Journal of Computer Science

Original Research Paper

Mapping Learning Algorithms on Data, a Useful Step for Optimizing Performances and Their Comparison

Filippo Neri

Department of Computer Science, University of Naples, Italy

Article history
Received: 19-03-2024
Revised: 02-07-2024
Accepted: 08-07-2024

Email: filippo.neri.email@gmail.com

Abstract: In this paper, we propose a novel methodology to map learning algorithms on data (performance map) in order to gain more insights into the distribution of their performances across their parameter space. This methodology provides useful information when selecting a learner's best configuration for the data at hand and it also enhances the comparison of learners across learning contexts. In order to explain the proposed methodology, the study introduces the notions of learning context, performance map, and high-performance function. It then applies these concepts to a variety of learning contexts to show how their use can provide more insights into a learner's behavior and can enhance the comparison of learners across learning contexts. The study is completed by an extensive experimental study describing how the proposed methodology can be applied.

Keywords: Learning Algorithms, Decision Trees, Support Vector Machines, Meta-Optimization of Learners, Comparing Learning Algorithms, Performance Maps of Learning Contexts

Introduction

The standard approach used in machine learning is to compare learning algorithms in contrast to their performances on a data set unseen during the learning phase. A learner's performance is expressed in the form of a single numeric value representing, for instance, its accuracy, the error rate, etc. Usually, a confidence interval around the mean performance value is also provided. However, in the end, a whole learner behavior is condensed into just one single number (i.e., the mean accuracy).

All other information about the learning process (i.e., how the search in the hypothesis space was conducted, what effect changing learning parameters produces, how human-readable the found concept, etc.,) is simply discarded. From the theoretical point of view, the user is then supposed to select one learner over the other just by considering a single number.

On the opposite, from the practical point of view, the literature papers may only be partially helpful as they usually hide away the important step of parameter selection that is, however, performed by the authors but generally not discussed in the paper.

When considering real data, we believe, instead, that a) the step of parameter selection should be considered a full part of the learning process and that b) a learner's parameter sensitivity should play a role in comparing learners across different learning contexts. In fact, if a learner's result is very sensitive to its settings, the user may want to consider selecting a lower-performing learner with stabler results to ensure a more robust behavior on future data.

Following the above considerations, this study describes a new methodology to compare learning systems by using performance maps that make explicit a learner's sensitivity to its parameter settings.

We define a performance map as the set of performance values, associated with the parameter settings that produced them when a leaner is applied to some data. Performance maps are functions of learning contexts. In order to understand how to build them, let us then define what a learning context is for the extent of this study. A learning context *LC* is a quadruple made of:

1. A learning algorithm *L*
2. A meta-optimization method *M*
3. The meta-optimized parameter space MOPS: The set of parameter settings for *L* considered during meta-optimization and
4. A data set D

Meta-optimization of learning systems, hyper-parameter learning, or meta-learning consists in finding the best-performing parameter settings for a learner by searching the space of all possible parameter settings (Blum and Roli, 2003; Grefenstette, 1986; Eiben *et al*., 1999; Reif *et al*., 2012; Feurer and Hutter, 2019; Lorenzo *et al*., 2017; Neri, 2022; 2024).

Meta-optimization of a learner *L* is achieved by performing multiple runs of *L* on *D*, using several parameter settings, in order to evaluate *L*'s performance for each considered parameter setting. Either an exhaustive search or a species meta-optimization algorithm *M* can be used. And, the set of *L*'*s* parameter settings evaluated during the meta-optimization process is the Meta-Optimized Parameter Space (MOPS). The collection of pairs <*s*, *L*'s performance >, with *s* in MOPS, allows us to create the performance map (*LC*) that we are interested in.

The selected meta-optimization method *M* determines the composition of MOPS and, in turn, of the performance map (*LC*). Performance maps can be either complete if MOPS is equal to the set of all parameter settings for *L*, or partial/approximated if MOPS is a proper subset of it.

Meta-optimization is very effective and can improve significantly the performance of a learning algorithm (Camilleri *et al*., 2014; Camilleri and Neri, 2014). We will show some instances of the case in the experimental part below. In this study, however, we do not focus on meta-optimization per se but we use it only as a tool to build performance maps. Complementary methodologies to meta optimization in learning include methods that helps understanding (i.e., explaining) the decision made by the learning system (Neri, 2023a; Neri, 2023c).

In the description of how performance maps are created, the machine learner expert can easily recognize a formalized version of the manual parameter tuning process accomplished by all authors in order to select the 'most suitable' configuration for running the learners discussed in their papers.

Novelties of this study include:

1. The notion of learning context and it used to compare learning algorithms or to tune their performance
2. The definition of performance maps and how they can be used to compare learners
3. The description of how to create approximate (partial) performance maps with relatively low computational cost yet providing' satisfactory' information
4. The suggestion that previous research in the literature has been implicitly using a weak version of the performance maps method, here described, usually performed informally by the authors before selecting the configuration to use in the learners discussed in their papers.
5. The suggestion that comparison tables among learners, presented in the literature, would benefit from being expanded and recalculated according to performance maps to provide more insights to the reader looking for the best learner/configuration when dealing with a species data set
6. The observation is that performance maps t nicely in the scope of the No Free Lunch Theorem (NFL) (Wolpert and Macready, 1997). The NFL theorem states that no learning algorithm can outperform all the others over all data sets. Our proposal makes explicit that changing parameter settings of a learning algorithm produces a different learner which usually has different a performance.

Finally, it is worthwhile to mention the challenging research environment where this research has originated (Neri, 2021c; 2023b).

Materials and Methods

State of the Art in Comparing Learning Algorithms

The standard procedure to compare learning algorithms consists of contrasting their performances on several data sets. It must be added that the comparison is done after an ad hoc selection of the better-performing parameter settings for the learners. Usually manually discovered by running some trial tests.

Traditional performance measures include Accuracy, error rate, *R*, etc. Their values are generally determined by using a statistical methodology called n-fold cross-validation (usually 5 or 10 folds are selected) on the whole available data in order to determine a performance interval (mean standard deviation) with known statistical confidence (Refaeilzadeh *et al*., 2009; Stone, 1974). Because performance measures reduce to a single value the whole learner's behavior, they may potentially miss important aspects of the underlying learning process like, for instance, the distribution of the performances over the parameter space of the learner.

In addition to traditional performance measures, other methodologies exist to evaluate a learner's performance. For instance: The Area Under the ROC Curve (AUC) (Bradley, 1997) or the rolling cross-validation (Racine, 2000; Bergmeir and Benítez, 2012). AUC is applicable to any classier producing a score for each case, but less appropriate for discrete classifiers like decision trees. Rolling cross-validation is only applicable to specie data types like time series or data streams (Racine, 2000; Bergmeir and Benítez, 2012). In fact, more recent performance measures are not generally applicable across learners or data types.

We then believe that, when learners need to be compared, the information provided by the above performance measures could be enhanced by including some insights about the distribution of performances on the learners' parameter spaces. The latter information would allow, for instance, to take into account the probability of achieving a high performance by randomly selecting a parameter from the learner's parameter space with uniform probability. Thus providing a measure of confidence or stability in the best performance achieved in the learning context under study.

Our Proposal: Comparing Learning Algorithms with Performances Maps and Their HP(k) Values

This study proposes to compare learning algorithms by confronting their performance maps and their *HP* (*k*) values.

As said, given a learning context *LC*, its performance map *Pmap* (*LC*) is the collection of pairs < s, *L* (*s*) >, with *s* in *MOPS* and *L* (*s*) as the performance of *L* run with settings *s*. From *Pmap*(*LC*), it is very simple to determine its best performance (*LC*) (the map's maximum).

The High-Performance function of a map $HP_{Pmap(LC)}(k)$ is defined as the ratio between the number of parameter settings in *MOPS* producing a performance with distance *k* from *best* (*LC*) and the cardinality of MOPS, as in Eq. (1):

$$HP_{Pmap(LC)}(k) = \frac{\left|\{p \mid p \in MOPS \wedge L(p) \geq best(LC) * (1-k)\}\right|}{|MOPS|} \quad (1)$$

Where, $p \in MOPS$; $0 < k < 1$ and *L* (*p*) is the performance observed by running *L* with parameter settings *p* on the data *D*. In the following, we will use $HP_{LC}(k)$, or simply *HP* (*k*) when the learning context is clear, as shorthand for $HP_{Pmap(LC)}(k)$.

$HP_{LC}(k)$ also represents the fraction of the map area above a certain performance level (*best*(*LC*) (1-*k*)) over the whole map extension. And, from another point of view, HP_{LC} (*k*) is an estimate of the cumulative distribution function $Prob_{LC}(X > best(LC)\ (1-k))$, where *X* is *L* (*s*) and *s* is randomly taken from *MOPS* with uniform distribution.

We will show, in the experimental session, the values of $HP_{LC}(k)$ for several learning contexts.

Learners and Meta-Optimization Methods

As said, the aim of our work is to compare learners across learning contexts by using performance maps. In order to practically show how our proposal works, we selected two learners and two meta-optimization methods so that we were able to present a full set of experiments.

Decision Trees (DT) (Quinlan, 2014) and Support Vector Machines (SVM) (Cortes and Vapnik, 1995) are selected as learners because they internally represent knowledge in very different ways, thus demonstrating the general applicability of our methodology. As meta-optimization methods, we selected Grid Search, which consists of the exhaustive enumeration of an input parameter space and Simple Genetic Algorithm (SGA) (Goldberg, 1989; Neri, 2005), in order to account for the case of partial search of the input parameter space and the ensuing partial performance map. We note that one can choose to build a partial performance map as it has a lower computational cost than a complete one. The pseudo-code for the used SGA and Grid Search can be found in Appendix A.

The Parameter Spaces for the Selected Learners

The chosen parameter spaces for DT and SVM are shown in Tables 1-2. These are the parameter spaces searched by the meta optimizer.

Table 1: Value ranges for the selected parameters of DT

Learner	Min impurity	Min samples	Max depth	Timeout (secs)
DT	{f i/10 for i = 0 to 6g}	{f i for i = 2 to 150 step 10g}	{f i for i = 1 to 160 step 10g}	40

Table 2: Value ranges for the selected parameters of SVM

Learner	Gamma	Kernel	C value	Timeout (secs)
SV M	scale auto	Linear poly rbf sigmoid	f i/100 for i = 1 to 200 steps 20 g S f i for i = 2 to 200 step 20 g	40

In the case of DT, the parameters that mostly affect its results have been identified as Minimum impurity decrease (decrease of a node's impurity to allow for a node splitting), minimum samples (the minimum number of samples required to split an internal node) and max depth (the maximum allowed depth of the tree). The parameter space for DT contains combinations of values for the three selected parameters. Similarly, for SVM, the chosen parameters are gamma, kernel, and C value, which affect the types of hyperplanes to be used and their boundary positions (margin distance). Again the combination of values for these three parameters dene the parameter space for SVM.

It is important to note that our methodology is not limited by the number of parameters used to define a parameter space. In this experimentation, we dene the parameter spaces with only three parameters per learner simply because this choice will allow us to draw a 3-dimensional representation of the performance maps built in the experiments. Thus facilitating the understanding of our work. If we had used more parameters it would have been difficult to show the results in a graphical form.

The Timeout columns in the tables report the maximum number of seconds an experiment will run before timing out. As an anticipation, an experiment consists of performing several 10-fold cross-validations of the selected learner on the available data in order to meta-optimize it.

Using a timeout is necessary for some data sets and learners given the long run time required. In this study, the time out is particularly needed when SVM is applied to the Pima Indians Diabetes and Abalone data sets which may require more than 30 min for each experiment. Resulting in a full experimentation running for several hours. The use of timeouts does not affect our comparison methodology though it may produce approximate performance maps. We denote a timeout experiment with a negative value equal to -0.2 on a performance map.

Parameter settings for the meta-optimization methods

In the case of Grid Search, no parameters affect its behavior because all points in the given parameter space are evaluated.

In the case of SGA, instead, it is known that the population size and the maximum number of generations can deeply affect the result found by a genetic algorithm. Here is why, in order to find the best parameter settings

for the SGA, we meta-optimized the SGA by using a Grid Search applied to the following parameter ranges: Population size (30, 50, 80), max number of generations (30, 50, 80), crossover probability (0.5, 0.7, 0.9) and learner (DT or SVM).

As a performance measure, we were interested in the genetic algorithm discovering a parameter setting performing as close as possible to the best performance discovered by Grid Search when used as a meta-optimizer in the learning contexts. Also by using the lowest possible population size and max generations.

The parameter settings for SGA are Population size equal to 50, max number of generations equal to 50, and crossover probability equal to 0.9. The fact that genetic algorithms, in general, are robust learners makes it quite easy to find one of the many suitable parameter settings (Neri, 2005; 2008).

We kept the remaining parameters of SGA to their default values as set in the Python library Genetic Algorithm (https://pypi.org/project/geneticalgorithm/) from which we built the SGA used in this study.

Data Set Descriptions

To perform the experiments in our study, we selected four data sets with varying characteristics from the UCI Machine Learning repository:

1. Mushrooms -8124 instances, 22 attributes (categorical), classification task: To predict if a mushroom is either edible or poisonous from some physical characteristics (Schlimmer, 1987)
2. Pima Indians Diabetes -769 instances, 8 attributes (categorical), classification task: To predict if the patients have or do not have diabetes based on some diagnostic measurements. Source: https://www.kaggle.com/uciml/pima-indians-diabetes-database
3. Congressional voting records -435 instances, 16 attributes (categorical), classification task: Predicting Republican or Democratic membership from vote record (Schlimmer, 1987)
4. Abalone -4177 instances, 8 attributes (categorical, integer, real), regression task: Predicting the age of abalone (a marine snail) from its physical measurements (Waugh, 1995)

An open research question is if the proposed methodology needs to be extended when different data types for instance financial time series (Neri, 2012a; 2012c; 2010; 2011) or unusual domains are considered (García-Magariño *et al*., 2019).

Building a performance map then could be particularly useful when selecting a learner for some novel data, because it provides information on the robustness of the learner when different configurations are used, a situation which is bound to happen in real-world usage of a learning system.

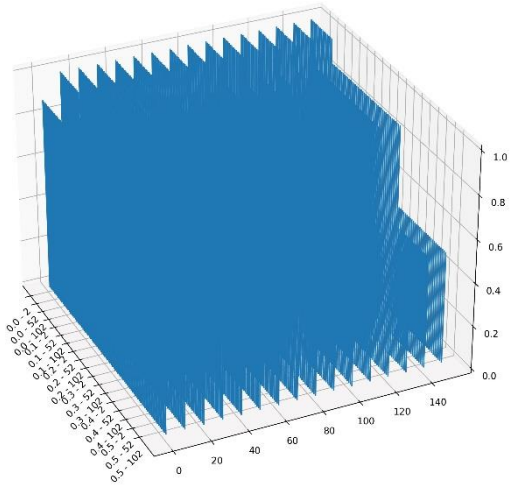
Here is why we believe that comparing learners by using performance maps provides more insights than the use of a single-valued performance measure as traditionally done in the literature.

Results and Discussion

Figures 1-4 show the performance maps for the learning contexts of Table 3. The peruse makes explicit that:

1. If we consider all learning contexts, DT performs better in a region of the parameter space where ’min impurity’ is close to 0, ’min sample' is below 50, and 'max depth' is above 20. When increasing the 'min impurity' value above 0.2, the performance decreases abruptly and significantly
2. If we consider all learning contexts, SVM performs better in a region or the parameter space where ’gamma’ is equal to ’scale’, ‘C-value' is lower than 1.0, and ‘kernel’ is ‘poly’, ‘rbf’ or ‘linear’
3. However, if we are interested in a species learner and data, the performance map shows the locations of the highest-performing parameter settings and it displays how these regions vary in location and extensions across the parameter space.
4. Performance maps do not need to be complete to be useful. Completeness may require a high computational cost to achieve. Indeed, even partial performance maps are very helpful in selecting high-performing parameter settings over just a blind selection of the same done by manually undertaking trial runs. Comparing performance maps using Grid Search with those using SGA demonstrates the point.

Moreover, by perusing the results in Table 3 and the performance maps, one can observe that even with relatively low computational costs, it is already possible to find high-performing parameter settings when an effective meta-optimizer, such as SGA, is applied to explore the learner’s parameter space.



(a)

DOI: 10.3844/jcssp.2024.1110.1120

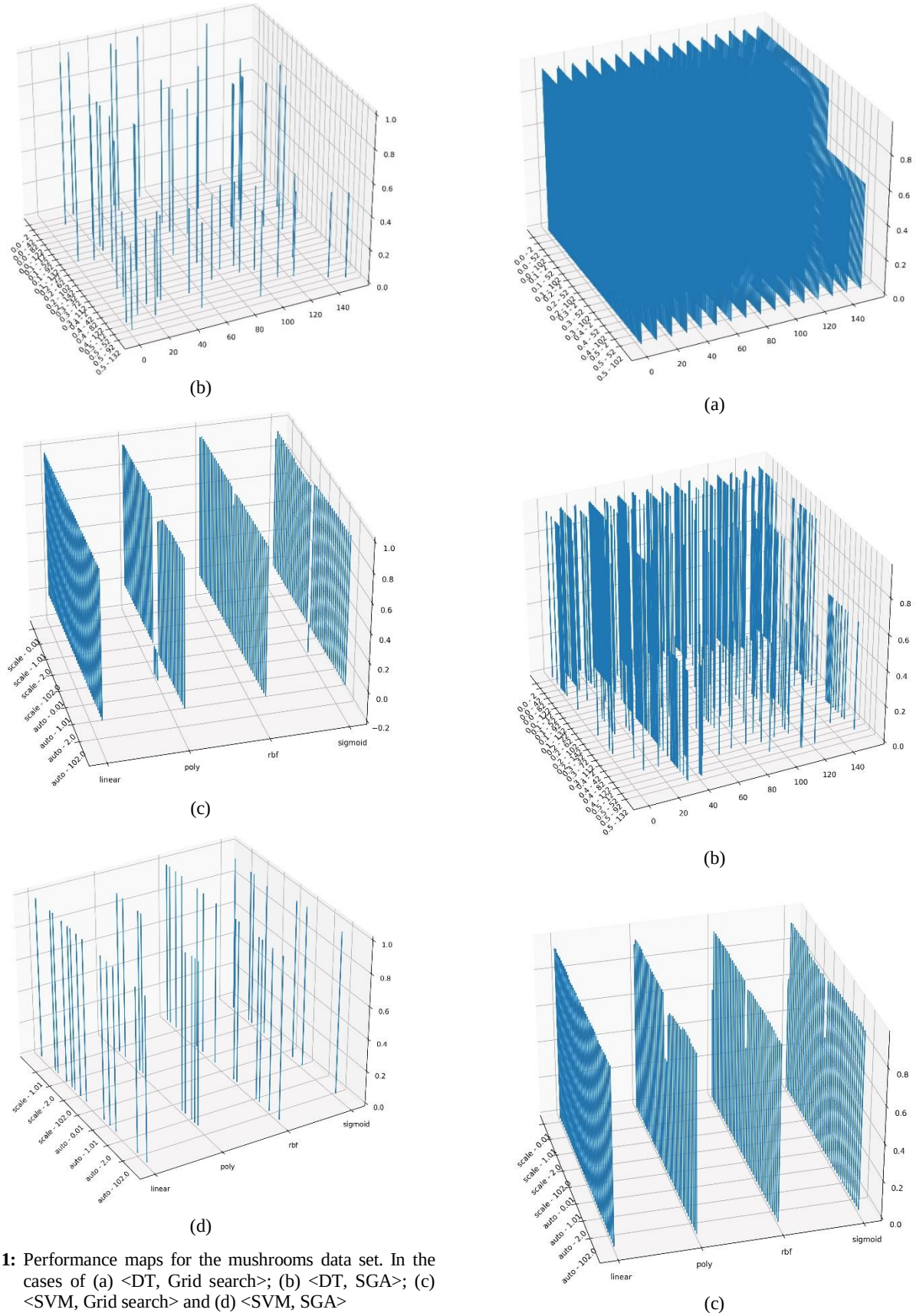
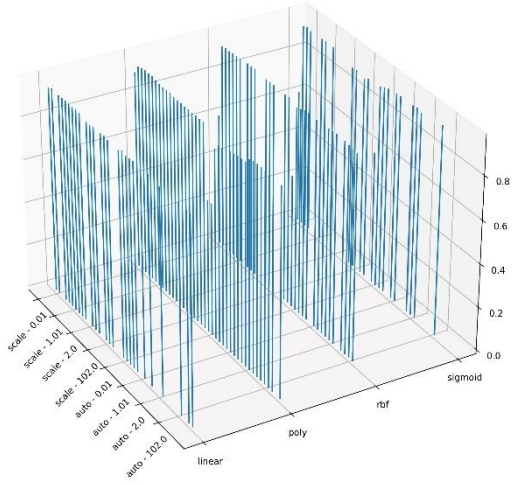
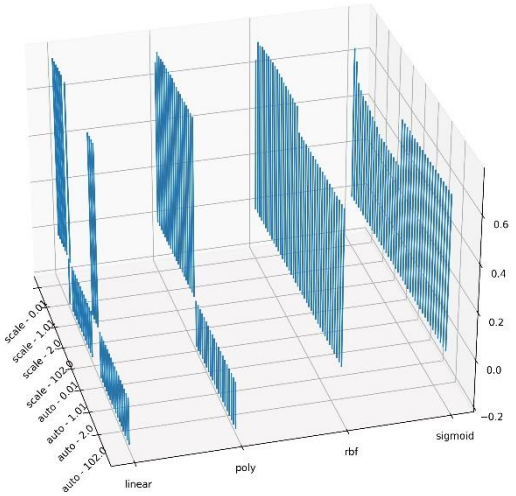


Fig. 1: Performance maps for the mushrooms data set. In the cases of (a) <DT, Grid search>; (b) <DT, SGA>; (c) <SVM, Grid search> and (d) <SVM, SGA>

DOI: 10.3844/jcssp.2024.1110.1120

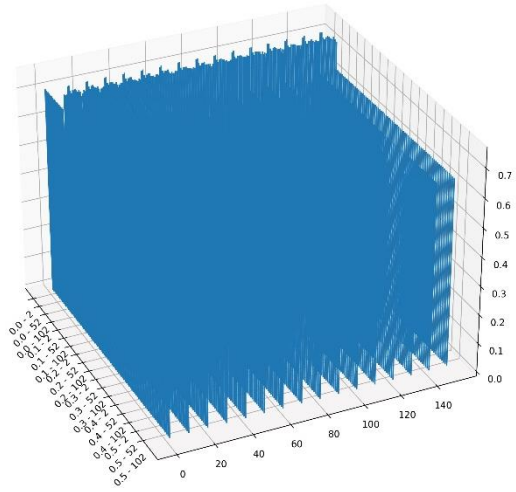


(d)

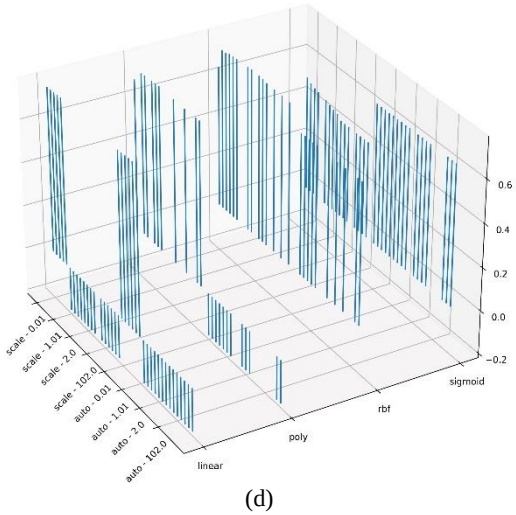


(c)

Fig. 2: Performance maps for the congressional voting records data set. In the cases of (a) <DT, Grid search>; (b) <DT, SGA>; (c) <SVM, Grid search> and (d) <SVM, SGA>

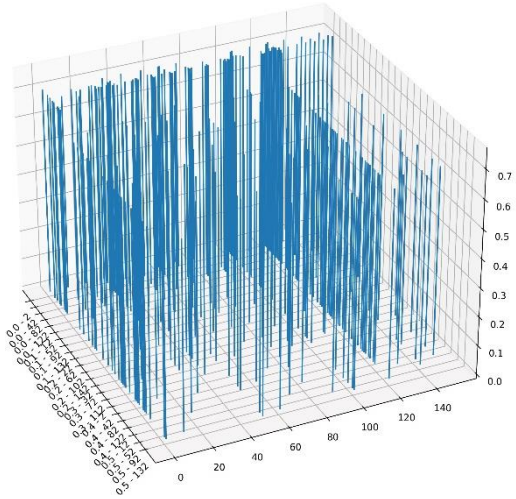


(a)

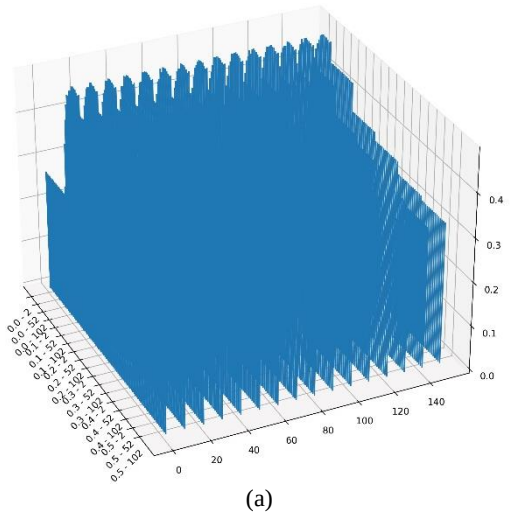


(d)

Fig. 3: Performance maps for the Pima Indians data set. In the cases of (a) <DT, Grid search>; (b) <DT, SGA>; (c) <SVM, Grid search> and (d) <SVM, SGA>

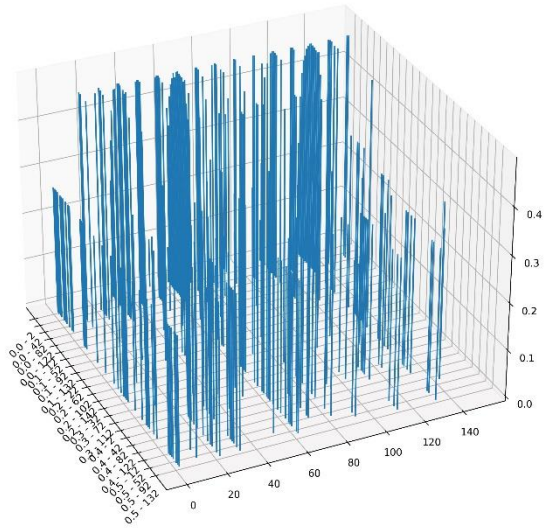


(b)

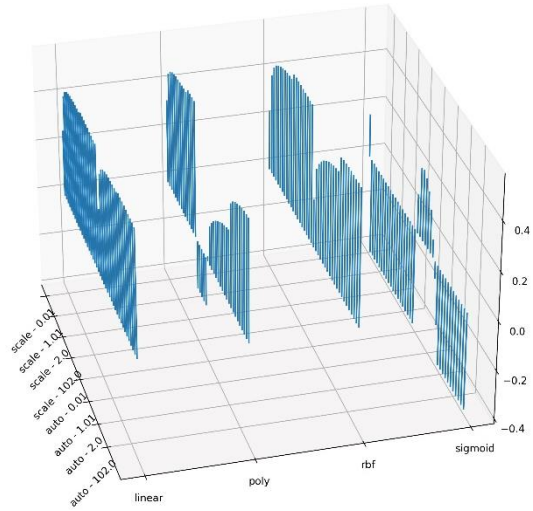


(a)

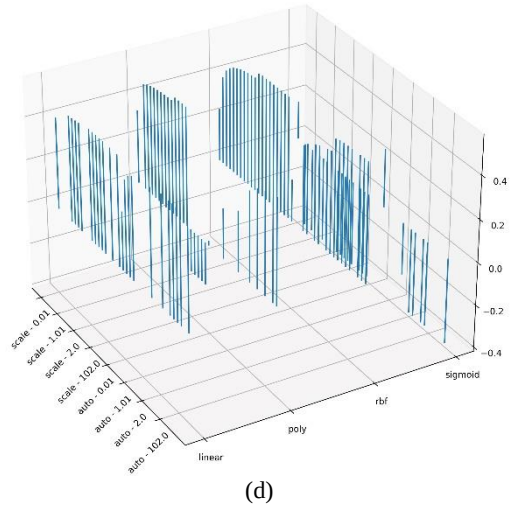
DOI: 10.3844/jcssp.2024.1110.1120



(b)



(c)



(d)

Fig. 4: Performance maps for the abalone data set. In the cases of (a) <DT, Grid search>; (b) <DT, SGA>; (c) <SVM, Grid search> and (d) <SVM, SGA>. Note that the highest points in (c) for 'poly' and 'rbf' reach the value of 0.56. Perspective makes them appear to be lower

High Performance in Learning Contexts

Table 4 introduces an additional measure to assist in comparing learners across learning contexts: The high-performance values *HP* (*k*). As introduced in Section 2.1, *HP* (*k*) measures how frequently are high-performing parameter settings within a (*k* 100)% distance from the maximum on a given performance map. *HP* (*k*) values thus allow us to express in short one of the main insights offered by a performance map: How easy or difficult is to find high-performing parameter settings for the learning context.

We could then compare two learning contexts in terms of their *HP* (*k*) values for a given selection of *k* distances. We could define a learning context as higher performant than another if it has higher *HP* (*k*) values for a given selection of *k* distances.

Table 3: Meta-optimization of learners in several learning contexts

Dataset	Learner and meta optimization	Best accuracy/R^2	Std	Evaluated points	Time
Mushrooms	DT-Grid	1.0	0.00	1440	197.45
	DT-SGA	1.0	0.00	49	6.70
	SVM-Grid	1.0	0.00	160	1000.25
	SVM-SGA	1.0	0.00	47	320.30
Congr. votes	DT-Grid	0.96	0.03	1440	18.08
	DT-SGA	0.96	0.03	272	5.28
	SVM-Grid	0.97	0.02	160	5.50
	SVM-SGA	0.96	0.02	129	6.11
Diabetes	DT-Grid	0.75	0.04	1440	62.53
	DT-SGA	0.75	0.04	241	12.50
	SVM-Grid	0.76	0.04	160	2312.26
	SVM-SGA	0.76	0.04	122	2127.17
Abalone (R^2)	DT-Grid	0.49	0.02	1440	133.13
	DT-SGA	0.49	0.02	291	35.22
	SVM-Grid	0.56	0.02	160	1512.06
	SVM-SGA	0.56	0.02	109	1068.36

Table 4: High-Performance values *HP* (*k*)) in several learning contexts

Data set	Learner and meta optimization	Best accuracy	HP (0.05) (within 5% of best)	HP (0.10) (within 10% of best)	HP (0.20) (within 20% of best)
Mushrooms	DT-Grid	1.00	0.16	0.16	0.66
Mushrooms	DT-SGA	1.00	0.25	0.25	0.65
Mushrooms	SVM-Grid	1.00	0.89	0.97	0.98
Mushrooms	SVM-SGA	1.00	0.89	0.93	1.00
Congr. voting rec.	DT-Grid	0.96	0.66	0.66	0.66
Congr. voting rec.	DT-SGA	0.96	0.78	0.78	0.78
Congr. voting rec.	SVM-Grid	0.97	0.91	0.96	0.96
Congr. voting rec.	SVM-SGA	0.96	0.91	0.96	0.96
Diabetes	DT-Grid	0.75	0.12	0.15	1.00
Diabetes	DT-SGA	0.75	0.32	0.39	1.00
Diabetes	SVM-Grid	0.77	0.31	0.32	0.58
Diabetes	SVM-SGA	0.77	0.30	0.30	0.57
Abalone	DT-Grid	0.49	0.09	0.23	0.28
Abalone	DT-SGA	0.49	0.25	0.40	0.45
Abalone	SVM-Grid	0.56	0.14	0.32	0.54
Abalone	SVM-SGA	0.56	0.17	0.36	0.59

From Table 4, one can observe that the learning contexts with SGA as the meta-optimizer have higher *HP* (*k*) values than those associated with Grid Search. This means that the performance maps associated with SGA contain more parameter settings performing closer to the maximums than performance maps associated with Grid search.

This finding is due to the capability of genetic algorithms to focus their search towards high-performing parameter settings and to avoid low-performing ones. On the Contrary, Grid Search will have to include all parameter settings in its exploration of the parameter space.

In addition, considering the Congressional Voting Records data set, one can note that the learning context with SVM and SGA dominates the learning context with SVM and Grid Search. Indeed, Table 4 shows that SVM is generally a more robust learner than DT across the considered learning contexts finding consistently higher HP-valued performance maps except in the case of Diabetes (the No Free Lunch theorem at works!).

Methodology results in conclusion, the better-performing pair <learner, meta optimizer> appears to be <SVM, SGA> over the considered learning contexts.

We complete our experimental study by repeating that using classic performance measures (accuracy, error rate, etc.) together with performance maps and HP values allows for a multi-faceted comparison of learning algorithms across data sets including robustness to varying parameter settings for the learner.

We believe that having more insight into the behavior of a learner is especially useful when dealing with novel, unseen data. Indeed, being able to calculate and possibly visualize its performance map provides more confidence in how the learner would behave in the future and what subset of parameter settings are likely to produce high-performing outcomes: the highest the *HP* (*k*) values, the highest the probability that the learner will operate within the [best performance *(1-*k*), best performance] range when variation to its configurations settings will happen in the future.

Conclusion

In the paper, we propose to map learning algorithms on data (performance map) in order to gain more insights into the distribution of their performances across their parameter space. This approach provides useful information when selecting the best configuration for a learning context and when comparing alternative learners. To formalize the above ideas, we introduced the notions of learning context, performance map, and high-performance function. We then applied the concepts to a variety of learning contexts to show their capabilities.

We showed that the proposed methodology can provide more information on the robustness of a learner in a given learning context thus enriching the traditional single-valued performance measures used in literature when comparing learners.

Future research directions are plentiful. Because meta-optimization is a separate learning task itself, it opens up a series of interesting research questions like How to better use relatively small data samples or data streams.

Another direction is to study the application of this methodology to more sophisticated learning systems such as agent-based systems for modeling complex time series in financial applications (Neri and García-Magariño, 2020; Neri, 2012b; 2018; 2019; 2021a-b). Or what will happen when neural networks are used as learners? How to select their most important parameters and how to deal with their long training time maybe in control applications (Marino and Neri, 2019)?

A Simple Genetic Algorithm and Grid Search

In the study, we use two meta-optimizers SGA and Grid search. The pseudo code for the SGA used in this study can be found in Table 5 and that of Grid search can be found in Table 6.

Table 5: Simple Genetic Algorithm

```
//Note: each individual codes for a parameter set for the Learner
//Function DoExperiment performs a 10 fold cross validation
//on Learner, configured with the parameters coded by an individual,
//applied on the data set Data
EvaluateFitness(Population, Learner, Data)
  for each individual in the Population
      Fitness(individual )= DoExperiment(Learner, individual, Data)
SGA(PopulationSize, MaxGenerations, Learner, Data) nGen = 1
  BestIndividual = {}
  Population = initPopulation(PopulationSize, Learner)
  EvaluateFitness(Population, Learner, Data) while nGen <
  MaxGenerations
      MatingPool = Select(Population)
      DoC rossOver(MatingPool)
      DoMutation(MatingPool)
      NextGenPopulation = ReplaceIn(MatingPool, Population)
      EvaluateFitness(NextGenPopulation, Learner, Data)
      Population = NextGenPopulation
      Maintain(BestIndividual,Population) //elitism
      BestIndividual = FindBestSolution(Population) nGen =
      nGen + 1
  end while return(BestIndividual)
```

Table 6: The Grid Search Algorithm.

```
//ParameterSpace contains all combinations of parameters for the Learner
  GridSearch(Learner, Data, ParameterSpace)
      BestParameterSettings = {}
      BestAccuracy = 0
      For each p in ParameterSpace
          Accuracy = DoExperiment(Learner, p, Data)
          if (Accuracy > BestAccuracy) then
              BestParameterSettings = p
          return(BestParameterSettings)
```

One of the meta-optimization methods used in our work is a Simple Genetic Algorithm (SGA) with elitism (Goldberg, 1989). SGA is a well-known algorithm therefore we will not explain it in detail. We implemented SGA in Python 3.8, by adapting the library genetic algorithm. In particular, we improved the SGA in the library by (1) adding a cache memory inside the fitness function to avoid repeated evaluations of the same individual and (2) adding a stopping criterion based on a minimum level of performance. The SGA stops when its best individual has a fitness equal to or above the given minimum. We did not add these two improvements in the code in Table 5 to improve its readability.

The parameters used to run the SGA in all the learning contexts are Max generation = 50, population size = 50, mutation rate = 0.1, crossover rate = 0.9, replacement rate = 0.9, crossover-type = uniform, stop-when-fitness-is-above = 0.99.

The second meta-optimization method used in this study is Grid Search. Grid Search consists of enumerating all the possible values inside a given search space and evaluating them. Also, Grid Search is a well-known algorithm so we will not comment on it.

Acknowledgment

We thanks the anonymous reviewers for their insightful comments.

Funding Information

The research here reported has not been funded.

Ethics

The research here reported has been respectful of any ethical consideration concerning machine learning research. Benchmark datasets have been used.

References

Bergmeir, C., & Benítez, J. M. (2012). On the use of cross-validation for time series predictor evaluation. *Information Sciences*, *191*, 192-213. https://doi.org/10.1016/j.ins.2011.12.028

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, *35*(3), 268-308. https://doi.org/10.1145/937503.937505

Bradley, A. P. (1997). The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, *30*(7), 1145-1159. https://doi.org/10.1016/S0031-3203(96)00142-2

Camilleri, M., & Neri, F. (2014). Parameter optimization in decision tree learning by using simple genetic algorithms. *WSEAS Transactions on Computers*, *13*, 582-591. https://core.ac.uk/download/pdf/55147372.pdf

Camilleri, M., Neri, F., & Papoutsidakis, M. (2014). An algorithmic approach to parameter selection in machine learning using meta-optimization techniques. *WSEAS Transactions on Systems*, *13*(1), 203-212. https://www.wseas.com/journals/systems/2014/a165702-311.pdf

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273-297. https://doi.org/10.1007/BF00994018

Eiben, Á. E., Hinterding, R., & Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, *3*(2), 124-141. https://doi.org/10.1109/4235.771166

Feurer, M., & Hutter, F. (2019). Hyperparameter optimization. *Automated Machine Learning: Methods, Systems, Challenges*, 3-33. https://library.oapen.org/bitstream/handle/20.500.12657/23012/1/1007149.pdf#page=15

García-Magariño, I., Plaza, I., & Neri, F. (2019). ABS-MindBurnout: An agent-based simulator of the effects of mindfulness-based interventions on job burnout. *Journal of Computational Science*, *36*, 101012. https://doi.org/10.1016/j.jocs.2019.06.009

Goldberg, D. E. (1989). Genetic Algorithm in Search, Optimization and Machine Learning, Addison. *Wesley Publishing Company, Reading, MA*, *1*(98), pp: 412. ISBN-10: 9780201157673.

Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, *16*(1), 122-128. https://doi.org/10.1109/TSMC.1986.289288

Lorenzo, P. R., Nalepa, J., Kawulok, M., Ramos, L. S., & Pastor, J. R. (2017). Particle swarm optimization for hyper-parameter selection in deep neural networks. In *Proceedings of the genetic and evolutionary computation conference* (pp. 481-488). https://doi.org/10.1145/3071178.3071208

Marino, A., & Neri, F. (2019). PID Tuning with Neural Networks. *Intelligent Information and Database Systems*, 476–487. https://doi.org/10.1007/978-3-030-14799-0_41

Neri, F. (2005). Traffic packet based intrusion detection: decision trees and genetic based learning evaluation. *WSEAS Transactions on Computers*, *4*(9), 1017-1024.

Neri, F. (2008). PIRR: A methodology for distributed network management in mobile networks. *WSEAS Transactions on Information Science and Applications*, *5*(3), 306-311. https://core.ac.uk/download/pdf/55017866.pdf

Neri, F. (2010). Software agents as a versatile simulation tool to model complex systems. *WSEAS Transactions on Information Science and Applications*, *7*(5), 609-618.

Neri, F. (2011). Learning and predicting financial time series by combining natural computation and agent simulation. In *Applications of Evolutionary Computation: EvoApplications 2011: EvoCOMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, and EvoTRANSLOG, Torino, Italy, April 27-29, 2011, Proceedings, Part II* (pp. 111-119). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-20520-0_12

Neri, F. (2012a). A comparative study of a financial agent based simulator across learning scenarios. In *Agents and Data Mining Interaction: 7th International Workshop on Agents and Data Mining Interation, ADMI 2011, Taipei, Taiwan, May 2-6, 2011, Revised Selected Papers 7* (pp. 86-97). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-27609-5_7

Neri, F. (2012b). Agent-based modeling under partial and full knowledge learning settings to simulate financial markets. *AI Communications*, *25*(4), 295-304. https://doi.org/10.3233/AIC-2012-0537

Neri, F. (2012c). Learning predictive models for financial time series by using agent based simulations. *Transactions on Computational Collective Intelligence vi*, 202-221. https://doi.org/10.1007/978-3-642-29356-6_10.

Neri, F. (2018). Case study on modeling the silver and Nasdaq financial time series with simulated annealing. In *Trends and Advances in Information Systems and Technologies: Volume 2 6* (pp. 755-763). Springer International Publishing. https://doi.org/10.1007/978-3-319-77712-2_71

Neri, F. (2019). Combining machine learning and agent based modeling for gold price prediction. In *Artificial Life and Evolutionary Computation: 13th Italian Workshop, WIVACE 2018, Parma, Italy, September 10–12, 2018, Revised Selected Papers 13* (pp. 91-100). Springer International Publishing. https://doi.org/10.1007/978-3-030-21733-4_7

Neri, F. (2021a). Domain specific concept drift detectors for predicting financial time series. *arXiv preprint arXiv:2103.14079*. https://doi.org/10.48550/arXiv.2103.14079

Neri, F. (2021b). How to Identify Investor's types in real financial markets by means of agent based simulation. *Proceedings of the 2021 6th International Conference on Machine Learning Technologies*, 141–149. https://doi.org/10.1145/3468891.3468913

Neri, F., & García-Magariño, I. (2020). Simulating and modelling the DAX index and the USO Etf financial time series by using a simple agent-based learning architecture. *Expert Systems*, *37*(4), e12516. https://doi.org/10.1111/exsy.12516

Neri, F. (2021c). Some key cultural obstacles to doing AI research in the Italian Academic System. *Available at SSRN*, id 3917044. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3917044

Neri, F. (2022). Coevolution and learning symbolic concepts: statistical validation: Empirical statistical validation of co-evolutive machine learning systems. *Proceedings of the 2022 7th International Conference on Machine Learning Technologies (ICMLT 2022)*, ACM press, pp. 244-248.

Neri, F. (2023a). Explainability and interpretability in agent based modelling to approximate market indexes. *Proceedings of the 2023 8th International Conference on Machine Learning Technologies (ICMLT 2023)*, ACM press, pp. 139-143.

Neri, F. (2023b). Peer Reviewing in Experimental Studies: a False Narrative Damaging Academic Research, Hiding Fiefdoms Buildings, and Supporting an Undeserving Publishing Business. A Manifest. Call for Support and Sharing. *Available at SSRN*, id 4621491. https://papers.ssrn.com/sol3/papers.cfm?abstract_id = 4621491

Layfield, C. & Neri, F. (2023c). Explainability and Interpretability in Decision Trees and Agent based Modelling when Approximating Financial Time series. A matter of balance with performance. *Proceedings of the 2023 8th International Conference on Computational Intelligence and Applications (ICCIA 2023),* IEEE press, pp. 42-46.

Neri, F. (2024). Preface to the conference proceedings. *Proceedings of the 2024 9th International Conference on Machine Learning Technologies (ICMLT 2024),* ACM press, pp. 1.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, *12*, 2825-2830. https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf?ref=https:/

Quinlan, J. R. (2014). *C4. 5: Programs for machine learning*. Elsevier. pp: 312. ISBN-10: 9780080500584.

Racine, J. (2000). Consistent cross-validatory model-selection for dependent data: Hv-block cross-validation. *Journal of Econometrics*, *99*(1), 39-61. https://doi.org/10.1016/S0304-4076(00)00030-0

Refaeilzadeh, P., Tang, L., & Liu, H. (2009). Cross-validation. *Encyclopedia of Database Systems*, 532-538.

Reif, M., Shafait, F., & Dengel, A. (2012). Meta-learning for evolutionary parameter optimization of classifiers. *Machine Learning*, *87*, 357-380. https://doi.org/10.1007/s10994-012-5286-7

Schlimmer, J. C. (1987). *Concept acquisition through representational adjustment*. University of California, Irvine. https://www.proquest.com/openview/6e555535561b995253cf5f0b79c4921a/1?pq-origsite=gscholar&cbl=18750&diss=y

Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society: Series B (Methodological)*, *36*(2), 111-133. https://doi.org/10.1111/j.2517-6161.1974.tb00994.x

Waugh, S. (1995). Extending and benchmarking cascade-correlation. Department *of Computer Science, University of Tasmania, Ph. D. Dissertation*.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67-82. https://doi.org/10.1109/4235.585893