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-D-17-00031
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 $c : \mathcal{P}(S) \to \mathbb{R}^+$, 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