

Better Software Analytics via “DUO”: Data Mining Algorithms Using/Used-by Optimizers

Amritanshu Agrawal · Tim Menzies ·
Leandro L. Minku · Markus Wagner · Zhe Yu

the date of receipt and acceptance should be inserted later

Abstract This paper claims that a new field of empirical software engineering research and practice is emerging: data mining using/used-by optimizers for empirical studies, or DUO. For example, data miners can generate the models that are explored by optimizers. Also, optimizers can advise how to best adjust the control parameters of a data miner. This combined approach acts like an agent leaning over the shoulder of an analyst that advises “ask this question next” or “ignore that problem, it is not relevant to your goals”. Further, those agents can help us build “better” predictive models, where “better” can be either greater predictive accuracy, or faster modeling time (which, in turn, enables the exploration of a wider range of options). We also caution that the era of papers that just use data miners is coming to an end. Results obtained from an unoptimized data miner can be quickly refuted, just by applying an optimizer to produce a different (and better performing) model. Our conclusion, hence, is that for software analytics it is possible, useful and necessary to combine data mining and optimization using DUO.

Keywords Software analytics, data mining, optimization, evolutionary algorithms

Authors listed alphabetically.

Amritanshu Agrawal, Tim Menzies, Zhe Yu
Department of Computer Science, North Carolina State University, Raleigh, NC, USA. E-mail: zyu9@ncsu.edu

Leandro L. Minku
School of Computer Science, University of Birmingham, Edgbaston, Birmingham, UK. E-mail: l.l.minku@cs.bham.ac.uk

Markus Wagner
School of Computer Science, University of Adelaide, Adelaide, SA, Australia. E-mail: markus.wagner@adelaide.edu.au

1 Introduction

After *collecting data* about software projects, and before *making conclusions* about those projects, there is a middle step in empirical software engineering where the data is *interpreted*. When the data is very large and/or is expressed in terms of some complex model of software projects, then interpretation is often accomplished, in part, via some automatic algorithm. For example, an increasing number of empirical studies base their conclusions on data mining algorithms (e.g. see [9, 67, 69–71]) or model-intensive algorithms such as optimizers (e.g. see the recent section on Search-Based Software Engineering in the December 2016 issue of this journal [53]).

This paper asserts that, for software analytics it is *possible, useful* and *necessary* to combine data mining and optimization. We call this combination DUO, short for data miners using/sed-by optimizers. Once data miners and optimizers are combined, this results in a very different and interesting class of interpretation methods for empirical SE data. DUO acts like an agent leaning over the shoulder of an analyst that advises “ask this question next” or “ignore that problem, it is not relevant to your goals”. Further, DUO helps us build “better” predictive models, where “better” can be for instance greater predictive accuracy, or models that generalize better, or faster modeling time (which, in turn, enables the faster exploration of a wider range of options). Therefore, DUO can speed up and produce more reliable analyses in empirical studies.

This paper makes the following claims about DUO:

- **Claim1:** *Optimization and data mining are very similar.* Hence, it is natural and simple to combine the two methods.
- **Claim2:** *Optimizers can greatly improve data miners.* A data miner’s default tuners can lead to sub-optimal performance. Automatic optimizers can find tunings that dramatically improve that performance [3, 4, 36].
- **Claim3:** *Data miners can greatly improve optimization.* If a data miner groups together related items, an optimizer can explore and report conclusions that are general across a set of solutions. Further, optimization for SE problems can be very slow (e.g. consider the 15 years of CPU needed by Wang et al. [111]). But if that optimization executes over the groupings found by a data miner, that inference can terminate orders of magnitude faster [57, 82].
- **Claim4:** *Data mining without optimization is not recommended. Conclusions reached from an unoptimized data miner can be changed, sometimes even dramatically improved, by running the same tuned learner on the same data [36]. Researchers in data mining should therefore consider adding an optimization step to their analysis.*

This paper extends a prior conference paper on the same topic [79]. That prior paper focused on case study material for DUO-like applications (see Figure 1). While a useful resource, it did not place DUO in a broader context. Nor did it contain the systematic literature review of this paper. That is, this paper is both more general (discussing the broader context) and more specific (containing a detailed literature review) than prior work.

The rest of this paper is organized as follows. The next section describes some related work. After that, we devote one section to each of **Claim1**, **Claim2**, **Claim3** and **Claim4**. To defend these claims we use evidence drawn from a systematic literature review of applications of DUO, described in Table 1 (Table 2 and Table 3 offer notes on the data miners and optimizers seen in the literature review). Finally, a *Research Directions* section discusses numerous open research issues that could be better explored within the context of DUO.


ai-se / ResourcesDataDrivenSBSE
Unwatch 2 Star 2 Fork 0

Code Issues 8 Pull requests 0 Projects 0 Wiki Insights Settings

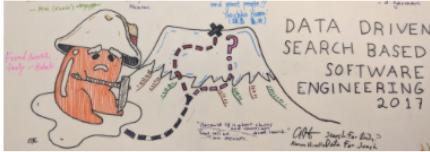
Branch: master ResourcesDataDrivenSBSE / README.md Find file Copy path

vivekaxl Update README.md cb2fc4d on Mar 13, 2018 3 contributors

24 lines (9 sloc) 1.22 KB Raw Blame History

Home | Contribute | Data | Models | Tools | Discuss | Citation | copyright ©2018
<http://tiny.cc/data-se>

DOI: 10.5281/zenodo.1156235

The recent seminar Data-Driven Search-Based Software Engineering (<https://goo.gl/f8D3EC>) concluded that (1) the mining software repositories can be improved using search-based methods; and that (2) search-based methods can be improved using tools from the MSR community. The resources of this repository are intended to encourage more experimentation in these two directions.



Resources:

Domain	Problem	Decision Space	C/D	Projects	Description	Links	Related Work
MSR	Defect Prediction	Numeric	D	10	CK Metric	raise_data_defect	[29]
				1	Citemap	raise_data_pits	
	Text Classification	Text	-	6	Pits	raise_data_pits	[1]
				1	StackOverflow	SOPProcess	
	Performance Optimization	Mixed	D	22	Performance Configuration optimization	raise_data_perf	[66, 67, 69]
SBSE	Software Product Lines	Boolean	D	5	Product Lines	raise_data_SPL	[15]
	General Optimization	Numeric	C	7	DTLZ	raise_dtlz_zdt	[65]
				6	ZDT	raise_dtlz_zdt	
	Workflow	Numeric	D	20	Workflow	raise_gen_workflow	[14]
	Text Discovery	Text	-	4	Reading Faster	raise_data_fastread	[103]
	Software Processes	Numeric	C	5	Xomo	raise_pom_xomo	[16, 65]
				4	POM3		
	Requirement Engineering	Numeric	D	8	Requirement Engineering	raise_short	[57]

References in the *related work* column come from the paper http://tiny.cc/dse_paper.

Fig. 1: Training and teaching resources for this work: <http://tiny.cc/data-se>. The authors of this paper invite the international research community to issue pull requests on this material.

Research questions	Q1. What papers have used DUO in the past? Q2. When were they published? Q3. What problems from what software engineering domains have they tackled? Q4. What optimizers have they used? Q5. What data miners have they used? Q6. What were the advantages offered by DUO?																																					
Search query	software engineering AND (“optimization” OR “evolutionary algorithm”) AND (“data mining” OR “analytics” OR “machine learning”)																																					
Search engines	Three widely used literature sources were adopted: – ACM Guide to Computing Literature (https://dl.acm.org/advsearch.cfm?coll=DL&dl=ACM) – IEEEExplore (https://ieeexplore.ieee.org/search/advsearch.jsp) – Google Scholar (https://scholar.google.com/#d=gs_asd&p=&u=)																																					
Inclusion criteria	– ACM and IEEEExplore: >3 citations per year OR published in 2017/2018. – Google Scholar: (>10 cites/year OR published in 2017/2018) AND in first 20 result pages. – For all search engines: paper relates to software engineering and must use DUO. The number of citations was more strict in Google Scholar, because this search engine usually retrieves more citations than the others. We restricted the Google Scholar results to the first 20 pages (200 papers) because Google Scholar allows papers that do not match the search query completely to be retrieved, resulting in 28,000 results that would need to be manually filtered for the software engineering domain and number of citations per year.																																					
Q1	The search query retrieved 393 results, when considering all ACM, IEEEExplore and the first 20 pages of Google Scholar results. After excluding papers that did not match the citation and year of publication criteria, we obtained 90 papers. After excluding any duplicates and papers that were not in the software engineering domain, we obtained 72 papers. Finally, among these 72, 29 really used DUO. These are: [1, 3–5, 19–21, 23, 26, 29, 34, 36, 48, 49, 60, 64, 65, 75, 76, 81, 82, 84, 85, 96, 98–100, 107, 121]. In addition, we found a systematic literature review related to DUO [2]. This systematic review is on the topic of genetic programming for predictive modeling in software engineering. Genetic programming can be seen as an optimization algorithm.																																					
Q2	The number of DUO papers published per year is shown below, with 2018 having the largest number of papers. One might claim that this is because we ignore the number of citations in 2018. However, this is also the case for 2017, which had a considerably smaller number of DUO papers. Therefore, DUO seems to have been recently attracting increased research interest. <div><div></div><table><tr><td>2002</td><td>1</td><td><div></div></td></tr><tr><td>2004</td><td>1</td><td><div></div></td></tr><tr><td>2006</td><td>1</td><td><div></div></td></tr><tr><td>2007</td><td>1</td><td><div></div></td></tr><tr><td>2008</td><td>2</td><td><div></div></td></tr><tr><td>2010</td><td>3</td><td><div></div></td></tr><tr><td>2012</td><td>2</td><td><div></div></td></tr><tr><td>2013</td><td>3</td><td><div></div></td></tr><tr><td>2015</td><td>1</td><td><div></div></td></tr><tr><td>2016</td><td>2</td><td><div></div></td></tr><tr><td>2017</td><td>2</td><td><div></div></td></tr><tr><td>2018</td><td>10</td><td><div></div></td></tr></table></div>		2002	1	<div></div>	2004	1	<div></div>	2006	1	<div></div>	2007	1	<div></div>	2008	2	<div></div>	2010	3	<div></div>	2012	2	<div></div>	2013	3	<div></div>	2015	1	<div></div>	2016	2	<div></div>	2017	2	<div></div>	2018	10	<div></div>
2002	1	<div></div>																																				
2004	1	<div></div>																																				
2006	1	<div></div>																																				
2007	1	<div></div>																																				
2008	2	<div></div>																																				
2010	3	<div></div>																																				
2012	2	<div></div>																																				
2013	3	<div></div>																																				
2015	1	<div></div>																																				
2016	2	<div></div>																																				
2017	2	<div></div>																																				
2018	10	<div></div>																																				
Q3	<div><div>Domain where DUO is applied:</div><div>Project management</div><div>Requirements</div><div>Design</div><div>Security</div><div>Software quality</div><div>Software configuration</div><div>Text mining: StackOverflow</div><div>Text mining: Defect Reports</div><div>Text mining: Software Artifact Search, Linking and Labeling</div></div>	<div><div>Specific problem:</div><div>Software effort estimation [19, 21, 48, 49, 60, 75, 76, 84, 99, 100], managing human resources [20].</div><div>Requirements optimisation [29].</div><div>Software architecture optimisation [26], extraction of products from very large product lines [19].</div><div>Intrusion detection [5, 96]</div><div>Software defect prediction [4, 23, 36, 64, 98, 106, 121], test case generation [1].</div><div>Software configuration optimisation [20, 81, 82].</div><div>Linking posts [3, 34], topic modeling [3, 107].</div><div>Defect reports topic modeling [3].</div><div>Traceability link recovery [85], locate features in source code [85], software artifacts labeling [85], topic modeling of software engineering papers [3], Stack Overflow/GitHub topic modeling [107].</div></div>																																				
Q4	See Table 2.																																					
Q5	See Table 3.																																					
Q6	See Sections 4, 5 and 6.																																					

Table 1: Exploring the DUO literature.

2 Definitions

Before discussing combinations of data miners and optimizers, we should start by defining each term separately, and discussing their relationship. *Optimizers* reflect over a model to learn inputs that most improve model output. *Data miners*, on the other hand, reflect over data to return a summary of that data. Whereas data miners usually explore a fixed set of data, optimizers can generate more data by re-running the model N times using different inputs.

One informal way to characterize the difference between data miners and optimizers is “slice” versus “zoom”.

- Data miners “slice” data such that similar patterns are found within each division.
- Optimizers “zoom” into interesting regions of the data, then they use a model to fill in any missing details about those regions.

There are many different kinds of data miners, each of which might produce a different kind of model. For example, regression methods generate equations; nearest-neighbor-based methods might yield a small set of most interesting attributes and examples; and neural net methods return a weighted directed graph. Some data miners generate combinations of models. For example, the M5 “model tree learner” returns a set of equations and a tree that decides when to use each equation [92].

The user of a data miner might learn a model from data that predicts for (say) a single target class. The user of an optimizer, on the other hand, might deploy a multi-objective optimizer to find what solutions score best on multiple target variables. Whereas data miners usually have “hard-wired goals” (e.g. improve accuracy), the goals of an optimizer can be adjusted from problem to problem. In this way, an optimizer can be tuned to the needs of different business users. For example:

- For requirements engineers, we can find the *least* cost mitigations that enable the delivery of *most* requirements [29].
- For project managers, we can apply optimizers to software process models to find options that deliver *more* code in *less* time with *fewer bugs* [66].
- For developers, our optimizers can tune data miners looking for ways to find *more* bugs in *fewer* lines of code (thereby reducing the human inspection effort required once the learner has finished [37].
- Etc.

While counter-examples exist, for the most part data miners are used by *software analytics* workers and optimizers are used by researchers into *search-based SE*. As shown in Figure 2, the field we call “DUO” combines software engineering work from both fields.

Search-based software engineering [46, 47] characterizes SE tasks as optimizing for (potentially competing) goals; e.g. designing a product such that it delivers the *most* features at *least* cost [102]. Software analytics [9, 70, 71], on the other hand, is a workflow that distills large amounts of low-value data into small chunks of very high-value data. For example, software analytics might build a model predicting where defects might be found in source code [42].

For the most part, researchers in these two areas work separately (as witnessed by, say the annual Mining Software Repositories conference and the separate symposium on Search-based Software Engineering). What will be argued in this paper is that insights

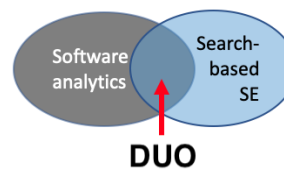


Fig. 2: About DUO.

and methods from these two fields can be usefully combined. We say this because papers that combined data miners and optimizers explore important SE tasks:

- Project management [19–21, 48, 49, 60, 75, 76, 84, 99, 100];
- Requirements engineering [29];
- Software design [19, 26];
- Software security [5, 96];
- Software quality [1, 4, 23, 36, 64, 98, 106, 121];
- Software configuration [20, 81, 82];
- Text mining of software-related textual artifacts [3, 85, 85, 107].

In our literature review, we have seen four different flavors of this combined DUO approach:

- *Mash-ups of data miners and optimizers*: In this approach, data miners and optimizers can be seen as separate executables. For example, Abdesslem et al. [1] generate test cases for autonomous cars via a cyclic approach where an optimizer reflects on the output of data miners that reflect on the output of an optimizer (and so on).
- *Data miners acting as optimizers*: In this approach, there is no separation between the data miner and optimizer. For example, Chen et al. [19] show that their recursive descent bi-clustering algorithm (which is a data mining technique) out-performs traditional evolutionary algorithms for the purposes of optimizing SE models.
- *Optimizers control the data miners*: In this approach, the data miner is a sub-routine called by the optimizer. For example, several recent papers improve predictive performance via optimizers that tune the control parameters of the data miner [4, 36, 106].
- *Data miners control the optimizers*: In this approach, the optimizer is a sub-routine called by the data miner. For example, Majumder et al. [65] use k-means clustering to divide up a complex text mining problem, then apply optimizers within each cluster. They report that this method speeds up their processing by up to three orders of magnitude.

The rest of this paper describes in detail what was learned from that literature review. For the purposes of exposition, those details are organized around the four claims listed in the introduction. In that discussion, we will also use other works that offer additional insight into DUO.

3 Claim1: Optimization and data mining are very similar

At first glance, the obvious connection between data miners and optimizers is that the former build models from data while the latter can be used to exercise those models (in order to find good choices within a model). However, a much more profound connection between data mining and optimization can be seen if we look “under the hood” to reflect on the core semantics of these supposedly separate kinds of algorithms. In order to explain this, we need to formally define optimization and data mining problems.

Without loss of generality, an optimization problem is of the following format [12]:

$$\begin{aligned} & \text{minimize } f_i(\mathbf{a}), i = 1, 2, \dots, n \\ & \text{subject to } g'_j(\mathbf{a}) \leq 0, j = 1, 2, \dots, n' \\ & \quad g''_k(\mathbf{a}) = 0, k = 1, 2, \dots, n'' \end{aligned} \tag{1}$$

where $\mathbf{a} = (a_1, \dots, a_m) \in \mathbf{A}$ is the optimization variable of the problem,¹ $f_i(\mathbf{a}) : \mathbf{A} \rightarrow \mathbb{R}$. are the objective functions (goals) to be minimized, $g'_j(\mathbf{a})$ are inequality constraints, and

¹ This definition has been generalized with respect to [12], not to be restricted to continuous optimization problems, where $\forall a_i, a_i \in \mathbb{R}$

Genetic Algorithms (GAs) execute over multiple generations. Generation zero is usually initialized at random. After that, in each generation, candidate items are subject to *select* (prune away the less interesting solutions), *crossover* (build new items by combining parts of selected items), and *mutate* (randomly perturb part of the the new solutions). Modern GAs take different approaches to the *select* operator e.g. dominance rank, dominance count, and dominance depth. Notable exceptions are MOEA/D that use a decomposition operator to divide all the solutions into many small neighborhoods where if anyone finds a better solution, all its neighbors move there as well [6, 21, 26, 48, 75, 84, 85, 98, 100].

Different evolution (DE) execute over multiple generations. Generation zero is usually initialized at random. After that, in each generation, candidate items are subject to *select* (prune away the less interesting solutions), *mutate* (build new items by combining with 3 other random candidates from the same generation) [3, 4, 34, 36, 65, 105].

MOEAs contains different types of multi-objective evolutionary algorithm such as MOEA/D, NSGA-II, and more. They differ based on either the diversity mechanism (such as, crowding distance or hypervolume contribution), or their sorting algorithm, or their use of target vectors, or etc. [1, 16, 20, 24, 50, 76, 100].

Particle Swarm Optimization (PSO) works by having a swarm of candidates where these candidates move around in the search space using simple formulae. The movements of the particles are guided by their own best known position in the search-space as well as the entire swarm’s best known position. When improved positions are being discovered these will then come to guide the movements of the swarm [5, 23, 90, 96].

Genetic programs (GPs) are like GAs except that while GAs manipulate candidates that are lists of options, GPs manipulate items that are trees. GPs can therefore be better for problems with some recursive structure (e.g. learning the parse tree of a useful equation) or when human-readable models are sought [2, 7, 56].

SWAY first randomly generates large number of candidates, recursively divides the candidates and only selects one. SWAY quits after the initial generation while GA reasons over multiple generations. It makes no use of reproduction operators so there is no way for lessons learned to accumulate as it executes [20].

Tabu Search uses a local or neighborhood search procedure to iteratively move from one potential solution x to an improved solution x' in the neighborhood of x , until some stopping criterion has been satisfied [20, 41].

FLASH, a sequential model-based method such as Bayesian optimization, is a useful strategy to find extremes of an unknown objective. FLASH is efficient because of its ability to incorporate prior belief as already measured solutions (or configurations), to help direct further sampling. Here, the prior represents the already known areas of the search (or performance optimization) problem. The prior can be used to estimate the rest of the points (or unevaluated configurations). Once one (or many) points are evaluated based on the prior, the posterior can be defined. The posterior captures the updated belief in the objective function. This step is performed by using a machine learning model, also called surrogate model [82].

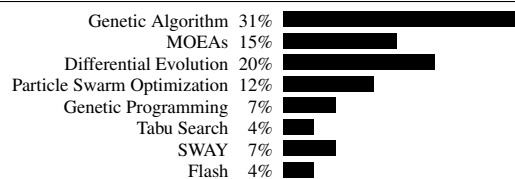


Table 2: Notes on different optimizers. Bar chart at bottom shows frequency of use in papers of Table 1’s literature review.

$g_k''(\mathbf{a})$ are equality constraints.² Sometimes, there are no constraints (so $n' = 0$ and $n'' = 0$). Also, we say a *multi-objective problem* has $n > 1$ objectives, as opposed to a *single-objective problem*, where $n = 1$.

One obvious question about this general definition is “where do the functions f, g', g'' come from?”. Traditionally, these have been built by hand but, as we shall see below, f, g', g'' can be learned via data mining. That is, optimizers can *explore* the functions *proposed* by a learner.

² The optimization variable is usually identified by the symbol \mathbf{x} , and the inequality and equality constraints are frequently identified by the symbols g and h in the optimization literature. However, we use the symbols \mathbf{a} , g and g'' here to avoid confusion with the terminology used in data mining, which is introduced later in this section.

An example of an optimization problem in the area of software engineering is to find a subset \mathbf{a} of requirements that maximizes value $f(\mathbf{a})$ if implemented³, given a constrained budget $g'_0(\mathbf{a}) \leq 0$, where $g'_0(\mathbf{a}) = \text{cost}(\mathbf{a}) - \text{budget}$ [97]. Many different algorithms exist to search for solutions to optimization problems. Table 2 shows the optimization algorithms that have been used by the software engineering community when applying DUO.

Data mining is a problem that involves finding an approximation $\hat{h}(\mathbf{x})$ of a function of the following format:

$$\mathbf{y} = h(\mathbf{x}) \quad (2)$$

where $\mathbf{x} = (x_1, \dots, x_p) \in \mathbf{X}$ are the input variables, $\mathbf{y} = (y_1, \dots, y_q) \in \mathbf{Y}$ are the output variables of the function $h(\mathbf{x}) : \mathbf{X} \rightarrow \mathbf{Y}$, \mathbf{X} is the input space and \mathbf{Y} is the output space. The input variables \mathbf{x} are frequently referred to as the independent variables or input features, whereas \mathbf{y} are referred to as the dependent variables or output features.

An example of a data mining problem in software engineering is software defect prediction [45]. Here, the input features could be a software component's size and complexity, and the output feature could be a label identifying the component as defective or non-defective. Many different machine learning algorithms can be used for data mining. Table 3 shows data mining algorithms used by the software engineering community when applying DUO.

The functions $h(\mathbf{x})$ and $\hat{h}(\mathbf{x})$ do *not* necessarily correspond to the optimization functions $f_i(\mathbf{a})$ depicted in Eq. 1. However, the true function $h(\mathbf{x})$ is unknown. Therefore, data mining frequently relies on machine learning algorithms to learn an approximation $\hat{h}(\mathbf{x})$ based on a set $D = \{(x_i, y_i)\}_{i=1}^{|D|}$ of known examples (data points) from $h(\mathbf{x})$. And, learning this approximation typically consists of searching for a function $\hat{h}(\mathbf{x})$ that minimizes the error (or other predictive performance metrics) on examples from D . Therefore, learning such approximation is an optimization problem of the following format:

$$\text{minimize } f_i(\mathbf{a}), i = 1, \dots, n \quad (3)$$

where $\mathbf{a} = (\hat{h}(\mathbf{x}), D)$, and f_i are the predictive performance metrics obtained by $\hat{h}(\mathbf{x})$ on D . The functions $f_i(\mathbf{a})$ depicted in Eq. 3 thus correspond to the functions $f_i(\mathbf{a})$ depicted in Eq. 1. An example of performance metric function would be the mean squared error, defined as follows:

$$f(\hat{h}(\mathbf{x}), D) = \frac{1}{|D|} \sum_{(x_i, y_i) \in D} (y_i - \hat{h}(x_i))^2 \quad (4)$$

As we can see from the above, solving a data mining problem means solving an optimization problem, i.e., optimization and data mining are very similar. Indeed, several popular machine learning algorithms *are* optimization algorithms. For example, gradient descent for training artificial neural networks, quadratic programming for training support vectors machines and least squares for training linear regression are optimization algorithms.

From the above, we can already see that optimization is of interest to data mining researchers, even though this connection between the two fields is not always made explicit in software engineering research. More explicit examples of how optimization is relevant to data mining in software engineering include the use of optimization algorithms to tune the parameters of the data mining algorithms, as mentioned in the beginning of this section and further explained in Sections 4 and 6. We consider such more explicitly posed connections between data mining and optimization as a form of DUO.

A typical distinction made between the optimization and data mining fields is data mining's need for generalization. Despite the fact that data mining uses machine learning

³ Any maximization problem can be re-written as a minimization problem.

Decision Tree learners such as CART and C4.5 seek attributes which, if we split on their ranges, most *reduces* the expected value of the diversity in splits. These algorithms then recurse over each split to find further useful divisions. For numeric classes, diversity may be measured in terms of variance. For discrete classes the Gini or entropy measures can assess diversity. Decision tree learners are widely applied in software engineering due to the simplicity and interpretability [1, 4, 15, 21, 23, 36, 64, 75, 81, 82, 84, 100, 106, 109].

Support Vector Machines (SVMs) are supervised learning trying to separate training items from two classes by a clear gap [104]. For linearly non-separable problems, SVMs either allow but penalize misclassification of training items (soft-margin) or utilize kernel tricks to map input data to a higher-dimensional feature space before learning a hyper-plane to separate the two classes. Many software engineering researchers have explored using SVM models to predict software artifacts [4, 23, 34, 65, 84, 98, 106, 109].

Instance-based algorithms: instead of fitting a model on the training data, instance-based algorithms stores all the training data as a database. When a new test item comes, the similarities between the test item and every training item are measured. The test item is then classified to the class where most of its similar training items belong to. Examples of instance-based algorithms include k-nearest neighbor [4, 100, 106, 109] and analogy algorithms [21, 48, 75, 100].

Ensemble algorithms use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone [91]. Usually, an ensemble algorithm builds multiple weak models that are independently trained and combines the output of each weak model in some way to make the overall prediction. Examples of ensemble algorithms include boosting [32, 106], bagging [13, 75, 106], and random forest [4, 15, 36, 63, 107].

Bayesian algorithms collect separate statistics for each class. Those statistics are used to estimate the prior distribution and the likelihood of each item in each class. When classifying a new test item, the estimated prior distribution and likelihood are applied to calculate its posterior distribution, which is then used to predict the class of the test item. Bayesian algorithms are widely applied in solving classification problems in software engineering [4, 5, 15, 23, 84, 96, 106].

Regression algorithms use predefined functions to model the mapping from input space to output space. Parameters of the predefined function are estimated by minimizing the error between the ground truth outputs and the function outputs. Regression algorithms can be applied to solve both regression (e.g. linear regression [100]) and classification problems (e.g. logistic regression [4, 36, 100, 106, 109]).

Artificial Neural Networks (ANNs) are models that are inspired by the structure and function of biological neural networks [39]. An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it. Such ANNs are usually applied in software engineering as baseline algorithms [21, 23, 75, 106].

Dimensionality reduction algorithms transforms the data in the high-dimensional space to a space of fewer dimensions [95]. The resulting low-dimensional space can be used as features to train other data mining models or directly used as a clustering result. Examples of dimensionality reduction algorithms include principle component analysis [52], linear discriminant analysis [94, 106], and latent Dirichlet allocation [3, 11, 85, 107].

Covering (rule-based) algorithms provide mechanisms that generate rules in a bottom-up, separate-and-conquer manner by concentrating on a specific class at a time and maximizing the probability of the desired classification. Examples of rule-based algorithms include PRISM [61] and RIPPER [22, 23, 106, 109].

Deep Learning methods are a modern update to ANNs that exploit abundant cheap computation. Deep learning uses a cascade of multiple layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input. In supervised or unsupervised manner, it learns multiple levels of representations that correspond to different levels of abstraction; the levels form a hierarchy of concepts [25]. Examples of instance-based algorithms include deep belief networks and convolutional neural network [34].

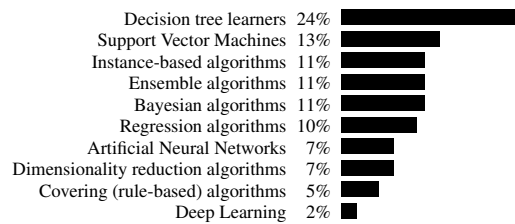


Table 3: Notes on the different data miners found by the literature review of Table 1. Bar chart at bottom shows frequency of use in papers of Table 1s literature review.

to search for approximations $\hat{h}(\mathbf{x})$ that minimize the error on a given dataset D , the true intention behind data mining is to search for approximations $\hat{h}(\mathbf{x})$ that minimize the error on unseen data D' from $h(\mathbf{x})$, i.e., being able to generalize. As D' is unavailable for learning, data mining has to rely on a given known data set D to find a good approximation $\hat{h}(\mathbf{x})$. Several strategies can be adopted by machine learning to avoid poor generalization despite the unavailability of D' . For instance, the performance metric functions $f_i(\hat{h}(\mathbf{x}), D)$ may use regularization terms [10], which encourage the parameters that compose $\hat{h}(\mathbf{x})$ to adopt small values, making $\hat{h}(\mathbf{x})$ less complex and thus generalize better. Another strategy is early stopping [10], where the learning process stops early, before finding an optimal solution that minimizes the error on the whole set D .

However, even the distinction above starts to become blurry when considering that generalization can also frequently be of interest to optimization researchers. For example, in software configuration optimization (Section 5), the true optimization functions $f_i(\mathbf{a})$ are often too expensive to compute, requiring machine learning algorithms to learn approximations of such functions. Optimization functions approximated by machine learning algorithms are referred to as surrogate models, and correspond to the approximation $\hat{h}(\mathbf{x})$ of Eq. 3. These approximation functions are then the one optimized, rather than the true underlying optimization function. Even though an optimization algorithm to solve this problem does not attempt to generalize, a data mining technique can do so on its behalf. We consider this as another form of DUO. Sections 4 and 5 explain how several other examples of software engineering problems are indeed both optimization and data mining problems at the same time, and how different forms of DUO can help solving these problems.

Overall, this section shows that optimization and data mining are very similar to each other, and that the typical distinction made between them can become very blurry when considering real world problems. Several software engineering problems require both optimization and generalization at the same time. Therefore, many of the ideas independently developed by the field of optimization are applicable to improve the field of data mining, and vice-versa. Sections 4 to 6 explain how useful DUO can and could be.

4 Claim2: Optimizers can greatly improve data miners

One of the most frequent ways to integrate data mining and optimization is via *hyperparameter optimization*. This is the art of tuning the parameters that control the choices within a data miner. While these can be set manually⁴ we found that several papers in our literature review used optimizers to automatically find the best parameters [4, 36, 64, 84, 98, 107, 121]. There are many reasons why this is so:

- These control parameters are many and varied. Even supposedly simple algorithms come with a daunting number of options. For example, the scikit-learn toolkit lists over a dozen configuration options for Logistic Regression⁵. This is an important point since recent research shows that the *more* settings we add to software, the *harder* it becomes for humans to use that software [118].
- Manually fine-tuning these parameters to obtain best results for a new data set, is not only tedious, but also can be biased by a human’s (mis-)understanding of the inner workings of the data miner.

⁴ Using a process called “engineering judgement”; i.e. guessing.

⁵ https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegressionCV.html, accessed 30 November 2018.

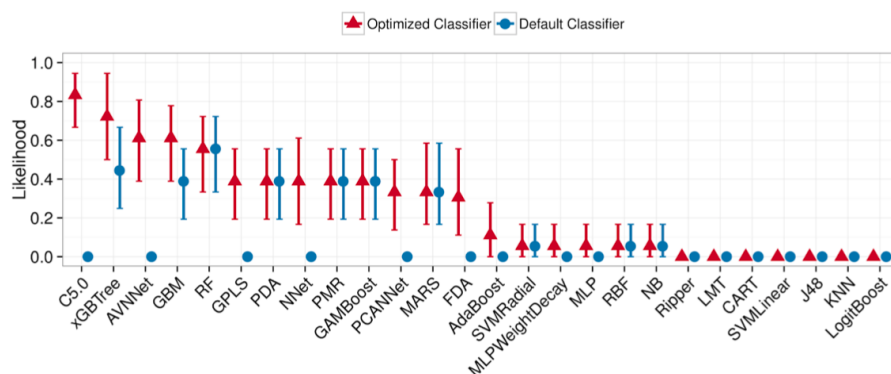


Fig. 3: Effects of hyperparameter optimization on control parameters of a learner from [106]. Blue dots and red triangle show the mean performance before and after tuning (respectively). X-axis shows different learners. Y-axis shows the frequency at which a learner was selected to be “top-ranked” (by a statistical analysis). Vertical lines show the variance of that selection process over repeated runs.

- The hurdle to implement or apply a “successful” heuristic for automated algorithm tuning is low since (a) the default settings are often not optimal for the situation at hand, and (b) a large number of optimization packages are readily available [27, 93]. Some data mining tools now come with built-in optimizers or tuners; e.g the SMAC implementation built into the latest versions of Weka [43, 51]; or the CARET package in “R” [59].
- Several results report spectacular improvements in the performance of data miners after tuning [4, 36, 64, 98, 106, 107, 121].

As evidence to the last point, we offer two examples. Tantithamthavorn et al. [106] applied the CARET grid search [59] to improve the predictive performance of classifiers. Grid search is an exhaustive search across on a pre-defined set of hyperparameter values. It is implemented as a set of for-loops, one for each hyperparameter. Inside the inner-most for-loop, some learner is applied to some data to assess the merits of a particular set of hyperparameters. Based on statistical methods, Tantithamthavorn et al. ranked all the learners in their study to find the top-ranked tuned learner. As shown in Figure 3, there is some variability in the likelihood of being top-ranked (since their analysis was repeated for multiple runs).

From Figure 3 we can see that:

- Hyperparameter optimization *never makes performance worse*. We say this since the red triangles (tuned results) are never lower than their blue dot counterpart (untuned results).
- Hyperparameter optimization is *associated with some spectacular performance improvements*. Consider the first six left-hand-side blue dots at $Y = 0$. These show all untuned learners that are never top-ranked. After tuning, however, the ranking of these learners is very different. Note once we tune two of these seemingly “bad” learners (C5.0 and AVNNet), they become part of the top three best learners.

For another example of the benefits of hyperparameter optimization, consider Figure 4. This work shown by Agrawal et al. [4] where differential evolution tuned a data pre-processor called SMOTE [17]. SMOTE rebalances training data by discarding items of the majority class and synthesizing artificial members of the minority class. As discussed in Table 2, differential evolution [105] is an evolutionary algorithm that uses a unique mutator that builds

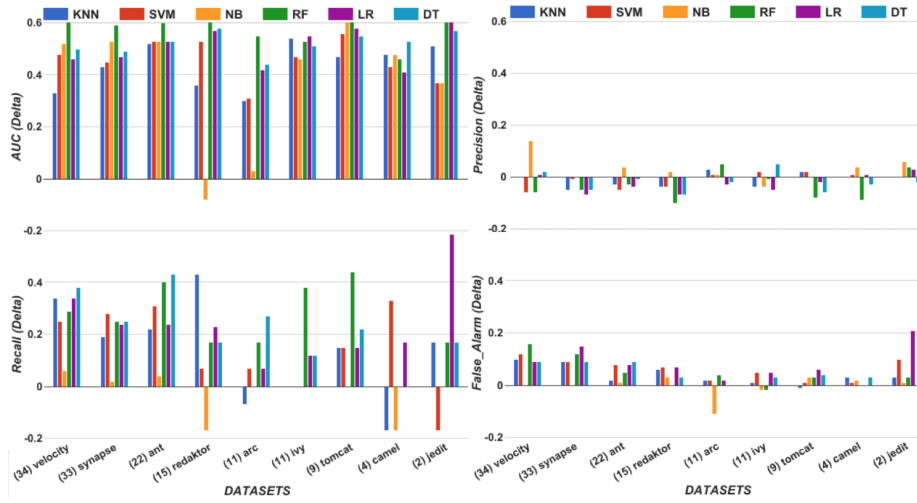


Fig. 4: Effects of hyperparameter optimization on control parameters of a data pre-processor from [4]. Different colored vertical bars refer to different learners: KNN=nearest neighbor; SVM= support vector machine; NB=naive bayes; RF=random forest; LR=logistic regression; DT=decision trees. X-axis shows different datasets. Y-axis shows the *after - before* values of four different performance scores: recall, precision, false alarm and AUC (area under the false positive vs true positive rate). For false alarms, *better* deltas are *smaller*. For all other measures, the *larger* the *better*.

new candidates by combining together three older candidates. From Figure 4 we can observe the same patterns seen in Figure 3:

- Hyperparameter optimization *rarely makes performance much worse*. There are some losses in precision and false alarms grow slightly. But overall, these changes are very small.
- Hyperparameter optimization is *associated with some spectacular performance improvements*. The improvements in recall can be large and the improvements in AUC are the largest we have ever seen of any method, ever, in the software analytics literature.

Another advantage of hyperparameter optimizers is that it can tune learners such that they succeed on multiple criteria. Standard learners have their goals “hard-wired” (e.g. minimize entropy). This can be a useful design choice since it allows algorithm designers to produce very fast systems that scale to very large data sets. That said, there are many situations where the business users have *multiple* competing goals; e.g. deliver *more* functionality, in *less* time, with *fewer* bugs. For further examples of multiple business goals in software analytics, see Table 23.2 of [67].

While the goals of data miners are often hard-wired, optimizers can accept multiple goals as part of the specification of a problem (see the g terms within Equation 1). This means optimizers can explore a broader range of goals than data miners. For example:

- Minku and Yao [72] used multi-objective evolutionary algorithms to generate neural networks for software effort estimation, with the objectives of minimizing different error metrics.

- Sarro et al. [100] used multi-objective genetic programming for software effort estimation, with the objectives of minimizing the error and maximizing the confidence on the estimates.

An interesting variant of the above is optimization (in the form of reinforcement learning) is model selection over time [73, 74]. Depending on the problem being tackled, the best predictive model to be used for a given problem may change over time, due to changes suffered by the underlying data generating process of this problem. For such problems, model choice has to be continuously performed over time, rather than performed only once, prior to model usage. To deal with that in the context of software effort estimation, Minku and Yao [73, 74] monitor the predictive performance of software effort estimation models created using software effort data from different companies over time. This predictive performance was monitored based on a time-decayed performance metric derived from the reinforcement learning literature, computed over their effort estimations provided for a given company of interest over time. The models whose predictive performance are recently the best are then selected to be emphasized when performing software effort estimations to the company of interest. When combined with transfer learning [73], this strategy enabled a reduction of 90% in the number of within-company effort data required to perform software effort estimation, while maintaining or sometimes even slightly improving predictive performance. This is a significant achievement, given that the cost of collecting within-company effort data is typically very high.

4.1 A Dozen Tips for Using Optimizers

The next section describes some of the problems associated with using optimizers. Before that, this section offers some rules of thumb for software engineers wishing to use *optimizers* in the manner recommend by this paper.

Just to say the obvious, we cannot *prove* the utility of the following heuristics. That said, when we work with our industrial partners or graduate students, we often say the following.

1. Start with a clear and detailed problem formulation. If you do not understand the problem well, then the proposed approach to solve the problem may not deliver what is desired.
2. Visualize first, optimize second. That is, try to visualize the trade offs between objectives before going into optimization by (e.g.) randomly generating some candidate solutions and plotting the generated performance scores (and for multi-goal reasoning, visualize the principal components of the objective space). We say this since (sometimes) glancing at such a visualization can lead to insights such as “this problem divides into two separate problems that we should explore separately”.
3. Optimizers of the kind explored here are found in many open source toolkits written in various languages (e.g. JAVA, C++, Python) such as jMETAL [27] or PAGMO/PyGMO (esa.github.io/pagmo2).
4. No optimizer works best for all applications [114]. That said, once it can be shown that one optimizer is better than another then the set of better optimizers is exponentially smaller [77]. Hence, when faced with a new problem, it is useful to try several optimizers and stop after two significant improvements have been achieved over some initial baseline result. In terms of algorithms, a useful set to try first are NSGA-2 [24] (since everyone tries this one), MOEA/D [120] (since it is fast), and differential evolution [105] (since it is so simple to use).

5. Further to the last point, any industrial application of this paper should try several optimizers. For that purpose, it is useful to apply faster optimizers (e.g. MOEA/D) before slower methods (e.g. NSGA-II). Note that if your preferred method is very slow, then it can be speed up via data mining (see point#1).

6. As said in §4, if optimizers run too slowly, use data miners to divide the data then and apply optimization to each segment [65].

7. If your users cannot understand what the optimizer is saying, use data mining to produce a summary of the results.

8. Watch out for changes in the problem over time – they may cause a previous optimal solution to become poor.

9. More specifically, insights can change with the computational budget. That is, conclusions that seem most useful after 1,000 evaluations might be superseded by the results from 5,000 evaluations on. Hence, if possible, before reporting a conclusion to users, try doubling the number of evaluations to see if your current conclusions still hold.

10. True multi-objective formulations are less biased than linear combinations of objectives. We say this since, sometimes, it is suggested to reduce a multi-optimization problem to a simpler single-objective problem by adding “magic weights” to each objective (e.g. “three times the speed of the car plus twice times the cost of the car”). Such “magic weights” introduce an unnecessary bias to the analysis. These magic weights can be avoided by using a true multi-objective algorithm (e.g. NSGA-II, MOEA/D, differential evolution (augmented with Algorithm 1)).

11. When optimizing for one or two goals, a simple predicate is enough to select which solution is better (specifically: x is not worse than y on both objectives; and x is better on at least one).

12. But when optimizing for more goals, Zitzler’s indicator method [122] might be needed [102]. This method reports what loses least: moving from x to y or y to x (and the solution x is preferred if moving to x results in the least loss; see Algorithm 1). Finally, for multiple goals, sometimes it is useful to focus first on a small number of most difficult goals (then use the results of that first study to “seed” a second study that explores the remaining goals) [101].

Algorithm 1: Zitzler’s indicator domination predicate [122]. Most useful when reasoning about more than two objectives [102]. In the weights array, -1,1 means “minimize, maximize” respectively. Objective scores are normalized 0..1 since, otherwise, the exponential calculation might explode.

```
def x_better_than_y(
    x,y,      # lists of candidate solutions
    weights,  # dictionary of objective weights,
    goals,    # list of objective indexes
    lo, hi):  # lists of low,high values of objectives
    xloss, yloss, n = 0, 0, len(w)
    for g in goals:
        a = normalize( x[g], lo[g], hi[g] )
        b = normalize( y[g], lo[g], hi[g] )
        w = weights[g]
        xloss += 10**( w * (a-b)/n )
        yloss += 10**( w * (b-a)/n )
    end
    return xloss < yloss

def normalize(z,lo,hi): return (z - lo) / (hi - lo + 0.00001)
```

4.2 Problems with Hyperparameter Optimization

In summary, optimization is associated with some spectacular improvements in data mining. Also, by applying optimizers to data miners, they can better address the domain-specific and goal-specific queries of different users.

One pragmatic drawback with hyperparameter optimization is its associated runtime. Each time a new hyperparameter setting is evaluated, a learner must be called on some training data, then tested on some separate “hold-out” data. This must be repeated, many times. In practice, this can take a long time to terminate:

- When replicating the Tantithamthavorn et al. [106] experiment, Fu et al. [38] implemented tuning using grid search and differential evolution. That study used 20 repeats for tuning random forests (as the target learner), and optimized four different measures of AUC, recall, precision, false alarm, that grid search required 109 days of CPU. Differential evolution and grid search required 10^4 and 10^7 seconds to terminate, respectively⁶.
- Xia et al. [116] reports experiments with hyperparameter optimization for software effort estimation. In their domain, depending on what data set was processed, it took 140 to 700 minutes (median to maximum) to compare seven ways to optimize two data miners. If that experiment is repeated 30 times for statistical validity, then the full experiment would take 70 to 340 hours (median to max).

While the above runtimes might seem practical to some researchers, we note that other hyperparameter optimization tasks take a very long time to terminate. Here are the two worst (i.e. slowest) examples that we know of, seen in the recent SE literature:

- E.g. decades of CPU time were needed by Treude et al. [107] to achieve a 12% improvement over the default settings;
- E.g. 15 years of CPU were needed in the hyperparameter optimization of software clone detectors by Wang et al. [111].

One way to address these slow runtimes is via (say) cloud-based CPU farms. Cross-validation experiments can be easily parallelized just by running each cross-val on a separate core. But the cumulative costs of that approach can be large. For example, recently while developing a half million US dollar research proposal, we guesstimated how much it would cost to run the same kind of hardware as seen in related work. For that three year project, two graduate students could easily use \$100,000 in CPU time – which is a large percentage of the grant (especially since, after university extracts its overhead charge, that \$500K grant would effectively be \$250K).

Since using optimizers for hyperparameter optimization can be very resource intensive, the next section discusses DUO to significantly reduce that cost.

5 Claim3: Data miners can greatly improve optimization

The previous section mentions the benefits of optimizers for data mining, but warned that such optimization can be slow. One way to speed up optimization is to divide the total problem into many small sub-problems. As discussed in this section, this can be done using data mining. That is, data mining can be used to optimize optimizers.

Another benefit of data mining is that, as discussed below, it can generalize and summarize the results of optimization. That is, data mining can make optimization results more comprehensible.

⁶ Total time to process 20 repeated runs across multiple subsets of the data, for multiple data sets.

5.1 Faster Optimization

It can be a very simple matter to implement data miners improving optimizers. Consider, for example, Majumder et al. [65] who were looking for the connections between posts to StackOverflow (which is a popular on-line question and answer forum for programmers):

- An existing deep learning approach [117] to that problem was so slow that it was hard to reproduce prior results.
- Majumder et al. found that they could get equivalent results 500 to 900 times faster (using one core or eight cores working in parallel) just by applying k-means clustering to the data, then running their hyperparameter optimizer (differential evolution) within each cluster.

Another example of data mining significantly improving optimization, consider the *sampling* methods of Chen et al. [19, 20]. This team explored optimizers for a variety of (a) software process models as well as the task of (b) extracting products from a product line description. These are multi-objective problems that struggle to find solutions that (e.g.) minimize development cost while maximizing the delivered functionality (and several other goals as well). The product extracting task was particularly difficult. Product lines were expressed as trees of options with “cross-tree constraints” between sub-trees. These constraints mean that decisions in one sub-tree have to be carefully considered, given their potential effects on decisions in other sub-trees. Formally, this makes the problem NP-hard and in practice, this product extraction process was known to defeat state-of-the-art theorem provers [89], particularly for large product line models (e.g. the “LVAT” product line model of a LINUX kernel contained 6888 variables within a network of 343,944 constraints [19]).

Chen et al. tackled this optimization problem using data mining to look at just a small subset of the most informative examples. **Chen et al. call this approach a “sampling” method.** Specifically, they used a recursive bi-clustering algorithm over a large initial population to isolate the superior candidates. As shown in the following list, this approach is somewhat different to the more standard genetic algorithms approach:

- Genetic algorithms (in SE) often start with an population of $N = 10^2$ individuals.
- On the other hand, samplers start with a much larger population of $N = 10^4$ individuals.
- Genetic algorithms run for multiple *generations* where useful variations of individuals in generation i are used to seed generation $i + 1$.
- On the other hand, samplers run for a single generation, then terminate.
- Genetic algorithms evaluate all N individuals in all generations.
- On the other hand, the samplers of Chen et al. evaluate pairs of distant points. If one point proves to be inferior then it is pruned along with all individuals in that half of the data. Samplers then recursively prune the surviving half. In this way, samplers only evaluate $O(2\log_2 N)$ of the population,

Regardless of the above differences, the goal of genetic algorithms and samplers is the same: find options that best optimize some competing set of goals. In comparisons with NSGA-II (a widely used genetic algorithm [24]), Chen et al.’s sampler usually optimized the same, or better, as the genetic approach. Further, since samplers only evaluate $O(2\log_2 N)$ individuals, sampling’s median cost was just 3% of runtimes and 1% of the number of model evaluations (compared to only running the genetic optimizer) [19].

For another example of data miners speeding up optimizers, see the work of Nair et al. [81]. That work characterized the software configuration optimization problem as ranking

a (very large) space of configuration options, without having to run tests on all those options. For example, such configuration optimizers might find a parameter setting to SQLite’s configuration files that maximized query throughput. Testing each configuration requires re-compiling the whole system, then re-running the entire test suite. Hence, testing the three million valid configurations for SQLite is an impractically long process.

The key to quickly exploring such a large space of options, is *surrogate modeling*; i.e. learning an approximation to the response variable being studied. The two most important properties of such surrogates are that they are much faster to evaluate than the actual model, and that the evaluations are precise. Once this approximation is available then configurations can be ranked by generating guesstimates from the surrogates. Nair et al. built their surrogates using a data miner; specifically, a regression tree learner called CART [14]. An initial tree is built using a few randomly selected configurations. Then, while the error in the tree’s predictions decreases, a few more examples are selected (at random) and evaluated. Nair et al. report that this scheme can build an adequate surrogate for SQLite after 30-40 evaluated examples.

For this paper, the key point of the Nair et al. work is that this data mining approach scales to much larger problems than what can be handled via standard optimization technology. For example, the prior state-of-the-art result in this area was work by Zuluga et al. [123] who used a Gaussian process model (GPM) to build their surrogate. GPMs have the advantage that they can be queried to find the regions of maximum variance (which can be an insightful region within which to make the next query). However, GPMs have the disadvantage that they do not scale to large models. Nair et al. found that the data mining approach scaled to models orders of magnitude larger than the more standard optimization approach of Zuluga et al.

5.2 Better Comprehension of User Goals

Aside from speeding up optimization, there are other benefits of adding data miners to optimizers. If we combine data miners and optimizers then we can (a) better understand user goals to (b) produce results that are more relevant to our clients.

To understand this point, we first note that modern data miners run so quickly since they are highly optimized to achieve a single goal (e.g. minimize class entropy or variance). But there are many situations where the business users have *multiple* competing goals; e.g. deliver *more* functionality, in *less* time, with *fewer* bugs. A standard data miner (e.g. CART) can be kludged to handle multiple goals reasoning, as follows: compute the class attribute via some *aggregation function* that uses some “magic weights” β , e.g., $class_value = \beta_1 \times goal1 + \beta_2 \times goal2 + \dots$. But using an aggregate function for the class variable is a kludge, for three reasons. Firstly, when users change their preferences about β_i , then the whole inference must be repeated.

Secondly, the β_i goals may be inconsistent and conflicting. A repeated result in decision theory is that user preferences may be nontransitive [30] (e.g. users rank $\beta_1 < \beta_2$ and $\beta_2 < \beta_3$ but also $\beta_3 < \beta_1$). Such nontransitivity means that a debate about how to set β to a range of goals may never terminate.

Thirdly, rather than to restrict an inference to the whims of one user, it can be insightful to let an algorithm generate solutions across the space of possible preferences. This approach was used by Veerappa et al. [110] when exploring the requirements of the London Ambulance system. They found that when they optimized those requirements, the result was a *frontier* of hundreds of solutions like that shown in Figure 5. Each member of that frontier is trying to

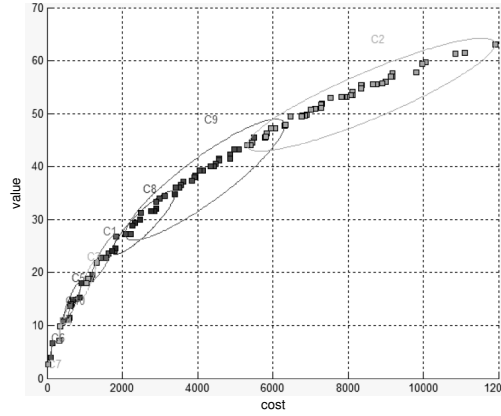


Fig. 5: Clusters of the solution frontier. Frontier generated by reasoning over the goals (in this case, *minimize* cost and *maximize* value). Clusters generated by reasoning over the decisions that lead to those goals. From [110].

push out on all objectives (and perhaps failing on some). Mathematicians and economists call this frontier the *Pareto frontier* [86]. Others call it the *trade-off space* since it allows users to survey the range of compromises (trade-offs) that must be made when struggling to achieve multiple, possibly competing, goals.

Figure 5 is an illustrative example of how data mining can help optimizers. In that figure, the results are grouped by a data miner (a clusterer). The decisions used to reach the centroid of each cluster are a specific example, each with demonstrably different effects. When showing these results to users, Veerappa et al. can say (e.g.) “given all that you’ve told us, there are less than a dozen kinds of solutions to your problem”. That is:

- Step#1: Optimize: here are the possible decisions;
- Step#2: Data mine: here is a summary of those decisions;
- Step#3: Business users need only debate the options in the summary.

Note that Figure 5 lets us comment on the merits of global vs local reasoning. *Global* reasoning might return the average properties of all the black points in that figure. But if we apply *local* reasoning, we can find specialized models within each cluster of Figure 5. The merits of local vs global reasoning are domain-specific but in the specific case of Figure 5, there seems to be some key differences between the upper right and lower left clusters.

Another approach to understanding trade-off space is contrast-rule generation as done by (e.g.) Menzies et al.’s STAR algorithm [66]. STAR divided results into a 10% “best” set and a 90% “rest” set. A Bayesian contrast set procedure ranks all N decisions in descending order (best to worst). STAR then re-runs the optimizer $i \in N$ times, each time pre-asserting the first i -th ranges. For example, suppose 10,000 evaluations lead to $B = 1000$ ‘best’ solutions and $R = 9000$ rest.

- Let a be “analyst capability” and let “ a =high” appear 50 times in best and 90 times in rest. So $b_{a.high} = 50/B = 50/1000 = 0.05$; $r_{a.high} = 90/R = 90/9000 = 0.01$;
- Let u be “use of software tools” and let “ u =high” appear 100 times in best and 180 times in rest. So $b_{u.high} = 100/1000 = 0.1$; and $r_{u.high} = 180/9000 = 0.02$.

- STAR sorts ranges via $s = b^n / (b + r)$, where n is a constant used to reward ranges with high support in “best”. E.g. if $n = 2$ then $s_{a.high} = 0.042$ and $s_{u.high} = 0.083$. That is, STAR thinks that the high use of software tools is more important than high analyst capability.

The output of STAR result is a graph showing the effects of taking the first best decision, the first two best decisions, and so on. In this way, STAR would report to users a succinct rule set advising them what they can do if they are willing to change just one thing, just two things, etc [66].

Before ending this section, we stress the following point: *it can be very simple to add a data miner to an optimizer*. For example:

- STAR’s contrast set procedure described above, is very simple to code (around 30 lines of code in Python).
- Recall from the above that Majumder et al. speed up their optimizer by 500 to 900 times, just by prepending a k-means clusterer to an existing optimization process.

5.3 A Dozen Tips for Using Data Mining

This section offers some rules of thumb for software engineers wishing to use *data miners* in the manner recommended by this paper.

As said above, just to say the obvious, we cannot *prove* the utility of the following heuristics. That said, when we work with our industrial partners or graduate students, we often say the following.

1. Data miners of the kind explored here are found in many open source toolkits written in various languages (e.g. JAVA, Python) such as WEKA [43] or Scikit-Learn [87].

2. If your data mining problem has many goals, consider replacing your data mining algorithms with an optimizer.

3. Avoid the use of the off-the-shelf parameters. Instead, use optimizers to select better settings for the local problem. For more on this point, see **Claim4**, below.

4. Check for conclusion stability. Once you make a conclusion, repeat the entire process ten times using a 90% random sample of the data each time. Do not tell business users about effects that are unstable across different samples. This point is particularly relevant for systems that combine data miners with optimizers that make use of any stochastic component.

5. No data miner works best for all applications [113]. Hence, we offer the same advice as with point#4,5 in §4.1. That is, when faced with a new problem, it is useful to try several data miners and stop after two significant improvements have been achieved over some initial baseline result. In terms of what data miners to try first, there is a large candidate list. For software analytics, we refer the reader to table IX of Ghotra et al. [40] that ranks dozens of different data mining algorithms into four “ranks”. To sample a wide range of algorithms, we suggest applying one algorithm for each rank.

6. Watch out for temporal effect of data – it may cause past models to become inadequate.

7. Strive to avoid overfitting. Try to test on data not used in training. If the data has time stamps, train on earlier data and test on later data. If no timestamps, then ten times randomly reorganized the data and divide it into ten bins. Next, make each bin the test set and all the other bins the train set.

8. Ignore spurious distinctions in the data. For example, the Fayyad-Irani discretizer [28] can simplify numeric columns by dividing up regions that best divide up the target class.

9. Ignore spurious columns. If a column ended up being poorly discretized, that is a symptom that that column is uninformative. By pruning the columns with low discretization scores, spurious data can be ignored [44].

10. Ignore spurious rows. Similarly, instance and range pruning can be useful. After discretization and feature selection, numeric ranges can be scored by how well they achieve specific target classes. If we delete rows that have few interesting ranges, we can reduce and simplify any process that visualizes or searches the data [88].

11. If there is very little data, consider asking the model inside the data miner to generate more examples. If that is not practical (e.g. the model is too slow to execute) then try transfer learning [58], active learning [119], or semi-supervised learning.

12. For more advice about using data miners, see “Bad Smells for Software Analytics” [68].

6 Claim4: Data mining without optimization is not recommended

There are many reports in the empirical SE literature where the results of a data miner are used to defend claims such as:

- “*In this domain, the most important concepts are X.*” For example, Barua et al. [8] used text mining called to conclude what topics are most discussed at Stack Overflow.
- “*In this domain, learnerX is better than learnerY for building models.*” For example, Lessmann et al. [62] reported that the CART decision tree performs much worse than random forests for defect prediction.

All the above results were generated using the default values for CART, random forests and a particular text mining algorithm. We note that these conclusions are now questionable given that tuned learners produce very different results to untuned learners. For example:

- Claims like “*In this domain, the most important concepts are X*” can be changed by applying an optimizer to a data miner. For example, Tables 3 and 8 of [3] show what was found before/after tuned text miners were applied to Stack Overflow data. In many cases, the pre-tuned topics just disappeared after tuning. Also, in defense of the tuned results, we note that, in “order effects experiments”, the pre-tuned topics were far more “unstable” than the tuned topics⁷.
- Claims like “*In this domain, learnerX is better than learnerY for building models*” can be changed by tuning. One example Fu et al. reversed some of the Lessmann et al. conclusions by showing that tuned CART performs much better than random forest [36]. For another example, recall Figure 3 where, before/after tuning the C5.0 algorithm was the worst/best learner (respectively).

For a small example of this effect (that optimizing a data miner leads to different results), see Figure 6. This figure ranks the importance of different static code features in a defect prediction decision tree. Here “importance” is computed as the (normalized) total reduction of the Gini index for a feature⁸. In this case, tuning significantly improved the performance of the learner (by 16%, measured in terms of the “utopia” index⁹). After tuning:

⁷ In “order effects experiments”, the training data is re-arranged at random before running the learner again. In such experiments, a result is “unstable” if the learned model changes just by re-ordering the training data.

⁸ The Gini index measures class diversity after a set of examples is divided by some criteria – in this case, the values of an attribute.

⁹ The distance to of a predictor’s performance to the “utopia” point of recall=1, false alarms=0.

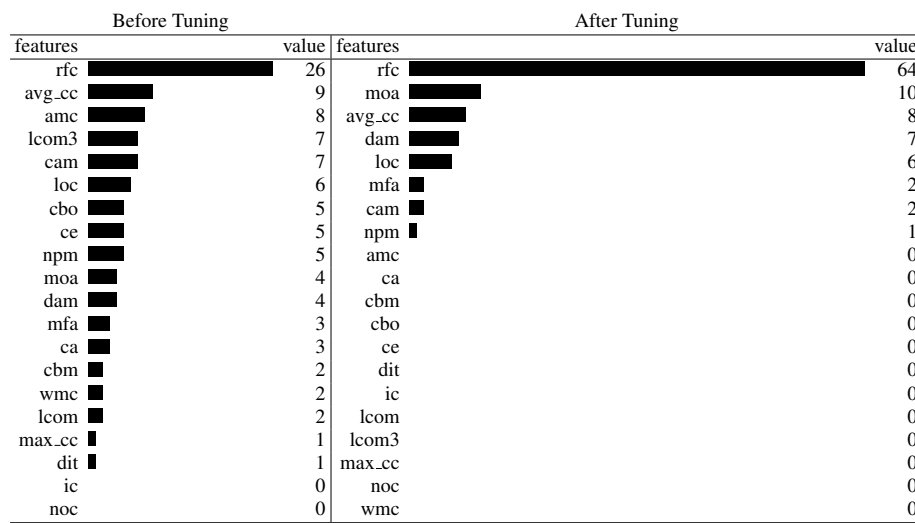


Fig. 6: Features importance shown for decision tree before and after tuning on jEdit defect prediction where it is optimized for *minimizing* distance to “utopia” (where “utopia” is the point recall=1, false alarms=0).

- Features that seemed irrelevant (before tuning) are found to be very important (after tuning); e.g. moa, dam.
- Many features received much lower ranking (e.g. amc, lcom3, cbo).

Overall, the number of “interesting” features that we might choose to report as “conclusions” from this study is greatly reduced.

Apart from the examples in this section, there are many other examples where (a) hyper-parameter optimization selects models with better performance and (b) those selected models report very different things to alternate models. For example, in §5.1, we saw the following:

- In the Nair et al. [81] case study, the CART decision tree was used to summarize the data seen so far. Such decision trees usually prune away most variables so that any human reading a CART model would see different things than if they read a logistic regression model (where every variable may appear in the logistic equation).
- In the Tantithamthavorn et al. [106] case study of Figure 3, depending on what learner was selected by what optimizer, that analysis would have reported models as a single decision tree, a forest of trees, a set of rules, the probability distributions within a Bayes classifiers, or as an opaque neural net model.
- In the Majumder et al. [65] case study, tuning made us select SVM over a deep learner. Note that that change (from one learner to another) also changes what we would report from that model. SVM models can be reported as the difference between their support vectors (which are few in number). A report of the deep learning model may be much more complicated (since such a summary would require any number of complex transforms, none of which are guaranteed to endorse the same model as the SVM),
- In the Chen et al. [19, 20] case study, tuning used a recursive clustering algorithm (that prune away most of the details in the original model). Such pruning is a very different

approach to that seen in other kinds of optimizers. Hence, any model learned from the Chen et al. methods would be very different to models learned from other optimizers.

Note that the above effect, where the nature of the model generated by a learner is effected by the optimizer attached to that learner, is quite general to all machine learning algorithms. Fürnkranz and Flach characterize learners as “surfing” a landscape of modeling options looking for a “sweet spot” that balances different criteria (e.g. false alarms vs recall). Figure 2 offers an insightful example of this process. In that figure, a learner adds more and more conditions to a model, thereby driving it to different places on the landscape. Depending on the hyperparameters of a learner, that learner will “surf” that space in different ways, terminate at different locations, and return different models.

The important point here is that, in domains where data miners can be optimized very quickly, it would take just a few minutes to hours to refine and improve untuned results. Further, when humans lean in to read what has been learned, these different tunings lead to different kinds of models and hence different kinds of conclusions. Hence, we do not recommend reporting on the models learned via data mining without optimization.

It is worth noting that the use of optimizers to tune the learners’ hyperparameters does not mean that we should ignore information on the range of results achieved using different hyperparameter choices. Analyses of sensitivity to hyperparameters are still important, specially considering that the best hyperparameters “right now” may not be the best hyperparameters for “later”. Moreover, optimizers have themselves’ hyperparameters, which could potentially affect their ability to suggest good hyperparameters to learners. Therefore, it is important to understand how sensitive the learners are to hyperparameter choice.

7 Research Direction

One way to assess any proposed framework such as DUO is as follows: is it sufficient to guide the on-going work of a large community of researchers? As argued in this section, DUO scores very well on this criterion.

Firstly, given that *data mining without optimization should be deprecated*, it is time to revisit and recheck every prior software analytics result based on untuned data miners. This will be a very large task that could consume the time of hundreds of graduate students for at least a decade.

Secondly, given that *data miners can greatly improve optimizers*, there are many research directions:

- *Better explanation and comprehension tools for AI systems*: Use the data miners to summarize complex optimizer output in order to convert opaque inference into something comprehensible. Some methods for that were seen above (Figure 5 and the STAR algorithm of §5.2) but they are just two early prototypes. Adding comprehension tools to AI systems that use optimization is a fertile ground for much future research. For a discussion on criteria to assess comprehensibility in software analytics, see [18, 78].
- *Auditable AI*: There is much recent interest in allowing humans to query AI systems for their biases and, where appropriate, to adjust them. Sampling tools like those of Chen and Nair et al. [19, 20, 82] offer functionality that might be particularly suited for that task. Recall that these tools optimized their models using just a few dozen examples – which is a number small enough to enable human inspection. Perhaps we could build human-in-the-loop systems where humans watch the samplers’ explored options – in the field of optimization, the concepts of interactive optimization, dynamic optimization, and

the multi-objective encoding of user preferences are well-established. At the very least, this might allow humans to understand the reasoning. And at the very most, these kinds of tools might allow humans to “reach in” and better guide the reasoning.

- *Faster Deep Learning*: One open and pressing issue in software analytics is the tuning of deep learning networks. Right now, deep learning training is so slow that it is common practice to download pre-tuned networks. This means that deep learning for software analytics may be prone to all the problems we discussed in claims 2 and 4. We gave some ideas here on how data mining can divide up and simplify the deep learning training problem (but, at this time, no definitive results).
- *Avoiding Hyper-hyperparameter Optimization*: While hyperparameter optimization is useful, it should be noted that the default parameters of hyperoptimizers may themselves need optimizing by other optimizers. This is a problem since if hyperparameter optimization is slow, then hyper-hyperparameter optimization would be even slower. So how can we avoid hyper-hyperparameter optimization?
 - One possible approach is *transfer learning*. Typically when something is tuned, we do so because it will be run multiple times. So instead of search *tabula rasa* for good tunings, perhaps it is possible to partially combine parts of old solutions to speed up the search for good hyperparameter values [80].
 - An analogous approach is to select from a portfolio of algorithms. This typically involves the training of machine learning models on performance data of algorithms in combination with instances given as feature data. In software engineering, this has been recently used as for the Software Project Scheduling Problem [103, 115]. The field of per-instance configuration has received much attention recently, and we refer the interested reader to a recent updated survey article [55].
 - Another approach, is to find shortcuts around the optimization process. For recent work on that, which we would characterize as highly speculative, see [35].

Thirdly, given that *optimizers can greatly improve data miners*, it is time to apply hyperparameter optimization to more data mining tasks within software analytics. As shown by this paper, such an application can lead to dramatically better predictors. Moreover, given that multi-objective perspective that can be given by optimizers, we can use optimizers to enable data mining to explore multiple objectives. For example:

- For software analytics, we could try to learn data miners that find highest priority bugs after the *fewest* tests, found in the *smallest* methods in code that is *most familiar* to the current human inspector. Such a data miner would return the bugs that are most important and easiest to fix (thus reducing issue resolution time for important issues).
- For project management, when crowd sourcing large software projects, we could allocate tasks to programmers in order to *minimize* development time while *maximizing* work assignments to programmers that have most familiarity with that area of the code.
- For refereeing new research results in SE, the tools described here could assign reviewers to new results in order to *minimize* the number of reviews per reviewer while *maximizing* the number of reviewers who work in the domain of that paper.

We could also extend SE to make it about using DUO for providing software engineering support for artificial intelligence systems. For example, using the optimization discussed above, Weise et al. [112] have run over 157 million experiments on over 200 instances of two classical AI combinatorial problems. They found that the naive configuration is a good baseline approach, but with effort it was possible to outperform it. Friedrich et al. [33] who studied a particular family of heuristic hill-climbers problems. Their empirical optimization results sparked extensive theoretical investigations (i.e., proper computational complexity

analyses) that showed that the new configuration is provably faster than what has been state-of-the-art. These can be seen as examples of software engineering to find better configurations. Other work in this area includes Neshat et al. [83] who studied wave energy equations. Their optimization results were translated into well-performing algorithms for large problems. This can be seen as software engineering to improve algorithms' performance.

Lastly, given *that data mining and optimization is essentially the same thing*, it is time to explore engineering principles for creating unified data mining/optimizer toolkits. We already have one exemplar of such a next-generation toolkit: see www.automl.org for the work on AutoML and its connections to Weka and scikit-learn. That said, this research area is wide open for experimentation. For two reasons, we would encourage researchers to “roll their own” DUO implementations before automatically turning to tools like AutoML:

- Some initial results suggest it may not be enough to just use AutoML [108] (in summary, given N hyperparameter optimizers, AutoML was “best” for a minority of goals and datasets).
- In terms of training ourselves on how easy it is to combine optimizers and data miners, there is no substitute for “rolling your own” implementation (at least once, then perhaps moving on to tools like AutoML).

8 Conclusion

For software analytics it is *possible, useful and recommended* to combine data mining and optimization using DUO. Such combination can lead to better (e.g., faster, more accurate or reliable, more interpretable, and multi-goals) analyses in empirical software engineering studies, in particular those studies that require automated tools for analysing (large quantities of) data. We support our arguments based on a literature review of applications of DUO.

We say it is *possible* to combine data mining and optimization since data mining and optimization perform essentially the same task (Section 3). Hence, it is hardly surprising that it can be a relatively simple matter to build a unified data mining/optimizer tool. For instance:

- Nair et al.'s approach [82] was just a for-loop around CART.
- STAR [66] was also a very simple learner (see §5.2) wrapped around a simulated annealer, then re-ran the simulated annealer after setting the first i -best ranges.
- Sampling is just a simple bi-recursive clustering algorithm (but see §3.4 of [19] for a discussion on some of the nuances of that process).
- There are many mature open source data mining and optimization frameworks¹⁰. Also, some of the data mining and optimization algorithms are inherently very simple to implement (e.g. naive bayes [31], differential evolution [105]). Hence it is easy to implement optimizer+data miner combinations.

As to *usefulness*, this paper has listed several benefits of DUO:

- Data miners can speed up optimizers by dividing large problems into several simpler and smaller ones.
- Also, when optimizers return many example solutions, data miners can generalize and summarize those into a very small set of exemplars (see for example Figure 5) or rules (see for example the STAR algorithm of §5.2).

¹⁰ E.g. in Python: scikit-learn and DEAP [87, 93]. E.g. in Java: Weka and (jMetal or SMAC) [27, 43, 51].

- Optimization technology lets data miners explore a much broader set of competing goals than just (e.g.) precision, recall, and false alarms. Using those goals, it is possible to better explore an interesting range of SE problems such as project management, requirements engineering, design, security problems, software quality, software configuration, mining textual SE artifacts, just to name a few.

Finally, as to *recommended*, we warn that it can be misleading to report conclusions from an untuned learner since those conclusions can be changed by tuning. Since papers that use untuned learners can be so easily refuted, this community should be wary of publishing analytics papers that lack an optimization component.

Having made this case, we acknowledge that DUO would require a paradigm shift for the software analytics community. Graduate subjects would have to be changed (to focus on different kinds of algorithms and case studies); toolkits would need to be reorganized; and journals and conferences will have to adjust their paper selection criteria. Looking into the future, we anticipate several years where DUO is explored by a minority of software analytics researchers. That said, by 2025, we predict that DUO will be standard practice in software analytics.

Acknowledgements

Earlier work ultimately leading to the present one was inspired by the NII Shonan Meeting on Data-Driven Search-based Software Engineering (goo.gl/f8D3EC), December 11-14, 2017. We thank the organizers of that workshop (Markus Wagner, Leandro L. Minku, Ahmed E. Hassan, and John Clark) for their academic leadership and inspiration.

Dr. Menzies’ work was partially supported by NSF grant No. 1703487.

Dr Minku’s work was partially supported by EPSRC grant No. EP/R006660/1.

Dr Wagner’s work was partially supported by the ARC grant DE160100850.

References

1. Abdesslem, R.B., Nejati, S., Briand, L.C., Stifter, T.: Testing vision-based control systems using learnable evolutionary algorithms. In: Proceedings of the 40th International Conference on Software Engineering, ICSE ’18, pp. 1016–1026. ACM, New York, NY, USA (2018). DOI 10.1145/3180155.3180160
2. Afzal, W., Torkar, R.: On the application of genetic programming for software engineering predictive modeling: A systematic review. *Expert Systems with Applications* **38**(9), 11,984–11,997 (2011)
3. Agrawal, A., Fu, W., Menzies, T.: What is wrong with topic modeling? and how to fix it using search-based software engineering. *Information and Software Technology* **98**, 74–88 (2018)
4. Agrawal, A., Menzies, T.: Is better data better than better data miners?: on the benefits of tuning smote for defect prediction. In: Proceedings of the 40th International Conference on Software Engineering, pp. 1050–1061. ACM (2018)
5. Ali, M.H., Al Mohammed, B.A.D., Ismail, A., Zolkipli, M.F.: A new intrusion detection system based on fast learning network and particle swarm optimization. *IEEE Access* **6**, 20,255–20,261 (2018)
6. Anderson-Cook, C.M.: *Practical genetic algorithms* (2005)
7. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: *Genetic programming: an introduction*, vol. 1. Morgan Kaufmann San Francisco (1998)
8. Barua, A., Thomas, S.W., Hassan, A.E.: What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering* **19**, 619–654 (2012)
9. Bird, C., Menzies, T., Zimmermann, T. (eds.): *The Art and Science of Analyzing Software Data*. Morgan Kaufmann, Boston (2015). DOI <https://doi.org/10.1016/B978-0-12-411519-4.09996-1>
10. Bishop, C.: *Pattern Recognition and Machine Learning*. Springer (2006)

11. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. *Journal of machine Learning research* **3**(Jan), 993–1022 (2003)
12. Boyd, S.P., Vandenberghe, L.: Section 4.1 – optimization problems. In: *Convex Optimization*. Cambridge University Press (2004)
13. Breiman, L.: Bagging predictors. *Machine learning* **24**(2), 123–140 (1996)
14. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA (1984)
15. Catal, C., Diri, B.: Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. *Information Sciences* **179**(8), 1040–1058 (2009)
16. Chand, S., Wagner, M.: Evolutionary many-objective optimization: A quick-start guide. *Surveys in Operations Research and Management Science* **20**(2), 35 – 42 (2015). DOI <https://doi.org/10.1016/j.sorms.2015.08.001>
17. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research* **16**, 321–357 (2002)
18. Chen, D., Fu, W., Krishna, R., Menzies, T.: Applications of psychological science for actionable analytics. In: *ESEC/SIGSOFT FSE* (2018)
19. Chen, J., Nair, V., Krishna, R., Menzies, T.: “Sampling” as a baseline optimizer for search-based software engineering. *IEEE Transactions on Software Engineering* (2018)
20. Chen, J., Nair, V., Menzies, T.: Beyond evolutionary algorithms for search-based software engineering. *Information and Software Technology* **95**, 281–294 (2018)
21. Chiu, N.H., Huang, S.J.: The adjusted analogy-based software effort estimation based on similarity distances. *Journal of Systems and Software* **80**(4), 628–640 (2007)
22. Cohen, W.W.: Fast effective rule induction. In: *Machine Learning Proceedings 1995*, pp. 115–123. Elsevier (1995)
23. De Carvalho, A.B., Pozo, A., Vergilio, S.R.: A symbolic fault-prediction model based on multiobjective particle swarm optimization. *Journal of Systems and Software* **83**(5), 868–882 (2010)
24. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation* **6**(2), 182–197 (2002). DOI [10.1109/4235.996017](https://doi.org/10.1109/4235.996017)
25. Deng, L., Yu, D., et al.: Deep learning: methods and applications. *Foundations and Trends® in Signal Processing* **7**(3–4), 197–387 (2014)
26. Du, X., Yao, X., Ni, Y., Minku, L.L., Ye, P., Xiao, R.: An evolutionary algorithm for performance optimization at software architecture level. In: *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pp. 2129–2136. IEEE (2015)
27. Durillo, J.J., Nebro, A.J.: jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software* **42**, 760–771 (2011). DOI [DOI:10.1016/j.advengsoft.2011.05.014](https://doi.org/10.1016/j.advengsoft.2011.05.014)
28. Fayyad, U., Irani, K.: Multi-interval discretization of continuous-valued attributes for classification learning (1993)
29. Feather, M.S., Menzies, T.: Converging on the optimal attainment of requirements. In: *Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on*, pp. 263–270. IEEE (2002)
30. Fishburn, P.C.: Nontransitive preferences in decision theory. *Journal of Risk and Uncertainty* **4**(2), 113–134 (1991). DOI [10.1007/BF00056121](https://doi.org/10.1007/BF00056121)
31. Frank, E., Trigg, L., Holmes, G., Witten, I.H.: Technical note: Naive bayes for regression. *Machine Learning* **41**(1), 5–25 (2000). DOI [10.1023/A:1007670802811](https://doi.org/10.1023/A:1007670802811)
32. Freund, Y., Schapire, R.E., et al.: Experiments with a new boosting algorithm. In: *Icml*, vol. 96, pp. 148–156. Citeseer (1996)
33. Friedrich, T., Quinzan, F., Wagner, M.: Escaping large deceptive basins of attraction with heavy-tailed mutation operators. In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '18*, pp. 293–300. ACM, New York, NY, USA (2018). DOI [10.1145/3205455.3205515](https://doi.org/10.1145/3205455.3205515)
34. Fu, W., Menzies, T.: Easy over hard: A case study on deep learning. In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pp. 49–60. ACM (2017)
35. Fu, W., Menzies, T., Chen, D., Agrawal, A.: Building better quality predictors using “ ϵ – dominance”. *arXiv preprint arXiv:1803.04608* (2018)
36. Fu, W., Menzies, T., Shen, X.: Tuning for software analytics: Is it really necessary? *Information and Software Technology* **76**, 135–146 (2016)
37. Fu, W., Menzies, T., Shen, X.: Tuning for software analytics: Is it really necessary? *Information and Software Technology* **76**, 135–146 (2016)
38. Fu, W., Nair, V., Menzies, T.: Why is differential evolution better than grid search for tuning defect predictors? *arXiv preprint arXiv:1609.02613* (2016)
39. van Gerven, M., Bohte, S.: Artificial neural networks as models of neural information processing. *Frontiers Media SA* (2018)

40. Ghotra, B., McIntosh, S., Hassan, A.E.: Revisiting the impact of classification techniques on the performance of defect prediction models. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 1, pp. 789–800 (2015)
41. Glover, F., Laguna, M.: Tabu search. In: Handbook of combinatorial optimization, pp. 2093–2229. Springer (1998)
42. Gondra, I.: Applying machine learning to software fault-proneness prediction. *Journal of Systems and Software* **81**(2), 186–195 (2008)
43. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: An update. *SIGKDD Explor. Newsl.* **11**(1), 10–18 (2009). DOI 10.1145/1656274.1656278
44. Hall, M.A., Holmes, G.: Benchmarking attribute selection techniques for discrete class data mining. *IEEE Transactions on Knowledge and Data Engineering* **15**(6), 1437–1447 (2003)
45. Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S.: A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering* **38**(6), 1276–1304 (2012)
46. Harman, M., Jones, B.F.: Search-based software engineering. *Information and software Technology* **43**(14), 833–839 (2001)
47. Harman, M., Mansouri, S.A., Zhang, Y.: Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)* **45**(1), 11 (2012)
48. Huang, S.J., Chiu, N.H.: Optimization of analogy weights by genetic algorithm for software effort estimation. *Information and software technology* **48**(11), 1034–1045 (2006)
49. Huang, S.J., Chiu, N.H., Chen, L.W.: Integration of the grey relational analysis with genetic algorithm for software effort estimation. *European Journal of Operational Research* **188**(3), 898–909 (2008)
50. Huang, V.L., Suganthan, P.N., Qin, A.K., Baskar, S.: Multiobjective differential evolution with external archive and harmonic distance-based diversity measure. School of Electrical and Electronic Engineering Nanyang, Technological University Technical Report (2005)
51. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: International Conference on Learning and Intelligent Optimization, pp. 507–523. Springer (2011)
52. Jolliffe, I.: Principal component analysis. In: International encyclopedia of statistical science, pp. 1094–1096. Springer (2011)
53. Kessentini, M., Ruhe, G.: A guest editorial: special section on search-based software engineering. *Empirical Software Engineering* **21**(6), 2456–2458 (2016). DOI 10.1007/s10664-016-9474-0
54. Kitchenham, B.: Guidelines for performing systematic literature reviews in software engineering, version 2.3. Tech. Rep. EBSE-2007-01, Keele University (2007)
55. Kotthoff, L.: Algorithm selection for combinatorial search problems: A survey. In: Data Mining and Constraint Programming, pp. 149–190. Springer (2016)
56. Koza, J.R.: Genetic programming as a means for programming computers by natural selection. *Statistics and computing* **4**(2), 87–112 (1994)
57. Krall, J., Menzies, T., Davies, M.: Gale: Geometric active learning for search-based software engineering. *IEEE Transactions on Software Engineering* **41**(10), 1001–1018 (2015)
58. Krishna, R., Menzies, T.: Bellwethers: A baseline method for transfer learning. *IEEE Transactions on Software Engineering* pp. 1–1 (2018)
59. Kuhn, M.: Building predictive models in r using the caret package. *Journal of Statistical Software, Articles* **28**(5), 1–26 (2008). DOI 10.18637/jss.v028.i05
60. Kumar, K.V., Ravi, V., Carr, M., Kiran, N.R.: Software development cost estimation using wavelet neural networks. *Journal of Systems and Software* **81**(11), 1853–1867 (2008)
61. Kwiatkowska, M., Norman, G., Parker, D.: Prism 4.0: Verification of probabilistic real-time systems. In: International conference on computer aided verification, pp. 585–591. Springer (2011)
62. Lessmann, S., Baesens, B., Mues, C., Pietsch, S.: Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering* **34**(4), 485–496 (2008). DOI 10.1109/TSE.2008.35
63. Liaw, A., Wiener, M., et al.: Classification and regression by randomforest. *R news* **2**(3), 18–22 (2002)
64. Liu, Y., Khoshgoftar, T.M., Seliya, N.: Evolutionary optimization of software quality modeling with multiple repositories. *IEEE Transactions on Software Engineering* **36**(6), 852–864 (2010)
65. Majumder, S., Balaji, N., Brey, K., Fu, W., Menzies, T.: 500+ times faster than deep learning (a case study exploring faster methods for text mining stackoverflow). arXiv preprint arXiv:1802.05319 (2018)
66. Menzies, T., Elrawas, O., Hihn, J., Feather, M., Madachy, R., Boehm, B.: The business case for automated software engineering. In: Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering, ASE ’07, pp. 303–312. ACM, New York, NY, USA (2007). DOI 10.1145/1321631.1321676

67. Menzies, T., Kocagineli, E., Minku, L., Peters, F., Turhan, B.: Data science for software engineering: Sharing data and models (2013)
68. Menzies, T., Shepperd, M.: 'bad smells' in software analytics papers. *Information and Software Technology* **112**, 35 – 47 (2019)
69. Menzies, T., Williams, L., Zimmermann, T.: *Perspectives on Data Science for Software Engineering*. Morgan Kaufmann, Boston (2016)
70. Menzies, T., Zimmermann, T.: Software analytics: so what? *IEEE Software* **4**, 31–37 (2013)
71. Menzies, T., Zimmermann, T.: Software analytics: Whats next? *IEEE Software* **35**(5), 64–70 (2018). DOI 10.1109/MS.2018.290111035
72. Minku, L., Yao, X.: Software effort estimation as a multi-objective learning problem. *ACM Transactions on Software Engineering and Methodology* **22**(4) (2013)
73. Minku, L., Yao, X.: How to make best use of cross-company data in software effort estimation? In: ICSE, pp. 446–456. Hyderabad (2014)
74. Minku, L., Yao, X.: Which models of the past are relevant to the present? a software effort estimation approach to exploiting useful past models. *Automated Software Engineering Journal* **24**(7), 499–542 (2017)
75. Minku, L.L., Yao, X.: An analysis of multi-objective evolutionary algorithms for training ensemble models based on different performance measures in software effort estimation. In: Proceedings of the 9th international conference on predictive models in software engineering, p. 8. ACM (2013)
76. Minku, L.L., Yao, X.: Software effort estimation as a multiobjective learning problem. *ACM Transactions on Software Engineering and Methodology (TOSEM)* **22**(4), 35 (2013)
77. Montaez, G.D.: Bounding the number of favorable functions in stochastic search. In: 2013 IEEE Congress on Evolutionary Computation, pp. 3019–3026 (2013). DOI 10.1109/CEC.2013.6557937
78. Mori, T., Uchihiro, N.: Balancing the trade-off between accuracy and interpretability in software defect prediction. *Empirical Software Engineering* (2018). DOI 10.1007/s10664-018-9638-1
79. Nair, V., Agrawal, A., Chen, J., Fu, W., Mathew, G., Menzies, T., Minku, L., Wagner, M., Yu, Z.: Data-driven search-based software engineering. In: Proceedings of the 15th International Conference on Mining Software Repositories, MSR '18, pp. 341–352. ACM, New York, NY, USA (2018). DOI 10.1145/3196398.3196442
80. Nair, V., Krishna, R., Menzies, T., Jamshidi, P.: Transfer learning with bellwethers to find good configurations. *CoRR abs/1803.03900* (2018)
81. Nair, V., Menzies, T., Siegmund, N., Apel, S.: Using bad learners to find good configurations. *arXiv preprint arXiv:1702.05701* (2017)
82. Nair, V., Yu, Z., Menzies, T., Siegmund, N., Apel, S.: Finding faster configurations using Flash. *arXiv preprint arXiv:1801.02175* (2018)
83. Neshtat, M., Alexander, B., Wagner, M., Xia, Y.: A detailed comparison of meta-heuristic methods for optimising wave energy converter placements. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '18, pp. 1318–1325. ACM, New York, NY, USA (2018). DOI 10.1145/3205455.3205492
84. Oliveira, A.L., Braga, P.L., Lima, R.M., Cornélio, M.L.: GA-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation. *Information and Software Technology* **52**(11), 1155–1166 (2010)
85. Panichella, A., Dit, B., Oliveto, R., Di Penta, M., Poshvanyk, D., De Lucia, A.: How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In: Proceedings of the 2013 International Conference on Software Engineering, pp. 522–531. IEEE Press (2013)
86. Pareto, V.: *Manuale di economia politica*, vol. 13. Societa Editrice (1906)
87. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* **12**, 2825–2830 (2011)
88. Peters, F., Menzies, T., Layman, L.: Lace2: Better privacy-preserving data sharing for cross project defect prediction. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 1, pp. 801–811. IEEE (2015)
89. Pohl, R., Lauenroth, K., Pohl, K.: A performance comparison of contemporary algorithmic approaches for automated analysis operations on feature models. In: 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), pp. 313–322 (2011). DOI 10.1109/ASE.2011.6100068
90. Poli, R., Kennedy, J., Blackwell, T.: Particle swarm optimization. *Swarm intelligence* **1**(1), 33–57 (2007)
91. Polikar, R.: Ensemble based systems in decision making. *IEEE Circuits and systems magazine* **6**(3), 21–45 (2006)
92. Quinlan, J.R.: Learning with continuous classes. In: Proceedings AI'92, pp. 343–348. World Scientific (1992)

93. Rainville, D., Fortin, F.A., Gardner, M.A., Parizeau, M., Gagné, C., et al.: Deap: A python framework for evolutionary algorithms. In: Proceedings of the 14th annual conference companion on Genetic and evolutionary computation, pp. 85–92. ACM (2012)
94. Riffenburgh, R.H.: Linear discriminant analysis. Ph.D. thesis, Virginia Polytechnic Institute (1957)
95. Roweis, S.T., Saul, L.K.: Nonlinear dimensionality reduction by locally linear embedding. *science* **290**(5500), 2323–2326 (2000)
96. Sadiq, A.S., Alkazemi, B., Mirjalili, S., Ahmed, N., Khan, S., Ali, I., Pathan, A.S.K., Ghafoor, K.Z.: An efficient ids using hybrid magnetic swarm optimization in wanets. *IEEE Access* **6**, 29,041–29,053 (2018)
97. del Sagrado, J., Águila, I.M., Orellana, F.J.: Requirements interaction in the next release problem. In: Proceedings of the 13th annual conference companion on Genetic and evolutionary computation, pp. 241–242. ACM (2011)
98. Sarro, F., Di Martino, S., Ferrucci, F., Gravino, C.: A further analysis on the use of genetic algorithm to configure support vector machines for inter-release fault prediction. In: Proceedings of the 27th annual ACM symposium on applied computing, pp. 1215–1220. ACM (2012)
99. Sarro, F., Ferrucci, F., Gravino, C.: Single and multi objective genetic programming for software development effort estimation. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC '12, pp. 1221–1226. ACM, New York, NY, USA (2012). DOI 10.1145/2245276.2231968
100. Sarro, F., Petrozziello, A., Harman, M.: Multi-objective software effort estimation. In: Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on, pp. 619–630. IEEE (2016)
101. Sayyad, A.S., Ingram, J., Menzies, T., Ammar, H.: Scalable product line configuration: A straw to break the camel's back. In: 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 465–474 (2013)
102. Sayyad, A.S., Menzies, T., Ammar, H.: On the value of user preferences in search-based software engineering: a case study in software product lines. In: Proceedings of the 2013 International Conference on Software Engineering, pp. 492–501. IEEE Press (2013)
103. Shen, X.N., Minku, L.L., Marturi, N., Guo, Y.N., Han, Y.: A q-learning-based memetic algorithm for multi-objective dynamic software project scheduling. *Information Sciences* **428**, 1 – 29 (2018). DOI <https://doi.org/10.1016/j.ins.2017.10.041>
104. Steinwart, I., Christmann, A.: Support vector machines. Springer Science & Business Media (2008)
105. Storn, R., Price, K.: Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization* **11**(4), 341–359 (1997). DOI 10.1023/A:1008202821328
106. Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K.: Automated parameter optimization of classification techniques for defect prediction models. In: Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on, pp. 321–332. IEEE (2016)
107. Treude, C., Wagner, M.: Per-corpus configuration of topic modelling for github and stack overflow collections. *arXiv preprint arXiv:1804.04749* (2018)
108. Tu, H., Nair, V.: Is one hyperparameter optimizer enough? In: SWAN 2018 (2018)
109. Vandecruys, O., Martens, D., Baesens, B., Mues, C., De Backer, M., Haesen, R.: Mining software repositories for comprehensible software fault prediction models. *Journal of Systems and software* **81**(5), 823–839 (2008)
110. Veerappa, V., Letier, E.: Understanding clusters of optimal solutions in multi-objective decision problems. In: 2011 IEEE 19th International Requirements Engineering Conference, pp. 89–98 (2011). DOI 10.1109/RE.2011.6051654
111. Wang, T., Harman, M., Jia, Y., Krinke, J.: Searching for better configurations: a rigorous approach to clone evaluation. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, pp. 455–465. ACM (2013)
112. Weise, T., Wu, Z., Wagner, M.: An improved generic bet-and-run strategy for speeding up stochastic local search. *CoRR* **abs/1806.08984** (2018). Accepted for publication at AAAI 2019.
113. Wolpert, D.H.: The lack of a priori distinctions between learning algorithms. *Neural Computation* **8**(7), 1341–1390 (1996). DOI 10.1162/neco.1996.8.7.1341
114. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* **1**(1), 67–82 (1997). DOI 10.1109/4235.585893
115. Wu, X., Consoli, P., Minku, L., Ochoa, G., Yao, X.: An evolutionary hyper-heuristic for the software project scheduling problem. In: J. Handl, E. Hart, P.R. Lewis, M. López-Ibáñez, G. Ochoa, B. Paechter (eds.) *Parallel Problem Solving from Nature – PPSN XIV*, pp. 37–47. Springer, Cham (2016)
116. Xia, T., Krishna, R., Chen, J., Mathew, G., Shen, X., Menzies, T.: Hyperparameter optimization for effort estimation. *CoRR* **abs/1805.00336** (2018)
117. Xu, B., Ye, D., Xing, Z., Xia, X., Chen, G., Li, S.: Predicting semantically linkable knowledge in developer online forums via convolutional neural network. In: 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 51–62 (2016)

118. Xu, T., Jin, L., Fan, X., Zhou, Y., Pasupathy, S., Talwadker, R.: Hey, you have given me too many knobs!: Understanding and dealing with over-designed configuration in system software. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, pp. 307–319. ACM, New York, NY, USA (2015). DOI 10.1145/2786805.2786852
119. Yu, Z., Kraft, N.A., Menzies, T.: Finding better active learners for faster literature reviews. *Empirical Software Engineering* **23**(6), 3161–3186 (2018)
120. Zhang, Q., Li, H.: Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation* **11**(6), 712–731 (2007)
121. Zhong, S., Khoshgoftaar, T.M., Seliya, N.: Analyzing software measurement data with clustering techniques. *IEEE Intelligent Systems* **19**(2), 20–27 (2004)
122. Zitzler, E., Künzli, S.: Indicator-based selection in multiobjective search. In: PPSN (2004)
123. Zuluaga, M., Krause, A., Sergeant, G., Püschel, M.: Active learning for multi-objective optimization. In: Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28, ICML'13, pp. I-462–I-470. JMLR.org (2013)

Response to Reviewers

We are most grateful to the Associate Editor and the anonymous reviewers for their constructive comments, which have helped to improve the quality of this paper significantly. In this response, the reviewers' comments were enumerated and are cited in blue italic, followed by our responses. We have also highlighted our modifications in the manuscript file using red to facilitate the editor's checking of our revisions.

Associate Editor

*Based on the 3 reviews (2 * major revision, 1 reject), we recommended a “Major Revision.” Special attention should be paid to:*

- Tone down the 4th claim and/or add more data to support the claim*
- Clarify how the paper evaluates the 1st claim*
- Discuss the SE community can make use of the 2nd, and 3rd claims*

*Please provide a detailed *point-by-point* response letter to identify how the reviewers' feedback has been addressed in the revision.*

In reply to the above:

- We have followed your direction and toned down Claim4 (and we have added more data to support it). Now we say “not recommended” instead of “deprecated”. Also, we have changed certain parts of the text that were too... ardent.. in this regard.
- Claim 1: machine learning and optimization problems are very similar by definition. As such, this is not the type of claim that is empirically evaluated. Therefore, we are willing to move the content of Claim 1 to the section about definitions, if you direct us to do so.
- To address your requirement about Claims2/3 we have included more material here to let the SE community make better use of our Claim2/3 content. For educators, we have reinstated the material about how to find and use case study material in this arena (see Figure 1). For practitioners, we have included lists of the kinds of business-level goals that can be explored with technology (see start of §2). Also, in §4.1 and §5.3 we have added 2 pages containing 24 heuristics for applying this kind of technology in an industrial setting.

As to other matters, Reviewer3 suggests we remove the second point in our Future Work section. Please, what is your direction on this? Reviewer3 says these items are not “software engineering” but we think that they are the new frontier of SE research. That said, we will remove them if you so require.

Reviewer 1

- 1.1. *This paper argues that there should be a new area of research in empirical software engineering research and practice where data mining and optimization should be combined and should assist each other for better and scalable outcome. The authors made four claims, (1) Optimization and data mining are very similar, (2) Optimizers can greatly improve data miners, (3) Data miners can greatly improve optimization and (4) Data mining without optimization should be deprecated.*

The paper reads well and the claims were made based on the survey data they collected. I also fully agree with the first three claims. However, I have a concern on the fourth claim, Data mining without optimization should be deprecated. I don't agree with this statement. MSR community has been exploring lots of novel ideas and insights from the Data without applying search based SE or optimization if you say. These studies have been useful as we see as a community. I agree that exploiting optimization would make them only better. However, I don't really agree that they should be deprecated.

Thank for you that comment. You are quite correct: we overstated Claim4. In this draft we have made numerous changes that moves “deprecated” to “not recommended”. Our text is now less ardent and now politely suggests to researchers that they might want to consider adding an optimization step to their work. Thank you for pointing out that we needed that clarification.

- 1.2. *While I was not fully happy with their conference submission, I even liked that one better than this version since that version provided some resources in these two areas. I was particularly happy that someone new in the area could be benefited from such resources. However, this extended version does not really add much. What is says we already know (the first three claims). Their support of data on those claims do not add much. As I noted, I could select another subset of data and draw some completely opposite claims. And for the fourth claim, I don't think they can even claim that with such a minimal set of data. Did they review the complete body of knowledge in empirical software engineering?*

Thank you for your comment. It was a mistake not adding the teaching and training resources from the prior version into this paper. This has now been fixed (see Figure 1). Regarding the support of data and the possibility of another subset of data leading to completely opposite claims, our conference paper was based on a set of articles collected based on our own knowledge about the field, being a potentially biased set of articles. Other authors selecting articles based on their own knowledge of the field might indeed reach different conclusions. Our extended paper performs a systematic literature review precisely to avoid conclusions being drawn based on a biased set of data. As explained by Kitchenham [54], “[s]ystematic reviews aim to present a fair evaluation of a research topic by using a trustworthy, rigorous, and auditable methodology”.

As to “claim that with such a minimal set of data”, that was also a mistake. We have added more material to the end of §6 showing how as a side effect of selecting better models, we also change what can be read or elared from that model.

As to “review the complete body of knowledge in empirical software engineering?”, no, we could not. Nevertheless, the use of optimization has shown to positively affect a number of different machine learning algorithms applied to software engineering problems (see the end of §6). In fact, hyperparameters only exist in machine learning algorithms *because* they are expected to affect the models that they generate. Therefore, we updated §6 to explain that we *recommend* data mining to be used with optimization.

- 1.3. *I would consider this paper if the authors tone down their claims and/or add more data to support the claims while they would need to judge papers of diverse areas in empirical software engineerings.*

We hope this current draft addresses your concerns. We have toned down claim4 and we have added more examples (and a little theory) that endorse that claim.

Reviewer 2

2.1. *# Summary: Data mining and optimization are very similar*

SE works should engage in optimization before publication. OPTimization of data-mining and optimization of parameters. Default parameters and no attempt to tease out proper performance of a learner should be avoided when one tries to make claims: Learner X doesn't work for SE problem Y. The authors describe methods of optimization and data mining, show their relationship, and the need for optimization in numerous avenues including hyper-parameter tuning. This is important because it melds 2 SE communities closer together with some helpful tips.

Thank for that comment.

2.2. *# Recommendation: Major Revision*

Why? I feel a lot of the discussion is lacking and the paper isn't clear enough for the average SE grad student to understand.

We agree—lately we have been writing the tutorial material required (to teach this paper to a graduate class of SE masters and Ph.D. students for Fall 2019). That material is fast becoming a book, all to itself (110 page at last count, and a few thousand likes of code. FYI, when we release that material, it will appear at <http://tiny.cc/data-se> and <https://github.com/txt/ase19/blob/master/README.md>).

Now the question is, how much of that introductory material can be usefully added to this paper? Clearly, we can't use it all (too long). Its a hard problem. Here is our solution: we have placed back into this paper the training resources material we built for the earlier MSR conference version of this paper (see Figure 1). Note that as we mature our teaching materials, so will those on-line materials mature (e.g. they should be extensive tutorial lecture notes added, and class exercises).

2.3. *# Is the work different from the MSR paper?*

As a prior reviewer for the MSR paper this work is significantly different.

Thank you for that comment.

2.4. *# Fixes needed*

Can you address the issue of locally optimized versus say general performance? Should all our results be optimized or should we report the range of results?

Not quite sure what you are after here but we have added a note around Figure 5 that comments on local-vs-global. We also added a new last paragraph to §6 to explain that reporting a range of results using different hyperparameter choices (i.e., analyses of sensitivity to hyperparameters) is still important.

2.5. *Page 2 line 31 needed by [100] please make sure that citations are nouns and not parentheticals, it makes it hard to read. e.g. So&So et al.[100] is better than [100] on its own.*

Thank you for your suggestion! That has been fixed.

2.6. *Page 15 Figure 4 is an unreadable screenshot. Please steal that image better and cite the authors names DIRECTLY in the reference.*

Replaced with original high resolution figure. Thank you for your comment!

2.7. *Why is the algorithm a figure. Figure 5. Why is it inline? It could be just text. This is weird. Please fix.*

Fixed! Thanks for the suggestion.

2.8. *# Clarity:*

The biggest barrier the original paper suffered from and that this paper faces is clarity. 2 communities are being involved so examples and clarifications across fields would be of value.

** Define data-mining up front. You’re using a subset of MSR work as data-mining and it is not exactly what the boundaries of data mining are.*

Great idea! Please see §2 for those notes.

- 2.9. ** Algorithm (1) the optimization needs to be clarified, the use of g' and g'' when g and h were already available. Give g' and g'' names. Not just purposes.*

We are not completely sure if we understood what the reviewer meant by giving names not just purposes. We attempt to address the reviewer’s comment below, but please advise if we misunderstood the reviewer’s request.

It is common to refer to the inequality and equality constraints using the symbols g and h in the optimization literature. However, we have opted for using g' and g'' to avoid confusion with the data mining terminology introduced in Section 3. Footnote 2 has now been updated to clarify that in Section 3. The “names” of g' and g'' are inequality and equality constraints, respectively, as explained below Equation 1.

- 2.10. ** Furthermore you define CONVEX optimization. This is SE, there’s no guarantees of convexity in many problems. So why limit the definition to convex optimization? Many forms of Heuristic search are heavily used in the rest of your paper and do not need convex optimization. Line 10 on page 5 referring to convex optimization should be more specific.*

Even though the book being cited when defining optimization problems is a book on convex optimization, the definition being given is not restricted to convex optimization problems. A convex optimization problem is an optimization problem where the objective and constraint functions are convex. To avoid confusion, we are now referring to the specific section of the book about general optimization problems, rather than referring to the book as a whole.

- 2.11. ** You are writing to readers of many fields. Using the term “samplers” for various forms of heuristic search either needs a clarification point whereby you name some individuals, or you use terminology from a particular field—such as search.*

§5.1 now takes care to define “sampling” before it is used.

- 2.12. *# Missing discussion*

- Evaluation measures that are SE relevant. THings like cost and time limits. We need to discuss that optimization should not be for IR/ML performance measures but task performance (like cost).

Great suggestion! We offer such a list in §2.

- 2.13. *# A general gripe*

I felt like other search, less optimization based, cross-overs were being ignored in favor of DUO. A lot of information and suggestions about priors mined from existing software was kind of glossed over. There were mentions in the paper but if you have time can you try to prop up the mining leg as priors are often valuable.

This paper was built around the query shown in Table 1; i.e. software engineering AND (“optimization” OR “evolutionary algorithm”) AND query(“data mining” OR “analytics” OR “machine learning”). So what you could be saying is that the field we are reporting is biased in the way your describe above. Now if that is true, we wonder how best to proceed. Advice please!

Reviewer 3

Thank you for for comments. They clearly show that you have carefully read this material, for which we thank you.

Just to save some time, we start this reply section by saying we are willing to delete Claim1, and the second set of research directions (the one with four sub-points). We make the case below that they should stay but, if the associate editor tells us otherwise, then we will happily remove those sections.

- 3.1. *Summary: The paper is about DUO: datamining using/used by optimizers. The authors argue that optimization and data mining go hand-in-hand and that every time some machine learning lagoon is used, there should be optimization. They have 4 claims in the paper. Some are supported by literature and one is argued logically. They also carry out a literature survey on the subject matter.*

Review:

I agree with the authors that optimization must happen whenever ML algos/models are used in SE. Additionally, I like the literature review done by the authors. Table 1 and claim 4 (and partly 3) are very cool contributions.

Thank you for those kind words.

- 3.2. *However, I find a considerable portion of this work highly misplaced. Why is this paper at EMSE? What is the value of claim 1 for SE folks (practitioners and researchers)? I can see how claim 4 is useful. Letting us know that as researchers we should not be using ML models without optimizing for hyperparameters. Let me present issues claim by claim.*
- As to “why is this paper at EMSE?”, we have decades of experience writing papers of the form “here is tiny experiment1 and here are its results”. This paper is an experiment that is painting a bigger picture— something we feel is lacking in this field.
 - We address your concerns on claims 1–3 over the next paragraphs.
- 3.3. *Claim 1: Optimization and data mining are very similar. Ask any core ML person and they would say all of it is nothing but optimization. In fact all ML theory people are essentially mathematicians who do optimization. Now I am not an ML theory person. And neither are any of the readers or reviewers of EMSE journal. So my questions are:*
- a) How can we evaluate the claim?*
 - b) Is this a new finding? Have ML theory people not done this? Why present in a SE journal?*
 - c) What is the claims connection to SE? No motivation at all here for the required audience.*

We respectfully disagree with the reviewer that any core ML person would say that all of it is nothing but optimization. The core of ML theory is tightly linked to the need for generalization, meaning that we must learn models that represent well an *unknown* function. This is not considered by the optimization theory. We have colleagues who are experts on ML theory and confirmed that ML is not about optimization. We also have colleagues who are experts on optimization theory. They are the ones who claim that ML is nothing but optimization. This is because they are not familiar with ML theory, and are only aware of the fact that optimization algorithms are frequently used to build machine learning models.

The differences in opinions between experts on ML and optimization illustrates well the lack of discussion on the similarities and differences between the two areas in the

usual literature. However, if one intends to work on the intersection between these two fields (DUO), it is important to understand what the similarities and differences are. And, given the benefits that DUO has been showing to bring to SE and the difficulty to find accessible literature providing such discussions, we consider it adequate and beneficial to present this in a SE journal.

We refer to Claim 1 as a claim because it is made based on our knowledge and experience of working on both areas for many years, rather than on a statement coming from textbooks. However, the two areas are very similar (not the same) by definition. As such, this is not the type of claim that can be empirically evaluated, except for the evidence that there are more and more works on combining these areas, as shown in this paper. For this reason, if the reviewer requests, we could move the content of Section 3 to Section 2, which is about definitions.

- 3.4. *Claim 2: Optimizers can improve data mining. I like the fact that SE research is used to support this claim. I would like to know if there are any findings that are specific to SE data? I do learn that there are quite a few papers that do this. So what? What ways are data mining SE data being improved by optimizers.*

Great question! As you suggest here, there are things very different about SE. You might be interested in Binkley et al’s recent paper *The need for software specific natural language techniques*. And, with respect to optimization, we have shown in another paper (currently under review at TSE¹¹ that:

- Generalizing from our results, perhaps it is time for a new characterization of software analytics:
- *Software analytics is that branch of machine learning that studies problems with large E outputs.*
- This new characterization is interesting since it means that machine learning algorithm developed in the AI community might not apply to SE. We suspect that understanding SE is a fundamentally different problem to understanding other problems that are more precisely controlled and restrained. Perhaps, it is time to design new machine learning algorithms that are better suited to large E SE problems. As shown in this article, such new algorithms can exploit the peculiarities of SE data to dramatically simplify software analytics.”

Now as to how much of that material should appear here, we are not clear. Our own paper, for example, is still under review. Is there something specific you think we should add in here from (say) our paper or the work of Binkley et al.?

- 3.5. *Claim 3: Data miners improve optimization. Similar to claim 1, is this a new finding? What is novel about this? Why should it be in SE journal?*

One reason to include Claim3 is that we found it a useful place to present many of the results from the literature review.

Another reason to include Claim3 is that it is an important idea. In many of our recent papers, this insight was the core idea that allowed use to significantly improve on multiple past results. It is a technique that is barely explored in the SE literature. We assert that it is very valuable to widely publicize that technique. Hence, we include it here.

- 3.6. *Claim 4: I really like this claim. There is value for an SE audience. The two examples are excellent.*

Thank you for your kind words. Note that we have extended the examples in this section.

¹¹ <https://arxiv.org/pdf/1902.01838.pdf>

We hope you do not mind but that another reviewer required us to tone down that claim a little. Now we say "not recommended" rather than "deprecated" – since that seems politer to the general analytics community.

- 3.7. *Research direction: The first one is relevant to SE. The 2nd one (with 4 directions) are not SE related. I am not sure how or why these directions came from the result of this paper. Are these an exhaustive list?*

We are willing to delete second set of research directions (the one with four sub-points). We make the case that they should stay but, if the associate editor tells us otherwise, then we will happily remove those sections. Our argument for keeping them is that we have been working in this area for sometime, and believe that those research directions will bring benefits to software engineering. In particular, these are the areas that are concerning us more and more when applying optimization to software engineering.