A Novel Tool For Fast Feature Selection

Ivan Alejandro Garcia Ramirez (✉ iagarcia@gdl.cinvestav.mx)

Centro de Investigacion y de Estudios Avanzados del Instituto Politecnico Nacional

<https://orcid.org/0000-0001-6260-7359>

Arturo Calderon

Centro de Investigacion y de Estudios Avanzados del Instituto Politecnico Nacional

Andrés Méndez

Centro de Investigacion y de Estudios Avanzados del Instituto Politecnico Nacional

Susana Ortega

Centro de Investigacion y de Estudios Avanzados del Instituto Politecnico Nacional

Software article

Keywords: Machine learning, Feature selection, Python tool

Posted Date: August 18th, 2020

DOI: <https://doi.org/10.21203/rs.3.rs-60546/v1>

License: This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

RESEARCH

A Novel Tool For Fast Feature Selection

Ivan Alejandro Garcia Ramirez^{*}
, Arturo Calderón
, Andrés Méndez Vásquez
and Susana Ortega Cisneros

^{*}Correspondence:

iagarcia@gdl.cinvestav.mx
Centro de Investigación y de
Estudios Avanzados del IPN, Av
del Bosque 1145, El Bajío, 45017
Zapopan, Jal., 45017 Zapopan,
México

Full list of author information is
available at the end of the article

[†]Equal contributor

Abstract

Motivation: Datasets with high dimensionality represent a challenge to existing learning methods. The presence of irrelevant and redundant features in a dataset can degrade the performance of the models inferred from it. In large datasets, manual management of features tends to be impractical. Therefore, the development of automatic discovery techniques to remove useless features has attracted increasing interest. In this paper, we propose a novell framework to select relevant features in supervised datasets.

Availability: This tool can be downloaded from
<https://github.com/ivangarcia88/ffselection>

Results: This tool allow to identify relevant and remove redundant features, reducing computation time on training a machine learning model while improving the performance.

Keywords: Machine learning; Feature selection; Python tool

Introduction

The current capacity of storage devices has allowed saving a large amount of data in diverse phenomena. The possibility of using this information more than just having records, has made *artificial intelligence* techniques based on pattern recognition quite attractive. Techniques such as machine learning have played a fundamental role in exploiting the potential of data that we have today. These techniques consist mainly of identifying patterns in data, and these patterns are usually essential to generate valuable insights in the fields that are being used.

Despite the success of machine learning algorithms in generating an approximate model to describe and predict such phenomena, the process usually depends on various preprocessing algorithms to reduce some of the issues in automatic modeling, one of them, which is known as the curse of dimensionality, which refers to the issue that arises when you have a high number of dimensions.

There are three kinds of features in a dataset: relevant, irrelevant, and redundant. Dimensionality reduction is one of the most popular techniques to remove noisy and unnecessary features. Dimensionality reduction techniques can be categorized into feature extraction [1] and feature selection [2]. Feature extraction approaches make combinations of original features to build new features. Meanwhile, feature selection aims to select a small subset of features that minimize redundancy and maximize relevance without incurring in loss of information. Both methods are capable of improving learning performance, lowering computational complexity, building better

generalizable models, and decreasing required storage. However, feature extraction is often more computationally expensive than feature selection.

This work focus on feature selection whose has two general approaches [3]. The first one is called *Individual Evaluation* also known as *Feature ranking*, and the second is *Subset Evaluation*. In Feature ranking, the weight of every feature is assigned according to its degree of relevance. In Subset Evaluation, candidate feature subsets are constructed using heuristic search strategies. The optimal subset selected by the filter approach is always relative to a certain evaluation function. Typically, an evaluation function tries to measure the discriminating ability of a feature or a subset to distinguish the different class labels [4]. In this paper, we present a novel framework that employs different measures in order to accelerate the process of ranking features.

1 Background

This section explains the concepts and theories needed to understand the development of this project. It contains the theoretical background that serves as the foundation for the developed method.

1.1 Pearson Correlation

The Pearson correlation coefficient has been widely used to understand relationships between pairs of variables. Its easy calculation and interpretation have made it a widely used metric in practice by statisticians [5]. To calculate Pearson coefficient r the following equation (Eq. 1) is used.

$$r_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} \quad (1)$$

Unfortunately, Pearson correlation is not a useful metric in general. This coefficient is limited to the detection of linear relationships between variables, and it is not suitable for detecting variable dependencies that are not linear. Fortunately, its computation is quite fast and gives similar results to MIC when there exists a linear relationship and low noise [6]. Figure 1 shows an example of two variables with linear relationship and its corresponding Pearson score.

1.2 Mutual Information

(MI) (Eq. 2) is a well-known correlation measure that uses the Shannon entropy [7] to obtain a wider range of relations between random variables.

$$I(X; Y) = I(Y; X) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \left[\frac{p(x, y)}{p(x)p(y)} \right]. \quad (2)$$

Although this measure is not normalized in the interval $[0, 1]$, there are versions where this is taken in account. For example, (Eq 3) is a normalized version of the Mutual Information.

$$R = \frac{I(X; Y)}{H(X) + H(Y)}, \quad (3)$$

where $H(X)$ is the entropy of X .

1.3 Maximal Information Coefficient (MIC)

The MIC is a measure of correlation between two variables. The general idea is that if two variables (features) are related, then a grid can be drawn over their dispersion graph to compute the degree of relationship between them [6]. Since the number of possible grids can be huge, it is necessary to find the best number of grid partitions (grid resolution) and their best location (partition placements). In order to compute MIC, all the grids (x, y) are explored, where: $x \cdot y \leq n^{0.6}$ y n is the total amount of points representing the data of the tuple. This restriction has been shown to work well in practice [8]. Similarly, for each resolution explored, the partition positions that produce the higher mutual information must be found (see figure 2).

The indicator of mutual information is computed with Eq. 4.

$$I(X; Y) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \left(\frac{p(x, y)}{p(x)p(y)} \right) \quad (4)$$

Where X and Y represent random variables, $p(x, y)$ is the joint probability for the regions x and y respectively, while $p(x)$ and $p(y)$ are their marginal probability distributions [6]. Figure 3 shows a possible 3×3 partition. The regions marked in blue correspond to the $x = 1$ column and row $y = 1$ respectively. In yellow the quadrant corresponding to the intersection of column x and row y is highlighted. The plot shows 20 data points in the interior of the grid, then $n = 20$. With this information it is easy to compute the following probabilities $p(x)$, $p(y)$ and $p(x, y)$. Substituting this probabilities in Eq. 4 the mutual information of this grid can be calculated. Figure 4 shows the complete procedure in order to compute the mutual information for this example.

This procedure is repeated for each resolution saving the higher of these values. This higher mutual information score is then normalized between $[0, 1]$ and stored in a characteristic matrix $M(x, y)$.

Finally, the value of MIC is considered as the higher normalized mutual information value contained in matrix M . Thus the MIC metric is a value in the interval $[0, 1]$.

Figure 2: Partition and placement for the computation of the MIC metric.

Figure 3: Partial mutual information computation for a 3 × 3 grid.

Figure 4: Mutual information computation for a 3 × 3 grid.

Unfortunately, the exhaustive computation of MIC is unpractical for large datasets. Therefore, it is preferable to use approximation algorithms to estimate the MIC metric. These algorithms are significantly faster than the exhaustive computation giving an approximation of the true MIC value. Usually, slower estimation algorithms have better accuracy than the faster ones.

1.4 Fast and Accurate MIC Estimation

In order to find a good balance between accuracy and speed, we proposed a method that uses a sequence of algorithms to speed up the computation of the MIC metric. Each algorithm of the sequence excels at dealing with a special case of correlation. When analyzing the correlation of all the possible pairs of features in the dataset, the sequence is applied from the fastest algorithm to the slowest one. At each step of the sequence, some features pairs are pruned when a reasonably good estimation

of its MIC metric is achieved. This prevents unnecessary computation of the slower algorithms over all the pairs. Thus accelerating the MIC estimation while retaining a good amount of accuracy.

The MIC estimation sequence is composed by the following algorithms:

- 1 Pearson Correlation
- 2 ParallelMIC
- 3 SAMIC

Section 2 shows the complete criteria used to prune feature pairs on each process stage. The next subsections detail the nature of the used algorithms and their role in the final computation of the MIC metric.

1.4.1 *ApproxMaxMI*

ApproxMaxMI [8] is an heuristic algorithm to approximate the optimal value of the MIC metric. The idea behind this heuristic is to consider one axis of the analyzed grids as being equally partitioned while optimizing the partition placement of the other axis. Then, the previously unfixed axis is fixed and equipartitioned repeating the process. The maximum of the two obtained scores at the end is used as the final MIC approximation [8]. The optimization made for each grid axis is performed using a dynamic programming approach.

In the context of ApproxMaxMI, an axis is equipartitioned if all of the regions induced by its partitions contain the same number of data points.

ApproxMaxMI has been implemented on various software packages including Minerva for R and MINEPY for Python [9].

1.4.2 *ParallelMIC*

ParallelMIC is a parallelized version of ApproxMaxMI, it accelerates the computation by calculating the MIC score for various feature pairs at the same time.

ParallelMIC is loosely based on RapidMIC [10]. RapidMIC is also a parallelized version of ApproxMaxMI but it computes MIC by calculating multiple grid resolutions at the same time. The performance of both ParallelMIC is slightly better than RapidMIC. In practice, both algorithms are equivalent when included in the MIC estimation sequence.

1.4.3 *SAMIC*

SAMIC is a MIC estimation algorithm based on Simulated Annealing. Just as ApproxMaxMI, SAMIC tries to find MIC by optimizing over every possible grid resolution (see Section 1.3 and Section 1.4.2). It is based on Simulated Annealing [11], meaning that SAMIC uses random choices over a temperature decay function to enforce the exploration of the whole solution space. In its simplest form SAMIC will proceed as follows at every grid resolution while keeping track of the maximum mutual information score MIC found at every step:

- 1 Set temperature $T = 1$ and equipartition grid in both axis.
- 2 Compute the mutual information score MI of the current grid placement.
- 3 Generate a random neighboring grid placement and compute its new score MI_{new} (More neighbors can be generated at this step if more precision is needed).

- 4 Compare both MI and MI_{new} :
 - If $MI < MI_{new}$, then $MI = MI_{new}$.
 - If $MI > MI_{new}$, generate a random choice $r \in [0, 1]$ and make $MI = MI_{new}$ if and only if $e^{(mi_{new}-mi)/T} > r$.
- 5 Update temperature $T = T \cdot c$ where c is a cooling factor between 0 and 1
- 6 Repeat steps 2 to 6 until $T < T_{min}$

In the context of SAMIC, given any grid G placement, a neighbor grid placement G' is a new placement with one and only one differently placed partition for each axis.

In theory, Simulated Annealing guarantees finding the optimal solution to the optimization problem if the temperature decay is sufficiently slow, and the amount of neighbors explored at each temperature change is big. However in practice the decay ratio and number of explored neighbors are constrained to ensure the termination of the algorithm in a reasonable time. SAMIC is way slower than ParallelMIC but is in fact way more precise.

2 A Novel Approach for Feature Selection

In order to solve the feature selection problem, the proposed method focuses on identifying and removing irrelevant and redundant variables from a given dataset. The MIC metric is used to identify irrelevant features by pruning those with low correlation against the target.

Then, the MIC metric is applied over all the remaining feature pairs. The resulting MIC scores are used to generate groups of correlated values from which only one element of each group is selected as its representative. In a group of features, every other feature besides the group representative is considered as being redundant. Figure 5 shows the entire process.

2.1 MIC Score

Since the proposed feature selection method relies heavily on the fast computation of MIC metrics. A fast and accurate MIC approximation technique has been developed for this purpose 1.4. This sequence is applied from the fastest algorithm to the slowest. At each step of the sequence, some feature pairs are pruned to prevent unnecessary computation of the slower algorithms.

Pearson correlation is used to find and prune those pairs of linearly related or highly dispersed features. Thus, accelerating the whole MIC estimation process. This is achieved by following the next two decision rules:

- If a pair of features scores high in Pearson, then it is considered to have a linear correlation.

- Analogously, if a pair of features scores very low in Pearson then it is considered to be very noisy and thus dispersed.

In both cases, the features with high and low Pearson scores are pruned from the input of next algorithm in the computation sequence (ParallelMIC), making the computation of this next algorithm faster. However, not every pair with high Pearson score is guaranteed to reflect a linear relationship between its pairs.

To ensure that no false positives are pruned, those pairs with high Pearson score must be tested to be truly linearly related. This can be done by comparing the Pearson score of each pair with the maximum mutual information score obtained from the equipartition of all the $k \times k$ grids where $k \cdot k < n^{0.6}$. If both scores are similar, then the pair is confirmed as being linearly correlated and then pruned from the input of the next algorithm in the sequence. The function that obtains the maximum mutual information coefficient from purely square grids is known as MaxEquipartitionGridScore, figure 6 illustrates the process of evaluating equipartitioned square grids.

Like Pearson, ParallelMIC can be used to prune the pairs of features to be computed on the next algorithm of the MIC estimation sequence (SAMIC). Those pairs whose ParallelMIC score is in a range of *good candidates* are passed to the next algorithm in the sequence. A *good candidate* is a pair of features that is suspected to be correlated but has not yet achieved a sufficiently high MIC value to prove it. The default range for ParallelMIC *good candidates* in the MICTools program is between 0.7 and 0.9.

Finally, if the SAMIC score for any given pair is less than its corresponding ParallelMIC score, the maximum among the two is reported as the true *MIC Score*.

2.2 MIC Based Feature Selection

The proposed feature selection method makes use of the MIC Score to identify and remove redundant features. First, compare every feature with the target to determine which features are relevant, ranking the features, then this rank is used to select a subset removing irrelevant features. The MIC Score is also used to generate groups between the subset to identify and remove redundant features.

The idea of using the MIC Score to evaluate the correlation between features correspond to a filter method in the literature of feature selection, which usually

is faster than other methods. Still, it can be combined with search strategies to improve the subset selection.

3 MICTools and MICSelect Software Architecture

To test and refine the feature selection and MIC estimation techniques presented in this project, a pair of software packages were developed. MICTools and MICSelect are computer programs to perform efficient MIC calculations and feature selection respectively.

MICTools is implemented in C++ and allows the parallel execution of the previous algorithm sequence (Pearson, ParallelMIC and SAMIC). MICSelect is implemented in Python and performs the feature selection tasks, it relies heavily on MICTools to perform its required MIC calculations. The communication between MICSelect and MICTools is achieved by a Python wrapper written using the Boost Python library and the Distutils build system. Even though MICSelect includes MICTools as part of its own, MICTools can be used independently and compiled as a standalone application.

To unify and guarantee the correct execution of both MICTools and MICSelect a software architecture has been developed. The developed architecture is detailed in figure 7.

Figure 7: Architecture overview.

This architecture defines a common set of routines and data structures to handle data between algorithms and manage their results. Also, it allows MICSelect to

make use of MICTools transparently through a Python wrapper.

The architecture design is composed of several layers. Each layer provides an specific function. The communication between layers is done through shared data structures, including execution configuration, input matrix and results array. The architecture layers are:

- 1 MICSelect layer.
- 2 Python Wrapper layer.
- 3 MICTools layer.
 - Presentation layer.
 - I/O layer.
 - Sequence of algorithms layer.
 - Dependencies layer.

The architecture allows to run each of the MIC estimation algorithms sequentially. However, each algorithm can run independently from the others. Furthermore, each algorithm of the sequence has been completely parallelized using POSIX threads (pthreads) and lock free policies [12, 13]. The parallelization model used to implement Pearson, ParallelMIC and SAMIC relies on a Single Instruction Multiple Data (SIMD) design over a Uniform Memory Access model (UMA) [14]. In practice, this means that every thread of the CPU will operate over a different set of feature pairs without overlapping. Figure 8 gives an intuition on how the parallelization model works.

Communication between layer is done by the follow data structures:

- Routine configuration
- Input matrix
- Results array

The next subsections explain in detail the role and functions of each architecture's component.

3.1 MICSelect Layer

The MICSelect layer performs the feature selection method. This layer relies on MICTools to read the datasets and to perform the required MIC operations. In

addition, it deals with the process of cutting, ordering, grouping and finally selecting the relevant features of the dataset. A user or program can interact with this layer by sending instructions to its argument parser. Table 1 shows the available commands recognized by MICSelect. This layer is implemented in Python as it allows easy data slicing and filtering.

Table 1: Parameters of MICSelect.

MICSelect Options			
Short command	Value	Range	Description
Mandatory options			
-i	Path		Input CSV file.
-y	String		Target Name of dataset.
Not mandatory options			
-r			Remove redundant features.
-x	Integer	[0,4]	Step wise feature selection
-s	Integer	[1,N]	Static cut, gives the number of features selected sorted by importance. N is the total number of features in the dataset.
-w			Write features correlation in a file.

3.2 Python Wrapper Layer

This layer allows the communication between MICSelect and MICTools. It is implemented as a Python module meaning that MICTools can be included in any Python program. The wrapper's functions are the follow:

- Instructions passing from MICSelect to MICTools.
- Results parsing to Python standard data structures.
- Automatic handling of memory allocation.

This layer is implemented in C++ using the Boost Python libraries and the Distutils build system.

3.3 MICTools Layer

The MICTools layer performs the MIC estimation sequence of algorithms explained in Section 2.1. It contains several sub-layers that perform simple tasks, these layers are:

- Presentation layer.
- I/O layer.
- Sequence of algorithms layer.
- Dependencies layer.

Since the performance of this layer is critical, it is implemented in C++ with POSIX Threads as parallelization library. This library ensures portability between Unix compliant operative systems. This layer can also be compiled as a standalone application independent of the rest of the architecture.

3.3.1 Presentation Layer

This layer provides the input interface for the MICTools program. A user or application is able to use this layer to specify commands, arguments and parameters to MICTools. These instructions may be introduced as a single command and interpreted by an argument parser. Table 2 shows the different options accepted as input by MICTools.

3.3.2 I/O Layer

This layer provides the input and output operations to interact with data sources. For example, datasets contained in CSV files or databases (not implemented). This layer guarantees the correct reading of datasets and writing of resulting data.

Table 2: MICTools supported parameters

MICTools Options				
Short command	Long command	Value	Range	Description
General options				
-a	-alpha	Double	(0, 1]	Alpha value for MIC calculations.
-i	-input	String		Input CSV file.
-o	-output	String		Output results file.
-f	-keys_file	String		Filter keys file. Restricts the generated pairs to be constructed only with those variables present in the file.
-g	-target	String		Name of the variable against which all of the remaining variables will be paired.
-h	-header			Indicates that the input file contains a Header line.
-t	-max_threads	Integer	>0	Max number of threads to use during the computation.
-d	-default			Forces the program to run Pearson, ParallelMIC and SAMIC using their default parameters.
Analysis options				
-P	-Pearson			Includes Pearson in the analysis schedule.
-R	-ParallelMIC			Includes ParallelMIC in the analysis schedule.
-S	-SAMIC			Includes SAMIC in the analysis schedule.
RapidMIC options				
-u	-clumps	Double	>0	Number of clumps (must be larger than 0).
-p	-min_pearson	Double	[0, 1]	Sets ParallelMIC to compute only those pairs whose Pearson coefficient absolute value is above the given value.
-e	-max_pearson	Double	[0, 1]	Sets ParallelMIC to compute only those pairs whose Pearson coefficient absolute value is below the given value.
SAMIC options				
-n	-neighbors	Integer	>0	SAMIC amount of candidate neighbors to consider for each temperature of the simulated annealing stage.
-c	-cooling	Double	(0, 1]	Temperature cooling factor to be used in SAMIC.
-m	-min_temp	Double	(0, 1]	Minimum temperature value to be used in SAMIC.
-j	-min_parallelmic	Double	[0, 1]	SAMIC min ParallelMIC value. Restricts computation to pairs with ParallelMIC score above the given value.
-k	-max_parallelmic	Double	[0, 1]	SAMIC max ParallelMIC value. Restricts computation to pairs with ParallelMIC score below the given value.

3.3.3 Sequence of Algorithms Layer

This layer contains the parallel implementations of the MIC estimation algorithms supported by MICTools. This layer allows every algorithm to processes every pair of features based on the algorithm sequence described in Section 2.1.

3.3.4 Dependencies Layer

This layer contains all the libraries needed by the Algorithms Layer. This libraries provide functionalities such as parallelization routines, grid management, thread safe random number, generators, a sequential implementation of ApproxMaxMI (implemented in C++ from scratch), etcetera.

3.4 MICTools Auxiliary Data Structures

The auxiliary data structures in MICTools provide a way of communication for all of its layers. These datasets contain mainly algorithms execution parameters, results and common input data.

3.4.1 Execution Configuration

This data structure contains the settings for the current planned execution of MICTools. It contains all the algorithms that are going to be executed in the current run as well as their pertinent parameters. The configurations includes:

- Input data source.
- Output data source.
- Algorithms execution parameters.

3.4.2 Mutual Input Matrix

This data structure contains reading registers acquired by the data source input. In addition, it provides the input data to all the algorithms in execution.

3.4.3 Array of Results

This data structure contains the computed pairs of variables, as well as the results obtained by each of the performed algorithms over them. When running any of the parallel algorithms in MICTools this data structure is segmented in many pieces as threads are available for the computation. Then, each thread will operate over its corresponding piece, reading the stored pairs and writing the results to this data structure.

4 Results

To evaluate the framework twelve datasets from UCI MLR [15] and LIBSVM [16] were used. In our experiments, we ran each dataset with three different scenarios. The first one uses the full set of features, the second one uses the subset of top-ten ranked features, and the last one uses the subset selected by a forward selection search. Every set was evaluated by four classification algorithm. Every classifier was executed three times, applying the stratified k-fold cross-validation strategy.

The figures reffig:results1, 10, 11 shows the roc curves of every dataset where the score of each scenario is displayed and the table 3 is a summary of the scores and execution time.

Table 3: Table of results of feature selection using MICSelect

dataset name	alias	full AUC	top10 time	top10 AUC	sw time	sw AUC
SPECTF Heart	ds1	0.81	6ms	0.89	4596ms	0.90
Cervical Cancer (R.F.)	ds2	0.80	12ms	0.78	4603ms	0.91
Breast Cancer (Diagnostic)	ds3	0.64	488m	0.96	5384ms	0.96
Breast Cancer (Prognostic)	ds4	0.57	23ms	0.62	4625ms	0.56
Sports articles	ds5	0.84	164ms	0.82	6492ms	0.82
Lung Cancer	ds6	0.84	1ms	0.86	4577ms	0.85
HCC Survival	ds7	0.70	23ms	0.73	4721ms	0.71
Duke Breast Cancer	ds8	0.88	182ms	0.95	5407ms	0.93
Colon Cancer	ds9	0.89	102ms	0.91	4878ms	0.93
Leukemia	ds10	0.92	93ms	0.99	5328ms	0.99
Sonar	ds11	0.59	127ms	0.75	4722ms	0.75
Splice	ds12	0.89	17ms	0.95	4847ms	0.95
average	-	0.781	103ms	0.851	5015ms	0.855

The software-hardware configuration to run the experiments were the following:

- AMD Ryzen 1700
- 2 x 8GB XPG Z1 DDR4 2400 Mhz
- Samsung 1TB 970 Evo Pro
- ROG STRIX RTX 2060
- Ubuntu Desktop 18.04 x86-64

5 Conclusion

We have presented a feature selection framework for large datasets based on correlation capable of detect nonlinear relationships between two variables. After a series of experiments, our proposal shows better accuracy with lower computational complexity when applied to different datasets. Our proposal is capable of detecting the relation between a pair of variables, but for the combined strong association between more than two variables, other methods should be applied.

References

1. Chao, G., Luo, Y., Ding, W.: Recent advances in supervised dimension reduction: A survey. *Machine learning and knowledge extraction* **1**(1), 341–358 (2019)
2. Kumar, V., Minz, S.: Feature selection: a literature review. *SmartCR* **4**(3), 211–229 (2014)
3. Bolón-Canedo, V., Sánchez-Marño, N., Alonso-Betanzos, A.: A review of feature selection methods on synthetic data. *Knowledge and information systems* **34**(3), 483–519 (2013)
4. Dash, M., Liu, H., Motoda, H.: Consistency based feature selection. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 98–109 (2000). Springer
5. Clark, M.: A comparison of correlation measures. *Center for Social Research, University of Notre Dame* **4** (2013)
6. Reshef, D.N., Reshef, Y.A., Finucane, H.K., Grossman, S.R., McVean, G., Turnbaugh, P.J., Lander, E.S., Mitzenmacher, M., Sabeti, P.C.: Detecting novel associations in large data sets. *science* **334**(6062), 1518–1524 (2011)
7. Gray, R.M.: *Entropy and Information Theory*. Springer, New York, NY, USA (1990)
8. Reshef, D.N., Reshef, Y.A., Finucane, H.K., Grossman, S.R., McVean, G., Turnbaugh, P.J., Lander, E.S., Mitzenmacher, M., Sabeti, P.C.: Detecting Novel Associations in Large Data Sets Support Online Material
9. Albanese, D., Filosi, M., Visintainer, R., Riccadonna, S., Jurman, G., Furlanello, C.: Minerva and minepy: a c engine for the mine suite and its r, python and matlab wrappers. *Bioinformatics*, 707 (2012)
10. Tang, D., Wang, M., Zheng, W., Wang, H.: Rapidmic: Rapid computation of the maximal information coefficient. *Evolutionary bioinformatics online* **10**, 11 (2014)
11. Kirkpatrick, S.: Optimization by simulated annealing: Quantitative studies. *Journal of statistical physics* **34**(5-6), 975–986 (1984)
12. Herlihy, M., Shavit, N.: *The Art of Multiprocessor Programming*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2008)
13. Williams, A.: *C++ Concurrency in Action: Practical Multithreading*. Manning Pubs Co Series. Manning, ??? (2012)
14. Hennessy, J.L., Patterson, D.A.: *Computer Architecture, Fifth Edition: A Quantitative Approach*, 5th edn. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2011)
15. Dua, D., Graff, C.: *UCI Machine Learning Repository* (2017). <http://archive.ics.uci.edu/ml>
16. Chang, C.-C., Lin, C.-J.: LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology* **2**, 27–12727 (2011). Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>

Figure 10: Comparative full-feature dataset vs feature selected subset (1)

Figures

Figure 1

A noisy linear relation and its corresponding Pearson score.

Figure 2

Partition and placement for the computation of the MIC metric.

$$p(x, y) \log \left(\frac{p(x, y)}{p(x)p(y)} \right) = 0.25 \log \left(\frac{0.25}{0.4 \times 0.5} \right) \approx 0.056$$

Figure 3

Partial mutual information computation for a 3×3 grid.

Figure 4

Mutual information computation for a 3 × 3 grid.

Figure 5

feature selection process

Figure 6

Evaluation of mutual information in equipartitioned square grids to ensure linearity. The cells with $p(x; y) \neq 0$ are highlighted.

Figure 7

Architecture overview.

Figure 8

Single Instruction Multiple Data (SIMD) model.

Figure 9

Comparative full-feature dataset vs feature selected subset (1)

Figure 10

Comparative full-feature dataset vs feature selected subset (1)