SoftwareX 6 (2017) 193–197

Contents lists available at ScienceDirect

SoftwareX

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

Original software publication

featsel: A framework for benchmarking of feature selection algorithms and cost functions

Marcelo S. Reis [a,b,*], Gustavo Estrela [b,c], Carlos Eduardo Ferreira [c], Junior Barrera [b,c]

[a] Laboratório Especial de Ciclo Celular, Instituto Butantan, Brazil
[b] Center of Toxins, Immune-response and Cell Signaling (CeTICS), Instituto Butantan, Brazil
[c] Instituto de Matemática e Estatística, Universidade de São Paulo, Brazil

ARTICLE INFO

Article history:
Received 18 April 2017
Received in revised form 18 July 2017
Accepted 19 July 2017

Keywords:
Feature selection
Benchmarking
Boolean lattice
Combinatorial optimization

ABSTRACT

In this paper, we introduce featsel, a framework for benchmarking of feature selection algorithms and cost functions. This framework allows the user to deal with the search space as a Boolean lattice and has its core coded in C++ for computational efficiency purposes. Moreover, featsel includes Perl scripts to add new algorithms and/or cost functions, generate random instances, plot graphs and organize results into tables. Besides, this framework already comes with dozens of algorithms and cost functions for benchmarking experiments. We also provide illustrative examples, in which featsel outperforms the popular Weka workbench in feature selection procedures on data sets from the UCI Machine Learning Repository.

Code metadata

Code metadata description	
Current code version	v1.3.
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_SOFTX-D-16-00023
Legal Code License	GNUGeneralPublicLicensev3
Code versioning system used	git.
Software code languages, tools, and services used	C++, Perl
Compilation requirements, operating environments & dependencies	C++ compiler, make, Perl interpreter, gnuplot, flex (optional), bison (optional), groff (optional)
Link to developer documentation/manual	github.com/msreis/featsel/wiki
Support email for questions	marcelo.reis@butantan.gov.br

1. Motivation and significance

In the context of Machine Learning and Statistics, feature selection is a procedure to select, from a set S of features, a subset $X \subseteq S$ such that X contains the most relevant features for a given classifier design process. If the relevance of X can be measured through a cost function $c : \mathcal{P}(S) \rightarrow \mathbb{R}^+$, then feature selection is reduced to a combinatorial optimization problem called *feature selection problem*, in which the objective is to minimize $c(X)$. It is a well-known fact that the feature selection problem is NP-hard [1]; however, in situations where a cost function c measures an estimation error and is computed with a fixed number of samples, a property of c arises due to the "curse of dimensionality": adding new features to the considered subset X decreases the estimation error $c(X)$, until the point that the limitation of samples increases $c(X)$, resulting in a chain of subsets whose graph describes a U-shaped curve. This observation is taken into account in a special case of the feature selection problem: given an instance $\langle S, c\rangle$, if for every $X \subseteq Y \subseteq Z \subseteq S$ it holds that $c(Y) \leq max\{c(X), c(Z)\}$, then $\langle S, c\rangle$ is also an instance of the *U-curve problem* [2]. Although the U-curve problem is also NP-hard [3], many feature selection algorithms rely on strategies that model the search space as an instance of this problem: among them there are suboptimal algorithms (i.e., algorithms that do not guarantee finding a global minimum) like the sequential forward search (SFS) [4], and also optimal ones, which include the U-Curve Search (UCS) algorithm [5]. However, in practical feature selection instances, subset chains do not have perfect U-shaped curves, since such chains often present oscillations (i.e., one or more violations of the U-shaped curve assumption). Depending the level of these oscillations, only an exhaustive search could guarantee a global minimum, though there are some suboptimal algorithms that were

* Correspondence to: Avenida Vital Brasil, 1500. ZIP 05503-900, São Paulo, Brazil.
E-mail address: marcelo.reis@butantan.gov.br (M.S. Reis).

http://dx.doi.org/10.1016/j.softx.2017.07.005

designed to circumvent that issue; one example is the Best-First Search (BFS) [6].

Nonetheless, different choices for the cost function c impact on the performance of a feature selection algorithm. For example, in instances whose chains of subsets have a tendency of decreasing cost toward a global minimum (a condition that can be approximated by the Hamming distance cost function [3]), then greedy strategies such as SFS would be good choices for feature selection. However, if the instances have their global minima distributed throughout the search space and their chains describe a perfect U-shaped curve, then algorithms such as UCS would be suitable for an optimal search. Moreover, if these instances have oscillations in their chains (which occurs when the mean conditional entropy is used to estimate morphological operators [7]), then algorithms such as BFS, whose dynamics includes backtracking, could be employed for a suboptimal search.

Once suitable choices of both the cost function and the algorithm have relevant impact on the performance of the feature selection procedure, new approaches have been continually proposed. However, generally these new algorithms and/or cost functions are introduced with their own implementations, and very often using different programming languages and/or data structures, which makes difficult experimental benchmarking of them. This is aggravated in situations where constant and/or polynomial factors associated to the implementation are not overwhelmed by the asymptotic complexity of both algorithm and cost function. Moreover, general-purpose Machine Learning workbenches that offer feature selection procedures either are not designed to allow the inclusion of new algorithms and cost functions (e.g., the MLC++ library [8]) or are not implemented taking into account the mitigation of the aforementioned constant factors (e.g., the Weka workbench [9]). Therefore, there is a need for an efficient, standardized environment to allow easily programming and testing of different algorithms and cost functions, especially to compare new proposed solutions with well-established ones, the so-called "gold standards".

In this paper, we introduce featsel, an open-source framework for benchmarking of feature selection algorithms and cost functions. The core of this framework was coded in C++ and allows the user to deal with the search space as a Boolean lattice $(\mathcal{P}(S), \subseteq)$; this property is very helpful, since a subset X of the set of features S can be efficiently described as a characteristic vector of X, thus allowing the application of Boolean operations. Moreover, featsel includes auxiliary Perl scripts to minimize the efforts in adding new algorithms and/or cost functions, generating random instances, plotting graphs and organizing the benchmarking results into both LaTeX and hypertext markup language (HTML) tables. The framework is under GNUGPLv3license and is available for download at github.com/msreis/featsel. The remainder of this paper is organized as follows: in Section 2, we make a high-level description of the framework's main features and list the algorithms and cost functions that are already available in this release(v1.3). In Section 3, we present the overall software architecture, which includes a pictorial component overview of the system through a class diagram and description of its main classes. In Section 4, we show some illustrative examples in which, for data sets of different sizes, the performance of featsel is compared against the one of the Weka workbench. In Section 5, we indicate the expected impact of this framework, both within and without the Machine Learning academic community. Finally, in Section 6, we make some conclusion remarks about this work and point out ideas for future improvements.

2. Software description

featsel is a framework designed to assist systematic comparisons among feature selection algorithms and cost functions: for a given cost function, each algorithm is executed against one or more sets of instances of same type (e.g., with same number of features), and the obtained results are averaged and organized into a summary table. The execution of a single feature selection procedure is provided by the framework core (main program); this core is wrapped by a main auxiliary script, which is responsible to launch each of the algorithm executions, to organize the results and to generate the output files.

The core of featsel was designed through class-based, object-oriented modeling. The chosen object-oriented programming language to code the core was C++. Since benchmarking is our main objective, from the computational efficiency point of view, C++ has a better payoff in comparison with interpreted languages such as Java, which is relevant to reduce constant factors that could impact the comparison among algorithms and cost functions.

The main auxiliary script was coded in Perl, and offers to the user the possibility for using instances stored in a temporary input directory or generating random instances through customizable modules. There are also other Perl scripts to assist the inclusion and the removal of algorithms and cost functions. In Supplementary Material Section 2, we provide detailed examples of how to use all those scripts; additional information can be obtained at featsel' suserguide, which is available online at the project'srepository.

This frameworkrelease(v1.3) includes implementation of dozens of algorithms and cost functions, which are all listed in Tables 1 and 2.

3. Software architecture

As we have described in the previous section, the framework core was designed under the object-oriented programming paradigm. The main object interactions in the core are as follows: Features are elements of the system; the aggregation of several elements yields a set, while each set can be associated to a subset of it; subsets can be aggregated into collections of subsets, and also are associated to cost functions; solvers are composed of collections of subsets (e.g., to store lists of visited subsets) and of a cost function to compute a subset cost. Both cost function and solver are abstract classes, which means they serve as a basis for concrete implementations of cost functions and algorithms, respectively.

In Fig. 1, we summarize these interactions into a class diagram, which contains six main classes. The most relevant properties of these classes are discussed below.

Element. This class represents a feature, and has attributes and methods to store and retrieve its relevant properties. For example, if a feature consists in a pixel of a window during a morphological operator estimation using k samples, then an object of this class stores an array of k integers, each one corresponding to the observed value for that pixel in each of those samples.

ElementSet. An aggregation of elements that compose the complete set S considered during the feature selection procedure. This class has methods to load element data of a given set S from either `dat` (flat file) or `xml` (extended markup language) instance files. To this end, it relies on auxiliary parser classes *DatParserDriver* and *XmlParserDriver*, respectively.

ElementSubset. A subset X of a complete set S is represented as an instance of this class, which is used to explicitly represent its corresponding node in the search space and also to compute the cost of X. The subset representation is accomplished through the

Table 1
List of algorithms available in this release (v1.3) of the featsel framework. For each algorithm, it is highlighted its argument code in the featsel core, its class name within the framework architecture, and the reference for the publication in which it was originally published; "none" indicates an algorithm developed for this paper, whose details can be seen on Supplementary Material Section 3.1.

Algorithm	Argument code	Class name	Original publication
Best-First Search	bfs	BFS	[6]
Exhaustive Search	es	ExhaustiveSearch	[3]
Random Chain	rc	RandomChain	None
Sequential Backward Floating Search	sbfs	SBFS	[10]
Sequential Backward Search	sbs	SBS	[11]
Sequential Forward Search	sfs	SFS	[4]
Sequential Forward Floating Search	sffs	SFFS	[10]
Spectral Relaxation on Conditional Mutual Information	spec_cmi	SpecCMI	[12]
U-curve Branch and Bound	ubb	UcurveBranchandBound	[3]
U-Curve Search	ucs	UCurveSearch	[5]

Table 2
List of cost functions available in the release (v1.3) of the featsel framework. For each cost function, it is highlighted its argument code in the featsel core, its class name within the framework architecture, and the reference for the publication in which it was originally published; "none" indicates a cost function developed for this paper, whose details can be seen on Supplementary Material Sections 3.2 and 3.3.

Cost function	Argument code	Class name	Original publication
Atashpaz-Gargari–Braga-Neto–Dougherty	abd	ABD	[13]
Correlation-based Feature Selection	cfs	CFS	[14]
Conditional Mutual Information	cmi	ConditionalMutualInformation	[12]
Explicitly-declared cost function	explicit	Explicit	None
Hamming distance	hamming_distance	HammingDistance	[15]
Mean Conditional Entropy	mce	MeanConditionalEntropy	[7]
Mutual Information	mi	MutualInformation	[16]
Point outlier in otherwise constant function	point	Point	None
Subset Sum polynomial reduction	subset_sum	SubsetSum	[3]
Tailor Convex Hull	tailor	TailorConvexHull	[17]

usage of the characteristic vector of X, which is a function $\chi^X : S \mapsto \{0, 1\}$ defined as:

$$\chi^X(s) = \begin{cases} 1, & \text{if } s \in X; \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

Once this representation of the subset is binary, Boolean operations can be performed on χ^X through methods of this class that implement union, intersection and complement operations.

Collection. This class implements a collection of objects of the class *ElementSubset*. The main objective of this class is to provide a way to store subsets of the search space $(\mathcal{P}(S), \subseteq)$ that were already visited, and also to register the subsets of minimum cost found along a search procedure. To this end, this class has methods to add, remove and search subsets stored in a given collection.

CostFunction. This is an abstract class, that is, a class that serves as a "scaffolding" for the implementation of a cost function within the framework. To this end, the cost function concrete class inherits from *CostFunction* a virtual constructor and also a virtual method to compute the cost of a given subset. Therefore, the user must program the internal logic of her/his own cost function. For instance, for the *HammingDistance* class depicted in Fig. 1, one can compute and return the Hamming distance between a subset X received as argument of the *cost* method against a constant subset Y received as argument by the constructor of that class.

Solver. An abstract class used to assist the implementation of an algorithm within the framework. An algorithm concrete class inherits a virtual method called *get_minima_list*, which launches a search whose exact procedure must be programmed by the user. The object representing the cost function to be used during the search must be passed to the algorithm as an argument of a method called *set_parameters*, which in turn must be called before *get_minima_list*. For example, according to the diagram depicted in Fig. 1, an instantiated object of *ExhaustiveSearch* can be initialized with an object of *HammingDistance* and then compute the cost function for all $X \subseteq S$. In this case, the element of minimum cost will be the subset $Y \subseteq S$ that was used to instantiate the cost function object.

4. Illustrative examples

To display the efficiency of a feature selection procedure using featsel, we carried out computational experiments with data sets of different number of features and samples. Those sets were acquired from the University of California at Irvine (UCI) Machine Learning Repository [18]. Each data set was discretized (if necessary), converted into the featsel `dat` format and used in feature selections with the Java API of Weka workbench (version 3.6.1) and the featsel framework. Both programs employed their own implementations of the Correlation-based Feature Selection (CFS) cost function and either the Exhaustive Search (ES) or the BFS algorithm. It is important to remark that the pair {CFS,ES} or {CFS,BFS} should yield the same search results in both featsel and Weka, since both implement the same algorithm and cost function. All experiments were performed in a computer server with sixty-four AMD Opteron™ cores at 2.1 GHz each, 256 GB RAM and Ubuntu 14.04.5 LTS operating system.

In Table 3, we summarize the results of experiments carried out on eight different data sets (Arrhythmia, Gisette, Madelon, Musk2, Optdigits, Promoters, Waveform1 and Zoo). From the computational point of view, for all considered data sets, the featsel framework consistently outperformed the Weka workbench. This comparison can be reproduced in other computational environments through a Perl program called *compare_featsel_against_Weka.pl*, which is included in this framework release.

The capabilities of the featsel framework are further illustrated in Supplemental Material Sections 4.1 and 4.2, with the application of featsel on subset sum and Zoo instances, respectively. These additional examples employ the SFS and UCS algorithms and also illustrate the graphs that are automatically-generated by the benchmarking program.

5. Impact

In the field of Machine Learning, new feature selection algorithms are being continually proposed. However, the lack of efficient software options for systematic comparisons of these new

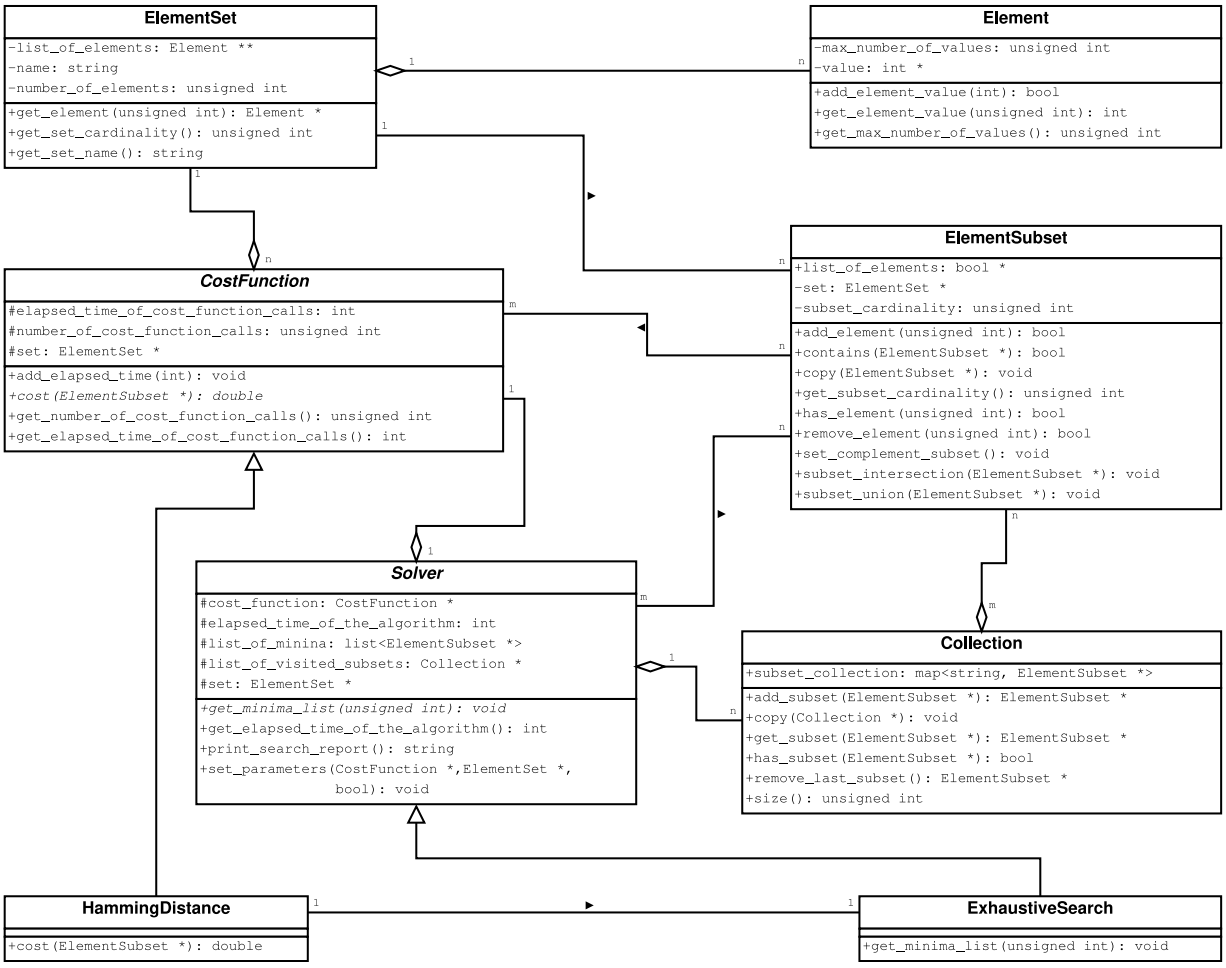

Fig. 1. Class diagram in unified modeling language (UML) of featsel core, including key class attributes and methods. In this diagram, it is exemplarily showed the derivation of a cost function concrete class *HammingDistance* and of an algorithm concrete class *ExhaustiveSearch*. An object of *ExhaustiveSearch* can be instantiated using an object of *HammingDistance* as the cost function.

Table 3
Performance comparison between the Weka workbench and this release (v1.3) of the featsel framework. For each data set and for each software, the indicated algorithm and cost function were executed 20 times. After that, it was calculated the mean and deviation for the required computational time.

Data set	# Features	# Samples	Algorithm	Cost function	Weka required time (s)	featsel required time (s)
Zoo	15	101	ES	CFS	0.4010 ± 0.0054	**0.1331 ± 0.0029**
Waveform1	21	5000	BFS[a]	CFS	0.5353 ± 0.0050	**0.0540 ± 0.0007**
Promoters	57	106	BFS[a]	CFS	0.3733 ± 0.0061	**0.0098 ± 0.0003**
Optdigits	64	5620	BFS[a]	CFS	0.8988 ± 0.0315	**0.3748 ± 0.0344**
Musk2	166	6598	BFS[a]	CFS	1.0985 ± 0.0213	**0.3722 ± 0.0045**
Arrhythmia	279	452	BFS[a]	CFS	1.8305 ± 0.3412	**0.5777 ± 0.0054**
Madelon	500	2600	BFS[a]	CFS	1.4246 ± 0.0207	**0.6118 ± 0.0100**
Gisette	5000	6000	BFS[a]	CFS	365.8821 ± 52.9896	**130.5534 ± 6.3817**

[a] BFS was executed in forward direction and terminating search after five non-improving nodes.

ideas against the already-established solutions impairs a proper evaluation of them. Therefore, this framework can be useful to assist the development of new methods in this field.

Moreover, feature selection is context-dependent, which implies that for a given set of problems (i.e., constraint on the possible cost functions to be adopted), algorithms that are more suited to tackle them can change dramatically—by "suited" it could be from the computational point of view, optimality of the obtained result, or both. Therefore, an efficient, systematic benchmarking among different pairs of algorithms and cost functions has the potential of enhancing the choice of the best suited ones for a feature selection problem in a specific context.

Finally, since feature selection is a procedure required in many fields (to mention a few of them: Bioinformatics, Statistics, Image Processing, Mathematical Morphology, etc.), we believe that the featsel framework has the potential to have a widespread usage also outside the Machine Learning academic community.

6. Conclusion

In this paper, we introduced featsel, an open-source, C++ coded framework to assist the benchmarking of algorithms and cost functions that are used to solve the feature selection problem. featsel includes auxiliary Perl scripts to assist the adding of new algorithms and/or cost functions, generating random instances, plotting graphs and organizing the benchmarking results into LaTeX and HTML tables. Additionally, we highlighted the performance

of featsel through computational experiments, in which it outperformed the Weka workbench during feature selection procedures on eight data sets from the UCI Machine Learning Repository

Currently, we are working on the addition of new algorithms and cost functions, as well as new types of graphs. Among other ongoing improvements on featsel are: usage of reduced ordered binary decision diagrams for a more efficient representation of elements removed from the search space [19]; parallelization of feature selection procedure calls in the main Perl auxiliary script; inclusion into the framework's C++ core of methods for implementation of parallelized algorithms, which will be achieved using the OpenMP library [20].

Acknowledgments

The authors thank Ronaldo F. Hashimoto and Edu Sanchez-Castro for their comments and suggestions, and Leo K. Iwai for the revision of this paper. We also thank the anonymous reviewers for their helpful remarks. This work was supported by scholarship #142039/2007-1 and grant #305705/2014-8 from CNPq, and by grants #2013/03447-6, #2013/07467-1, #2015/01587-0 and #2016/25959-7, São Paulo Research Foundation (FAPESP).

Appendix A. Supplementary data

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.softx.2017.07.005.

References

[1] Guyon I, Elisseeff A. An introduction to variable and feature selection. J Mach Learn Res 2003;3(Mar):1157–82.
[2] Ris M, Barrera J, Martins-Jr DC. U-curve: A branch-and-bound optimization algorithm for U-shaped cost functions on Boolean lattices applied to the feature selection problem. Pattern Recognit 2010;43(3):557–68.
[3] Reis MS, Minimization of decomposable in U-shaped curves functions defined on poset chains –algorithms and applications, Ph.D. thesis, Institute of Mathematics and Statistics, University of São Paulo, Brazil, (in Portuguese), 2012.
[4] Whitney AW. A direct method of nonparametric measurement selection. IEEE Trans Comput 1971;20(9):1100–3.
[5] Reis MS, Ferreira CE, Barrera J, The U-curve optimization problem: improvements on the original algorithm and time complexity analysis, 2014. ArXiv Preprint ArXiv:1407.6067, pp. 1–30.
[6] Kohavi R, John GH. Wrappers for feature subset selection. Artif Intell 1997;97(1–2):273–324.
[7] Martins-Jr DC, Cesar-Jr RM, Barrera J. W-operator window design by minimization of mean conditional entropy. PAA Pattern Anal. Appl. 2006;9(2):139–53.
[8] Kohavi R, John G, Long R, Manley D, Pfleger K. MLC++: A machine learning library in C++. In: Sixth international conference on tools with artificial intelligence. IEEE; 1994. p. 740–3.
[9] Holmes G, Donkin A, Witten IH. Weka: A machine learning workbench. In: Proceedings of the second Australian and New Zealand conference on intelligent information systems. IEEE; 1994. p. 1–5.
[10] Pudil P, Novovicová J, Kittler J. Floating search methods in feature selection. Pattern Recognit Lett 1994;15(11):1119–25.
[11] Marill T, Green D. On the effectiveness of receptors in recognition systems. IEEE Trans. Inform. Theory 1963;9(1):11–7.
[12] Nguyen XV, Chan J, Romano S, Bailey J. Effective global approaches for mutual information based feature selection. In: Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining. ACM; 2014. p. 512–21.
[13] Atashpaz-Gargari E, Braga-Neto UM, Dougherty ER. Improved branch-and-bound algorithm for U-curve optimization. In: IEEE international workshop on genomic signal processing and statistics. IEEE; 2013. p. 100–1.
[14] Hall MA, Correlation-based feature selection of discrete and numeric class Machine Learning, Tech. rep., University of Waikato, Department of Computer Science, 2000, pp. 1–10.
[15] Hamming RW. Error detecting and error correcting codes. Bell Syst Tech J 1950;29(2):147–60.
[16] Barrera J, Cesar-Jr RM, Martins-Jr DC, Vêncio RZN, Merino EF, Yamamoto MM, Leonardi FG, Pereira CAB, del Portillo HA. Constructing Probabilistic Genetic Networks of *Plasmodium falciparum* from dynamical expression signals of the intraerythrocytic development cycle. In: Methods of microarray data analysis V. Springer; 2007. p. 11–26.
[17] Reis MS, Barrera J. Solving problems in Mathematical Morphology through reductions to the U-curve problem. In: International symposium on mathematical morphology and its applications to signal and image processing. Springer; 2013. p. 49–60.
[18] Lichman M. UCI machine learning repository. Irvine: University of California, School of Information and Computer Sciences; 2013 URL http://archive.ics.uci.edu/ml.
[19] Bryant RE. Graph-based algorithms for Boolean function manipulation. IEEE Trans Comput 1986;100(8):677–91.
[20] Dagum L, Menon R. OpenMP: An industry standard API for shared-memory programming. IEEE Comput Sci Eng 1998;5(1):46–55.