juliacon

GlobalSearchRegression.jl: Building bridges between Machine Learning and Econometrics in Fat-Data scenarios

Demian Panigo[1, 3], Pablo Glüzmann[1, 2], Esteban Mocskos[1, 5], Adan Mauri Ungaro[4], Valentin Mari[4], and Nicolás Monzón[3]

[1]National Scientific and Technical Research Council (CONICET)
[2]Center For Distributive, Labor and Social Studies (CEDLAS-UNLP)
[3]Laboratory of Economics, Technology, Innovation and Finance (LETIF-UNLP)
[4]National University of La Plata
[5]National University of Buenos Aires

ABSTRACT

The aim of this paper is twofold. The first one is to describe a novel research-project designed for building bridges between machine learning and econometric worlds (`ModelSelection.jl`). The second one is to introduce the main characteristics and comparative performance of the first Julia-native all-subset regression algorithm included in `GlobalSearchRegression.jl` (v1.0.5). As other available alternatives, this algorithm allows researchers to obtain the best model specification among all possible covariate combinations - in terms of user defined information criteria-, but up to 3165 and 197 times faster than STATA and R alternatives, respectively.

Keywords

Julia, Parallel computing, Econometrics, all-subset regression, Machine Learning, Fat-Data

1. Introduction

While allowing for better and more accurate analysis (and forecasts), fat-data availability generates a revival of Bellman's [9] -and newer- high-dimensionality concerns:

(1) Exponential increase in required information and execution time [34] ;
(2) Overfitting risk [17];
(3) Less-informative euclidean distances [4]; and
(4) Unfeasible estimators [10].

Notwithstanding, the advantage of having thousands/millions of features to deal with complex phenomena stimulates an unprecedented number of methodological -and technological- improvements to manage the 'curse of dimensionality' [11].

In Economics, this process has a dual approach with machine-learning (ML) and econometric (EC) algorithms emerging for different purposes: the former for prediction/forecasts (focusing on $\hat{y}$) and the latter for estimation/causal inference (interested in $\hat{\beta}$). Alternatively, the same distinction can be expressed in Diebold's terms as non-causal vs causal prediction (see [7]), where ML algorithms are designed to reduce prediction sampling-risks -i.e. learning through cross-validation techniques- and EC methods to identify unbiased multivariate relationships -i.e. avoiding consistency issues through residual and coefficient tests for model selection.

In turn, ML feature selection algorithms can be classified into three different families: Filters, Wrappers and Embedded -depending on whether they use some classifier/response variable information or not, or how variable selection is made along with the learning process, see [14]. Similarly, most EC dimensionality reduction approaches can be classified into three different groups: Exhaustive, General-to-Specific and Specific-to-General -depending on the search pattern; see [22].

Despite significant improvements in recent econometric developments -like PCGIVE/Autometrics or Retina algorithms, which combine some ML and EC characteristics-, available alternatives fails to fully exploit cross-validation and model averaging capabilities -see [19], [31], [15], [18], [29], [25], and [13].

Conversely, newer ML algorithms -like Convolutional Neural Networks or Bootstrap-Based LASSO, which improve complex non-linear adjustment or model selection under regularization schemes- achieved unprecedented forecast accuracy but disregarding model interpretability and/or parameter estimation issues -i.e omitting residual and coefficient tests; see [12].

Following Varian's [33] advices, about ML and EC complementarities -i.e. merging algorithms from different families to reduce both sampling and model uncertainty-, we are developing a novel multi-layer-multi-algorithm methodology combining two reinforcing paradigms: The LSE "Testimation" approach -to obtain information about residual properties, see [2]- and the Bayesian-like "Double-model averaging" -across different covariates and subsamples, see [26]. This methodology includes five complementary layers -handling cross-section, time series and panel data-: 1) Preprocessing: with outlier detection, missing values identification, seasonal adjustment and normalization/standardization functions; 2) Feature extraction: creation of logs, squares, inverses and interactions from selected variables; 3) Feature pre-selection: using

filter and embedded ML algorithms like CFS, Variance threshold and LASSO functions; 4) Final feature selection: with a modified all-subset regression approach, including residual tests and model averaging capabilities; 5) Post-estimation fine-tuning: coefficient re-evaluation through cross-validation techniques and model averaging across different k-fold results.

The objective of this paper is to introduce the main characteristics and comparative performance of a key layer of our methodology: the modified all-subset regression algorithm included in GlobalSearchRegression.jl (v1.0.5). As other available alternatives (like MuMin-pdredge in R, or GSREG Stata alternatives), this Julia algorithm allows researchers to obtain the best model specification among all possible covariate/feature combinations - in terms of user defined information criteria-, but up to 3165 times faster than Stata and 197 times faster than R.

2. Package's main features

Written in Julia, GlobalSearchRegression is a parallel (and improved) version of the Stata-GSREG all-subset regression command (get the original code here). The package structure is quite simple, as shown in figure 1:

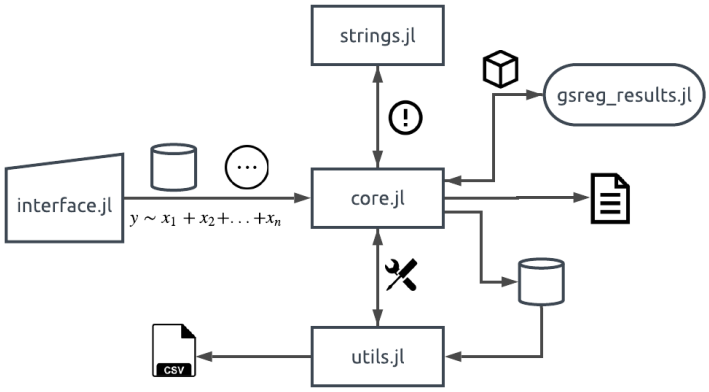

Fig. 1. GlobalSearchRegression.jl Structure Flowchart

Through the gsreg function of the `interface.jl` internal package, users set the appropriate database to be used, the general unrestricted model -GUM, which defines the search space- and additional options for model selection. With this information and complementary supporting functions and definitions provided by `strings.jl` -i.e. error messages-, `utils.jl` -i.e. equation formatting, combinatorial analysis, database manipulation, sorting results, etc.- and `gsreg_results.jl` -i.e. the structure to save estimation results-, the `core.jl` package perform the all-subset-regression algorithm explained in the pseudocode below, to obtain the following outputs: 1) a matrix -optionally exported to a csv file through `utils.jl`- including regression coefficients, selection criteria, observations and (optionally) t-test, residual tests, averaging-weights and out-of-sample metrics for every alternative model; 2) a text file -also displayed on screen- which contains the best model specification and (optionally) model averaging results in terms of he user-selected information criteria -see multiple examples in `runtest.jl-`.

Pseudo-code: Core.jl

1. ***function*** *gsreg_single_proc_results!(args...)*
   1.1. Select regression covariates
   1.2. Perform QR Decomposition
   1.3. Estimate regression results
   1.4. Calculate additional tests
   1.5. Save individual results in a Shared Array
   end *function gsreg_single_proc_results!(args...)*

2. ***function*** *gsreg_proc_results!(args...)*
   2.1. **for** $j = 1 : ((k^2 - 1)/nworkers)$ *# where nworkers is the user defined number of cores, and k is the numbers of covariates in the GUM*
   2.1.1. $order = (j - 1) * nworkers + workerID$
   2.1.2. *gsreg_single_proc_results!(order, args...)*
   end for
   end *function gsreg_single_proc_results!(args...)*

3. ***function*** *proc!(args...)*
   3.1. Create environment (Shared Array and other objects)
   3.2. for $workerID = 1 : nworkers$
   3.2.1 spawn jobs among workerIDs to perform gsreg_proc_results!
   end for
   3.3. Perform vector operations
   3.4. Call `utils.jl` sort function
   3.5. Create results array
   end *function proc!(args...)*

4. Export results using `gsreg\_results.jl` structure
5. Export summary results to a txt file
6. Optionally send the results structure to `utils.jl` to export a csv file.

end module `Core.jl`.

3. Comparative performance against *R* and *STATA*

In table 1, we present a performance comparison of our GLobalSearchRegression Julia-package against its main alternatives: MuMin-pdredge and GSREG (written in Stata[1], espectively). Execution times were obtained from a HED architecture using a Threadripper 1950x build, with 16 cores (32 threads) overclocked to 3.8GHz and 64 GiB of DDR4-RAM at 3200Mhz. Comparative scripts were implemented on Julia 1.0.3, R 3.6.0 and Stata 15 IC, running on Ubuntu 18.04.2 LTS -Linux kernel 4.15-[2]

For experimental -random variable- databases with a few covariates -up to 15 explanatory variables-, our Julia algorithm only provides significant time improvement in standard personal computers -e.g. 4 cores-, being up to twice faster than R and 4 times faster than Stata. For HED computers or HPC nodes, there is almost no difference among the best result obtained for each alternative.

However, for databases with 20 or more covariates, our Julia all-subset-regression code is always faster, irrespective of the number of observations or threads -up to 3165 times faster than STATA

[1] The parallel version of Stata-GSREG is still under development. A preliminary version is available upon request from authors
[2] All these test are available here

Table 1. Average execution time and speed-up metrics -over 5 runs-

Cov.	Obs	Threads	Execution time (in seconds)			Parallel Speed-up			Julia Speed-up	
			Julia	R	Stata	Julia	R	Stata	Over R	Over Stata
15	100	1	14.0	69.1	646.6	1.0	1.0	1.0	4.9	46.2
15	100	4	17.0	33.4	80.3	0.8	2.1	8.0	2.0	4.7
15	100	16	22.0	23.6	22.1	0.6	2.9	29.2	1.1	1.0
15	100	32	31.4	22.3	15.1	0.4	3.1	43.0	0.7	0.5
15	1000	1	18.6	74.9	667.5	1.0	1.0	1.0	4.0	35.9
15	1000	4	18.5	35.3	81.6	1.0	2.1	8.2	1.9	4.4
15	1000	16	22.2	24.6	23.3	0.8	3.1	28.7	1.1	1.0
15	1000	32	31.5	21.5	16.0	0.6	3.5	41.6	0.7	0.5
15	10000	1	61.6	142.0	809.0	1.0	1.0	1.0	2.3	13.1
15	10000	4	30.5	61.4	114.7	2.0	2.3	7.1	2.0	3.8
15	10000	16	25.7	38.4	37.3	2.4	3.7	21.7	1.5	1.5
15	10000	32	33.4	35.5	26.7	1.8	4.0	30.3	1.1	0.8
20	100	1	169.2	10538.9	535693.2	1.0	1.0	1.0	62.3	3165.1
20	100	4	65.6	9275.4	75510.2	2.6	1.1	7.1	141.4	1151.1
20	100	16	44.7	8855.2	4526.3	3.8	1.2	118.4	197.9	101.2
20	100	32	51.2	8732.4	1658.2	3.3	1.2	323.0	170.7	32.4
20	1000	1	362.4	10765.9		1.0	1.0		29.7	
20	1000	4	119.0	9352.3		3.0	1.2		78.6	
20	1000	16	58.8	8926.4		6.2	1.2		151.8	
20	1000	32	59.8	8753.9		6.1	1.2		146.4	
20	10000	1	2338.4	13218.9		1.0	1.0		5.7	
20	10000	4	629.4	10287.9		3.7	1.3		16.3	
20	10000	16	198.6	9392.3		11.8	1.4		47.3	
20	10000	32	149.5	9041.6		15.6	1.5		60.5	
25	100	1	6700.7			1.0				
25	100	4	1937.7			3.5				
25	100	16	913.7			7.3				
25	100	32	803.9			8.3				
25	1000	1	14257.4			1.0				
25	1000	4	3954.2			3.6				
25	1000	16	1471.4			9.7				
25	1000	32	1159.9			12.3				
25	10000	1	93450.0			1.0				
25	10000	4	23783.8			3.9				
25	10000	16	6825.6			13.7				
25	10000	32	4660.4			20.1				

and 197 times faster than R-. Execution time differences exponentially increase with the number of covariates and slightly decrease with available observations. Moreover, Stata and R alternatives become unfeasible for databases with 25 or more covariates. For a confirmatory analysis, in figure 2 we present execution-time kernel densities for the 20 covariates - 100 observations - 32 threads case, obtained from 300 independent (non-cached) runs. Kernel density results show that execution time differences are not an artifact obtained from noisy runs. Our Julia algorithm is consistently faster han R and Stata alternatives.

A detailed speed-up analysis is also available for the 25-covariate case. In figure 3, it's shown that our Julia all-subset-regression algorithm scales almost linearly for large databases -while the number of threads is not higher than the number of physical cores-. With small databases, Amdhal's law [6] inputs change. Parallel tasks become lighter and speed-up efficiency degrades consistently with additional threads, because the marginal overhead cost of a larger environment creation is not overcompensated by parallelism gains obtained from additional threads. Notwithstanding, using only physical cores speed-up efficiency is always above 45% -with an average of 84% for 2, 4, 8 and 16 threads-.

4. Why GlobalSearchRegression.jl is faster than existing alternatives?

4.1 JULIA platform

Despite some `GlobalSearchRegression.jl` specific features to be examined below, our all-subset-regression algorithm benefits from the well-known Julia language efficiency for High Performance Computing tasks.

Table 2. Julia speed-up over STATA and R in basic functions

Function	Speed-up over R	Speed-up over Stata
recursion_fibonacci	251.28	76.92
recursion_quicksort	32.36	48.54
matrix_multiply	11.6	14.07
iteration_pi_sum	14.33	26.89
matrix_statistics	7.65	44.92
parse_integers	28.4	49.38
userfunc_mandelbrot	175.44	368.42
Simple-Average	74.44	89.88
Weighted-Average	12.15	22.01

Note: Weighted-average were estimated as the Ratio between total execution time for R or Stata, and Julia's total execution time -for all basic functions-. Benchmarkings were obtained using a i7-4700MQ quad-core build, with 8 GiB of DDR3 1600 Mhz RAM, under Windows 8 and running Julia 1.1.0, Stata 15 and R 3.5.3.

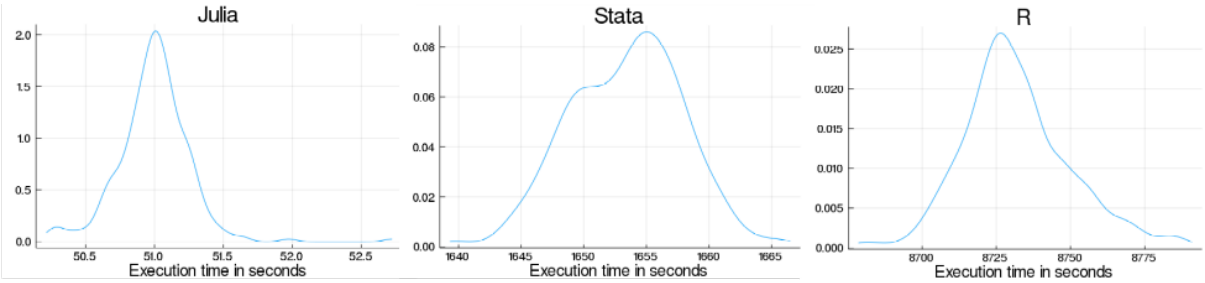

Fig. 2. Execution time, speed-up and speed-up efficiency for the 20-covariate case using GlobalSearchRegression.jl

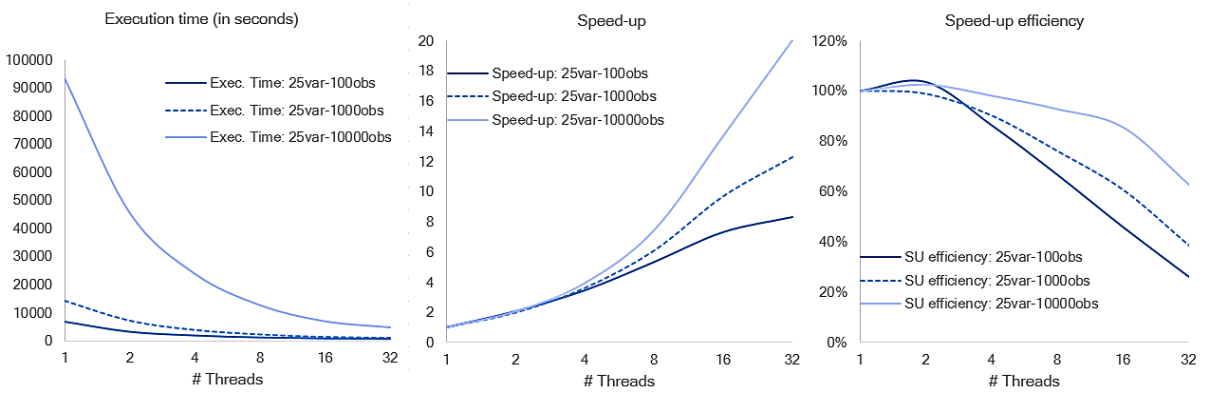

Fig. 3. Execution time, speed-up and speed-up efficiency for the 25-covariate case using GlobalSearchRegression.jl

Julia JIT-compilation allows packages to run faster than those executed using interpreted or byte-compiled languages (like R or Stata-Mata). Indeed, basic functions running on Julia can be up to 251 and 368 times faster than those running in R and Stata, respectively -with an average speed-up of 75 and 90, see Table 2-.

4.2 Parallel strategy and memory setup

It is well know that multicore architectures can be used to speed-up execution times through two main paradigms: data and task-parallelism [35]. The preferred strategy critically depends on:

(1) Database structure;
(2) Algorithm serial portion; and
(3) Interthread communication costs.

The first discussion is about tall vs fat data. While tall databases are more suitable for data-parallelism [8], fat-structures improve relative performance of task-parallelism (because tasks increase exponentially with covariates/columns in feature selection problems, see [20]).

Additionally, the choice between alternative paradigms takes into account the Amdahl's Law for the specific algorithm to be used. Task-parallelism is usually better for econometric and machine learning algorithms requiring some specific serial optimization paths (i.e. arima-arfima models), while data-parallelism performs better under linear-algebra solutions (i.e. OLS-family estimators, see [24]).

Finally, we have the problem of intercomunication costs. Data-parallelism -generally- involves intense needs of inter-thread-communication. While task-parallelism communication costs depends on Load-Balancing choices (i.e. Dynamic vs Static), they are usually lower than data-parallelism ones [3] [30].

For feature selection problems, pros and cons of alternative strategies often determine that available cores should be used for task parallelism. Databases are fat, data-mining algorithms can include large serial portions, and intercomunication costs can be huge for large-multicore architectures. These reasons explain why all available all-subset-regression packages use taks-parallism to speed-up execution times.

As for the memory setup, there are also alternative methodologies. First, it's necessary to choose among Static vs. Dynamic memory allocations [32]. To improve speed-up, Static "once-and-for-all" allocations are often preferred (even at a cost of higher average memory utilization). Second, shared-memory strategies must be determined. Depending on both object-size and CPU architecture -cache size and its distribution among cores-, it could be optimal to use large shared arrays or -alternatively- smaller core-specific objects. Splitting output matrices to work with smaller non-shared arrays could be useful for cache optimization purposes, but it could also entail additional communication costs and higher memory requirements [5]. In practice, feature selection algorithms usually prefer shared-arrays for high performance computing.

[3] At least when Static Load Balancing is implemented, because it allows for Coarse-grained granularity [23]

`GlobalSearchRegression.jl` (execution-time) advantages have been obtained combining:

a) Task-parallelism
b) Static Load-Balancing
c) Coarse-grained granularity
d) Static memory allocation
e) Efficient shared-array implementation

While some of these characteristics are shared with R and Stata alternatives (`MuMin-pdredge` and `GSREG`, respectively), our Static Load-Balancing algorithm outperform the round-robin R scheduling (implemented by the clusterapply function included in the `parallel` R-package see [16]) and the Shared-Array strategy significantly improves I/O performance against Stata. By construction, pure Static Load-Balancing in GlobalSearchRegression.jl avoids Round-Robin communication costs and execution gaps. This advantage overcompensate minimal[4] load-asymmetries, associated with any static scheduling. In turn, using efficient shared-arrays to store all-subset-regression results allows us to outperform the `Stata-GSREG` methodology which heavily relies on slower I/O disk operations (because multiple instances must be launched to enable task-parallelism and, therefore, shared-arrays become unfeasible).

4.3 OLS estimation

Efficient Ordinary Least Squares (OLS) algorithms rely on Linear Algebra operations (matrix decomposition, matrix inversion, etc.).[5] The traditional $(X'X)^{-1}$ operation could be time-expensive with unstable solutions under certain conditions [27]. A preferred method is the QR-decomposition developed by Francis [21] and Kublanovskaya [28]. The QR factorization allows us to decompose any full rank $N \times p$ matrix $\bar{X}$ as:

$$X = QR \tag{1}$$

where

Q is a $N \times p$ matrix with $Q'Q = I$; and
R a $p \times p$ upper triangular matrix.

The QR decomposition is fast and provides stable numerical solutions under rank-deficient matrices. Alternative factorizations are either slower (SVD decomposition) or potentially unstable (Cholesky decomposition) [3].

`GlobalSearchRegression.jl` OLS estimation through QR-decompositions outperform existing Julia alternatives (like `GLM.jl`) allowing for large speed-ups compared with `R-lm` and `Stata-regress` commands. Table 3 shows execution time differences for a 200 covariates - 1000 observations - multivariate linear regression [6].

It must be notice that execution times are compilation-free, because all-subset-regression algorithms must perform thousand to millions of regressions where compilation time is absent (it only affects the first regression).

[4] Minimal because our static scheduler guarantees that average task-complexity will not be too different among workers

[5] Optimization alternatives (i.e. Gradient descent), while quite inefficient for linear models, can also be used for non-linear estimations [1]

[6] Execution times were calculated for alternative OLS algorithms available for all-subset-regression packages in Julia, and Stata.

Table 3. Execution times for different OLS algorithms in R, Stata and Julia

	GlobalSearchRegression.jl	R-lm	Stata-reg	GLM.jl
Execution time	0.012	0.06	0.2	0.023
Speed-up	1	5	16.67	1.92

Note: Execution times were obtained using a i7-7500 dual-core build, with 8 GiB of DDR4 RAM, under Windows 10 and running Julia 1.1.0, Stata 13 and RStudio 3.6.0. Speed-ups are obtaining dividing each execution time by the GlobalSearchRegression.jl execution time.

5. Conclusion

> *"There are a number of areas where there would be opportunities for fruitful collaboration between econometrics and machine learning [... and] the most important area for collaboration involves causal inference"*. [33]

As Hal Varian emphasizes, there is a need for mutual collaboration between machine learning developers/practitioners and econometricians. In this paper we describe a research-project aimed at building bridges between machine learning and econometric worlds (`ModelSelection.jl`) and introduce the main characteristics of its first outcome: a Julia-native all-subset regression algorithm (`GlobalSearchRegression.jl`) which runs up to 3165 and 197 times faster than existing Stata and R alternatives, respectively.

Throughout the paper, it has been shown that execution-time gains are explained by multiple efficient strategies combined in `GlobalSearchRegression.jl` (i.e. task parallelism, static load-balancing, coarse-grained granularity, static memory allocation, efficient shared array implementation, and OLS estimation using neat QR decompositions). However, the main 'explanatory variable' is the impressive speed-up in atomic operations obtained using the Julia language.

Notwithstanding, increasing availability of Big and -more challenging- Fat-data, force us to go beyond pure all-subset-regression approaches and combine it with machine learning feature selection algorithms. Work in progress include a new hybrid package merging LASSO capabilities, all-subset-regression robustness nd K-fold cross validation strengths.

6. Acknowledgments

This research benefited from the financial support of 1) the National Scientific and Technical Research Council and the YPF-Foundation (PIO CONICET-YPF 13320150100020CO); and 2) the National Agency for Scientific and Technological Promotion (PICT-2017-0867). We would also like to thank PhD. Jorge Carrera for his technical assistance on Big-data econometric algorithms.

7. References

[1] Hervé Abdi. The method of least squares. *Encyclopedia of Measurement and Statistics. CA, USA: Thousand Oaks*, 2007.

[2] Felix Abramovich and Claudia Angelini. Bayesian maximum a posteriori multiple testing procedure. *Sankhyā: The Indian Journal of Statistics (2003-2007)*, 68(3):436–460, 2006.

[3] Monika Agarwal and Rajesh Mehra. Review of matrix decomposition techniques for signal processing applications. *Int. Journal of Engineering Research and Applications*, 4(1):90–93, 2014.

[4] Charu Aggarwal, Alexander Hinneburg, and Keim Daniel. On the surprising behavior of distance metrics in high dimensional space. In Jan Van den Bussche and Victor Vianu, editors, *International conference on database theory*, Lecture Notes in Computer Science, vol 1973, pages 420–434. Springer, Berlin, Heidelberg, 2001.

[5] Muhammad Waqas Ahmed and Munam Ali Shah. Cache memory: An analysis on optimization techniques. *International Journal of Computer and IT*, 4(2):414–418, 2015.

[6] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM.

[7] Susan Athey. Machine learning and causal inference for policy evaluation. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 5–6. ACM, 2015.

[8] Shivnath Babu, Herodotos Herodotou, et al. Massively parallel databases and mapreduce systems. *Foundations and Trends® in Databases*, 5(1):1–104, 2013.

[9] Richard Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

[10] Peter Bhlmann and Sara Van De Geer. *Statistics for high-dimensional data: methods, theory and applications*. Springer-Verlag Berlin Heidelberg, 2011.

[11] Verónica Bolón-Canedo, Noelia Sánchez-Maroño, and Amparo Alonso-Betanzos. *Feature selection for high-dimensional data*. Progress in Artificial Intelligence. Springer-Verlag Berlin Heidelberg, 2016.

[12] Danilo Bzdok, Naomi Altman, and Martin Krzywinski. Statistics versus machine learning. *Nature methods*, 15(4):233–234, 2010.

[13] Jennifer Castle. *Empirical modeling and model selection for forecasting inflation in a non-stationary world*. Ph.d. thesis, Nuffield College University of Oxford, 2006.

[14] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

[15] James E.H. Davidson and David F. Hendry. Interpreting econometric evidence: The behaviour of consumers' expenditure in the uk. *European Economic Review*, 16(1):177–192, 1981.

[16] Briti Deb and Satish Narayana Srirama. Parallel k-means clustering for gene expression data on snow. *International Journal of Computer Applications*, 71(24), 2013.

[17] Marianne Defernez and Katherine Kemsley. Avoiding overfitting in the analysis of high-dimensional data with artificial neural networks (anns). *Analyst*, 124(11):1675–1681, 1999.

[18] Shelley Derksen and H. J. Keselman. Backward, forward and stepwise automated subset selection algorithms: Frequency of obtaining authentic and noise variables. *British Journal of Mathematical and Statistical Psychology*, 45(2):265–282, 1992.

[19] Jurgen A. Doornik. Autometrics. In *in Honour of David F. Hendry*, pages 88–121. University Press, 2009.

[20] Ian Foster. Task parallelism and high-performance languages. *IEEE Concurrency*, (3):27–36, 1996.

[21] John GF Francis. The qr transformation a unitary analogue to the lr transformation—part 1. *The Computer Journal*, 4(3):265–271, 1961.

[22] Pablo Glüzmann and Demian Panigo. Global search regression: A new automatic model-selection technique for cross-section, time-series, and panel-data regressions. *The Stata journal*, 15(2):325–349, 2015.

[23] Michael I. Gordon, William Thies, and Saman Amarasinghe. Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. *SIGPLAN Not.*, 41(11):151–162, oct 2006.

[24] Guangbao Guo. Parallel statistical computing for statistical inference. *Journal of Statistical Theory and Practice*, 6(3):536–565, 2012.

[25] Helmut Herwartz. A note on model selection in (time series) regression models – general-to-specific or specific-to-general? *Applied Economics Letters*, 17(12):1157–1160, 2010.

[26] Jennifer A. Hoeting, David Madigan, Adrian E. Raftery, and Chris T. Volinsky. Bayesian model averaging: A tutorial. *Statistical Science*, 14(4):382–401, 1999.

[27] Antony Jameson and Eli Turkel. Implicit schemes and decompositions. *Mathematics of Computation*, 37(156):385–397, 1981.

[28] Vera N Kublanovskaya. On some algorithms for the solution of the complete eigenvalue problem. *USSR Computational Mathematics and Mathematical Physics*, 1(3):637–657, 1962.

[29] Maximiliano Marinucci. *Automatic prediction and model selection*. Ph.d. thesis, Facultad de Ciencias Económicas y Empresariales, Universidad Complutense de Madrid, 2008.

[30] N. Moreano and A.C.M.A.de Melo. Chapter 6 - biological sequence analysis on gpu. In Hamid Sarbazi-Azad, editor, *Advances in GPU Research and Practice*, Emerging Trends in Computer Science and Applied Computing, pages 127 – 162. Morgan Kaufmann, Boston, 2017.

[31] Teodosio Perez-Amaral, Giampiero M. Gallo, and Halbert White. A flexible tool for model building: the relevant transformation of the inputs network approach (retina). *Oxford Bulletin of Economics and Statistics*, 65(s1):821–838, 2003.

[32] Mr Ramesh Prajapati, Mr Dushyantsinh Rathod, and Samrat Khanna. Comparison of static and dynamic load balancing in grid computing. *International Journal For Technological Research In Engineering*, 2015.

[33] Hal R. Varian. Big data: New tricks for econometrics. *Journal of Economic Perspectives*, 28(2):3–28, May 2014.

[34] Lei Yu and Huan Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *In Proceedings of the 20th international conference on machine learning*, pages 856–863. The publisher of the proceedings, 2003.

[35] Mohammed J Zaki. Parallel sequence mining on shared-memory machines. *Journal of Parallel and Distributed Computing*, 61(3):401–426, 2001.