 8. In doing this, no requester–requester or target–target comparisons are performed, which saves on a great number of operations, as does division of each bucket into smaller and more specific buckets by appending h_r to hashes.

We refer to the resulting algorithm that integrates all these parts as ReliefF-LSH, for which a distributed implementation in the Apache Spark framework is available for download from https://github.com/eirasf/ReliefF-LSH. Although the ReliefF-LSH algorithm is designed to work with any family of LSH functions, in this study we implement and test it for a commonly used family, sensitive to Euclidean distance, which performs hashing through random projections onto one-dimensional lines.[38] As shown in Section 5, satisfactory results are obtained with this family of functions.

3.1 | Spatial and computational complexity

Given a data set with n elements, each described with d variables, and a hasher that generates hashes of length l for each element t, the greatest memory demand on the algorithm is from

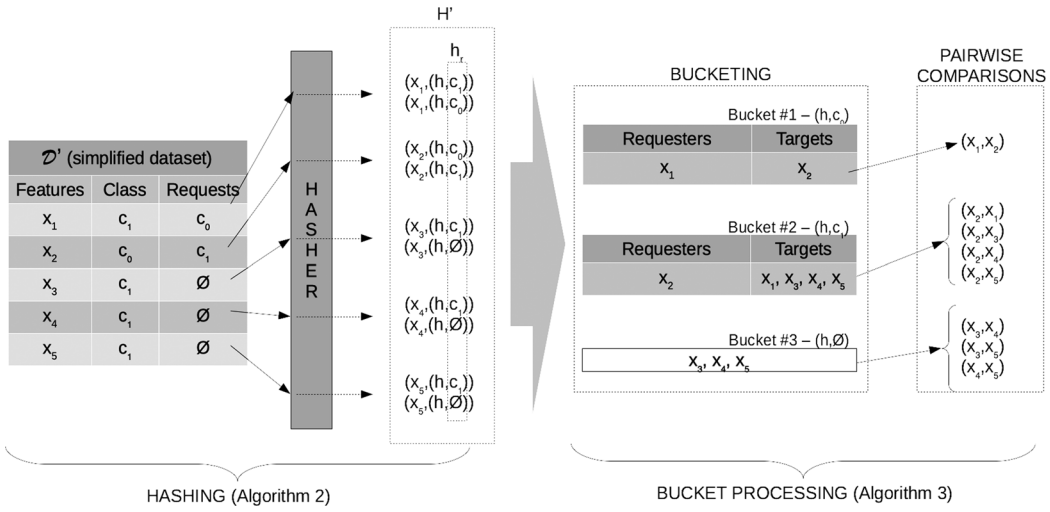

FIGURE 1 Proposed hashing and bucket-forming procedure. x_i are, in this example, similar enough to be assigned the same hash value h and the hasher outputs only one hash per element. This hash, augmented with the h_r value, informs the bucket-forming procedure. A different number of pairwise comparisons are performed for each type of bucket, as described in Algorithm 3

storing the generated hashes. The proposed algorithm therefore requires memory in proportion to the product of n, l, and t, that is, $\mathcal{O}(nlt)$. The memory requirement is managed by distributing calculations across various computing nodes. Furthermore, given that the number of variables d in the data set does not directly impact on memory requirements, for high-dimensionality data the space needed to store the hashes and the space occupied by the data set are comparable. The memory overhead is therefore smaller for high-dimensionality data sets, which is the focus of our approach. The computational complexity of the method cannot be analytically established, as the probabilistic nature of the hashing function (and, therefore, of the bucketing process) makes it difficult to establish an upper bound for the complexity of both the parameter tuning and bucketing procedures. Computational complexity also depends on the data set, so even an average-case study is not possible. Nonetheless, the experimental results reported in Section 5 highlight the efficiency of this algorithm in terms of computational complexity.

4 | EXPERIMENTAL SETTINGS

We report two sets of experiments performed to validate the effectiveness of ReliefF-LSH, whose performance is compared with those of ReliefF and of DiReliefF—the best alternative to ReliefF,[26] as discussed in Section 2. We also compared results to those for ReliefF with an approximate kNN-graph precomputed using the most popular method for this task, namely, the Fast Library for Approximate Nearest Neighbors (FLANN),[39] which relies on randomized kd-trees[40] to build the graph. In this composite method, the algorithm, which we call ReliefF-FLANN, was run over the whole data set for each class and the resulting graphs were merged to obtain the graph required for ReliefF.

In a first set of experiments, for real data sets we recorded the execution time required to compute exact weights using ReliefF and then compared that time with the times taken by ReliefF-LSH, DiReliefF, and ReliefF-FLANN. Since ReliefF-LSH, DiReliefF, and ReliefF-FLANN are all approximate methods, the accuracy of their rankings was determined by measuring the level of agreement of the approximate rankings with the ground-truth rankings calculated by ReliefF. In a second set of experiments, we determined the level of scalability of ReliefF-LSH, reporting the ReliefF-LSH runtimes when applied to the same problem with varying amounts of computational resources assigned to the task.

4.1 | Equipment and data sets

Each experiment was executed in a computer cluster consisting of machines with 12 computing cores. The cluster nodes are described in Table 1. We ran ReliefF-LSH on Apache Spark version 2.4.0 on Hadoop 3.0.0-6.1.0, and DiReliefF on Apache Spark version 1.6.1 on Hadoop 2.7.1.2.4.2.0-258, because the available implementation required it. CentOS Linux release 7.4.1708 is the operating system used by those machines. ReliefF-FLANN was run on a single machine using the Python bindings contained in PyFLANN 1.6.14.

For the experiments we selected nine real-world high-dimensional data sets, as described in Table 2, selected to represent all problems that ReliefF can handle, namely, regression ($Year_{small}$), binary classification ($Higgs$,[41] $Higgs_{small}$, $Epsilon_{small}$), and multiclass classification ($KDD99_{small}$, CT_{small}, $Cifar10$,[42] $SVHN$,[43] $Sensorless$[44]). The data sets reflect various problems, including computer vision and intrusion detection.*

TABLE 1 Cluster description

32 Nodes. Individual specifications	
Processor	2× Intel Xeon E5-2620 v3 and 2.40 GHz
Cores	6 per processor (12 per node)
Threads	2 per core (24 total per node)
Storage	12× 2TB NL SATA 6 Gbps 3.5″ G2HS
RAM	64 GB
Network	1 x 10 Gbps + 2 x 1 Gbps

TABLE 2 Data set description

Data set	Attributes	Elements	Classes
$Year_{small}$	90	46,371	–
Higgs	28	11,000,000	2
$Higgs_{small}$	28	55,000	2
$Epsilon_{small}$	2000	50,000	2
$KDD99_{small}$	41	48,984	23
CT_{small}	54	58,101	7
Cifar10	3072	50,000	10
SVHN	3072	73,257	10
Sensorless	84	58,509	11

Some of the data sets contain a very large number of elements, making their processing with the original ReliefF unfeasible (several weeks would be required, even using 12 computing cores). In the first set of experiments, to compare runtimes for ReliefF, DiReliefF, and ReliefF-LSH, therefore, we used trimmed versions of the largest data sets, that is, only the top N elements were used: $Year_{small}$ is the top 10% of the YearPredictionMSD[44] data set, $Higgs_{small}$ is the top 0.5% of the *Higgs* data set (55,000 samples), $Epsilon_{small}$ is the top 10% (50,000 elements) of Epsilon,[45] $KDD99_{small}$ is the top 1% (48,984 elements) of KDD99,[46] and CT_{small} is the top 10% of CoverType.[44] In our second set of experiments we used the full *Higgs* data set to test the capability of our method to handle large data sets.

4.2 | Methodology

We used three different measures to compare the results achieved with ReliefF and with DiReliefF, ReliefF-LSH, and ReliefF-FLANN: runtimes, recall, and weight error. First, we evaluated time efficiency by measuring algorithm runtimes for each data set. Second, we measured the accuracy of the results, using the recall measure at various selection levels to determine the correctness of the retrieved rankings of attributes. Since the main purpose of the ReliefF ranking is to determine a subset of relevant attributes, comparing the retrieved subsets

shows the effectiveness of the approximate algorithms, that is, DiReliefF, ReliefF-LSH, and ReliefF-FLANN. For a given selection level t, attribute ranking $\mathcal{E}$ obtained with ReliefF and attribute ranking $\mathcal{A}$ computed by an approximate algorithm, we define recall as

$$recall(t) = \frac{|\mathcal{E}.first(t) \cap \mathcal{A}.first(t)|}{t}, \tag{2}$$

where $\mathcal{X}.first(t)$ symbolizes the top t elements of list $\mathcal{X}$. Third, to give a more fine-grained account of the accuracy of the rankings, a distinction had to be drawn between sets of attributes with the same number of wrong selections. Since ReliefF is a ranking method, attribute selection is based on ranking attributes by weight and keeping the top N. For some data sets, many attributes may have similar weights; hence, keeping one of these similarly weighted attributes does not have as adverse an impact as keeping an attribute with a much smaller weight. We quantified the magnitude of errors for a given selection level by measuring the difference between the weights in the exact list and the weights in the approximate list, that is,

$$WD(t) = -\left(\sum_{a \in \mathcal{E}.first(t)} \mathcal{E}(a) - \sum_{a \in \mathcal{A}.first(t)} \mathcal{E}(a) \right), \tag{3}$$

where $\mathcal{E}$ is the ReliefF attribute ranking, $\mathcal{A}$ is the approximate attribute ranking being assessed, and $\mathcal{E}(a)$ represents the weight given to attribute a by ReliefF. Changing the sign is merely an aesthetic device to obtain a measure with a positive value that should be minimized.

Note that the random nature of some of the steps in the DiReliefF, ReliefF-LSH, and ReliefF-FLANN algorithm makes their results nondeterministic, that is, output rankings may vary in different executions with the same data. To mitigate the impact of randomness in our measurements, we reported the average value for four separate runs for each of the three methods. We also used the same method of listing the average runtime of four separate executions to mitigate the slight variation in runtimes for the random bucketing procedure at the heart of ReliefF-LSH.

It is also noteworthy that, unlike ReliefF-LSH and DiReliefF, ReliefF-FLANN is not implemented in a distributed framework, and, while it leverages multithreading to use all available cores in a machine, it cannot take advantage of a cluster of computers to handle larger data sets or to expedite the processing of small data sets. This lack of distributed computing, however, allows ReliefF-FLANN to compute the approximate kNN-graph without the overhead of cluster handling procedures, resulting in faster times for smaller data sets that can be processed by a single machine. Therefore, the above factors need to be kept in mind when comparing ReliefF-FLANN with ReliefF-LSH and DiReliefF in terms of execution times.

Finally, DiReliefF allows the user to define a sampling level to indicate how precise the computed ranking should be. In all experiments with DiReliefF we used 1000 samples so as to achieve the most precise ranking possible without exceeding the ReliefF runtime. ReliefF-FLANN was configured to compute each graph using eight random kd-trees; larger values were also tested but offered no improvement despite the increased computational effort.

5 | EXPERIMENTAL RESULTS

As detailed in Section 3, the kNN-graph-building procedure was modified to enable the linking of all elements to their k nearest neighbors in each class and to retrieve the graph required for ReliefF to be applied to classification problems. The challenge is exacerbated as the number of

classes grew. We therefore report the results of our first set of experiments in two separate groups: (1) regression and binary classification data sets, and (2) multiclass classification data sets.

5.1 | Regression and binary classification

In this first set of experiments, we performed feature selection on regression and binary classification data sets using the DiReliefF, ReliefF-FLANN, and ReliefF-LSH algorithms, with the results pointing to the superiority of ReliefF-LSH over the other methods in terms of recall and weight error, as depicted in Figures 2 and 3, respectively, at various threshold selection levels for DiReliefF, ReliefF-FLANN, and ReliefF-LSH. Since regression data sets are not supported by DiReliefF, no results are listed for DiReliefF for the $Year_{small}$ data set.

ReliefF-LSH obtained results that are clearly superior to ReliefF-FLANN in all cases and to DiReliefF for both $Epsilon_{small}$ and $Higgs_{small}$. ReliefF-LSH also outperformed ReliefF-FLANN for the regression data set, recalling perfectly the five most relevant attributes and accurately retrieving the rest of the list (the lowest recall, achieved at selection level 15, was a competent 0.82, that is, 12 or 13 correct attributes out of 15 depending on the execution, with a weight error of only $5 * 10^{-5}$). Of the three compared alternatives, the ReliefF-LSH rankings were the most reliable approximations to the exact rankings.

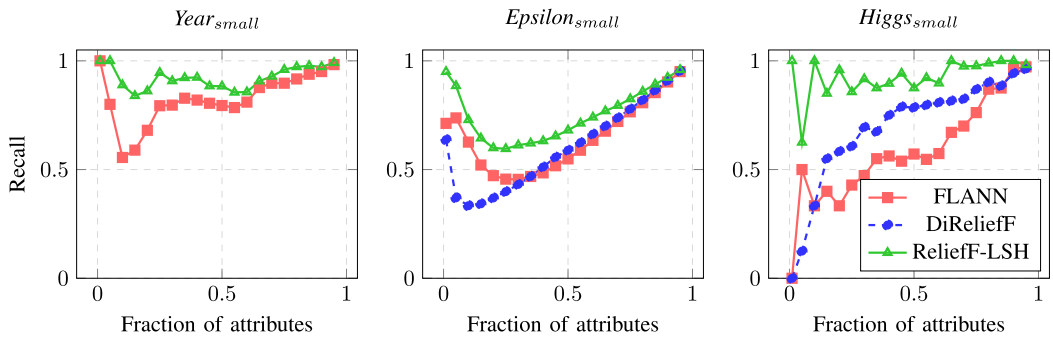

FIGURE 2 Recall obtained for data sets $Year_{small}$, $Epsilon_{small}$, and $Higgs_{small}$. LSH, locality-sensitive hashing [Color figure can be viewed at wileyonlinelibrary.com]

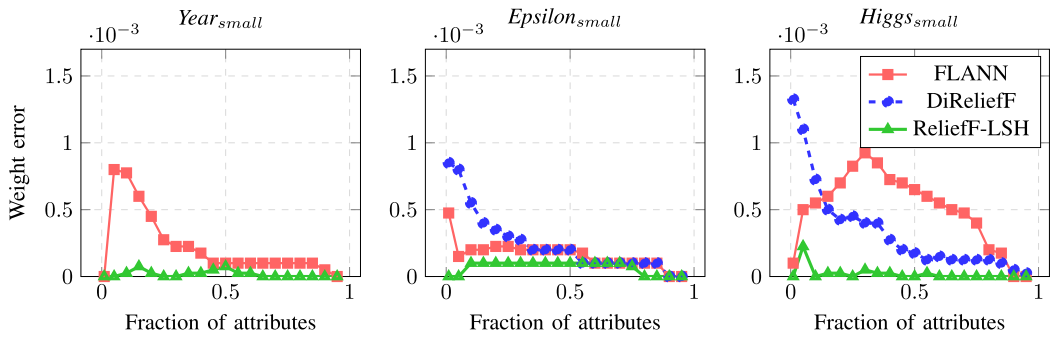

FIGURE 3 Weight error for data sets $Year_{small}$, $Epsilon_{small}$, and $Higgs_{small}$. LSH, locality-sensitive hashing [Color figure can be viewed at wileyonlinelibrary.com]

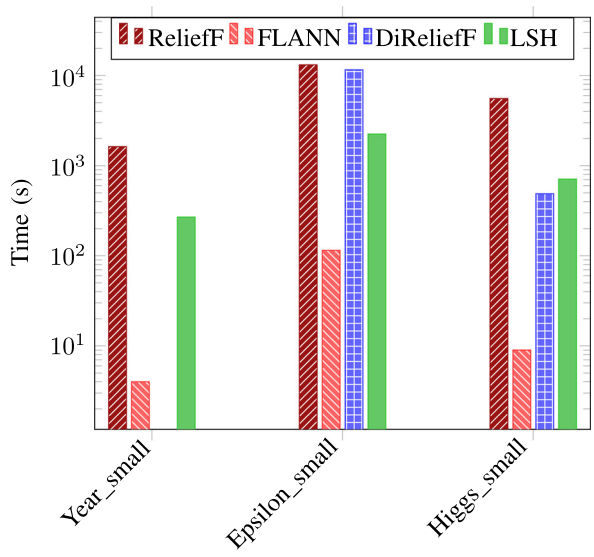

FIGURE 4 Execution times (logarithmic scale) of ReliefF, ReliefF-FLANN, DiReliefF, and ReliefF-LSH for data sets $Year_{small}$, $Epsilon_{small}$, and $Higgs_{small}$. LSH, locality-sensitive hashing [Color figure can be viewed at wileyonlinelibrary.com]

Comparisons of total processing runtimes for each data set are depicted in Figure 4, which shows that ReliefF-LSH offered a consistent time reduction over ReliefF and competed well with DiReliefF and ReliefF-FLANN. Execution times with ReliefF-LSH were always significantly lower than those for ReliefF. Compared with DiReliefF, while ReliefF-LSH was significantly faster in the case of higher dimensionality data sets, it was slightly slower with the $Higgs_{small}$ data set, consisting of elements with only 28 attributes. This effect, which only appeared with small data sets, can be attributed to the overhead caused by the hashing and grouping stages in VRLSH, which is notable only in low-dimensionality and small data sets, like $Higgs_{small}$. However, since real-life data sets typically have many attributes, this effect becomes negligible because pairwise comparisons are more costly. Similarly, for data sets containing many instances, the time saved by ReliefF-LSH for pairwise comparisons compensates for and greatly exceeds any overhead. Note that the ReliefF-LSH runtime to obtain the ranking for $Epsilon_{small}$ was significantly smaller, yet the accuracy of its results was clearly superior. While ReliefF-FLANN obtained the approximate kNN-graphs very rapidly, when used by ReliefF those graphs performed poorly. Overall, ReliefF-LSH, in offering the best balance between recall and execution time, achieves the most accurate approximations with a consistent time reduction.

5.2 | Multiclass data sets

Figures 5 and 6 show accuracy results for DiReliefF, ReliefF-FLANN, and ReliefF-LSH for multiclass data sets. ReliefF-LSH again obtained the most accurate approximations while competently handling all types of data. Results were superior in all cases except for CT_{small} between selection levels 3 and 30, for which DiReliefF achieved better results. The execution times shown in Figure 7 confirm that ReliefF-LSH took significantly less time than the original ReliefF. As for the previously mentioned overhead caused by ReliefF-LSH, this adverse effect was only noted for the smallest data sets. Thus, while the time overhead was perceptible,

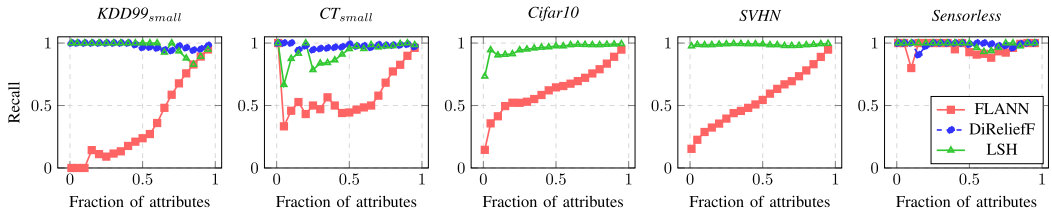

FIGURE 5 Recall obtained for data sets *KDD99*$_{small}$, *CT*$_{small}$, *Cifar10*, *SVHN*, and *Sensorless*. *LSH, locality-sensitive hashing* [Color figure can be viewed at wileyonlinelibrary.com]

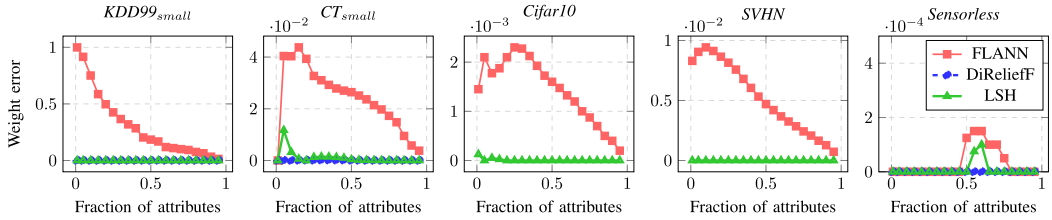

FIGURE 6 Weight error for data sets *KDD99*$_{small}$, *CT*$_{small}$, *Cifar10*, *SVHN,* and *Sensorless*. LSH, locality-sensitive hashing [Color figure can be viewed at wileyonlinelibrary.com]

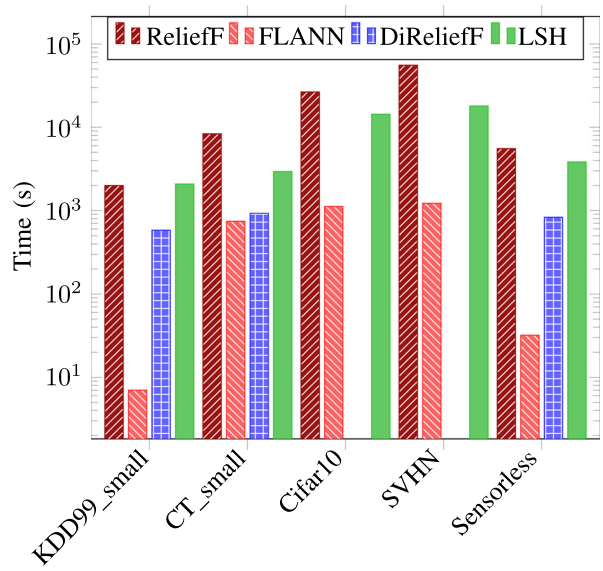

FIGURE 7 Execution times (logarithmic scale) for ReliefF, ReliefF-FLANN, DiReliefF, and ReliefF-LSH for data sets *KDD99*$_{small}$, *CT*$_{small}$, *Cifar10*, *SVHN*, and *Sensorless*. LSH, locality-sensitive hashing [Color figure can be viewed at wileyonlinelibrary.com]

execution did not take as long as it did for ReliefF. As data set dimensionality and/or size grew, the time overhead was dwarfed by the time saved in unnecessary pairwise comparisons, improving the time reduction accordingly.

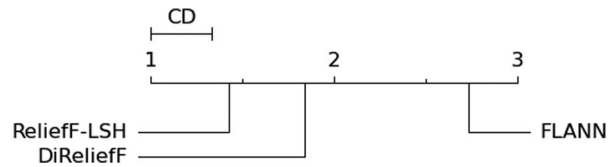

FIGURE 8 Nemenyi test for recall: critical distance diagram highlighting the statistically significant advantage of ReliefF-LSH over ReliefF-FLANN and DiReliefF. Since differences were only minor for weight error, the corresponding diagram has been omitted in the interest of brevity. LSH, locality-sensitive hashing

Contrary to what happened with the binary classification data sets, for the smallest data sets, ReliefF-LSH was slower than DiReliefF, which highlights the fact that multiclass problems demand more effort from ReliefF-LSH. However, the available implementation of DiReliefF could not handle the larger and therefore more computationally demanding *Cifar*10 and *SVHN* data sets, because the memory available in the computational nodes was exceeded, independently of how many partitions were used for the distributed task. Although ReliefF-FLANN retained its advantage in terms of processing time, its rankings differed vastly from the ground truth, to the point of being unusable in most cases. Overall, even if the multiclass data sets demanded more computational effort from ReliefF-LSH, our method maintained a competitive advantage by achieving high-quality approximations with consistent time reductions across all types of data.

To determine whether the performance advantage of our method was statistically significant we implemented Nemenyi testing covering all the results across the different selection levels for the data sets (the data sets that could not be processed by DiReliefF were excluded to maintain DiReliefF in the comparison). The Nemenyi test results (depicted in Figure 8) show that ReliefF-LSH had a statistically significant advantage over DiReliefF and ReliefF-FLANN. Moreover, the closest competitor, DiReliefF, was unable to process data sets that ReliefF-LSH handled correctly.

5.3 | Scalability

In another experiment we evaluated the scalability of ReliefF-LSH in terms of its ability to leverage distributed computing, for which purpose we used the full *Higgs* data set, comprising 11 million instances. The attribute ranking was calculated repeatedly, with the number of computing cores used increasing in each run. Table 3, which lists the resulting runtimes, shows that ReliefF-LSH could process data sets that were beyond the reach of ReliefF. While the original algorithm took 5550 s to select features for $Higgs_{small}$, ReliefF-LSH was able to tackle the full *Higgs* data set, with 200 times more elements, in just 48,283 s and with the same number of computing cores. Note that, since the original ReliefF has quadratic computational complexity, execution would be expected to take in the order of 10^7 s, which is inoperative in practice.

These results also highlight that, since many of the operations required by ReliefF-LSH could be independently calculated in parallel, the use of computational nodes involved in the run was efficient, as shown by an inversely proportional relationship between number of nodes and runtime, with a slope close to the desired value of −1. A larger experiment with a synthetic data set comprising 50 million elements, also reported in Table 3, yielded similar results, demonstrating that the efficiency of computational nodes was maintained when larger data sets were tackled.

TABLE 3 Efficient use of computational nodes as a measure of scalability

# Nodes	Time (s)	Scan rate	Ops/s	Speed-up
Higgs				
1	48,283	$2.42 \cdot 10^{-3}$	$3.03 \cdot 10^{6}$	1.00
2	19,864	$1.71 \cdot 10^{-3}$	$5.19 \cdot 10^{6}$	1.72
4	15,402	$2.5 \cdot 10^{-3}$	$9.96 \cdot 10^{6}$	3.29
Synth-50M				
2	73,478	$5.43 \cdot 10^{-5}$	$0.92 \cdot 10^{6}$	1.00
4	44,533	$6.81 \cdot 10^{-5}$	$1.91 \cdot 10^{6}$	2.07
8	26,300	$7.00 \cdot 10^{-5}$	$3.33 \cdot 10^{6}$	3.60

Note: Values correspond to the ReliefF-LSH algorithm run on the full *Higgs* data set and on a synthetic data set with 50 million elements. Scan rate refers to the fraction of operations executed with respect to the exact brute-force calculation, while speed-up is the ratio between operations per second for a given execution and operations per second for a single computational nodes (12 cores).

6 | CONCLUSIONS

Currently available feature selection algorithms are either incapable of handling very large data sets or are limited to a specific type of input data. We describe our modification of the widely used ReliefF algorithm aimed at enabling the processing of large data sets. In our approach, the computationally costly process of obtaining the *k*NN graph is approximated using a modification of the VRLSH algorithm, called ReliefF-LSH, which greatly reduces execution time while preserving accuracy. A distributed implementation of ReliefF-LSH using the Apache Spark framework is available for free download.[#] The described experiments confirm the following advantages of our method:

1. The approximate ranking by ReliefF-LSH, in terms of both recall and weight error, is more accurate than competing methods to a statistically significant degree.
2. The distributed computing approach of ReliefF-LSH leverages computing power to scale up to very large data sets, far beyond the reach of the original ReliefF and achieving better approximations than competing methods.
3. The efficient use of memory by ReliefF-LSH allows the processing of very high-dimensionality data sets that are too memory-demanding for the most scalable alternative, that is, DiReliefF.
4. ReliefF-LSH is not restricted to certain types of input data and can be implemented for any classification or regression data set, unlike other approaches.
5. ReliefF-LSH lacks any additional hyperparameters with respect to ReliefF, which means that tuning processes that are computationally costly when dealing with large data sets can be sidestepped.
6. ReliefF-LSH is customizable and can prioritize runtime over accuracy and vice versa, giving users the option to output results suited to their needs.

As future work, the strategies used by ReliefF-LSH to process multiclass data sets could be analyzed in an attempt to develop refinements aimed at further reducing computational demands.

ACKNOWLEDGMENTS

This study has been supported in part by the Spanish Ministerio de Economía y Competitividad (projects PID2019-109238GB-C2 and TIN 2015-65069-C2-1-R and 2-R), partially funded by FEDER funds of the EU and by the Xunta de Galicia (projects ED431C 2018/34 and Centro Singular de Investigación de Galicia, accreditation 2016-2019). The authors wish to thank the Fundación Pública Galega Centro Tecnolóxico de Supercomputación de Galicia (CESGA) for the use of their computing resources. Funding for open access charge: Universidade da Coruña/CISUG.

ENDNOTES

*All data sets are publicly available for download at https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/ except KDD99, which can be downloaded from http://www.kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

#https://github.com/eirasf/ReliefF-LSH.

ORCID

Carlos Eiras-Franco https://orcid.org/0000-0001-6322-7593
Bertha Guijarro-Berdiñas https://orcid.org/0000-0001-8901-5441
Amparo Alonso-Betanzos https://orcid.org/0000-0003-0950-0012
Antonio Bahamonde https://orcid.org/0000-0002-2188-9035

REFERENCES

1. Mayer-Schönberger V, Cukier K. *Big Data: a Revolution That Will Transform How We Live, Work, and Think*. USA: Houghton Mifflin Harcourt; 2013.
2. Guyon I, Gunn S, Nikravesh M, Zadeh LA. *Feature Extraction: Foundations and Applications*. Vol. 207. USA: Springer; 2008.
3. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. *Commun ACM*. 2008;51(1): 107-113.
4. Apache Foundation. *Apache Hadoop Project*. Wilmington, DE: The Apache Software Foundation; 2006. http://hadoop.apache.org/. Accessed April 19, 2019.
5. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: cluster computing with working sets. *HotCloud*. 2010;10(10-10):95.
6. Apache Foundation. *Apache Hadoop Project*. Wilmington, DE: The Apache Software Foundation; 2009. http://mahout.apache.org/. Accessed April 19, 2019.
7. Meng X, Bradley J, Yavuz B, et al. Mllib: machine learning in Apache Spark. *J Mach Learn Res*. 2016;17(1): 1235-1241.
8. Eiras-Franco C, Guijarro-Berdinas B, Alonso-Betanzos A, Bahamonde A. Seleccion de caracterısticas escalable con ReliefF mediante el uso de hashing sensible a la localidad. In: *XVIII Conferencia de la Asociación* Española para la Inteligencia Artificial; 2018.
9. Kenji K, Rendell LA. A practical approach to feature selection. In: *Machine Learning Proceedings 1992*. Netherlands: Elsevier; 1992:249-256.
10. Kononenko I. Estimating attributes: analysis and extensions of relief. In: *European Conference on Machine Learning*. USA: Springer; 1994:171-182.
11. Urbanowicz RJ, Meeker M, LaCava W, Olson RS, Moore JH. Relief-based feature selection: introduction and review. 2017. arXiv preprint arXiv:1711.08421.
12. Robnik-Šikonja M, Kononenko I. An adaptation of relief for attribute estimation in regression. In: *Proceedings of the Fourteenth International Conference Machine Learning (ICML '97)*. USA: ACM; 1997:296-304.
13. Spolaôr N, Cherman EA, Monard MC, Lee HD. ReliefF for multi-label feature selection. In: *2013 Brazilian Conference on Intelligent Systems (BRACIS)*. IEEE; 2013:6-11.
14. Slavkov I, Karcheska J, Kocev D, Kalajdziski S, Džeroski S. ReliefF for hierarchical multi-label classification. In: *International Workshop on New Frontiers in Mining Complex Patterns*. Springer; 2013:148-161.

15. Bolón-Canedo V, Remeseiro B, Sánchez-Maroño N, Alonso-Betanzos A. mC-ReliefF—an extension of ReliefF for cost-based feature selection. In: *ICAART (1)*; 2014:42-51.
16. Robnik-Šikonja M, Kononenko I. Theoretical and empirical analysis of ReliefF and RReliefF. *Mach Learn*. 2003;53(1-2):23-69.
17. Bolón-Canedo V, Sánchez-Maroño N, Alonso-Betanzos A. A review of feature selection methods on synthetic data. *Knowl Inf Syst*. 2013;34(3):483-519.
18. Zhang J, Chen M, Zhao S, Hu S, Shi Z, Cao Y. ReliefF-based EEG sensor selection methods for emotion recognition. *Sensors*. 2016;16(10):1558.
19. Zhang Y, Ren X, Zhang J. Intrusion detection method based on information gain and ReliefF feature selection. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. USA: IEEE; 2019:1-5.
20. Li B, Wen T, Hu C, Zhou B. Power system transient stability prediction algorithm based on ReliefF and lSTM. In: *International Conference on Artificial Intelligence and Security*. USA: Springer; 2019:74-84.
21. Kononenko I, Šimec E, Robnik-Šikonja M. Overcoming the myopia of inductive learning algorithms with ReliefF. *Appl Intell*. 1997;7(1):39-55.
22. Zheng X, Liu X, Zhang Y, Cui L, Yu X. A portable HCI system-oriented EEG feature extraction and channel selection for emotion recognition. *Int J Intell Syst*. 2021;36(1):152-176.
23. Sun L, Yin T, Ding W, Qian Y, Xu J. Multilabel feature selection using ML-ReliefF and neighborhood mutual information for multilabel neighborhood decision systems. *Inf Sci*. 2020;537(4):401-424.
24. Bolón-Canedo V, Rego-Fernández D, Peteiro-Barral D, Alonso-Betanzos A, Guijarro-Berdiñas B, Sánchez-Maroño N. On the scalability of feature selection methods on high-dimensional data. *Knowl Inf Syst*. 2018;56(2):395-442.
25. Eiras-Franco C, Bolón-Canedo V, Ramos S, González-Domínguez J, Alonso-Betanzos A, Touriño J. Multithreaded and spark parallelization of feature selection filters. *J Comput Sci*. 2016;17:609-619.
26. Palma-Mendoza R-J, Rodriguez D, de Marcos L. Distributed ReliefF-based feature selection in spark. *Knowl Inf Syst*. 2018:1-20.
27. Eppstein MJ, Haake P. Very large scale ReliefF for genome-wide association analysis. In: *IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology, 2008 (CIBCB '08)*. USA: IEEE; 2008:112-119.
28. Xu S, Li X, Lu WF. Randomized K-d tree ReliefF algorithm for feature selection in handling high dimensional process parameter data. In: *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. USA: IEEE; 2016:1-8.
29. Bentley JL. Multidimensional binary search trees used for associative searching. *Commun ACM*. 1975;18(9):509-517.
30. Anastasiu DC, Karypis G. L2Knng: fast exact k-nearest neighbor graph construction with l2-norm pruning. In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. USA: ACM; 2015:791-800.
31. Chen J, Fang H-r, Saad Y. Fast approximate kNN graph construction for high dimensional data via recursive Lanczos bisection. *J Mach Learn Res*. 2009;10:1989-2012.
32. Dong W, Moses C, Li K. Efficient k-nearest neighbor graph construction for generic similarity measures. In: *Proceedings of the 20th International Conference on World Wide Web*. USA: ACM; 2011:577-586.
33. Zhang Y-M, Huang K, Geng G, Liu C-L. Fast kNN graph construction with locality sensitive hashing. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. USA: Springer; 2013:660-674.
34. Eiras-Franco C, Martínez-Rego D, Kanthan L, et al. Fast distributed kNN graph construction using autotuned locality-sensitive hashing. *ACM Trans Intell Syst Technol (TIST)*. 2020;11(6):1-18.
35. Arnaiz-González Á, Díez-Pastor J-F, Rodríguez JJ, García-Osorio C. Instance selection of linear complexity for big data. *Knowl-Based Syst*. 2016;107:83-95.
36. Wang J, Qian T, Yang A, Wang H, Qian J. LSR-forest: an locality sensitive hashing-based approximate k-nearest neighbor query algorithm on high-dimensional uncertain data. *Concurrency Comput: Pract Exper*. 2020. https://doi.org/10.1002/cpe.5795
37. Eiras-Franco C, Kanthan L, Alonso-Betanzos A, Martínez-Rego D. Scalable approximate k-NN graph construction based on locality sensitive hashing. In: *25th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*; 2017.
38. Andoni A, Indyk P. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: *47th Annual IEEE Symposium on Foundations of Computer Science, 2006 (FOCS '06)*. USA: IEEE; 2006:459-468.

39. Muja M, Lowe DG. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*. 2009;2(331-340):2.
40. Baldi P, Sadowski P, Whiteson D. Searching for exotic particles in high-energy physics with deep learning. *Nat Commun*. 2014;5(1):4308.
41. Silpa-Anan C, Hartley R. Optimised KD-trees for fast image descriptor matching. In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE; 2008:1-8.
42. Krizhevsky A, Hinton G. *Learning Multiple Layers of Features from Tiny Images* [Technical Report]. Citeseer; 2009.
43. Netzer Y, Wang T, Coates A, Bissacco A, Wu B, Ng AY. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*; 2011.
44. Dua D, Graff C. *UCI Machine Learning Repository*; 2017.
45. Sonnenburg S, Franc V, Yom-Tov E, Sebag M. Pascal large scale learning challenge. In: *25th International Conference on Machine Learning (ICML 2008) Workshop*. Vol. 10, 2008:1937-1953. http://largescale.first.fraunhofer.de.J.Mach.Learn.Res
46. Lee W, Stolfo SJ. A framework for constructing features and models for intrusion detection systems. *ACM Trans Inf Syst Secur (TiSSEC)*. 2000;3(4):227-261.

How to cite this article: Eiras-Franco C, Guijarro-Berdiñas B, Alonso-Betanzos A, Bahamonde A. Scalable feature selection using ReliefF aided by locality-sensitive hashing. *Int J Intell Syst*. 2021;36:6161-6179. https://doi.org/10.1002/int.22546

APPENDIX A: SYMBOLS USED

See Table A1.

TABLE A1 List of symbols used

Symbol	Description
A	Attribute. Each of the input variables
$W[A]$	Weight assigned to attribute A by ReliefF
$\mathcal{D}$	Data set consisting of many input elements
x	Vector. Bold, small letter
$\sigma(x_i, x_j)$	Similarity between x_i and x_j (real number)
$\mathcal{O}(n)$	Computational complexity of order n
$\lvert\mathcal{D}\rvert$	Number of elements in $\mathcal{D}$
$b.\ size$	Number of elements contained in bucket b
$b.\ elems$	Set of elements contained in bucket b
$p.\ neighbors$	Neighbors of element p
$p.\ class$	Class of element p
c_v	Class label with value v
h	Hash
h_r	Augmented hash component for multiclass data sets
$\varnothing$	Empty set