SN Computer Science (2020) 1:40
https://doi.org/10.1007/s42979-019-0037-5

ORIGINAL RESEARCH

Multidimensional Feature Selection and High Performance ParalleX

A tool for detection of informative variables for big data

Karol Niedzielewski[1] · Maciej E. Marchwiany[1] · Radoslaw Piliszek[2] · Marek Michalewicz[1] · Witold Rudnicki[1,2,3]

Received: 22 July 2019 / Accepted: 2 September 2019
© The Author(s) 2019

Abstract
Great amount of stored information used in connection with Machine Learning and statistical methods enables high quality insight and analysis of data that leads to design of high precision predictive and classification systems. In the process of analysis, selection of most informative features is crucial for later quality of the designed system. In this report, we propose two implementations of multidimensional feature selection (MDFS) algorithm (Piliszek et al. in Mdfs-multidimensional feature selection. arXiv preprint. arXiv:1811.00631, 2018) that can be used in distributed environments for detection of all-relevant variables in data sets with discrete decision variable. While most methods discard information about interactions between features, MDFS is designed towards identification of informative variables that are not relevant when considered alone but are relevant in groups. We have developed software using C++ and High Performance ParalleX (HPX) (Kaiser et al. in STEllAR-GROUP/hpx: HPX V1.3.0: the C++ Standards library for parallelism and concurrency. 2019. https://doi.org/10.5281/zenodo.3189323, 2019) to achieve best performance, great scalability and portability. HPX is a library that uses lightweight threads, asynchronous communication, and asynchronous task submission based on the declarative criteria of work. These features enabled us to deeply explore granularity and parallelism of the MDFS algorithm. Software is prepared entirely in C++; therefore, calculations can be performed using CPUs on desktops, distributed systems, and any system with C++ compiler support. During testing on Cray XC40 (Okeanos) using artificially prepared data, we achieved 196 times acceleration on 256 nodes compared to a single node. From this point, ICM computing facility is capable of massively parallel feature engineering. The main purpose of the software is to enable researchers for more accurate genomics data analysis in search for multiple correlations in potential sources of the diseases.

Keywords Multidimensional feature selection · Mutual information · HPX · Distributed systems · Big data · Genomics

Introduction

In the twenty-first century, we have been introduced with new fourth scientific discovery paradigm that is based on data [4]. Growth of computing capabilities in connection with big data manipulation and analysis gives us new tools for broadening knowledge and bringing new scientific breakthroughs.

Every data analysis begins with data introspection and evaluation. Finding information that is relevant for phenomena under scrutiny is inseparable part of every knowledge development; additionally, reduction of problem dimensionality and disposing of all irrelevant information lead to storage savings and allows faster data manipulation during next steps of the analysis. The analysed data can contain variables that are relevant only when considered

This article is part of the topical collection "Modelling methods in Computer Systems, Networks and Bioinformatics" guest edited by Erol Gelenbe.

✉ Karol Niedzielewski
k.niedzielewski@icm.edu.pl

[1] Interdisciplinary Centre for Mathematical and Computational Modelling (ICM), University of Warsaw, Warsaw, Poland

[2] Computational Centre, University of Bialystok, Białystok, Poland

[3] Institute of informatics, University of Bialystok, Białystok, Poland

Published online: 24 October 2019

in groups, which is complicating the case even further. These facts are not negligible, but investigation of new phenomena when domain specific knowledge is missing or limited is hard and feature engineering is even harder. In such cases, it is highly likely that additional irrelevant information will be stored or relevant information will be disposed.

There are many methods dedicated to the problem of feature selection and they fall into three main categories [2]:

- filters where the identification of informative variables is performed before data modelling and analysis,
- wrappers where the identification of informative variables is achieved by analysis of the models,
- embedded methods which evaluate utility of variables in the model and select the most useful variables.

An example of feature selection method is multidimensional feature selection (MDFS) algorithm that was invented for identification of all variables in data set that have influence on discrete decision variable. It was specifically designed towards identification of informative variables that are not relevant when considered alone but are relevant in groups. Its design introduces great granularity of calculations that allows computations to be intensively parallelized across multiple processing units and in this case multiple computational nodes.

In this report, we introduce the first known implementation of this method for distributed environments. We want to provide scientists with a new tool that can accelerate their feature engineering workflows or enable bigger data sets to be analysed in search for high dimensional correlations.

Genomics Motivation

The exhaustive search for non-trivial interactions is particularly important for analysis of genetic data. The synergistic effects of double mutations have been established for example in connection with insectide resistance in pests [12] or various phenotypic effects in *Arabidopsis thaliana* [9].

Traditional approach for identification of synergies requires lengthy experimental work in genetics targeted on a handful of genes. On the other hand, analysis of large data sets collected in population studies opens possibility to perform large-scale analysis of entire genomes without a priori knowledge which genes are interesting. Such analysis involves non-trivial computations, since the number of variables that could be possibly relevant in such analysis may even reach several millions. Consequently, exhaustive analysis of pairwise interaction may require computation of $10^{12} - 10^{14}$ pairwise interactions, which will be possible using massively parallel computers.

Theory

Variable $x_i \subset X$ is considered weakly relevant when there exists such a subset of variables $X_{\text{sub}} \subset X : x_i \notin X_{\text{sub}}$ that one can increase information about the decision variable y by extending this subset with the variable x_i [7]. To restrict and strictly define dimensionality of analysed problem, Mnich and Rudnicki [8] introduced the notion of k-weak relevance that encloses original definition to $(k-1)$-element subsets X_{sub}. Dimensionality equals k in this case.

By applying this notion to information theory and by calculating difference between conditional entropies of subsets X_{sub} and $X_{\text{sub}} \cup \{x_i\}$, we are able to measure k-weak relevance of variable x_i. This difference in (conditional) information entropy H is known as (conditional) mutual information and was named information gain (IG) by authors of MDFS method. After computation of mutual information for each subset X_{sub} (identified by value m) from all subsets, we are able to find maximum information gain that reflects the biggest mutual information of that variable. This procedure is described by formula (1).

$$\begin{aligned} \text{IG}^k_{\max}(y; x_i) = N \max_m (H(y|x_{m_1}, \ldots, x_{m_{k-1}}) \\ - H(y|x_i, x_{m_1}, \ldots, x_{m_{k-1}})). \end{aligned} \tag{1}$$

For one-dimensional analysis ($k = 1$), formula (1) reduces to Eq. (2).

$$\text{IG}^1_{\max}(y; x_i) = N(H(y) - H(y|x_i)). \tag{2}$$

To find the biggest mutual information gain, it is required to generate all unique k-element subsets. The number of all unique k-element subsets is given by equation:

$$S = \binom{I}{k} = \frac{I!}{k!(I-k)!}, \tag{3}$$

where S represents the number of unique subsets, I represents the number of variables, and k represents the number of elements in subset (number of dimensions). Therefore, computation complexity grows rapidly with number of variables I and dimensions k.

Computation of entropy requires variables to have discrete values. Therefore, continues variables need to be discretized in order to be tested by MDFS method.

Algorithm Implementation

Algorithm

Mathematical formula of MDFS method (1) reveals high granularity of the algorithm. Each conditional entropy can

be computed independently and no information exchange is required until search for maximum IG. Therefore, computations can be heavily parallelized.

We prepared two versions of the algorithm. The first one is optimized for minimalization of preformed computations and the second one is optimized for minimalization of communication and data synchronization.

First Algorithm Description

For every k-element subset X_{sub} (this is different notion of X_{sub} than in Section "Theory") computation can be carried indepently and obey the following steps:

1. Compute contingency table for X_{sub},
2. Compute reduced contingency table for each $X_{sub} \backslash x_i : x_i \subset X_{sub}$,
3. Compute IG from contingency tables for each x_i using following formula (5).

$$IG(y; x_i) = N(H(y|X_{\text{sub}} \backslash x_i) - H(y|X_{\text{sub}})). \tag{4}$$

4. Search for maximum IG for each x_i and store it in the vector.

We are aware that information entropies of reduced contingency tables are computed redundantly in the first algorithm. In computationally optimal scenario, we are required to compute information entropies of reduced contingency tables and store them in a vector. Then, we can continue with computations of contingency tables X_{sub}. Unfortunately, this approach requires more iterations over data (data movement is the most computationaly expensive part), more synchronization, and more data dependencies.

Second Algorithm Description

1. Compute contingency table for $X_{\text{sub}} \cup \{x_i\}$,
2. Compute contingency table for X_{sub},
3. Compute IG from contingency tables using following formula (5).

$$IG(y;x_i) = N(H(y|X_{\text{sub}}) - H(y|X_{\text{sub}} \cup \{x_i\})), \tag{5}$$

4. Store IG for x_i.

In the end search maximum IG for each x_i.

Algorithm proposed by Mnich and Rudnicki [8] is applied to compute conditional entropy from formula (5).

High Performance ParalleX

High Performance ParalleX (HPX) is an implementation of a theoretical execution model for extreme-scale computing systems called Parallex [5] and offers several new approaches to parallelism:

- Active Global Address Space (AGAS)—to provide great control over a flow of a program, processes and variables in a system have their unique addresses,
- Constraint-based synchronization—parallel programming is based on declarative criteria of work. In other words, programmer is required to define dependencies between data and computations in order to allow runtime system to build dependency graph between tasks. Then, during program execution, finishing of one task automatically activates others and can be compared to dominoes falling and pushing each other in a sequence. It provides great asynchronism in a system but does ensure synchronization,
- Lightweight threads—to ensure high concurrency and granularity of tasks, HPX provides lightweight threads. This approach makes dynamic scheduling faster and fundamentally reduces overheads of switching between threads,
- Data directed execution—to avoid movement of data whenever possible, HPX is designed to allow asynchronous commissioning of tasks on a remote nodes to do calculations there and to receive answers that are relevant locally. This approach can be summarized in a maxim: "move work to data, keep data local".

HPX library is completely written in C++. It is available on github[1] under a liberal open-source license and has an open, active, and thriving developer community.

We regard HPX library as a perfect fit to our needs, because it allows deep exploitation of granularity of the algorithm. A programmer is required to provide data tasks dependencies, afterwards scheduling with synchronization is provided by a runtime system. It allows the developer to put more effort in solving the problem and consequently reduces development time. Therefore, we used this library in development of distributed version of MDFS method.

Parallelization and Distribution

To calculate $IG^k_{\max}(y;x_i)$ from formula (1), we need to generate all k-element subsets X_{sub} and than apply algorithm from Section "Algorithm". The subsets should be generated in a way that provides distribution of computations, avoids repetition of generated subsets, allows reasonable load balancing and maintains enough concurrency to utilize computing resources efficiently.

[1] https://github.com/STEllAR-GROUP/hpx.

We propose the following method to fulfil the above requirements for the "First Algorithm Description:

1. Generation of all k-element subsets X_{sub} is completed with iteration over all possible variables. An example of generated sequence for three-element subset with five variables is the following:
   $[[x_1, x_2, x_3], [x_1, x_2, x_4], [x_1, x_2, x_5], [x_1, x_3, x_4], [x_1, x_3, x_5], [x_2, x_3, x_4], [x_2, x_3, x_5], [x_2, x_4, x_5], [x_3, x_4, x_5]]$
2. Distribution of work is achieved by spliting a list of variables between nodes. Each node acquires list of variables. Then, for each variable on the list it generates all unique subsets of variables that start with variable in question. The first element of a subset is not incremented in this case. An example of generated sequence for three-element subset and five variables with x_2 in a list on a single node is the following: $[[x_2, x_3, x_4], [x_2, x_3, x_5], [x_2, x_4, x_5]]$.
3. To achieve decent load balancing, variables are assigned to nodes during an "up&down" iteration over nodes IDs. An example of assignement of ten variables to three nodes is the following:
   $x_1 \rightarrow n_1, x_2 \rightarrow n_2, x_3 \rightarrow n_3, x_4 \rightarrow n_3, x_5 \rightarrow n_2, x_6 \rightarrow n_1, x_7 \rightarrow n_1, x_8 \rightarrow n_2, x_9 \rightarrow n_3, x_{10} \rightarrow n_3$ In result following mapping is optained:
   $n_1 : [x_1, x_6, x_7], n_2 : [x_2, x_5, x_8], n_3 : [x_3, x_4, x_9, x_{10}]$.
   Data reading and discretization is performed on each node independently. It is possible to perform multiple discretizations during one run.
4. In order to provide concurrency on nodes, each element in an assigned list is computed in parallel. In other words, each subset sequence generation and processing is seperated into n tasks where n is the number of variables in the list.

The disadvantages of this approach are: requirement for a vector of maximum IGs that needs to be updated synchronously (critical section) and fact that algorithm will not be scalable over $N / 2$ processors where N is the number of variables. However, our tests in the next chapter prove that software achieves great scalability and performance with this parallelization method.

We propose the following method to fulfil the above requirements for the "Second Algorithm Description":

1. Distribution of work is achieved by spliting list of variables between nodes. To achieve load balancing, variables are assigned to nodes in equal chunks. An example of ten variables divided between three nodes is the following: $n_1 : [x_1, x_2, x_3], n_2 : [x_4, x_5, x_6, x_7], n_3 : [x_8, x_9, x_{10}]$. Please note that 10 is not divisible by 3; therefore, node no. 2 has four elements on a list. The distribution is designed in a way that provides consistent partition and no variable is omitted. Data reading and discretization are performed on each node independently. It is possible to perform multiple discretizations during one run.
2. Each variable on the list is processed independently. Generating of all k-element subsets X_{sub} for each variable from the list is completed with iteration over all possible variables without incrementing the first element. A variable from the list x_i is the first element and remains constant in this case. An example of generated sequence for three-element subset and four variables with x_2 as the first element is the following:
   $[[x_2, x_1, x_2], [x_2, x_1, x_3], [x_2, x_1, x_4], [x_2, x_2, x_3], [x_2, x_2, x_4], [x_2, x_3, x_4]]$,
3. To provide concurrency on nodes, each maximum IG of variables on an assigned list is computed in parallel. In other words, each subset sequence generation and processing is seperated into N tasks where N is the number of variables in the list.

The disadvantage of this approach is the fact that there are redundant computations but the algorithm will be scalable to N sockets where N is the number of variables.

For big data use cases (when data cannot fit into a single node operating memory), the software is equipped with a chunk processing option. In this scenario, the data are read and processed in chunks and should already be discretized, because discretization cannot be performed in this scheme.

Our tool can read files in CSV and HDF5 format.

Results

The tests were performed on Cray XC40 machine (Okeanos) that is installed at ICM facility. It is composed of 1084 computing nodes. Each node has 24 Intel Xeon E5-2690 v3 CPU cores with a 2-way Hyper Threading (HT) with 2.6 GHz clock frequency. The nodes are connected with a Cray Aries Network with a Dragonfly topology.

For demonstration and testing of the MDFS implementation, two datasets were used.

1. Madelon [3] is a synthetic dataset with 2000 objects and 500 variables that can be accessed from the UCI Machine Learning Repository [1],
2. Neuroblastoma is data set containing information on expression levels of 340414 exon/intron junctions measured for 498 neuroblastoma patients with the help of RNA-seq method [11].

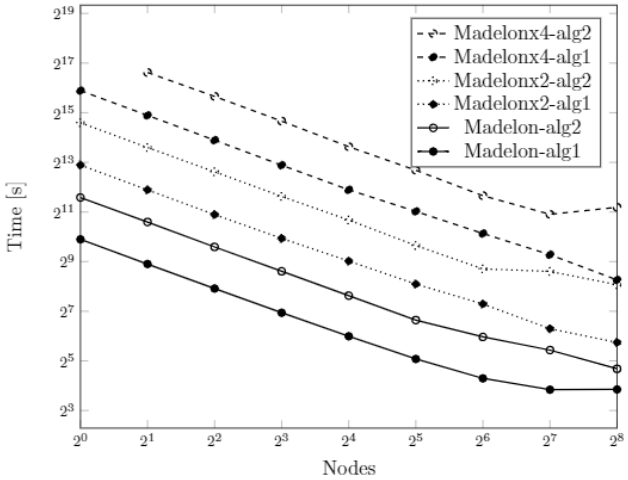

Fig. 1 Measured time of three-dimensional analysis with 100 discretizations on different Madelon dataset sizes and with two algorithm implementations (e.g. dataset that is twice as big has two times the number of variables. Additional variables are copies of variables from the original data set)

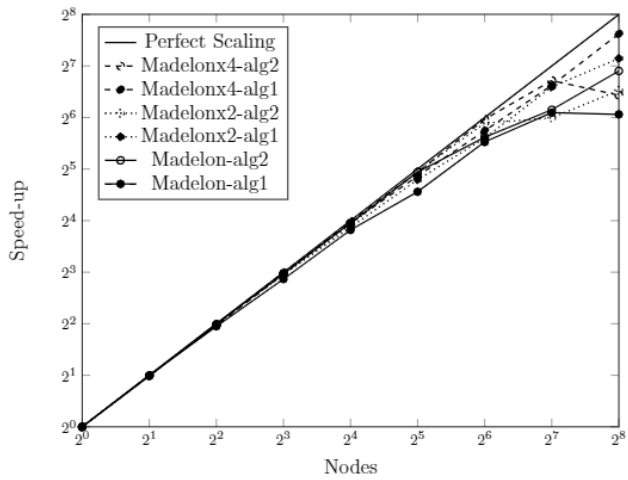

Fig. 2 Measured speedup of three-dimensional analysis with 100 discretizations on different Madelon dataset sizes and with two algorithm implementations (e.g. dataset that is twice as big has two times the number of variables. Additional variables are copies of variables from the original data set)

To provide multiple sizes of datasets for testing, additional datasets were generated via duplication of columns and rows from the original datasets.

Madelon Tests

We decided to perform three-dimensional analysis tests because it is the minimal size of the problem that allows presentation of performance up to 256 nodes. The measured time of the analysis performed on different sizes of Madelon dataset is presented in Fig. 1.

We can observe linear scalability of both algorithms up to 256 nodes (6144 cores). The first algorithm is faster and presents better scalability. Please note that Madelon analysis time using the first algorithm is stagnant for 128 (2^7) and 256 (2^8) nodes. These phenomena appear because the first application is not scalable over N / 2 processors where N is the number of variables (please see "First Algorithm Description"). Speed-up of analysis is shown in Fig. 2.

Tests of big data scenario were performed on artificially increased Madelon data sets (the sets had 1GB, 2GB, 4GB, 8GB, 16GB, 32GB, 64GB). In this case, additional rows were duplicated. We present perfect weak scaling results for various dataset sizes ranging from 1 to 64 GB with constant processing time of around 1 h 45 min (~ 6500 s) in Fig. 3. We can see a growth of performance in Fig. 4. The performance is defined as the ratio of the size of the problem and the analysis time.

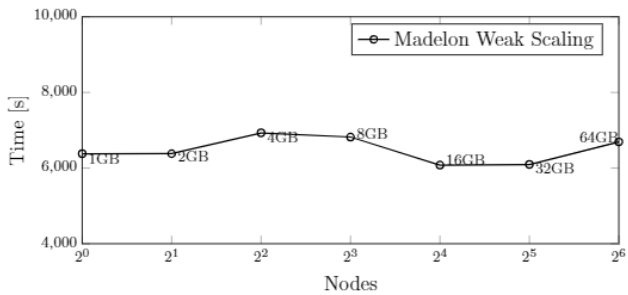

Fig. 3 Measured time of three-dimensional analysis of different sizes of Madelon data set in big data scenario(the sets had 1GB, 2GB, 4GB, 8GB, 16GB, 32GB, 64GB. In this case, increase was in the number of rows in the dataset. New rows are copies of the rows from the original Madelon dataset)

Neuroblastoma Tests

Measured time of one- and two-dimensional analysis performed on Neuroblastoma data set using the first algorithm is presented in Fig. 5. We did not perform 3 and more dimensional analysis because of big processing time requirements. Please note that the size of the problem is proportional to formula:

$$S \sim N^D, \tag{6}$$

where S represents the size of the problem, N represents the number of variables, and D represents the number of dimensions. Please note that one-dimensional analysis time is reduced below 100 s. It is the reason of scalability "saturation" beause IO and other sequential sections of implementation start to prevail.

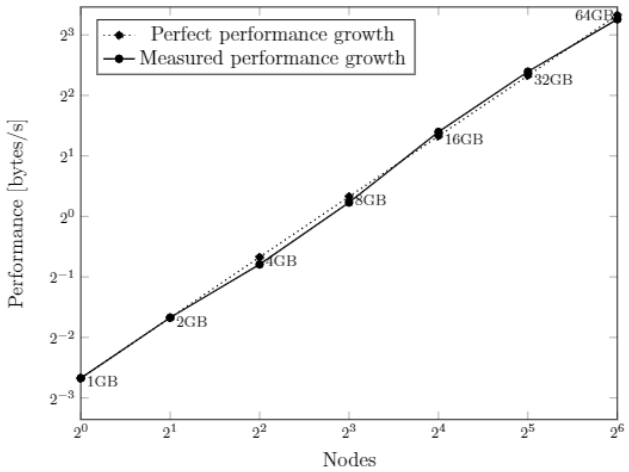

Fig. 4 Measured performance defined as ratio of the size of the problem and the analysis time. Presented plot data is acquired with three-dimensional analysis of different sizes of Madelon dataset in big data scenario (the sets had 1GB, 2GB, 4GB, 8GB, 16GB, 32GB, 64GB. In this case, increase was in number of rows in the dataset. New rows are copies of the rows from the original Madelon dataset)

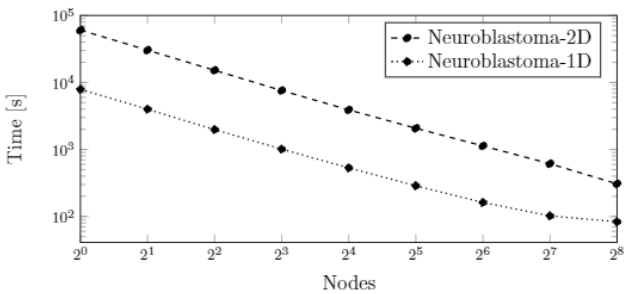

Fig. 5 Measured time of one- and two-dimensional neuroblastoma dataset analysis

Conclusions

We presented MDFS algorithm for distributed systems in two implementations. The first one with less computations but with more synchronization and the second one with more computations, but with less synchronization. It turns out that the first implementation performs better and presents better scalability up to 256 nodes (strong scaling).

Parallelization with ParalleX leads to the reduction of processing time of feature analysis of genomics data for Neuroblastoma dataset. We demonstrate that our approach is able to reduce time of feature engineering analysis of genomics data by two orders of magnitude from ~ 6.5 h (22933.3s) to ~ 3 min (175.4 s). We also present perfect weak scaling results for various dataset sizes ranging from 1 to 64 GB with constant processing time of around 1 h 45 min (~ 6500 s).

We are planning to use this research tool for analysis of a variety of genomics data sets at ICM.

Future Work

In future work, we will pursue further optimization of presented MDFS implementation. This will include optimization of present algorithm as well as seeking new ways of solving the problem.

Additionally, we want to expand tool with new features. One such opportunity is detection of subsets that give the biggest information gain for each variable.

Acknowledgements This research was carried out with the support of the Interdisciplinary Centre for Mathematical and Computational Modelling (ICM) University of Warsaw under Grant no G66-21.

Compliance with Ethical Standards

Conflict of Interest The authors declare that they have no conflict of interest.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

1. Dua D, Graff C. Uci machine learning repository. http://archive.ics.uci.edu/ml. 2017. Accessed 18 Oct 2019.
2. Guyon I, Elisseeff A. An introduction to variable and feature selection. J Mach Learn Res. 2003;3:1157–82.
3. Guyon I, Gunn S, Ben-Hur A, Dror G. Result analysis of the nips 2003 feature selection challenge. In: L.K. Saul, Y. Weiss, L. Bottou, editors. Advances in neural information processing systems 17. MIT Press; 2005. pp. 545–552. http://papers.nips.cc/paper/2728-result-analysis-of-the-nips-2003-feature-selection-challenge.pdf. Accessed 18 Oct 2019.
4. Hey AJ, Tansley S, Tolle KM, et al. The fourth paradigm: data-intensive scientific discovery, vol. 1. Redmond: Microsoft Research; 2009.
5. Kaiser H, Brodowicz M, Sterling T. Parallex an advanced parallel execution model for scaling-impaired applications. In: 2009 International Conference on Parallel Processing Workshops, pp. 394–401. IEEE. 2009.
6. Kaiser H, aka wash B.A.L, Heller T, Berg A, Simberg M, Biddiscombe J, Bikineev A, Mercer G, Schfer A, Serio A, Kwon T, Huck K, Habraken J, Anderson M, Copik M, Brandt S.R, Stumpf M, Bourgeois D, Blank D, Jakobovits S, Amatya V, Viklund L, Khatami Z, Bacharwar D, Yang S, Diehl P, Schnetter E, Gupta N, Wagle B. Christopher: STEllAR-GROUP/hpx: HPX V1.3.0: the C++ Standards library for parallelism and concurrency. 2019. https://doi.org/10.5281/zenodo.3189323.
7. Kohavi R, John GH. Wrappers for feature subset selection. Artif Intell. 1997;97(1–2):273–324.

8. Mnich K, Rudnicki WR. All-relevant feature selection using multidimensional filters with exhaustive search. CoRR abs/1705.05756. 2017. arXiv:abs/1705.05756.
9. Pérez-Pérez JM, Candela H, Micol JL. Understanding synergy in genetic interactions. Trends Genet. 2009;25(8):368–76.
10. Piliszek R, Mnich K, Migacz S, Tabaszewski P, Sulecki A, Polewko-Klim A, Rudnicki W. Mdfs-multidimensional feature selection. arXiv preprint. 2018. arXiv:1811.00631.
11. Zhang W, Yu Y, Hertwig F, Thierry-Mieg J, Zhang W, Thierry-Mieg D, Wang J, Furlanello C, Devanarayan V, Cheng J, et al. Comparison of rna-seq and microarray-based models for clinical endpoint prediction. Genome Biol. 2015;16(1):133.
12. Zhang Y, Meng X, Yang Y, Li H, Wang X, Yang B, Zhang J, Li C, Millar NS, Liu Z. Synergistic and compensatory effects of two point mutations conferring target-site resistance to fipronil in the insect gaba receptor rdl. Sci Rep. 2016;6:32335.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.