Journal of Machine Learning Research 18 (2017) 1-39 Submitted 5/16; Revised 3/17; Published 4/17

Bridging Supervised Learning and Test-Based Co-optimization

Elena Popovici ELENA@ICOSYSTEM.COM
Icosystem Corp.
222 Third Street, Suite 0142
Cambridge, MA 02142, USA

Editor: Una-May O'Reilly

Abstract

This paper takes a close look at the important commonalities and subtle differences between the well-established field of supervised learning and the much younger one of co-optimization. It explains the relationships between the problems, algorithms and views on cost and performance of the two fields, all throughout providing a two-way dictionary for the respective terminologies used to describe these concepts. The intent is to facilitate advancement of both fields through transfer and cross-pollination of ideas, techniques and results. As a proof of concept, a theoretical study is presented on the connection between existence / lack of free lunch in the two fields, showcasing a few ideas for improving computational complexity of certain supervised learning approaches.

Keywords: supervised learning, active learning, co-optimization, free lunch, optimal algorithms

1. Introduction

It is not uncommon in science that separate fields independently develop similar ideas but describe them with different language, or that they need to solve similar problems but approach them from different perspectives and thus derive different methods of solving them. Either field would benefit from learning about the latest and greatest in the other, but achieving such benefits requires overcoming a couple of barriers: lack of awareness of the existence of another field concerned with similar issues and lack of understanding of its language and perspective. The goal of this paper is to break such barriers between the fields of supervised learning and co-optimization.

Supervised learning has been around for a long time, thus knowledge and understanding of it has permeated other disciplines. Loosely speaking, it is concerned with generalizing from given examples to unseen ones. Co-optimization is a much newer area, not widely known outside its community. The term has been introduced by Service and Tauritz (2008a), though the ideas had already been floating around in the field of coevolution (Popovici et al., 2010), a sub-field of evolutionary computation (De Jong, 2006) that has been around since the early 90's (Hillis, 1990). There are multiple possible goals in co-optimization, of which a common one is finding solutions that would perform well in many different situations (tests), given the ability to do only a limited number of performance assessments. This type of goal is the focus of *test-based co-optimization*. To draw a quick parallel to supervised learning,

un-assessed situations are similar in nature to unseen examples, but their space can and must be explored. The present work delves into this parallel in much greater detail.

We start in Section 2.1 with a handful of examples of problems from both fields, first presented in natural language and then formalized to highlight both the commonalities and the differences between them. Then in Section 2.2 we discuss the relationship between the typical algorithmic approaches applied to solve such problems in their respective fields. Two themes thread through both sections: how much and what kind of information is available to the problem-solver, and what are the various kinds of cost the problem-solver might incur. The performance-versus-cost issue is further detailed in Section 2.3, highlighting how both fields lack a performance-evaluation approach encompassing all cost types.

Thus Section 2 should provide a useful guide to those newcomers to one of the fields that already have a background in the other, but it is *not a detailed survey* of either field; many books and review papers are available for that purpose, e.g., by Abu-Mostafa et al. (2012) for supervised learning, Popovici et al. (2010) for co-optimization. Rather, it presents a *mapping between the core issues and concepts* that researchers in the two fields concern themselves with and the language they use in the process (summarized in Table 3). State-of-the-art approaches in either field should be describable in terms of such core notions; while not the main focus, a couple of supervised learning examples, such as deep learning (LeCun et al., 2015) and the reusable holdout (Dwork et al., 2015), are mentioned in sections 2.3.1 and 2.3.2.

The motivation behind this type of presentation is to enable researchers in the two fields to better understand each other and provide a bridge over which methods, ideas and results can travel both ways and contribute to the advancement of both fields. In particular, for machine learning researchers, the co-optimization paradigm should reveal the formalization of new and non-trivial learning tasks and of methods for assessing performance on those tasks which offer interesting free lunches. To illustrate the potential, in Section 3 we derive the counterpart for binary classification of a recent line of work in co-optimization concerning the exploitation of free lunch to design algorithms that have optimal performance with respect to a particular type of cost. We discuss the implications for the other cost types described in Section 2.3 and also point out how this new paradigm suggests alternative approaches to instance selection and active learning.

2. Parallel Between Fields

There are three main aspects of any computer science field: the problems to be solved, the algorithms used to solve them and the paradigms used to assess and compare different algorithms' performance with respect to the problem-solving goal. For each of these aspects, we explain what's common and what's different between supervised learning and co-optimization. In the process, we also build a two-way dictionary between the terminologies of these two fields, summarized in Table 3.

2.1 Problems

We first present some problem examples through their real-world formulations, but explaining or replacing domain jargon with simple words. This should make it easier to spot both commonalities and differences, which are then formalized later in Section 2.1.3.

2.1.1 Examples of Co-optimization Problems

Resilient Piping – Automated design of robust physical systems (Co-optimization). Consider the problem of placing valves in a network of water-carrying pipes so as to maximize the resilience of the network in case of damage. Such piping networks carry water from pumps to a number of usage sites; this could be for human consumption, for equipment cooling or for putting out fires (Popovici et al., 2007). Should damage occur somewhere in the network, such as broken pipes causing local leaks and global pressure drop, the damaged areas need to be isolated by automatically closing nearby "smart" valves, so as to stop the leak and restore pressure to as much of the network as possible. Smart valves are expensive and can only be placed sparingly throughout the network. Where they are placed has a big impact on how tightly we can isolate a given damage, i.e., how resilient the network is to that damage. For any given placement of valves and any given damage, we can compute this resilience value using graph-traversal algorithms to identify the valves closest to damaged spots and to determine, once these valves are closed, how many usage sites still have an unobstructed path to a pump and thus can get water. The goal is to find, for a given piping network and number of smart valves, a placement that provides as good as possible resilience across many/all damage scenarios. More broadly, the problem described here is that of automating the design of physical systems that perform well under many different circumstances.

Sorting Networks – Automated algorithm design (Co-search). Consider the task of sorting in increasing order a sequence of numbers by repeatedly comparing pairs of numbers and, if the first is bigger than the second, swapping them. An algorithm performing such sorting is called a sorting network, because it can be represented as a network of compare-and-swap gates. Sorting networks have both software and hardware applications. Each gate incurs a cost, either computational or actual, so one would like to use as few of them as possible. For a given number of available gates and a given length of input sequences, the goal is to find a network with that number of gates that correctly sorts all input sequences of that length (Hillis, 1990). More broadly, the problem described here is that of automating the design of computer programs that for any possible input produce the correct output, where correctness is according to the computational task at hand (e.g., sorting).

2.1.2 Examples of Supervised Learning Problems

Photo Labeling – Automated item classification (Supervised binary classification). Consider the problem of automating the labeling of photos as to whether or not they contain a particular thing, say a balloon. We are provided with a set of examples of photos that have been manually labeled by humans as 'yes' if they contain a balloon or 'no' if they don't. The goal is to find a computer program that outputs the correct label for any possible photo.

Protein Structure – Outcome prediction (Discrete regression). Consider the problem of predicting higher-level protein structure. The primary structure of a protein is specified by a chain of amino-acids of 20 possible types. In nature, this chain folds into a convoluted shape, such that neighboring amino-acids become part of local formations of three different types: coil, helix and beta-strand. Determining this secondary-structure sequence of a given protein (i.e., what type of formation each amino-acid becomes part of) requires lab experiments and it is known only for a small percentage of all known proteins. The goal is

to find a computer program that outputs the correct secondary-structure sequence for any amino-acid sequence (Cheng et al., 2008).

More broadly, these two problems are about learning from samples in order to generalize to new, as yet unseen cases.

2.1.3 Problem Comparisons and Formalisms

While these problems span a varied set of domains and at first glance may not appear particularly related, closer inspection reveals common aspects as well as key differentiators.

All of the descriptions above state that the problem-solving goal is finding something (a placement of valves, a network of compare-and-swap gates, a computer program). This 'object' of interest needs to 'perform' in a certain way (or have some desirable properties) across all/many 'contexts' (damages, number sequences, photos, proteins). While not explicitly stated in the above descriptions, for each of these problems the number of contexts of interest can be very large: there are millions of proteins with known amino-acid sequence but unknown structure; there is an infinite number of photos, or at least a combinatorial number of them if we restrict the size; there is an exponential number of sorting-relevant sequences;[1] and the number of possible damage locations in a water-delivery network can also be significant, with the number of damage scenarios increasing in combinatorial fashion if we account for co-occurrences at different locations (which can happen for instance in case of earthquakes).

Additionally, for each of these problems, the set of objects of interest can also be very large: if there are n possible locations where we could place valves, but we can only afford $m < n$ valves, then the number of possible placements is $\binom{n}{m}$; the number of possible networks with n compare-and-swap gates is also combinatorial in n; and the number of input-output mappings for the computer programs predicting photo labels or protein structures is exponential.

The descriptions above also hint at differences in the information available to help us solve the respective problem. To see this better, we turn to formalisms. Table 1 provides a summary of the notation. Throughout the rest of Section 2 we underline key terminology which is then collated in Table 3.

Co-optimization Formalisms. Consider the problem of designing resilient piping networks. Let S denote the set of all possible placements of a given number of valves in a given piping network. Let T denote the set of all possible/likely damages to that piping network. For any valve placement $s \in S$ and any damage $t \in T$ we can compute the resilience of s to t, as described earlier. Let this computation be denoted by a <u>metric</u> $M : S \times T \to V \subset \mathbb{R}$. The goal is to find a valve placement $s \in S$ that maximizes, say, the *average* resilience over all damages $t \in T$, $g(s) = avg_{t \in T}\, M(s,t)$.[2] We call this goal the <u>maxiavg</u> <u>solution concept</u> (i.e., definition of what constitutes a solution) and we call g the <u>quality function</u>; $g(s)$ is the quality of potential solution s. Equivalently, we could be concerned with the *total* resilience over all damages, $g(s) = \sum_{t \in T} M(s,t)$, a solution concept called <u>maxisum</u>. Other interpretations of 'as good as possible resilience across many/all

1. Specifically, 2^n, where n is the length of the sequence, since any network which correctly sorts all binary sequences of length n also correctly sorts any arbitrary numeric sequence of length n.
2. Assuming such average (or perhaps an expectation) is well defined and that g has a maximum.

damage scenarios' include, for instance, aggregating over damages by taking the worst value rather than the average (worst-case optimization, also called maximin solution concept) or requiring Pareto-dominance, like in multi-objective optimization with the damages playing the role of objectives (Popovici et al., 2010).

Thus, resilient piping design is clearly an optimization problem: we have a set of potential solutions S and we are searching for the element of S that maximizes a function $g : S \to \mathbb{R}$. More specifically, in the field of optimization, this type of problem is called a test-based co-optimization problem. This is because the function g is in fact defined via the metric M, whose definition involves an additional set of entities T. The elements of T are called tests: they do not contribute to the definition of the set of potential solutions S, but they help test which elements of S are solutions and which are not, via the metric M. For brevity, from here on we will use just the word co-optimization when we actually mean test-based co-optimization; co-optimization problems that are not test-based exist, but are outside the scope of this paper.[3]

Note also that we do not have a closed-form mathematical formula for M. Rather, the resilience of a design to a damage is determined through a computational procedure (this could involve graph-traversal algorithms as mentioned above, or, in some cases, a fluid dynamics simulation). While we might know what this procedure is, this knowledge does not easily (or even at all) provide us with a roadmap for solving the problem. What we *can* always do though is to evaluate M to get its output for one input-pair $\langle s, t \rangle$ at a time; $\langle s, t \rangle$ is typically called an interaction or event and determining $M(s, t)$ is referred to as performing a metric evaluation or M-evaluation or, for short, an evaluation; we also say that s has seen t (and t has seen s). For each such evaluation we incur the computational cost of running M (ranging from milliseconds for graph-traversal algorithms to minutes or hours for hydraulics simulations); this is the basic unit of cost for co-optimization algorithms. We say that M is accessible in black-box fashion. In some domains, M may in fact be the outcome of a real-world experiment which we do not know how to simulate computationally, in which case M is truly black-box and even more costly (e.g., days and dollars).

A key point in co-optimization is that the function whose maximum we are trying to find is not the black-box M, but g, which is an aggregate of $|T|$ values of M. As previously discussed, the size of T can be such that computing the actual value of g for even a single $s \in S$ can be very costly or completely infeasible. Even estimating $g(s)$ will typically require evaluating multiple interactions involving s. This is in contrast with traditional black-box optimization, where the quality of a potential solution is given by a single evaluation, since it is potential solutions that are the object of black-box evaluation, not interactions. It is also in contrast with multi-objective optimization, where the number of objectives is so small that evaluating each potential solution on *all* objectives is not an issue and may in fact be considered the basic unit of cost; in co-optimization the set T is something that needs to be explored. Single- and multi-objective optimization could also be seen as instantiations of the more generic framework of co-optimization.[4]

3. In particular, the field of co-optimization also comprises a subfield concerned with so-called *compositional* co-optimization problems (Popovici et al., 2010), some of which have been approached via cooperative coevolutionary algorithms (Potter, 1997).
4. But see also work mapping test-based co-optimization to multi-objective optimization with fewer *underlying* objectives than the size of T (Bucci et al., 2004).

With a bit of reshaping, the sorting network problem can similarly be cast. Let S denote the set of all possible networks made up of a fixed number of compare-and-swap gates and taking as inputs sequences of a fixed length. Let T denote the set of all binary sequences of that length (see footnote 1 on page 4). For any network $s \in S$ and any input sequence $t \in T$, we can determine sorting correctness by running the sequence through the network, then checking if the output is in fact sorted. Let $M : S \times T \to \{0, 1\} \subset \mathbb{R}$ be a metric such that $M(s, t) = 1$ if s correctly sorts t and 0 otherwise. Like for the piping network design problem, M is not given by a closed-form mathematical formula, but for any input-pair $\langle s, t \rangle$ we can compute $M(s, t)$ via the above procedure. As originally stated, the goal for sorting network design is to find a network $s \in S$ such that $M(s, t) = 1$ for all $t \in T$. This is an example of a test-based co-search problem, in that we are searching for an element with a particular property. A co-optimization problem is a co-search problem in which the property of the element we're looking for is that the element optimizes some function. The sorting network problem can be re-phrased as a test-based co-optimization problem, since an s satisfying the above property also maximizes $g : S \to \mathbb{R}$, $g(s) = \sum_{t \in T} M(s, t)$. Also note that for this problem an M-evaluation has fairly modest cost (linear in the number of compare-and-swap gates and likely measured in milliseconds), but the size of T is exponential (2^n, where n is the length of the input sequence).

A subtle difference between the resilient piping problem and the re-stated sorting networks problem is that for the latter it is really important that we find a maximum of g and we might be able to tell whether or not we have, whereas for the former we would not recognize a maximum even if we found one, so we're really looking for as high a value of g as we can find. Nonetheless, any method we can apply to the generic, black-box formulation of the resilient piping problem we can also apply to the sorting networks problem. Problems like these are the object of study in the field of co-optimization (Service and Tauritz, 2008a; Popovici et al., 2010; Popovici and Winston, 2015).

Supervised Learning Formalisms and Mapping to Co-optimization. We now turn our attention to the latter two problems, which come from the field of supervised machine learning (Abu-Mostafa et al., 2012). For the photo labeling problem, let X be the set of all possible photos (up to a certain size limit). Let $Y = \{yes, no\}$ be the set of labels, denoting presence or absence of balloons. Let $f : X \to Y$ denote the correct assignment of labels to photos, where correctness can always be judged by humans. This is often referred to as the target function to be learned. We are provided with a set of n labeled examples $\mathrm{D} = (x_i, y_i = f(x_i))_{i=1..n} \subset X \times Y$ and the goal is to find the full f, or, more precisely, to find a computer program that implements f. Such a program is often referred to as a classifier or predictor, the examples D are also referred to as data points or simply as data or a data set, and the problem solving goal is also referred to as (binary) classification.

We can map this problem to a test-based co-optimization problem as follows. Let S be the set of all computer programs that take as input a photo $x \in X$ and output a label $y \in Y$. For any $s \in S$ and any $x \in X$ we denote by $s(x)$ the output of program s for input photo x. Note $|S|$ is larger than $|Y^X|$, since multiple programs can implement the same functionality. Let $T = X$ (i.e., photos constitute tests) and define $M : S \times T \to \mathbb{R}$ via $M(s, t) = \delta(s(t), f(t))$, where δ is the Kronecker delta function or some other function measuring the agreement between $s(t)$ and $f(t)$. Then the photo labeling goal is to find a program $s \in S$ such that $M(s, t) = 1$ for all $t \in T$. This is essentially the same goal as

for sorting network design and can be restated as maxisum, which is useful since for this particular application we do not actually expect to find a perfect program. Equivalently and more commonly, M is defined as $1 - \delta(s(t), f(t))$ and called error, which is then to be minimized.

The problem of predicting secondary protein structure is similar: X is the set of all possible primary-structure amino-acid sequences, Y is the set of all possible secondary-structure sequences and f represents the real-world mapping of primary to secondary structure sequences. When mapping to co-optimization, S is the set of all computer programs that take as input a primary-structure amino-acid sequence and output a secondary-structure sequence, $T = X$ and $M(s, t)$ is defined to measure agreement between $s(t)$, the secondary-structure sequence output by program s for amino-acid sequence t, and the true secondary-structure sequence given by $f(t)$; agreement could be binary (1 if sequences are identical, 0 otherwise) or it could give partial credit for those positions that do match between the two sequences. The problem-solving goal, especially for the second choice of M, is once again maxisum.

Comparison. While these formalisms show us how we can express the problem-solving goals of typical supervised learning tasks in a similar fashion to those of co-optimization, they also expose subtle but important differences in the information available to any algorithm attempting to solve such problems. One such difference is the access we have to the function M. For generic co-optimization problems like resilient piping and sorting network design, we can evaluate M for any $\langle s, t \rangle$, thus the main concern is with the cost of each such evaluation and, consequently, with the number of evaluations that can be afforded and how good a potential solution (as judged via g) we can find given such an evaluation budget. For supervised learning problems like photo labelling and protein structure prediction, we can only get at the value of M for pairs $\langle s, x_i \rangle_{i \in 1..n}$ corresponding to the data set D (which is provided as input to the algorithm for free) and the main concern is how good a potential solution we can find given only those n specific labeled examples. Note how n does not impose a hard limit on the number of $\langle s, x_i \rangle$ pairs for which we can compute M, but see more on cost in Section 2.3.1.

Having made this difference explicit, we can now see that the related field of supervised active learning (Settles, 2012) concerns itself with problems whose information availability lies on a spectrum between the above two extremes.

2.1.4 Active Learning Problems

For instance, consider the following slight variation of the photo labeling problem. The goal is the same as before, but instead of a set of labeled photos we have a much larger set of unlabeled photos (obtained, for instance, via internet crawling). With the advent of the Amazon Mechanical Turk (Amazon, 2005) obtaining a label for a photo can be relatively cheap and prompt, though it is not completely free, and the time required to get it is not computational in nature, but rather depends on human availability (Maji, 2011). Both to save money and to speed up the process, it would be beneficial if we could ask for a label only for those images most useful for the learning task. This is referred to as a pool-based active learning problem. Formally, we are given X the set of all possible photos, Y the set

Symbol	Description
S	Space of potential solutions.
T	Space of tests.
$V \subseteq \mathbb{R}$	Space of metric values.
$M : S \times T \to V$	Interaction metric; $M(s,t)$ is the result of evaluating potential solution s with test t; it can be completely black box or based on labels in supervised learning.
$g : S \to \mathbb{R}$	Quality function that is the target of optimization; $g(s)$ gives quality of potential solution $s \in S$; examples include taking the average, sum or minimum of M over T.
X	Space of inputs, from which data is sampled.
Y	Space of outputs (labels for inputs).
$f : X \to Y$	Target function from Y^X, labeling inputs; it defines M when viewing supervised learning as co-optimization.
$\mathrm{D} \subset X \times Y$ $\mathrm{D} = (x_i, f(x_i))_{i=1..n}$	Data set of size n from which f must be inferred.
$\mathcal{H}$	In supervised learning, set of hypotheses considered (also called a model). In co-optimization, set of potential solutions actually being searched. Its mapping to Y^X or, respectively, S is typically not one-to-one.

Table 1: Notation summary for Section 2.1 (Problems) and Section 2.2 (Algorithms).

of labels, a strict subset $X' = (x_i)_{i=1..n}$ of X and access to f in black-box fashion with the restriction that we can only query f for $x_i \in X'$.

If mapping this to co-optimization, the restriction on access to f means in turn that we can only get the value of M for pairs $\langle s, x_j \rangle$ for those $x_j \in X'$ for which we have obtained a label. However, while in generic co-optimization one metric evaluation is the basic unit of cost, in active learning if we perform multiple $M(s, x_j)$ evaluations with different s but the same x_j we incur the cost of obtaining the label $f(x_j)$ only once overall and the costs for $s(x_j)$ and $\delta(s(x_j), f(x_j))$ multiple times, once for each s.

This is true for two additional flavours of active learning. One further variation of the above problem consists of the case when we are not provided with a set $X' \subset X$, but with a stream of elements from X: each element is available only for a limited time, during which the algorithm can ask for a label for that element, thus it must decide quickly whether or not to do so—and we cannot ask for all labels. This setup is of interest for domains such as part-of-speech tagging (Dagan and Engelson, 1995).

Let us also revisit the protein structure problem and relax the requirement that we use only data previously generated by lab experiments. Rather, we are allowed to perform further experiments for any existing amino-acid chain to determine its secondary structure, though for any such experiment we incur costs both in money and in time required to perform the experiment. Note that for this setting f is truly black-box and we have no restrictions as to for which examples $x \in X$ we can obtain the label $f(x)$—examples

can be generated de-novo. Consequently, if we model this as co-optimization, we have no restrictions in terms of which pairs $\langle s, x \rangle$ we can compute M for. Thus, this problem is the most similar to the co-optimization problems of resilient piping design and sorting network design. The difference is that for those problems it was M directly that was given in black-box fashion, whereas for the de-novo active learning problem we have that M is defined through f and it is f that we have black-box access to. The impact of this difference will be discussed in Section 3.

For all active learning problems there may be a budget constraining how many labels the algorithm can obtain.

2.2 Algorithms

We now turn to discussing the various types of algorithms that have historically been used to approach supervised learning and co-optimization problems. Even state-of-the art algorithms tend to fall under these broad types or employ as components the core approaches described here and hybrids thereof; when they don't, this framework should make it easy to express how they differ. Our categorization is mainly driven by the way algorithms make use of the information available to them. Sections 2.2.1 and 2.2.2 will read familiar to the machine learning researcher; we review some basic notions so that: 1) whenever possible, we can directly describe them using co-optimization terminology; and 2) we can refer back and contrast with them the co-optimization-specific notions we introduce later in Section 2.2.3 and thus build towards the vocabulary mapping in Table 3.

2.2.1 Supervised Learning Algorithms

Recall from the formalization of supervised learning in Section 2.1.2 that we are looking for an element $s \in S$ that minimizes $g(s) = \sum_{t \in T} M(s,t) = \sum_{t \in T}(1 - \delta(s(t), f(t)))$, but we only have access to the value of $f(t)$ for $t \in (x_i)_{i=1..n}$ from the data D. Thus, a first issue to be addressed is that we *cannot* actually compute $g(s)$ for any s. Supervised learning algorithms deal with this by working with restrictions of the sum inside g to subsets of the data. State-of-the-art methods employ meta-approaches such as regularization or cross-validation (discussed at the end of this section), but at their core they incorporate and expand upon the following procedure. Data is randomly partitioned into a training set and a test set. An $s \in S$ is identified that minimizes g^{tr}, the variant of g obtained by summing $M(s,t)$ only over the t in the training set, also called in-sample error or training error. It is hoped that $g^{tr}(s)$ will correlate with the actual value of $g(s)$, though of course the former is an optimistically-biased estimator of the latter. For that reason, for the best s *only*, an unbiased estimator is computed by summing $M(s,t)$ only over the t in the test set. Overfitting or over-training is said to have occurred if an s was chosen that had low training error, but high test set error. The term generalization error is used to refer either to the actual value of $g(s)$ or to the out-of-sample error defined as the sum of $M(s,t)$ only over the $t \in T \setminus (x_i)_{i=1..n}$ (the symbol '\' stands for set difference).

In minimizing g^{tr}, the next issue concerns the set S. For supervised learning problems, the true space of all potential solutions S is the set of all computer programs implementing a function from X to Y. In practice though, supervised learning algorithms only consider a subset $\mathcal{H} \subset S$ of programs, but one with a structure that facilitates finding a program

that implements f or a good approximation of f. The elements of $\mathcal{H}$ are referred to as hypotheses. For instance, $\mathcal{H}$ could be the set of neural networks of a particular size and shape, or the set of decision trees of a certain depth, etc. Such a set $\mathcal{H}$ is typically called a model.[5] There may be multiple elements of $\mathcal{H}$ that implement the same function from X to Y and there may be functions from X to Y that no element of $\mathcal{H}$ implements (i.e., the mapping from $\mathcal{H}$ to Y^X is not one-to-one). The choice of $\mathcal{H}$ (and an algorithm to search it) has a strong bearing on the expected generalization error and carries with it a so-called bias-variance tradeoff (Geman et al., 1992), where bias and variance represent two different sources of said error that are difficult to minimize simultaneously. Bias is the average, over possible data sets, of the distance between the target function and the best hypothesis found by the algorithm for that data set. The variance measures how much this best hypothesis varies with the data set. Larger, more *complex* sets $\mathcal{H}$ tend to have lower bias but higher variance and be prone to high generalization error via overfitting. Smaller and simpler sets $\mathcal{H}$ tend to have lower variance but be prone to high generalization error due to high bias (underfitting).

For many supervised learning approaches, the choice of $\mathcal{H}$ requires that the data is either directly given as or can be transformed into tabular form, i.e., the set X can be written as $X^{\mathrm{f}} = X_1^{\mathrm{f}} \times X_2^{\mathrm{f}} \times ... \times X_k^{\mathrm{f}}$ where the sets X_i^{f} are numeric, ordinal or categorical in nature (the superscript 'f' stands for 'feature'). These sets represent so-called variables or features or (in statistics) covariates. In certain domains the elements of X in fact have richer structure, which cannot be represented in tabular form. When mapping such an X into an X^{f} there will inevitably be information loss; additionally, the choice of mapping could have an impact on the difficulty of the problem.

At the core of even sophisticated machine learning methods there are two main ways in which the structure of $\mathcal{H}$ helps locate an element of $\mathcal{H}$ that minimizes g^{tr} and they differ in the way they make use of the training data: constructive versus exploratory.[6]

Constructive approaches rely on a mapping of X to X^{f} and define $\mathcal{H}$ in such a way that the training data can be used to directly construct a *single* element $s \in \mathcal{H}$ with a good g^{tr}-value (low training error). From a co-optimization standpoint, this constructive process used during training does not involve any $M(s,t)$ evaluations. For instance, for decision trees (Rokach and Maimon, 2014) this is done by repeatedly partitioning the examples based on one feature X_i^{f} at a time, by using the relationship between the feature's values for the training data points and the respective labels of those data points; a new node is added to the tree with each partition until a maximum depth is reached. For support vector machines (SVMs, Cristianini and Shawe-Taylor, 2000), $\mathcal{H}$ is a set of polynomials which, along with some constraints, makes the problem of minimizing training error into a quadratic programming problem.

Exploratory approaches work by computing the training error g^{tr} for *multiple* elements of $\mathcal{H}$ in order to find one with a good value. In co-optimization lingo, they perform a series of $M(s,t)$ evaluations for various $s \in \mathcal{H}$ and various t in the training set portion of $(x_i)_{i=1..n}$ in order to decide which s to pick. This search process can be strongly or loosely guided.

5. In some works, the word 'model' is used to mean a single hypothesis. To avoid confusion, in this paper 'model' only refers to a set of hypotheses $\mathcal{H}$.
6. These have also been referred by De Jong (2006) as bottom-up and, respectively, top-down.

One way to provide strong guidance to the search is to map X to X^{f} and define the space $\mathcal{H}$ so that the error g^{tr} has a closed form with easily computable derivative that allows gradient descent techniques to be used for minimization. Neural networks are an example of this (Rumelhart et al., 1986): the X_i^{f} are mapped to the input nodes of the network, $\mathcal{H}$ is the set of all possible weight combinations for the edges and the error function is linear in these weights. Forward propagation constitutes performing one M-evaluation.[7] Back-propagation of the error to update the weights via gradient descent generates a new network $s' \in \mathcal{H}$. Using a single example for each back-propagation step generates a series of evaluations of the type $M(s_1, t^1), M(s_2, t^2), M(s_3, t^3), \ldots$. Using multiple examples and aggregating the error before propagating it back leads to a series of evaluations of the type $M(s_1, t_1^1), M(s_1, t_2^1), \ldots, M(s_1, t_i^1), M(s_2, t_1^2), M(s_2, t_2^2), \ldots, M(s_2, t_j^2),\ M(s_3, t_1^3), \ldots$. In either case, the s with best g^{tr}-value (often last in the series) is output. Choosing the order in which to use the examples is typically done stochastically and orderings resulting from different random seeds lead to different exploratory paths through $\mathcal{H}$.

An example of a loosely-guided exploratory approach is the application of genetic programming (Koza et al., 2006) to supervised learning. Genetic programming is a set of techniques from the field of evolutionary computation (De Jong, 2006) that draw inspiration from the principle of evolution by natural selection to search for computer programs achieving some task. Programs can take various shapes, such as rule-sets in the field of learning classifier systems (Lanzi et al., 2003), expression trees in symbolic regression (Icke and Bongard, 2013), even neural networks with variable structure in neuro-evolution (Floreano et al., 2008), etc. The set X, representing inputs to the programs, need not necessarily take tabular X^{f} form and the space $\mathcal{H}$ of programs is not constrained to lead to a closed-form error function. Instead, $\mathcal{H}$ is endowed with a topology such that neighboring points (programs) in $\mathcal{H}$ have close error values. This topology is induced by so-called genetic operators that generate new programs in $\mathcal{H}$ from previous programs, in a manner inspired by how mutation and crossover work on DNA. A program $s \in \mathcal{H}$ is evaluated by computing $M(s, t)$ for some (usually all) t from the training-set portion of $(x_i)_{i=1..n}$ and aggregating these measurements into one value, often referred to as fitness. Programs with good fitness (low error) are selected and operators are applied to them to produce a population of new programs; the process then repeats for multiple iterations called generations. These techniques are slower at climbing any given local optima than pure-gradient methods—since they essentially have to indirectly estimate the gradient by performing $M(s, t)$ evaluations—but better at avoiding getting stuck in one. Note also that they are general optimization techniques applicable to other tasks besides supervised learning and therefore not tuned to it.

Meta-approaches. Both exploratory and constructive approaches need to be concerned about over-/under-fitting. One heuristic is to try out sets $\mathcal{H}$ of different complexities: for example, put a hard constraint on the degree of the polynomials or the depth of the trees considered, then investigate what happens when varying the value of the constraint. Another heuristic is to use "soft" constraints, i.e., allow an all-encompassing $\mathcal{H}$, but include in the training error g^{tr} a penalty term for the complexity of individual hypotheses in that $\mathcal{H}$; examples include: allowing trees of high maximum depth, but including a penalty term that grows with the depth; or allowing polynomials of high degree, but including a penalty

7. Feeding example t through network s computes $s(t)$ and determining disagreement between the network's output and the example's label computes $M(s, t) = 1 - \delta(s(t), f(t))$.

term that is low when most weights are close to 0 and high otherwise. This latter heuristic is generally referred to as regularization; in genetic programming, penalizing tree depth is called parsimony pressure. Different levels must be tried out for the relative contribution of the penalty term versus the main error term.

In the above, the expression 'try out' refers to the fact that the overall algorithm applies multiple basic methods (either exploratory or constructive), or applies a parameterized basic method multiple times with different parameter values, then selects one of the variants for producing the final output—a process known as model selection. To perform the selection, (semi)unbiased empirical estimates are computed for the output of each variant by performing additional $M(s,t)$ evaluations with tests not used during the training of that variant—a process known as (cross-)validation (Refaeilzadeh et al., 2009; Geisser, 1975; Stone, 1974). Thus, during this process, what was originally designated as a training set gets further split (possibly multiple times for cross-validation) into an *actual* training set and a validation set. The term holdout set has also been used in the literature, but with inconsistent meaning, referring sometimes to a validation set and sometimes to a true test set. The test set is still to be used only once to get an unbiased estimate for the quality of the final output. Such uses of estimates for output selection and performance evaluation are relevant for co-optimization algorithms as well, as discussed in sections 2.2.3 and 2.3.2.

2.2.2 Active Learning Algorithms

Typical approaches to active learning treat the problem as an extension of supervised learning. As such, they use a query strategy to decide for which points $x \in X$ to obtain a label, then apply one of the supervised learning methods described above on the labeled data accumulated so far, then re-iterate these two steps. Some query strategies are generic and could be used with any supervised learning approach, whereas others are specific to the supervised learning algorithm used (Settles, 2012). Whether the core supervised learning approach is constructive or exploratory, *the query strategy of active learning algorithms can be thought of as exploring the set X.* The resemblance of active learning to co-optimization suggests there may be approaches that are more direct than the standard iterative one. We will revisit this at the end of Section 3.

2.2.3 Co-optimization Algorithms

Because in co-optimization it is the metric M that is given in black-box fashion (as opposed to an f that further defines M), it is not obvious if or how one could find a good element of S through a constructive approach. Therefore, co-optimization algorithms (Service and Tauritz, 2008a) are exploratory by necessity and involve adaptations of stochastic search and optimization heuristics (Luke, 2013) like simulated annealing (SA), hill-climbing (HC) and evolutionary algorithms (EAs)—the genetic programming described in Section 2.2.1 being an example of the latter.

Note that in this case the search space S does not necessarily contain computer programs, but possibly other kinds of objects or structures (e.g., subsets of valves in the case of resilient piping design or, often, simply vectors of parameter values characterizing a structure or design). Recall that supervised learning algorithms actually search a hypothesis set $\mathcal{H}$ that does not have a one-to-one mapping with Y^X; similarly, in co-optimization, the

way the space S is represented in the computer often means that what's actually being explored by the algorithm is a set other than S and whose mapping to S is not one-to-one. To limit notation, we'll denote this set by the same symbol $\mathcal{H}$. In the field of evolutionary computation (De Jong, 2006), the choice of $\mathcal{H}$ is called a representation and the mapping from $\mathcal{H}$ to S is called the genotype-to-phenotype mapping. One still needs to consider the tradeoff between the ability of $\mathcal{H}$ to represent many/all the elements of S and the ease of searching $\mathcal{H}$ (which may decrease as the size of $\mathcal{H}$ increases). This can be considered a counterpart concept to supervised learning's bias-variance tradeoff.

As in supervised learning, an important issue is the fact that we might not be able to compute the actual value of the quality function g that we are to optimize, because $g(s)$ is defined as the sum of $M(s,t)$ for *all* $t \in T$ and T is very large and/or M is very expensive.

In contrast to supervised learning, there are no restrictions as to which elements $t \in T$ we can use in $M(s,t)$ evaluations. Consequently, *co-optimization algorithms explore not just the space S but also the space T, as well as the cross-product of the two.* Thus, a co-optimization algorithm can be formalized as having two components: 1) the exploration mechanism,[8] deciding which interaction between a potential solution $s \in S$ and a test $t \in T$ should be evaluated next via the metric M, given the sequence of interactions previously evaluated and their M-values, also referred to as the history[9]; and 2) the output mechanism,[10] deciding, based on such a history, which potential solution s to output. This formalism can be used to describe exploratory approaches to supervised learning as well; and most meta approaches perform some kind of output selection even when incorporating constructive components.

Exploration mechanisms. A basic but sometimes successful approach is to use any of the previously-mentioned stochastic search heuristics for exploring S, by evaluating any considered $s \in S$ with a random sample of tests $t \in T$ and using as fitness an aggregation of the resulting M-values, a placeholder for the true quality of s. This sample may contain a portion that is fixed throughout the entire algorithm, a portion that is regenerated for each iteration and a portion that is regenerated for each s.

Another popular approach is to apply the evolutionary metaphor to both spaces. The resulting *coevolutionary* algorithms (Popovici et al., 2010) maintain a population of potential solutions $s \in S$ and a population of tests $t \in T$. Elements in the former population are paired with elements in the latter and the resulting interactions are evaluated via the metric M. Various interaction methods can be used to determine who is paired (interacts) with whom (Popovici, 2006). For the maxiavg/maxisum solution concepts, fitness is typically assigned to each element s in the S-population by averaging $M(s,t)$ for all the interactions that particular s was involved in; such fitness represents a subjective quality, internal to the algorithm. Much research has gone into how to assign fitness to the elements of T, leading to two main paradigms: 1) if high values of M are good for the elements of S, then reward tests in T for low values of M (i.e., for being *difficult* tests)—a paradigm traditionally

8. This has also been called just an *algorithm* (Wolpert and Macready, 2005; Service and Tauritz, 2008b) or a *search heuristic* (Service and Tauritz, 2008b; Service, 2009a,b; Popovici and De Jong, 2009; Popovici et al., 2011).
9. This has also been called a *sample* (Wolpert and Macready, 2005) or simply a *sequence* (Service and Tauritz, 2008b).
10. This has also been called a *champion-selection rule* (Wolpert and Macready, 2005), a *champion selection function* (Service and Tauritz, 2008b), a *candidate selection function* (Service and Tauritz, 2008b; Service, 2009a,b) or *output selection* (Popovici and De Jong, 2009; Popovici et al., 2011).

referred to as *competitive* coevolution (Angeline and Pollack, 1993; Popovici et al., 2010); and 2) reward elements of T for their ability to differentiate between elements of S, that is for being *informative* tests (Bucci, 2007). In addition to the tests in the current population, archives of hard/informative tests can be kept and used in evaluations (Rosin and Belew, 1997; de Jong, 2005). Co-optimization problems emerged as a formalization of the kinds of problems coevolutionary algorithms were being applied to.

Both approaches (random sampling of tests versus coevolution) typically use the same *number* of tests to evaluate all potential solutions considered; these potential solutions are called *partially evaluated*, as said number of tests is usually much smaller than $|T|$.

Output mechanisms. A typical approach is *greedy*, which outputs the partially-evaluated potential solution with the best average of M-values over the tests it was evaluated with. It is also common to maintain archives of promising potential solutions (as identified by the greedy method) and then do another round of M-evaluation for them, with either freshly sampled or similarly archived tests, and pick the output based on the resulting *empirical* g-estimates (Panait and Luke, 2002); this is similar to the process of validation for model selection used in supervised learning. Newer work (Wolpert and Macready, 2005) has introduced optimal output mechanisms whose definition makes use of *theoretical* (Bayes) estimates of g consisting of aggregations (e.g., averages) over all possible values that yet-unperformed metric evaluations could result in. Interestingly—and unlike greedy—such output mechanisms even consider outputting *completely-unevaluated* potential solutions, i.e., ones for which the algorithm hasn't yet performed any M-evaluations, if their Bayes estimates are better. We will discuss these in more detail in sections 2.3.2 and 3.

Estimates of g, whether empirical or theoretical, can also be used to judge the quality of the outputted potential solution. In particular, using freshly sampled tests to obtain an unbiased empirical estimate (Chong et al., 2008) is similar to the way a test set is used in supervised learning; such an estimate represents an objective quality and is external to the algorithm itself. As with over/under-training in supervised learning, this is necessary because a disconnect can occur between internal and external quality measurements (Popovici and De Jong, 2005; de Jong, 2007).

2.3 Cost and Performance

In computer science the terms cost and performance are sometimes used interchangeably, especially when algorithms are acceptable only if they produce correct output and differentiating between them is done by measuring their execution speed. In supervised learning and co-optimization the notion of correctness is replaced by as-good-as-possible, thus we have to differentiate between two terms. We use the word 'performance' to refer to quantities having to do with the quality of potential solutions outputted by the algorithm. We use the word 'cost' to refer to time or money. Performance and cost are typically inter-related, since the performance of an algorithm may vary with the cost expended. We start by formalizing the various types of cost encountered in co-optimization and supervised learning, and hint at possible tradeoffs between different types. Then we review ways of assessing performance, which may itself incur costs.

The study in Section 3 relies upon and provides examples for several of the notions of cost and performance introduced here.

2.3.1 Cost

In general, costs can take several high-level forms: 1) computational time, that is the time required by a computer to perform a calculation; it can vary from one hardware to another, but may be expressible in terms of basic operations via the big-O notation of traditional computational complexity—this tends to be more difficult to achieve when the calculation is in fact a complex simulation; 2) real-world time, for instance time required to run a physical or chemical experiment, or time required to get a human to provide a label; and 3) money, for instance to pay for a lab, materials and people.

Algorithms for supervised learning and co-optimization problems can incur these kinds of costs while carrying out various activities. Some are directly driven by the problem specification: obtaining labels for active learning and evaluating $M(s,t)$ for co-optimization; all algorithms attacking these problems will necessarily incur the associated costs. Other cost-incurring activities are dependent on the specific choice of algorithm and include: exploration decisions, construction, output selection. Last but not least, in supervised learning and co-optimization evaluating the performance of any algorithm often incurs additional costs. Table 2 provides an overview in concise form.

Labeling costs are relevant only for *active learning* and are discussed extensively in Section 2.1.4, so we only re-summarize them on the top of Table 2. The only additional note is that while for all the problem examples in Section 2.1.4 labeling costs were in the form of real-world time or money, they can occasionally be measured in computational time: for instance, trying to find a quick-to-compute approximation for a very-slow-to-compute simulation (Sacks et al., 1989), such as one involving computational fluid dynamics or a weather model.

Metric-evaluation costs are part of the problem specification in *co-optimization*, as detailed in Section 2.1.1; their type and magnitude can vary, but even when computational and small it is a concern due to the very large sizes of S and T (often combinatorial or exponential) requiring a lot of M-evaluations to find good potential solutions.

Metric-evaluation costs are also relevant in *supervised learning*. They are a key concern for all exploratory algorithms, since exploration decisions are made based on the outcome of such evaluations. Additionally, incurring $M(s,t)$ evaluation costs may also be necessary for two other algorithmic activities discussed later in this section, namely output selection and performance assessment, which are relevant even for constructive approaches to supervised learning. The cost of evaluating $M(s,t)$ is always computational time and has two components: 1) cost of computing $s(t)$, that is the computational time required to run example t through program s (e.g., forward-propagation through a neural network, or top-down propagation through a decision tree, or evaluation of a polynomial for a support vector machine); and 2) cost of computing the agreement $\delta(s(t), f(t))$. The latter cost is typically small, e.g., constant when simply comparing photo labels or linear in the number of amino-acids when comparing protein structures. The cost of computing $s(t)$ varies with the size of individual hypotheses, e.g., number of edges in a neural network, levels in a decision tree (and trees in a random forest, Ho, 1995) or terms in a polynomial; it has grown in recent years for models such as deep learning's convolutional neural networks with millions of connections (LeCun et al., 2015). Using models with a low cost of computing $s(t)$ is desirable because this cost is also always incurred once a solution is deployed in the real world.

			Supervised Learning				Test-based Co-optimization
			Passive	Active			
				Pool-based	Stream-based	De-novo	
Labeling costs			Free ("gift" from someone else)	Pay for label Type: usually money or time, occasionally computational. Magnitude: varies widely across domains			Not applicable (N/A), there are no labels
Label availability		*How many* ...	Restricted by problem specification	Restricted mainly by labeling budget, then by pb. spec.	Restricted by labeling budget		
		... and *which* examples we can get labels for:	Restricted			Unrestricted	
Metric-evaluation (M(*s*,*t*)) costs			Type: always computational, two components *s(t)* and *δ(s(t), f(t))*; exclusive of cost of obtaining label *f(t)* Incurred during training by exploratory algorithms Incurred during testing and validation by all algorithms Cost of s(t) always incurred in the real world by deployed solution Magnitude: *s(t)* varies with hypothesis size, *δ* with label size				Main cost of interest, all algorithms incur it Type: usually computational, sometimes real-world time and/or money Magnitude: varies widely, a concern even if small due to extremely large *S* and *T*
			May also be incurred as part of ouput-selection and performance-assesment				
M(*s*,*t*) availability		*How many* ...	Restricted by metric-evaluation budget				
		... and *which* interactions <*s*,*t*>:	Restricted to *t* in training data	Restricted to *t* that we have choosen to obtain a label for (see label availability)			Unrestricted
Type: allways computational	**Exploration-decision costs**	... to decide which interactions to evaluate next:	Incurred by exploratory approaches only Magnitude: for backprop, at most as much as metric-evaluation costs; for others, see co-opt. box to the right				Always incurred, as all approaches are exploratory Magnitude: negligible compared to metric-evaluation costs for traditional approaches (EAs, SA, HCs), not so for newer ones (CMA-ES, EDAs, Bayes-optimal)
		... to decide which examples to label next:	N/A	Incurred by query strategy			N/A
	Construction costs		Incurred by constructive approaches Magnitude: big-O complexity as function of data size				N/A
				Incurred many times if full retraining is required			
Type: usually computational, unless we collect new data or perform new experiments	**Output-selection costs**		Incurred to determine which of the considered potential solutions to output as the algorithm's best guess for a solution				
			Magnitude: negligible when picking the output of core training components; multiple of metric-evaluation costs when performing (cross-)validation for model selection				Magnitude: negligible for greedy methods; multiple of metric-evaluation costs when using empirical estimates; negligible to prohibitive for theoretical Bayes estimates
	Performance-evaluation costs		Incurred to estimate the actual, "true" quality of the outputted potential solution				
			Magnitude: metric-evaluation costs over a single-use test set				Magnitude: multiple of metric-evaluation costs when using empirical estimates; negligible to prohibitive for theoretical Bayes estimates

Table 2: Summary of the various types of costs encountered in test-based co-optimization and supervised learning.

For *active learning*, note that the cost of obtaining the label $f(t)$ is not included in the cost of $M(s,t)$, because a label can be used for multiple computations of $M(s,t)$ for the same t but different s.

Exploration-decision costs are incurred to decide which problem-specification costs to incur next (e.g., which evaluations $M(s,t)$ an exploratory approach would perform, or, in the case of active learning, which labels a query strategy would request). They are always computational.

Back-propagation in neural networks for supervised learning is an example of an exploration decision incurring a cost: it produces a new network s for which to then evaluate $M(s,t)$ via forward propagation. The cost of one backward propagation has similar magnitude to that of one forward propagation and both vary with the size of the network, as described earlier. If multiple rounds of forward propagation are performed for each backward propagation, then, overall, exploration-decision costs are accordingly smaller than metric-evaluation costs.

Evolutionary approaches, whether used for *supervised learning* or *co-optimization*, usually incur exploration-decision costs while performing selection and breeding to produce a new generation of potential solutions, which will then be involved in $M(s,t)$ evaluations. Similarly, simulated annealing and hill-climbers incur exploration costs when generating and sampling from a local neighborhood. Historically, these costs have been considered much lower than $M(s,t)$ costs. But newer stochastic black-box optimization heuristics tend to have non-negligible exploration costs, such as: quadratic matrix computations for covariance-matrix adaptation evolutionary strategies (CMA-ES, Hansen and Ostermeier, 2001), estimating distributions for estimation of distribution algorithms (EDAs, Larranaga and Lozano, 2002), recursive optimization for optimal exploration mechanisms (Popovici and Winston, 2015; Popovici, 2017).

In *active learning*, exploration-decision costs are also incurred via the query strategy used to decide the next $x \in X$ for which to obtain a label. Such costs vary widely with the strategy (Settles, 2012) and have to be weighted against the costs of obtaining labels, since reducing the latter is the main goal of sophisticated query strategies .

Construction costs, as the name suggests, are incurred by constructive approaches to supervised learning, whether passive or active; examples include the construction of a decision tree and solving quadratic programming equations for support vector machines. They are computational costs, usually measured in basic operations and expressed via big-O complexity as a function of properties of the problem, such as size of data. They are of particular concern for the increasingly more common "big data", as well as in active learning, where these costs are incurred multiple times if full retraining is necessary each time new labels are obtained.

Output-selection costs are incurred in order to decide which potential solution to output out of the possibly multiple ones considered. This activity is closely-related to performance assessment, since the performance of an algorithm always relies in some manner on the quality of potential solutions it outputs. As discussed in Section 2.2, one distinguishing feature—and potential difficulty—of co-optimization and supervised learning problems is the inability to compute the actual quality of any potential solution for the problem instance at hand. Consequently, assessing quality of potential solutions must be done by means of estimation, which carries a cost.

In co-optimization, simple output selection methods just side-step the issue, by using the greedy heuristic of outputting the potential solution with the best average over only the tests seen so far, a highly-biased empirical estimate of g. Keeping track of this potential solution is trivial, so output selection costs have historically been considered negligible compared to metric-evaluation costs. As described in Section 2.2.3, there are also approaches that acknowledge the issue. Some compute new empirical estimates of g for a handful of potential solutions identified via the greedy method, then use these estimates (which could be unbiased or at least less biased) to select which one to output; this requires performing additional $M(s,t)$ evaluations and incurring the respective metric-evaluation costs. Others use theoretical g-estimates whose computation does not involve performing additional metric evaluations, but rather aggregating (e.g., averaging) over all possible values that such evaluations could result in. These aggregates can be computationally-prohibitive; but output selection only relies on comparisons between them, the outcome of which can sometimes be determined without actually computing the aggregates themselves (Popovici and De Jong, 2009; Popovici et al., 2011; Popovici, 2017), in which case the costs are minimal.

Unless we collect new data or perform new experiments for the purpose of additional $M(s,t)$ evaluations, output selection costs are only computational. This is the case for supervised learning as well. Here too, output selection costs can be negligible if the algorithm simply outputs the outcome of the training process for a particular method, for example, the decision tree constructed, or the final set of weights for a neural network (which is often the greedy best-so-far, given the gradient-based nature of exploration via back-propagation). More commonly though, a model is selected first via (cross-)validation, as described in Section 2.2.1. To compute the empirical estimates involved in validation, $M(s,t)$ costs are incurred by performing additional metric evaluations for the outputs of the models to be selected amongst.

2.3.2 Performance

By performance of an algorithm we mean the quality of the algorithm's outputted potential solutions with respect to the problem solving goal. But because in supervised learning and often in co-optimization as well exact quality is not actually computable, assessing algorithmic performance is an uncertainty-laced, more difficult task in these fields compared to traditional function optimization.

Perhaps the simplest performance question one can answer—and one that is quite important to answer in practice—is: on the problem instance at hand, what is the quality of the potential solution that constitutes our algorithm's final choice as *the solution* to be deployed in the real-world?

This is a quality-estimation question, so it can be answered the same way as when part of the output selection process (see previous section). But subtle care must be taken. Computing unbiased empirical estimates (by performing additional $M(s,t)$ evaluations and incurring the associated costs) must be done with fresh tests in co-optimization (Chong et al., 2008) or, in supervised learning, with labeled data points (a test set) not used for any other purpose.[11] Moreover, the resulting estimate *should not* be used for any further

11. Note the different usage of the word 'test' in the two parts of this sentence.

tweaking of the chosen potential solution, or else it ceases to be an unbiased estimate; in other words, a true test set is always single-use—but see the work of Dwork et al. (2015) for a novel technique where the bias resulting from reuse is small and quantifiable as a function of the amount of reuse.

We may also wish to know how performance of an algorithm varies as a function of given resources/cost expended, so we can determine how we may be able to boost performance. Ideally, such characterizations would take into account *all costs* listed in the previous section. In reality, different approaches have focused on different aspects of the performance-cost dependence.

In passive supervised learning, the main concern is with performance as a function of resources, in the form of size of available data. However, the cost of acquiring data is not counted as a cost incurred by the learning algorithm. Indirectly, this dependency on data size does describe a relationship between performance and algorithmic cost, because making use of more data takes more time, whether it is done in constructive or exploratory fashion. Because of this, a whole subfield of supervised learning concerns itself with instance selection, that is determining a subset of the labeled examples that is small, yet using it would result in similar output quality as using the whole set (Reinartz, 2002). In co-optimization, the main concern is with performance as a function of metric-evaluation costs. In active learning, the main concern is with performance as a function of labeling costs. Unlike passive supervised learning, the latter two fields *do* concern themselves with the dependency of performance on the cost of acquiring data, whether the data consists of values of M or values of f.

Algorithms often have one or more parameters whose values must be set. Thus another important issue is how performance varies with changes in parameters. In supervised learning, a common example of this comes from binary classification in the form of receiver operating characteristic (ROC) curves (Spackman, 1989) which plot how the error for one class changes as a function of the error for the other class while varying the value of the algorithm parameter investigated.

Being able to compare the performance obtained for two different values of one parameter of an algorithm is akin to being able to compare two different algorithms. While empirical quality estimates allow us to do that and to determine if one one algorithm is better than another for the one problem instance at hand, the question that immediately arises is whether some algorithms are better than others in a general sense (i.e., across all, or a large class of problem instances). This is important because designing or tuning a new algorithm for each problem instance is a time-consuming manual activity that might not be feasible in practice. For instance, imagine a factory that must schedule jobs on a set of machines susceptible to various breakdowns (Jensen, 2001) and must produce a new schedule every day for the jobs of that day. One cannot re-tune with such high frequency.

To answer the above question, one must make precise the meaning of 'better' and of 'general'. Theoretical Bayes estimates are intrinsically about performance on multiple problem instances. By definition, they are aggregations (e.g., expectations) over all possible hypotheses f or metrics M, assuming a certain probability distribution over such functions. Wolpert (1992, 1996) first introduced this view of algorithmic performance and argued that, to the extent that the goal of supervised learning is to generalize to *unseen* examples, the meaning of 'better' should really be based on out-of-sample error (off-training set (OTS) error)

Supervised Learning	**Co-optimization**
labeled example data point data set data	test interaction event
data size	budget
training set	history
test set	fresh tests
hypotheses classifiers predictors	potential solutions population
model variables features covariates	representation search space
bias-variance tradeoff	genotype-to-phenotype mapping
unknown target	black-box metric
forward propagation labeling	evaluation
IID error generalization error out-of-sample, off-training set (OTS) error	quality function external, objective quality
in-sample error validation error	fitness internal, subjective quality
overfitting (over-training) underfitting	internal-external quality disconnect
regularization	parsimony pressure
gradient descent back-propagation query strategy instance selection	exploration mechanism genetic operators (e.g., mutation, crossover) interaction methods generation
model selection (cross-)validation	output mechanism
supervised learning (binary) classification active learning (pool-based, stream, de novo)	solution concept (e.g., maxiavg, maxisum, maximin) test-based co-optimization problem co-search problem
no free lunch (NFL)	

Table 3: Terminology map. Terms in respective cells for the two fields are not synonymous, but represent concepts that are similar or related (e.g., back-propagation can be thought of as being a type of exploration mechanism).

rather than on the overall error g (IID error, where IID stands for independently identically distributed). For the OTS notion, he showed that there is no free lunch (NFL), meaning that for any given data set: 1) if all hypotheses consistent with that data set are equally likely (i.e., uniformly distributed), then all algorithms have the exact same aggregated performance; and 2) for any two algorithms, there are as many distributions over hypotheses for which one algorithm is aggregately-better than the other for that data set, as there are distributions where the opposite relation between algorithms holds. If by 'better' we mean better actual g value (i.e., over all tests in T), then there is free lunch, but of a rather unsatisfying type: the only aggregate-performance differences between algorithms come from their in-sample error. And while overall error is non-increasing as a function of the data set size n, out-of-sample error can actually increase with n (Wolpert, 1992). All of this holds regardless of the amount of computation expended by the algorithms.

Wolpert and Macready (2005) adapted and applied this approach to co-optimization. Since in co-optimization there is no given data set, the meaning of 'better' *must* rely on the actual value of g over all tests in T and the "fair ground" on which we compare algorithms is not a data set but the budget of $M(s,t)$ evaluations they are allowed to perform. Under this setup, there is free lunch (i.e., some algorithms are aggregately-better than others) *even if* all metrics M are equally likely.[12]

Active learning has not been studied to a similar extent from a free lunch perspective. Cursory treatment by Wolpert (1996) claims negative implications of no free lunch theorems apply just as they do for passive supervised learning. The details of this are unclear though, as there are several different NFL theorems for passive supervised learning and they are conditioned on a given data set (or its size) and are generally concerned with error outside this data set, whereas in active learning there is no data set given upfront, generating data is a key part of the algorithms. The claim is also intriguing given active learning's similarities with co-optimization and the existence of free lunch in that field. Investigating this issue is subject for future work and the treatment in Section 3 should provide a base towards that: it uses the common ground built in Section 2 to get a deeper understanding of the commonalities and differences between the two fields from a free lunch perspective.

3. Free Lunch Study

Having laid out the various concerns and approaches to measuring performance and cost, we now use them to present a proof-of-concept method transfer from co-optimization to supervised learning. Specifically, we focus on the free lunch view of performance versus cost, where quality is measured by means of theoretical Bayes estimates and metric-evaluation costs are constrained by a budget. We show how optimal *exploratory* algorithms can be designed within this framework, but also emphasize how they might incur significant costs in some of the other categories listed in Section 2.3.1, such as exploration-decision costs and output-selection costs, thus arguing for the need of a more encompassing view on costs to be developed.

12. Wolpert and Macready (2005) originally showed this for the worst-case (maximin) solution concept, but similar results have later been shown for other solution concepts (Service and Tauritz, 2008b; Service, 2009a,b), including the maxisum that most resembles supervised learning (Popovici and De Jong, 2009; Popovici, 2017).

We structure the presentation in 3 incremental steps with respect to metric-evaluation costs: in Section 3.1 we describe the small-scale case of only one or two M-evaluations in total at most one of which is remaining in the budget; in Section 3.2 we progress to histories of arbitrarily-many already-evaluated interactions, but still at most one remaining to be evaluated; finally, in Section 3.3 we discuss the most general case of arbitrarily-large spent budgets and arbitrarily-large remaining budgets. In each section we first review the co-optimization perspective and results pertaining to free lunch and optimality, then derive and contrast their counterparts for supervised learning. We focus specifically on binary classification, but many of the ideas presented would apply in a multi-class situation. Throughout, we differentiate between the nature of free lunches for output mechanisms versus exploration mechanisms. We keep the presentation as informal as possible and support it with small but concrete examples; precise mathematical definitions of all concepts involved and proofs of the results can be found in the accompanying Online Appendix A.

3.1 Small Overall Budget

In this section we look at a spent budget of 0 or 1 evaluations and a remaining budget of 1.

3.1.1 Co-optimization

Consider a domain with $S = \{a, b, c\}$, $T = \{t_1, t_2\}$, $V = \{0, 1\}$ and $M : S \times T \to V$, meaning there are 3 potential solutions, 2 tests and binary outcomes for interactions between solutions and tests. To ease comparison with supervised learning, suppose the values in V represent some kind of penalty, such that lower values are better and we're trying to find the potential solution minimizing the sum of M over the tests (i.e., a minisum solution concept rather than maxisum). The total number of interactions is $|S| \cdot |T| = 3 \cdot 2 = 6$ and therefore the total number of possible metrics M for this domain is $|V|^{|S| \cdot |T|} = 2^6 = 64$. These are shown in columns 1–7 of Table 4, one per row, numbered from 0 to 63.

Output mechanisms concern themselves with which potential solution to output given the history of interactions evaluated so far and their M-values; this history reflects the budget spent so far; the remaining budget is not a concern. If the current history is an empty sequence H = [] (i.e., we haven't yet performed any evaluations, spent budget is 0) and we have no other information, then all 64 metrics are possible and, we assume, equally likely (more on this in Section 3.4). Columns 8–10 in the table show, for each metric, the actual value of g (i.e., the sum of M over all tests) for each of the 3 potential solutions. At the bottom, the expected value of g, averaged over all metrics, shows that in the absence of any data or information all 3 potential solutions are equally promising, with an expected g-value of 1. Due to the equal likelihood of all metrics, this can also be thought of as follows: the expected M-value for a potential solution on a test it hasn't yet seen is the average of the values in V, which is 0.5; summing that over two tests gives $2 \cdot 0.5 = 1$.

Consider now the situation where an algorithm has evaluated the interaction $\langle a, t_1 \rangle$ and observed value $M(a, t_1) = 1$, so now the history is H = $[\langle \langle a, t_1 \rangle, 1 \rangle]$ (spent budget is 1). What should the algorithm output as the most promising potential solution? As we can see from the columns 11–13 in Table 4, metrics from 0 to 31 are no longer possible, since they have $M(a, t_1) = 0$. Averaging g only over the remaining metrics (which we say are *consistent* with the history) yields a different expected value for potential solution a (1.5)

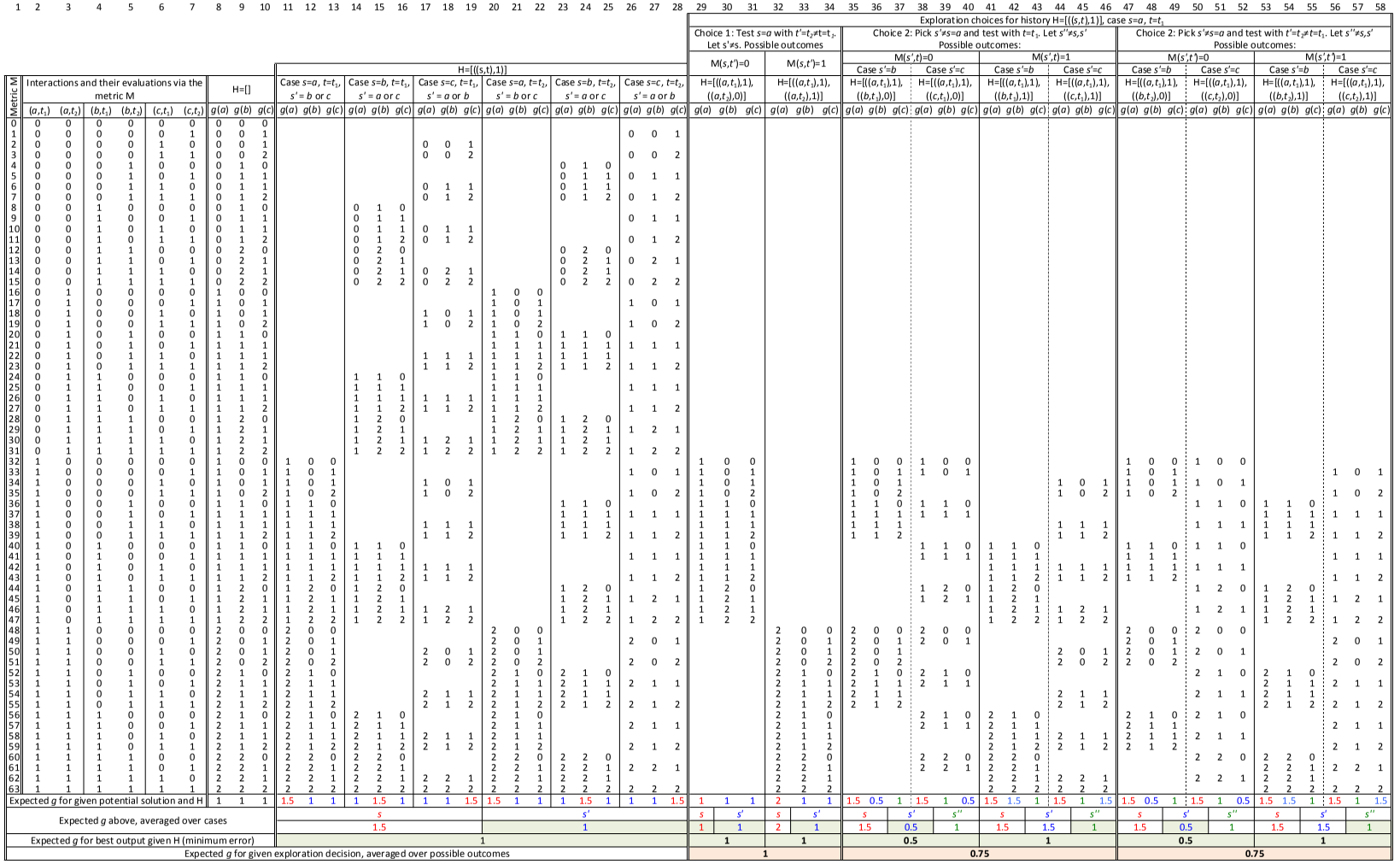


Table 4: Examples of co-optimization free lunch in domain with $S = \{a, b, c\}$, $T = \{t_1, t_2\}$, $V = \{0, 1\}$. Output mechanism free lunch (columns 11–28): for history $\mathrm{H} = [\langle\langle s, t\rangle, 1\rangle]$ with $s \in \{a, b, c\}$ and $t \in \{t_1, t_2\}$, outputting a new potential solution $s' \neq s$ yields strictly better expected performance (1) than outputting s (1.5). Exploration mechanism free lunch (right half of table): for history $\mathrm{H} = [\langle\langle s, t\rangle, 1\rangle]$, instantiated with $s = a$ and $t = t_1$, testing a new potential solution $s' \neq s$ with either the same test t or a new test $t' \neq t$ yields strictly better expected performance (0.75) than further testing s (1). Font style indicates which values were averaged to compute an expectation, green cell shading indicates where the minimum value is coming from and orange cell shading highlights the expected performance for each exploration choice.

than for b or c (1). Given these estimates, it follows that outputting one of b or c is the strictly best choice from the perspective of the algorithm's aggregated performance. This is an example of existence of *output mechanism free lunch* in co-optimization. Note this does not mean that outputting b or c is the best thing to do for each and every metric; for instance, if M happens to be M_{47}, outputting a would be the better choice.

Often, an algorithm might not be able to pick a specific potential solution and/or a specific test, rather it picks at random from a set of previously unexplored possibilities. Consider for instance a history $\mathrm{H} = [\langle\langle s, t\rangle, 1\rangle]$, where s is any of a, b or c and t is either t_1 or t_2, but we do not know which. There are 6 possible cases, depicted in columns 11–28 of Table 4. In each of these cases, the expected g-value of s is 1.5, so overall the expectation is also 1.5 and the expected g-value of any $s' \neq s$ is 1, thus outputting s' is the better choice. These values can also be arrived at with the kind of reasoning described a few paragraphs back: for s, the expected g-value is the sum of the actual M-value observed for t, which is 1, plus the expected value of M on the remaining test, 0.5; for $s' \neq s$, since it hasn't seen any tests yet, the expected g-value is the same as before: we simply multiply the expected M-value on a given test, 0.5, by the total number of tests, 2.

Exploration mechanisms concern themselves with which interaction to evaluate next given the history; this decision may be influenced by the remaining budget of M-evaluations. Suppose that the algorithm having produced history $\mathrm{H} = [\langle\langle s, t\rangle, 1\rangle]$ (spent budget of 1) is now allowed one additional evaluation (remaining budget of 1). It is the job of the exploration mechanism to decide which of the remaining 5 interactions to run through the metric M. And to determine the performance expected to result from a given interaction choice (exploration decision), we must aggregate over the potential outcomes of evaluating that interaction (0 or 1) the expected value of g for the best output for the ensuing lengthened history for that outcome. The exploration decision can be thought of as having two parts: choosing which potential solution to evaluate and choosing which test to evaluate it with. If s is chosen, then there is only one test it hasn't seen yet, $T \setminus \{t\}$. If an $s' \neq s$ is chosen, then there is a choice between evaluating s' with t or with the other test, $t' \neq t$. These choices are shown in columns 29–58 of Table 4, instantiated only with with $s = a$ and $t = t_1$, due to space constraints; the other instantiations yield the same kind of results. We can see that once an interaction is chosen and its M-value observed, the number of consistent metrics is further cut in half: there are as many metrics producing an M-value of 1 as of 0. Thus aggregating over outcomes is simply by averaging. And within each outcome, for each potential solution we average over the consistent metrics the g-value for that solution, then choose the potential solution with the best average. We see that the choice of evaluating an $s' \neq s$, regardless of whether with test t or t', leads to a better expected g-value (0.75) than evaluating s again with the remaining test t' (expected g-value 1). This is an example of existence of *exploration mechanism free lunch* in co-optimization.

3.1.2 Supervised Learning

Consider now a small binary classification domain with inputs $X = \{t_1, t_2, t_3\}$ and outputs $Y = \{y, n\}$ (short-hand for *yes*, *no*). There are a total of $|Y|^{|X|} = 2^3 = 8$ possible functions $f : X \to Y$. These are shown on the top left side of Table 5 (columns 1–4), one per row, numbered from 0 to 7. The problem is to guess f given some data D in the form

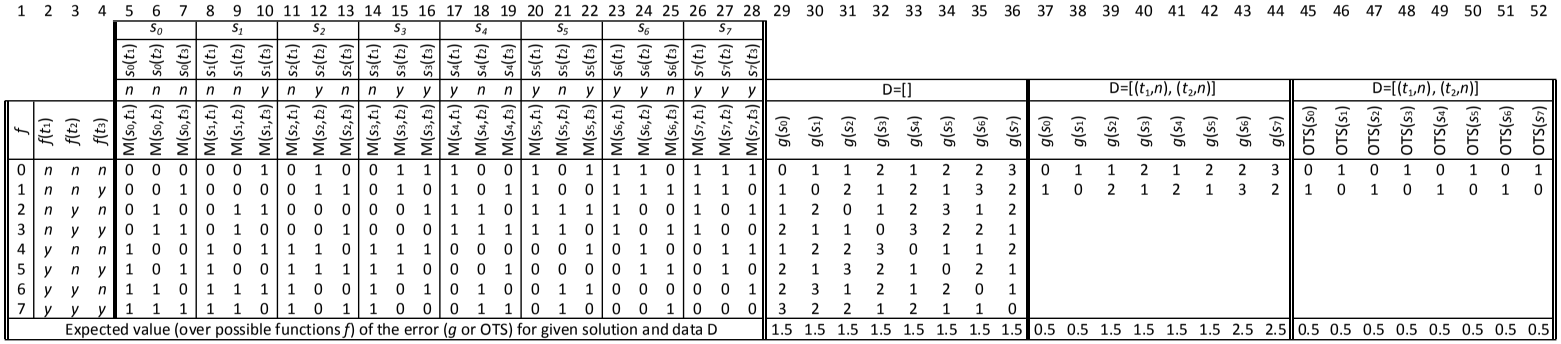
of pairs from $X \times Y$. Columns 5–28 in Table 5 show how this problem can be mapped to a co-optimization problem. We've already denoted the elements of X via the letter t because from a co-optimization perspective they serve as tests. For simplicity, let the space of potential solutions S have a one-to-one mapping with Y^X—though in practice this will often not be the case, as discussed in Section 2.1.3. So let $S = \{s_0, ..., s_7\}$ where s_i corresponds to f_i. Typically, an algorithm attempting to find an $s \in S$ that approximates f would know there is some mapping between S and Y^X, but may not be able to pick $s \in S$ guaranteeing specific outputs for given inputs (e.g., we don't know how to directly set the weights of a neural network to guarantee a certain error for a certain input). We have $T = X = \{t_1, t_2, t_3\}$ and $M : S \times T \to V = \{0, 1\}$, $M(s, t) = 1 - \delta(s(t), f(t))$ where δ is the Kronecker delta function. There are $|S| \cdot |T| = 8 \cdot 3 = 24$ possible interactions. In generic co-optimization, there would be $|V|^{|S| \cdot |T|} = 2^{24}$ possible metrics M, but since here M is fully determined by f, there are only 8 possible metrics M, fully-specified, one per row, in columns 5–28 of Table 5.

One way to think about it is that supervised learning with an unrestricted set of functions f maps to co-optimization with a restricted set of metrics M; the specifics of this restriction lead to differences in the design of optimal algorithms that we point out throughout the entire Section 3. Note also that in supervised learning there is a dependence between the number of tests and the number of theoretically-possible distinct potential solutions, as $|Y^X| = |Y|^{|X|}$, whereas that's typically not the case in co-optimization (for instance, in the piping network problem, the set S of possible valve placements would be the same whether tests consist of pairs of co-occurring damages or triples thereof, the latter of which are more numerous).

Output mechanisms. If we have no data (D = {}), then all these metrics (functions) are possible and, we assume, equally likely. We discuss this assumption in more detail in Section 3.4. Columns 29–36 in the table show, for each metric, the actual value of g (i.e., the sum of errors over all tests) for each of the 8 potential solutions. At the bottom, the expected value of g, averaged over all metrics, shows that in the absence of information all potential solutions are equally promising, with an expected g-value of 1.5.

Consider being given 2 data points, namely D = $\{(t_1, n), (t_2, n)\}$. Columns 37–44 in Table 5 show that only 2 out of the 8 functions f (metrics M) are still possible (consistent with the data D). Averaging the expected value of g over only these functions shows that potential solutions s_0 and s_1 look more promising than the rest. So any algorithm that would output one of them, by whatever means, would be better overall than an algorithm outputting one of the other six potential solutions. This constitutes *free lunch with respect to g*. By contrast, the *no free lunch* that Wolpert (1992) showed was *for off-training-set (OTS) error*. This is exemplified in columns 45–52 of Table 5, where OTS(s_i) is simply equal to the error over the remaining test t_3, which is $M(s_i, t_3)$. Averaging this over the two possible functions shows that all potential solutions have the same expected error outside of the sample D.

Nonetheless, one question is: how might we exploit the free lunch with respect to g? Answering this for the equal-likelihood case can provide a base from which to then study non-uniform distributions over problems, where there should be free lunch with respect to OTS as well. We can approach this question in a manner similar to co-optimization by considering only algorithms that use an exploratory approach, where they can sample



f	$f(t_1)$	$f(t_2)$	$f(t_3)$	$M(s_0,t_1)$	$M(s_0,t_2)$	$M(s_0,t_3)$	$M(s_1,t_1)$	$M(s_1,t_2)$	$M(s_1,t_3)$	$M(s_2,t_1)$	$M(s_2,t_2)$	$M(s_2,t_3)$	$M(s_3,t_1)$	$M(s_3,t_2)$	$M(s_3,t_3)$	$M(s_4,t_1)$	$M(s_4,t_2)$	$M(s_4,t_3)$	$M(s_5,t_1)$	$M(s_5,t_2)$	$M(s_5,t_3)$	$M(s_6,t_1)$	$M(s_6,t_2)$	$M(s_6,t_3)$	$M(s_7,t_1)$	$M(s_7,t_2)$	$M(s_7,t_3)$	D=[] $g(s_0)$	$g(s_1)$	$g(s_2)$	$g(s_3)$	$g(s_4)$	$g(s_5)$	$g(s_6)$	$g(s_7)$	D=[(t_1,n), (t_2,n)] $g(s_0)$	$g(s_1)$	$g(s_2)$	$g(s_3)$	$g(s_4)$	$g(s_5)$	$g(s_6)$	$g(s_7)$	D=[(t_1,n), (t_2,n)] $OTS(s_0)$	$OTS(s_1)$	$OTS(s_2)$	$OTS(s_3)$	$OTS(s_4)$	$OTS(s_5)$	$OTS(s_6)$	$OTS(s_7)$
0	n	n	n	0	0	0	0	0	1	0	1	0	0	1	1	1	0	0	1	0	1	1	1	0	1	1	1	0	1	1	2	1	2	2	3	0	1	1	2	1	2	2	3	0	1	0	1	0	1	0	1
1	n	n	y	0	0	1	0	0	0	0	1	1	0	1	0	1	0	1	1	0	0	1	1	1	1	1	0	1	0	2	1	2	1	3	2	1	0	2	1	2	1	3	2	1	0	1	0	1	0	1	0
2	n	y	n	0	1	0	0	1	1	0	0	0	0	0	1	1	1	0	1	1	1	1	0	0	1	0	1	1	2	0	1	2	3	1	2																
3	n	y	y	0	1	1	0	1	0	0	0	1	0	0	0	1	1	1	1	1	0	1	0	1	1	0	0	2	1	1	0	3	2	2	1																
4	y	n	n	1	0	0	1	0	1	1	1	0	1	1	1	0	0	0	0	0	1	0	1	0	0	1	1	1	2	2	3	0	1	1	2																
5	y	n	y	1	0	1	1	0	0	1	1	1	1	1	0	0	0	1	0	0	0	0	1	1	0	1	0	2	1	3	2	1	0	2	1																
6	y	y	n	1	1	0	1	1	1	1	0	0	1	0	1	0	1	0	0	1	1	0	0	0	0	0	1	2	3	1	2	1	2	0	1																
7	y	y	y	1	1	1	1	1	0	1	0	1	1	0	0	0	1	1	0	1	0	0	0	1	0	0	0	3	2	2	1	2	1	1	0																
Expected value (over possible functions f) of the error (g or OTS) for given solution and data D																												1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	0.5	0.5	1.5	1.5	1.5	1.5	2.5	2.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

History H=[((s,t),y)] (i.e. $s(t)$=y). Case t=t_1

Output choices (s or $s'\neq s$)

s	$s'\neq s$	$E_f(g(s))$	$E_f(g(s'))$	$E_{s,s'}\{E_f(g(s))\}$	$E_{s,s'}\{E_f(g(s_{new}\neq s))\}$	Best $E_{s,s'}\{E_f(g(.))\}$
s_4	s_0	1.5	0.5			
s_4	s_1	1.5	0.5			
s_4	s_2	1.5	1.5			
s_4	s_3	1.5	1.5			
s_4	s_5	1.5	1.5			
s_4	s_6	1.5	2.5			
s_4	s_7	1.5	2.5			
s_5	s_0	1.5	0.5			
s_5	s_1	1.5	0.5			
s_5	s_2	1.5	1.5			
s_5	s_3	1.5	1.5			
s_5	s_4	1.5	1.5			
s_5	s_6	1.5	2.5			
s_5	s_7	1.5	2.5			
s_6	s_0	2.5	0.5			
s_6	s_1	2.5	0.5			
s_6	s_2	2.5	1.5			
s_6	s_3	2.5	1.5			
s_6	s_4	2.5	1.5			
s_6	s_5	2.5	1.5			
s_6	s_7	2.5	2.5			
s_7	s_0	2.5	0.5			
s_7	s_1	2.5	0.5			
s_7	s_2	2.5	1.5			
s_7	s_3	2.5	1.5			
s_7	s_4	2.5	1.5			
s_7	s_5	2.5	1.5			
s_7	s_6	2.5	2.5			

Exploration choices

Choice 1: pick t'=$t_2\neq t$=t_1 and evaluate $s(t')$

Outcome of $s(t')$	s	Outcome probability	$E_f(g(s))$	$E_f(g(s_{new}\neq s))$	$E_s\{E_f(g(s))\|s(t')\}$	$E_s\{E_f(g(s_{new}\neq s))\|s(t')\}$	Best $E_s\{E_f(g(.))\|s(t')\}$	Best $E_s\{E_f(g(.))\}$
n	s_4	**0.5**	1.50	1.50	1.50	1.50	**1.50**	**1.43**
	s_5		1.50	1.50				
y	s_6	**0.5**	2.50	1.36	2.50	1.36	**1.36**	
	s_7		2.50	1.36				

Choice 2: pick $s'\neq s$ and evaluate $s'(t=t_1)$

Outcome of $s'(t)$	s	s'	Outcome probability	$E_f(g(s))$	$E_f(g(s'))$	$E_f(g(s_{new}\neq s,s'))$	$E_{s,s'}\{E_f(g(s))\|s'(t)\}$	$E_{s,s'}\{E_f(g(s'))\|s'(t)\}$	$E_{s,s'}\{E_f(g(s_{new}\neq s,s'))\|s'(t)\}$	Best $E_{s,s'}\{E_f(g(.))\|s'(t)\}$	Best $E_{s,s'}\{E_f(g(.))\}$
n	s_4	s_0	**0.57**	1.5	0.5	1.67	2	1	1.5	**1**	**1.14**
	s_4	s_1		1.5	0.5	1.67					
	s_4	s_2		1.5	1.5	1.50					
	s_4	s_3		1.5	1.5	1.50					
	s_5	s_0		1.5	0.5	1.67					
	s_5	s_1		1.5	0.5	1.67					
	s_5	s_2		1.5	1.5	1.50					
	s_5	s_3		1.5	1.5	1.50					
	s_6	s_0		2.5	0.5	1.50					
	s_6	s_1		2.5	0.5	1.50					
	s_6	s_2		2.5	1.5	1.33					
	s_6	s_3		2.5	1.5	1.33					
	s_7	s_0		2.5	0.5	1.50					
	s_7	s_1		2.5	0.5	1.50					
	s_7	s_2		2.5	1.5	1.33					
	s_7	s_3		2.5	1.5	1.33					
y	s_4	s_5	**0.43**	1.5	1.5	1.50	2	2	1.33	**1.33**	
	s_4	s_6		1.5	2.5	1.33					
	s_4	s_7		1.5	2.5	1.33					
	s_5	s_4		1.5	1.5	1.50					
	s_5	s_6		1.5	2.5	1.33					
	s_5	s_7		1.5	2.5	1.33					
	s_6	s_4		2.5	1.5	1.33					
	s_6	s_5		2.5	1.5	1.33					
	s_6	s_7		2.5	2.5	1.17					
	s_7	s_4		2.5	1.5	1.33					
	s_7	s_5		2.5	1.5	1.33					
	s_7	s_6		2.5	2.5	1.17					

Choice 3: pick $s'\neq s$ and t'=$t_2\neq t$=t_1 and evaluate $s'(t')$

Outcome of $s'(t')$	s	s'	Outcome probability	$E_f(g(s))$	$E_f(g(s'))$	$E_f(g(s_{new}\neq s,s'))$	$E_{s,s'}\{E_f(g(s))\|s'(t')\}$	$E_{s,s'}\{E_f(g(s'))\|s'(t')\}$	$E_{s,s'}\{E_f(g(s_{new}\neq s,s'))\|s'(t')\}$	Best $E_{s,s'}\{E_f(g(.))\|s'(t')\}$	Best $E_{s,s'}\{E_f(g(.))\}$
n	s_4	s_0	**0.50**	1.5	0.5	1.67	2.07	0.93	1.50	**0.93**	**1.14**
	s_4	s_1		1.5	0.5	1.67					
	s_4	s_5		1.5	1.5	1.50					
	s_5	s_0		1.5	0.5	1.67					
	s_5	s_1		1.5	0.5	1.67					
	s_5	s_4		1.5	1.5	1.50					
	s_6	s_0		2.5	0.5	1.50					
	s_6	s_1		2.5	0.5	1.50					
	s_6	s_4		2.5	1.5	1.33					
	s_6	s_5		2.5	1.5	1.33					
	s_7	s_0		2.5	0.5	1.50					
	s_7	s_1		2.5	0.5	1.50					
	s_7	s_4		2.5	1.5	1.33					
	s_7	s_5		2.5	1.5	1.33					
y	s_4	s_2	**0.5**	1.5	1.5	1.50	1.93	1.93	1.36	**1.36**	
	s_4	s_3		1.5	1.5	1.50					
	s_4	s_6		1.5	2.5	1.33					
	s_4	s_7		1.5	2.5	1.33					
	s_5	s_2		1.5	1.5	1.50					
	s_5	s_3		1.5	1.5	1.50					
	s_5	s_6		1.5	2.5	1.33					
	s_5	s_7		1.5	2.5	1.33					
	s_6	s_2		2.5	1.5	1.33					
	s_6	s_3		2.5	1.5	1.33					
	s_6	s_7		2.5	2.5	1.17					
	s_7	s_2		2.5	1.5	1.33					
	s_7	s_3		2.5	1.5	1.33					
	s_7	s_6		2.5	2.5	1.17					

Table 5: Examples of binary classification free lunch in domain with $X = \{t_1, t_2, t_3\}$, $Y = \{y, n\}$. Font style indicates which values and probabilities were aggregated, green cell shading indicates where the minimum value is coming from and orange cell shading highlights the expected performance for each exploration choice.

$s \in S$ and $t \in T$ that are present in the data and pay for each $M(s,t)$ evaluation. While traditional exploratory approaches to supervised learning always perform *full evaluation*, i.e., for each s considered they evaluate $M(s,t)$ for all t in the data, there is no constraint to do so. In fact, as we shall see later in this section, full evaluation can be suboptimal.

Consider the history $\mathrm{H} = [\langle\langle s,t\rangle, y\rangle]$, meaning the algorithm randomly selected an $s \in S$ and a $t \in \{t_1, t_2\}$, evaluated $s(t)$ and observed value y (spent budget is 1). The output mechanism has two choices: output the partially-evaluated s or output a randomly selected, completely-unevaluated potential solution $s' \neq s$. These choices are shown in the bottom left part of Table 5 (instantiated only with with $t = t_1$, due to space constraints; the other instantiation yields the same kind of results). Since $s(t) = y$, s can only be one of s_4, s_5, s_6, s_7, with respective expected g-values of 1.5, 1.5, 2.5, 2.5. For each possible s, there are seven possibilities for $s' \neq s$, of which four (s_0, s_1, s_2, s_3) have expected g less than or equal to 1.5 and three $(\{s_4, s_5, s_6, s_7\} \setminus \{s\})$ have expected g greater than or equal to 1.5. Averaging over all situations yields an overall expectation of 2 for $g(s)$ and ≈ 1.43 for $g(s')$, thus outputting s' is the strictly better choice and we have *output mechanism free lunch*.

Exploration mechanisms. Suppose now that the algorithm having produced history $\mathrm{H} = [\langle\langle s,t\rangle, y\rangle]$ (spent budget of 1) is allowed one additional M-evaluation (remaining budget of 1). It is the job of the exploration mechanism to decide which interaction to run through the metric M. The rest of Table 5 outlines the three exploration choices: 1) evaluate s again with the remaining test present in the data (in this case, $t' = t_2$); 2) pick a random $s' \neq s$ and evaluate it with the test already seen by s (that is $t = t_1$); and 3) pick a random $s' \neq s$ and evaluate it with the other test ($t' = t_2$). And to determine the expected performance for a given exploration choice, we must: consider each of the two potential outcomes, y or n, that could be observed when evaluating the chosen interaction; determine the best (minimum) g-value estimate for the history resulting from that outcome; then aggregate the estimates for the two outcomes. This aggregate is a *weighted* average, since the two outcomes might not be equally likely; for instance, if we choose to evaluate $s'(t = t_1)$, there are 16 cases with outcome n and only 12 with outcome y, so the respective probabilities are $16/28 \approx 0.57$ and $12/28 \approx 0.43$. The calculations reveal that evaluating an interaction involving s', whether with t or t', yields a strictly better expected g-value (1.14) than further evaluating s (1.43). This constitutes *exploration mechanism free lunch*.

It is insightful to look more closely at these calculations so as to understand how the estimates for the various potential solutions involved change when a new interaction is evaluated. Recall that after having evaluated $s(t)$ and observing y the expected $g(s)$ is 2 and the expected $g(s')$ for any completely-unevaluated potential solution $s' \neq s$ is 1.43. The outcome of evaluating $s(t')$ changes the estimate for s: if we observe $s(t') = n$, the new estimate goes down to 1.5 (we've just observed a match between s and f, so s looks more promising now), whereas if we observe $s(t') = y$, then the estimate goes up to 2.5. This is not surprising, since by evaluating $s(t')$ we are, after all, learning more about s. What is interesting is that the estimate of g for a completely-unevaluated potential solution also changes: either to 1.5 if $s(t') = n$ or to 1.36 if $s(t') = y$. This is pointedly different from generic co-optimization. Referring back to Table 4, for history $\mathrm{H} = [\langle\langle a, t_1\rangle, 1\rangle]$ the expected g-value of b or c is 1 and it does not change when we evaluate $M(a, t_2)$ no matter what outcome we observe. Similar impacts can be noted in Table 5 for the other two exploration choices: evaluating $s'(t)$ for $s' \neq s$ changes the expected value for a random

remaining completely-unevaluated potential solution, but not for s, which has already seen t; evaluating $s'(t')$ for $s' \neq s, t' \neq t$ changes both expectations. In contrast, for co-optimization solution concepts like maxisum and maximin the outcome of evaluations involving one potential solution never changes the expectation for any other potential solutions (and this is independent of which test is used in the evaluation).

This difference can be explained through the prism of the possible metrics M, which in the case of supervised learning are much fewer than $|V|^{|S|\cdot|T|}$. Note for instance how in Table 5 we have no metric M assigning 0 to all interactions, nor one assigning 1 to all interactions. In fact, each of the metrics induced by possible functions f has the property that half the interactions get assigned a 0 and half get assigned a 1; and this is also true at the test level: for any test $t \in T$, there are $|S|/2$ potential solutions $s \in S$ such that $M(s,t) = 0$ and just as many for which $M(s,t) = 1$. Thus for any given test t, the expected value of M for a random potential solution that hasn't seen t yet depends on how many other potential solutions have seen t and what the outcomes were. We provide a generic characterization of this in the following section.

3.2 Large Spent Budget, Small Remaining Budget

When the budget spent so far consists of more than a couple of M-evaluations, the structure of the resulting histories can become more complex in terms of which potential solutions have interacted with which tests. We extend the treatment from the small examples above to these more generic cases; when discussing exploration mechanisms we still keep the remaining budget at just one M-evaluation.

3.2.1 Co-optimization

The issue of free lunch and optimality for the maxisum solution concept of co-optimization has first been investigated by Popovici and De Jong (2009), for output mechanisms only. As shown there and alluded to in the simple example of Section 3.1.1, given a history of evaluated interactions H, the expected value of g for a potential solution s is obtained by adding two components: 1) the sum of actual $M(s,t)$ values over those tests t seen by s as part of H; and 2) the sum of *expected* $M(s,t)$ values over tests t not seen by s. Due to the assumption of equally-likely metrics, the later is simply the product of the number of unseen tests and the average of V. To express this formally, assume T and V are finite, $|T| = m$, $T = \{t_1, \ldots, t_m\}$, $|V| = p$, $V = \{v_1, \ldots, v_p\}$. Given a history H and a potential solution s, let $0 \leq k \leq m$ denote the number of *distinct* interactions in H involving s: $\langle\langle s, t_{i_1}\rangle, v_{j_1}\rangle, \ldots \langle\langle s, t_{i_k}\rangle, v_{j_k}\rangle$, where $i_1, \ldots, i_k \in 1..m$, all distinct, and $j_1, \ldots, j_k \in 1..p$, $v_{j_\ell} = M(s, t_{i_\ell})$. Let the *current sum* of s given H, $g_{\mathrm{H}}(s) = v_{j_1} + \cdots + v_{j_k}$. Then the expected g-value of s given H is $\mathbb{E}(g(s)|\,\mathrm{H}) = g_{\mathrm{H}}(s) + (m-k)\cdot \mathrm{avg}(V)$. For brevity, we sometimes leave H implicit and simply write $\mathbb{E}(g(s))$. Notably, this value does not depend on any interactions involving potential solutions other than s. It also does not depend on which specific distinct k tests s has interacted with as part of H, only how many and what the outcomes of those interactions were. For a potential solution that is completely unevaluated (i.e., $k = 0$), the expectation is simply $m \cdot avg(V)$.

Output mechanisms are optimal if for any given H they output the potential solution s with the best $\mathbb{E}(g(s)|\,\mathrm{H})$ value; this may be a partially-evaluated potential solution ($k > 0$)

or a completely-unevaluated one. The computational cost of this, alluded to in Section 2.3.1, is small[13] : updating $\mathbb{E}(g(s))$ after each new M-evaluation involving s is trivial, as is determining the potential solution with the best $\mathbb{E}(g(\cdot))$-value; the cost of optimal output-selection is at most $O(|\,\mathrm{H}\,|)$. But the feasibility of implementation is dependent on knowing $\mathrm{avg}(V)$. Note that we do not need to know the actual value of m, since the the term $m \cdot \mathrm{avg}(V)$ is common to all estimates and thus doesn't impact the outcome of comparisons between them.

Exploration mechanisms have the nature of choices available to them influenced by the above-mentioned lack of dependencies, as follows. To assess an interaction choice, for each possible outcome in V we assess the resulting history and determine the potential solution with the best g-value estimate for that history, using the $\mathbb{E}(g(s)|\,\mathrm{H})$ formula in the previous paragraph; then we aggregate these values over the possible outcomes; under the assumption of all possible and equally-probable metrics, all outcomes for a given interaction are also equally likely, so the aggregation is via uniform averaging. These mathematical derivations reveal that for any interaction $\langle s, t\rangle$ that is new with respect to history H, the expected performance after evaluating $\langle s, t\rangle$ is independent of the specific t used, as long as s has not previously seen it (Popovici, 2017). Thus the only actual choice to be made is which potential solution to evaluate; the test can be picked randomly amongst those yet unseen by that solution (can be one seen by other potential solutions). For a history containing measured interactions involving α distinct potential solutions, the number of choices is $\alpha + 1$: one of α partially-evaluated potential solutions or a random completely-unevaluated one. And the computational cost of assessing one choice is the sum of p optimal output mechanism costs (as described in the previous paragraph), one for each possible outcome in V. This is considerably more expensive than the optimal output mechanism, though explicitly implementing it could still be tractable for small values of p and α. Furthermore, such explicit implementation might not be necessary in some situations: for domains where V is symmetric around 0, if the history H does not contain any fully-evaluated potential solutions (ones that have seen all m tests), then the best choice is mathematically-provable to be evaluating a new interaction for a still-testable potential solution with the best $\mathbb{E}(g(\cdot))$ (Popovici, 2017)—and the ensuing exploration-decision cost is as small as described above for the optimal output mechanism.

3.2.2 Supervised Learning

Let us now return to the case of binary classification. Given a history of evaluated interactions H, the expected value of g for a potential solution s is still obtained by adding the same two components as in co-optimization: the sum of $M(s, t)$ values known from the history ($g_{\mathrm{H}}(s)$) and the sum of expected $M(s, t)$ values. The difference is that for the second component, the expected value of $M(s, t)$ varies with t.

To see this, consider the example in Table 6. For a binary classification domain with $|X| = m$, assuming a one-to-one relationship between S and Y^X, the total number of potential solutions is 2^m. For any test and any metric M, half of these potential solutions will produce an M-value of 1, while the other half will produce an M-value of 0. As part of the history H, test t has interacted with potential solutions s and s', both times resulting

13. For other co-optimization solution concepts, like maximin, output-selection cost can be much larger.

	M(.,.)										
	t	t'	t''								
s	1			E($g(s)$)	=	M(s,t)	+	E(t')	+	E(t'')	+ (m-3)*0.5
s'	1	0		E($g(s')$)	=	M(s',t)	+	M(s',t')	+	E(t'')	+ (m-3)*0.5
s''			1	E($g(s'')$)	=	E(t)	+	E(t')	+	M(s'',t'')	+ (m-3)*0.5
s_{new}				E($g(s_{new})$)	=	E(t)	+	E(t')	+	E(t'')	+ (m-3)*0.5
# 1s seen (β_1)	2	0	1								
# pot. sols. seen (β)	2	1	1								
E(.) = $\frac{2^{m-1} - \beta 1}{2^m - \beta}$	$\frac{2^{m-1} - 2}{2^m - 2}$	$\frac{2^{m-1}}{2^m - 1}$	$\frac{2^{m-1} - 1}{2^m - 1}$								

Table 6: Examples of computing expected g-values for potential solutions for a given history. Binary classification domain with $|X| = m > 4$, $|Y| = 2$ and $3 \leq |D| < m$, with $\mathrm{D} \supseteq \{(t, n), (t', n), (t'', y)\}$ and history $\mathrm{H} = [\langle\langle s, t\rangle, y\rangle, \langle\langle s', t\rangle, y\rangle, \langle\langle s', t'\rangle, n\rangle, \langle\langle s'', t''\rangle, n\rangle]$. $\mathbb{E}(\cdot)$ is shorthand notation for $\mathbb{E}(\cdot|\,\mathrm{D}, \mathrm{H})$, whether applied to tests or to g-values for potential solutions.

in value 1. Thus of the remaining $2^m - 2$ potential solutions that have not been evaluated with t yet, there are only $2^{m-1} - 2$ left that can produce a value of 1, while the other 2^{m-1} give a value of 0. So the expected value of $M(\cdot, t)$ for any potential solution other than s and s' is $\mathbb{E}(t|\,\mathrm{D}, \mathrm{H}) = \mathbb{E}(t) = \frac{(2^{m-1}-2)\cdot 1 + 2^{m-1}\cdot 0}{2^m - 2} = \frac{2^{m-1}-2}{2^m-2}$. In general, if as part of H a test has interacted with β other potential solutions, of which β_1 produced an outcome of 1, then the probability of observing 1 upon evaluating another potential solution with this test is $\frac{2^{m-1}-\beta_1}{2^m-\beta}$ and the probability of observing 0 is $\frac{2^{m-1}-(\beta-\beta_1)}{2^m-\beta}$; so the expected value for evaluating a new interaction with that test is $\frac{2^{m-1}-\beta_1}{2^m-\beta} \cdot 1 + \frac{2^{m-1}-(\beta-\beta_1)}{2^m-\beta} \cdot 0 = \frac{2^{m-1}-\beta_1}{2^m-\beta}$. $\mathbb{E}(t')$ and $\mathbb{E}(t'')$ in Table 6 were also derived using this formula. The expected value for any test not seen by any potential solution is still $\mathrm{avg}(V) = 0.5$, which can also be derived as $\frac{2^{m-1}-0}{2^m-0}$. This is regardless of whether such a test is part of the data D or not. Then the g-value estimates for potential solutions are $\mathbb{E}(g(s)|\,\mathrm{D}, \mathrm{H}) = \mathbb{E}(g(s)) = M(s,t) + \mathbb{E}(t') + \mathbb{E}(t'') + (m-3)\cdot 0.5$ and respectively $\mathbb{E}(g(s'))$, $\mathbb{E}(g(s''))$ and $\mathbb{E}(g(s_{new}))$ shown in Table 6. More thorough mathematical derivations of these quantities can be found in Online Appendix A.2.

Output mechanisms are optimal if they output the potential solution with minimum $\mathbb{E}(g(\cdot))$. To implement that, the estimates are inexpensive to compute, albeit more involved than for co-optimization: for each new $M(s,t)$ evaluation, first update the expected value for $\mathbb{E}(t)$, then update $\mathbb{E}(g(s))$ as well as $\mathbb{E}(g(\cdot))$ for any other potential solutions that haven't yet seen t (including a representative completely-unevaluated one) then re-determine the potential solution with the best g-value estimate, for a total output-selection cost of $\mathrm{O}(|\,\mathrm{H}\,|)$. However, these computations do require knowing the value of m and care must be taken to avoid potential computer precision errors for large m.

Exploration mechanisms are exemplified in Table 7, which shows how Table 5's performance estimates for the various exploration choices can be re-derived using a more compact computation approach: for a given choice, for each of the two possible outcomes, consider the resulting history and compute $\mathbb{E}(g(\cdot))$ for all potential solutions involved, using

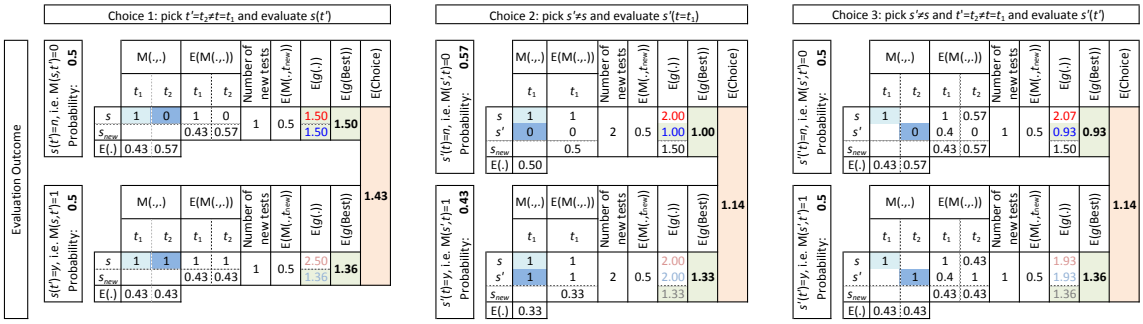


Table 7: Examples of computing expected g-values for potential solutions after various exploration choices and outcomes. Binary classification domain with $X = \{t_1, t_2, t_3\}$, $Y = \{y, n\}$, data $\mathrm{D} = \{(t_1, n), (t_2, n)\}$ and history $\mathrm{H} = [\langle\langle s, t_1\rangle, y\rangle]$. Font style and green & orange cell shading link to the respective cells in Table 5. Light blue shading corresponds to the current history and dark blue shading shows the next interaction chosen for evaluation and its M-outcome. $\mathbb{E}(\cdot)$ is shorthand notation for $\mathbb{E}(\cdot \,|\, \mathrm{D}, \mathrm{H})$. For a given s and t, the expected value $\mathbb{E}(M(s,t))$ is equal either to the actual value of $M(s,t)$, if it has been evaluated, or to $\mathbb{E}(t)$ otherwise.

formulas like the ones in Table 6; then take the best (minimum) estimate; then use the probabilities of the two outcomes to compute a weighted average of the best estimates.

For example, consider the middle panel of Table 7, where $m = 3$ and we choose to evaluate a new potential solution s' with the test $t = t_1$, which has previously interacted only with s (so $\beta = 1$) and produced a value $M(s,t) = 1$ (so $\beta_1 = 1$). The probability of outcome 1 for $M(s', t_1)$ is $\frac{2^2-1}{2^3-1} = \frac{3}{7} \approx 0.43$ and the probability of outcome 0 is $\frac{2^2-0}{2^3-1} = \frac{4}{7} \approx 0.57$. For outcome 1 the best g-value estimate is 1.33 for s_{new} and for outcome 0 it is 1 for s', resulting in an expected performance of $1.14 = 1.33 \cdot 0.43 + 1 \cdot 0.57$ for the $\langle s', t_1\rangle$ exploration choice.

For this particular example, the expected performance for evaluating a new potential solution s' was the same whether using the test t_1 already seen by s or the new test t_2. This is not the case in general, as exemplified in Table 8 for a slightly larger domain and history. When choosing to evaluate a new potential solution s'', we get different expected performance whether we use the test t previously seen by s and s', the test t' previously seen by s' only or a new test t'' not previously seen by any potential solution. Similarly, if choosing to further evaluate s, we get different expected performance whether we use the test t' previously seen by s' or a new test t''.

Thus for binary classification the number of relevant choices available for the exploration mechanism for a given history is considerably larger than in co-optimization. If a history H contains η distinct interactions involving α distinct potential solutions and γ distinct tests ($\eta \leq \alpha \cdot \gamma$), then there could be as many as $(\alpha + 1) \cdot (\gamma + 1) - \eta$ relevant choices, as opposed to at most $\alpha + 1$ in co-optimization. The computational cost of assessing one choice is the sum of two optimal output mechanism costs, one for each of the possible outcomes $\{y, n\}$. Consequently, the total exploration-decision cost is significantly larger than the optimal output mechanism cost, though still polynomial with a naive explicit implementation of $\mathrm{O}(|\,\mathrm{H}\,|^3)$, which could be tractable for moderate-size histories.

Table 8: Examples of how the choice of test impacts the expected resulting performance. Binary classification domain with $|X| = m = 4$, $Y = \{y, n\}$, data $\mathrm{D} = \{(t, n), (t', n), (t'', y)\}$ and history $\mathrm{H} = [\langle\langle s, t\rangle, n\rangle, \langle\langle s', t\rangle, y\rangle, \langle\langle s', t'\rangle, n\rangle]$. Bold font shows which values and probabilities were aggregated. Green cell shading indicates where the best (minimum) value is coming from, orange cell shading highlights the expected performance for each exploration choice, light blue shading corresponds to the current history and dark blue shading shows the next interaction chosen for evaluation and its M-outcome. $\mathbb{E}(\cdot)$ is shorthand notation for $\mathbb{E}(\cdot \mid \mathrm{D}, \mathrm{H})$. For a given s and t, the expected value $\mathbb{E}(M(s,t))$ is equal either to the actual value of $M(s,t)$, if it has been evaluated, or to $\mathbb{E}(t)$ otherwise. $\mathbb{E}(g(s))$ is equal to the sum of the $\mathbb{E}(M(s,t))$ over tests seen by any potential solution plus the number of new tests times $\mathbb{E}(M(\cdot, t_{new})) = \mathbb{E}(t_{new}) = 0.5$.

Since math-based reductions of exploration-decision costs have been proven for co-optimization, we did a preliminary analysis to see if the same were possible for binary classification. We were able to prove a couple of ways to analytically prune the set of relevant exploration choices while maintaining optimality. Specifically, if t_{new} is a test present in the data but previously unseen by any potential solution, then out of the exploration choices $\langle s, t_{new}\rangle, s \in S$ we only need to assess the expected performance for $\langle s^{best}, t_{new}\rangle$, where s^{best} is the potential solution with the best expectation $\mathbb{E}(g(\cdot))$ for the current H (Theorem 1 in Appendix A.3). Moreover, if s^{best} is a completely-unevaluated potential solution, then out of all choices of type $\langle s^{best}, t\rangle, t \in T$ we only need to assess $\langle s^{best}, t^*\rangle$ for a t^* easily-identifiable as maximizing the quantity $\mathbb{E}(t) \cdot \big(1 - \mathbb{E}(t)\big) \cdot \frac{2^m - \beta}{2^m - \beta - 1}$, where β is as in Section 3.2.2, the number of potential solutions seen by t (Theorem 2 in Appendix A.3).

While helpful, these results allow us to prune at most $\alpha + \gamma$ exploration choices, which in the worst case (if $\alpha \cdot \gamma \gg \eta$) could still leave us with $\mathrm{O}(|\,\mathrm{H}\,|^2)$ interaction choices to be assessed, each costing $\mathrm{O}(|\,\mathrm{H}\,|)$. Further research is needed to determine if additional analytical pruning can be performed. Moreover, these results only concern the interaction choice optimizing the expected g-value for the best output right after evaluating that interaction, i.e., a remaining budget of $n = 1$. Which interaction choice is optimal could vary with n and we briefly discuss this case in the next section; the pruning results above were obtained by first generalizing to the formulas below and then instantiating them with $n = 1$.

3.3 Large Remaining Budget

Since output mechanisms are not impacted by remaining budgets and we have already studied large spent budgets in Section 3.2, in this section we focus on exploration mechanisms. Their optimality and free lunch for large remaining budgets was previously investigated in co-optimization for the maxisum (Popovici, 2017) and maximin (Popovici and Winston, 2015) solutions concepts. Following the reasoning from those works, exemplified for $n = 1$ in the previous section, the expected performance for an optimal exploratory algorithm for binary classification, given data D, current history H and remaining budget n, denoted by $\Phi_{\mathrm{a}}^{\min}(\mathrm{D}, \mathrm{H}, n)$, can be recursively expressed as:

$$\begin{aligned}
\Phi_{\mathrm{a}}^{\min}(\mathrm{D}, \mathrm{H}, 0) &= \min_{s \in S} \; \mathbb{E}(g(s)|\,\mathrm{D}, \mathrm{H}),\\
\Phi_{\mathrm{a}}^{\min}(\mathrm{D}, \mathrm{H}, n) &= \min_{\langle s,t\rangle \in \mathcal{E}'(\mathrm{D},\mathrm{H})} \; \mathop{\mathbb{E}}_{v \in \{0,1\}} \; \Phi_{\mathrm{a}}^{\min}(\mathrm{D}, \mathrm{H} \oplus \langle\langle s, t\rangle, v\rangle, n-1)\\
&= \min_{\langle s,t\rangle \in \mathcal{E}'(\mathrm{D},\mathrm{H})} \; \sum_{v \in \{0,1\}} P(M(s,t) = v|\,\mathrm{D}, \mathrm{H}) \cdot \Phi_{\mathrm{a}}^{\min}(\mathrm{D}, \mathrm{H} \oplus \langle\langle s, t\rangle, v\rangle, n-1)
\end{aligned}$$

where

- $\mathcal{E}'(\mathrm{D}, \mathrm{H})$ represents the set of all new interactions that could be evaluated next;
- the probabilities $P(M(s,t) = v|\,\mathrm{D}, \mathrm{H})$ of outcomes 0 and 1 are as described at the beginning of Section 3.2.2;
- $\oplus$ denotes adding a measured interaction to a history; and
- the subscript 'a' refers to 'algorithms'.

In words, the optimal performance for a given history and budget is dependent on the optimal performance for histories lengthened by one and budgets decreased by one; it is obtained by optimizing over interaction choices the expected value, aggregated over possible outcomes for the interaction, of the optimal performance for the resulting lengthened history and reduced budget.

Going back to the expression of $\mathbb{E}(g(s))$ from Table 6, note that it can be re-written as $M(s,t)+\mathbb{E}(t')+\mathbb{E}(t'')+(|D|-3)\cdot 0.5\big)+(m-|D|)\cdot 0.5$. The last term is the expected out-of-sample (OTS) error and is the same for all potential solutions: this is the no free lunch of Wolpert (1992). The sum of the other terms represents the *expected* in-sample error. Note this differs from traditional exploratory approaches to supervised learning, where each considered potential solution is evaluated with all tests in the data and therefore the actual in-sample error is known. Such an approach can still be used if the budget n is large in relation to the size of the data D, but given the existence of exploration mechanism free lunch, this could conceivably be a suboptimal approach.

To show that this can indeed happen, we apply the above recursive formula to a small example. Consider a domain with 3 inputs ($|X| = m = 3$) and a situation where we are given 2 data points ($|\,\mathrm{D}\,| = 2$) and a budget of 2 M-evaluations. Whenever output needs to be produced, we use the optimal output mechanism, which chooses the potential solution with minimum $\mathbb{E}(g(\cdot))$ given the history at that point. The optimal exploration mechanism takes into account the outcome of the first interaction in order to determine what the second interaction should be; the respective aggregate performance, given by the above formula, is ≈ 1.071. Full evaluation simply evaluates a potential solution with the two available tests and there are 4 equally-likely combinations of outcomes (11, 01, 10, 00); averaging the respective minimum $\mathbb{E}(g(\cdot))$ values (1.357, 1.5, 1.5, 0.5) yields an aggregate performance of ≈ 1.214, which is strictly worse.

However, implementing the optimal exploration mechanism appears daunting for large budgets n, as it seemingly involves solving a very time-consuming recursive optimization problem. Interestingly, the co-optimization studies of optimal exploration mechanisms for maxisum (Popovici, 2017) and maximin (Popovici and Winston, 2015) showed that for some situations an analytical solution to this recursive optimization can be proven, making the optimal algorithm straightforward to implement, with low exploration-decision and output-selection costs of $\mathrm{O}(|\,\mathrm{H}\,|)$.

The preliminary analysis we performed for binary classification so far only yielded the pruning results for $n = 1$ presented in the previous section, which could still leave us with more than one choice, requiring recursing down to $n = 0$. This is not an issue for such a small n, but for large n the exponential cost of recursive optimization can be infeasible. The additional complexity introduced in binary classification by the interdependence between different potential solutions makes it unclear at the moment whether a full analytical solution for the recursive optimization (or at least some amount of pruning) can be proven for large remaining budgets—and thus whether the costs of the optimal exploration mechanism are feasible or not. This serves to highlight the importance of acknowledging *all* types of costs, as optimizing with respect to one type (e.g., metric-evaluation costs) can incur another (e.g., exploration-decision costs).

3.4 Discussion

The above analysis made two core assumptions: 1) that all functions in Y^X are equally likely; and 2) that there is a of one-to-one mapping between S and Y^X. Additionally, algorithms were constrained to use only a limited budget of M-evaluations; such algorithms might not know the actual in-sample error (error on the data D) for all/any of the potential solutions considered and thus must decide what to output given only such incomplete information.

For this setup, given a certain budget n of M-evaluations, an optimal algorithm is one that has the smallest average over all functions in Y^X of the error on the data D (in-sample error) for the potential solution outputted after using up the budget; and there is free lunch, in that some algorithms produce strictly smaller such average values than others. Unfortunately, this is not useful for generalization, since under the above assumptions, all potential solutions have the same average over functions of OTS-error (error outside the data D): $(m - |\,\mathrm{D}\,|) \cdot 0.5$.

In practice though, we often have reason to believe that the first assumption is not true, rather that there is a non-uniform P on Y^X. But if we do not know anything more specific about P besides its existence, this in itself is not sufficient reason for dismissing OTS no free lunch, as there are as many distributions over metrics for which one algorithm is aggregately-better than the other as there are distributions where the opposite relation holds (Wolpert, 1992). On the other hand, if the knowledge we have about P and the way we choose S allow us to express theoretical Bayes estimates, then we can perform the same type of analysis as presented here to derive and investigate algorithms that are aggregately-optimal from a generalization perspective as well.

Here are several ways in which this could become useful. First, it could be used in passive supervised learning to provide a new approach for dealing with data sets so large that it is too costly to perform evaluation with their entire contents for every potential solution considered. Rather than performing instance selection (Reinartz, 2002) and then performing full evaluation using the selected subset of examples, computational resources could be used more effectively by selecting at a more granular level, namely that of interactions $\langle s, t\rangle$ rather than that of examples (tests) t.

Second, it could be extended for usage in active learning, as a direct exploratory approach instead of the current methods which perform retraining after new labels are observed. To achieve this, the analysis will have to be adapted to take into account labeling costs; whether a new interaction $\langle s, t\rangle$ will incur both an M-evaluation cost and a labeling cost or only the former depends on whether or not the test t has been previously seen by other potential solutions and thus already labeled. This could be formalized as having two different budgets, one for labeling and one for M-evaluations, or, if both costs are in fact measured in the same "currency", have a single budget and allow the algorithm to determine the best way to allocate it to the two activities.

Last but not least, recall from the discussion on page 11 that some exploratory approaches do not rely on the error function having a closed form with easily computable derivative and therefore they also do not require that the data be in (or transformed into) tabular form. The optimal algorithms described in Section 3 have this desirable property and thus would be applicable to supervised learning for more complex data structures.

4. Conclusions

Metaphorically speaking, this paper laid out a deciphered Rosetta Stone for the languages of supervised learning and co-optimization, then used it to translate free lunch research from the latter field to the former. While practical optimal algorithms based on free lunch are still a ways away, the hope of this presentation is that the different perspective provided by co-optimization could spark ideas for new types of heuristics for supervised learning (promising directions were described in Section 3.4) and also that this translation will become only one of many, the beginning of cross-pollination of ideas between the fields.

Several paths for future research have already been pointed out: reducing the time complexity of optimal exploration mechanisms in the base setup comprising uniformity assumptions, expanding to non-uniform distributions and adapting to the multiple types of costs incurred in active learning. Additionally, there are yet more fields concerned with similar problems and constructing similar mappings for them could lead to further advances; the most obvious ones are reinforcement learning, design of experiments and statistical learning in general.

Acknowledgments

This material is based upon work supported by the Office of Naval Research under Contract No. N00014-12-C-0131. The author would also like to thank Siegfried Martens and the anonymous reviewers for reading earlier versions of the manuscript and providing valuable and constructive feedback.

References

Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning From Data*. AMLBook, 2012.

Amazon. Amazon Mechanical Turk Documentation, 2005. URL `https://aws.amazon.com/documentation/mturk/`. Accessed on 2016-03-08.

Peter J. Angeline and Jordan B. Pollack. Competitive environments evolve better solutions for complex tasks. In *International Conference on Genetic Algorithms*, pages 264–270. Morgan Kaufmann Publishers Inc., 1993.

Anthony Bucci. *Emergent Geometric Organization and Informative Dimensions in Co-evolutionary Algorithms*. PhD thesis, Michtom School of Computer Science, Brandeis University, USA, 2007.

Anthony Bucci, Jordan B Pollack, and Edwin De Jong. Automated extraction of problem structure. In *Genetic and Evolutionary Computation Conference*, pages 501–512. Springer, 2004.

Jianlin Cheng, Allison N. Tegge, and Pierre Baldi. Machine learning methods for protein structure prediction. *IEEE Reviews in Biomedical Engineering*, 1:41–49, 2008.

Siang Yew Chong, Peter Tino, and Xin Yao. Measuring generalization performance in coevolutionary learning. *IEEE Transactions on Evolutionary Computation*, 12(4):479–505, 2008.

Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000.

Ido Dagan and Sean P. Engelson. Committee-based sampling for training probabilistic classifiers. In *International Conference on Machine Learning*, pages 150–157, 1995.

Edwin D. de Jong. The MaxSolve algorithm for coevolution. In *Genetic and Evolutionary Computation Conference*, pages 483–489. ACM Press, 2005.

Edwin D. de Jong. Objective fitness correlation. In *Genetic and Evolutionary Computation Conference*, pages 440–447. ACM Press, 2007.

Kenneth A. De Jong. *Evolutionary Computation: A Unified Approach*. MIT press, 2006.

Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Roth. The reusable holdout: preserving validity in adaptive data analysis. *Science*, 349 (6248):636–638, 2015.

Dario Floreano, Peter Dürr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

Seymour Geisser. The predictive sample reuse method with applications. *Journal of the American Statistical Association*, 70(350):320–328, 1975.

Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

W. Daniel Hillis. Co-evolving parasites improve simulated evolution in an optimization procedure. *Physica D*, 42:228–234, 1990.

Tin Kam Ho. Random decision forests. In *Third International Conference on Document Analysis and Recognition*, volume 1, pages 278–282. IEEE, 1995.

Ilknur Icke and Josh C. Bongard. Improving genetic programming based symbolic regression using deterministic machine learning. In *Congress on Evolutionary Computation*, pages 1763–1770. IEEE, 2013.

Mikkel T. Jensen. Finding worst-case flexible schedules using coevolution. In *Genetic and Evolutionary Computation Conference*, pages 1144–1151. Morgan Kaufmann, 2001.

John R Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*, volume 5 of *Genetic Programming*. Springer Science & Business Media, 2006.

Pier L. Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson. *Learning Classifier Systems: From Foundations to Applications*. Springer, 2003.

Pedro Larranaga and Jose A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, volume 2 of *Genetic Algorithms and Evolutionary Computation*. Springer Science & Business Media, 2002.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015.

Sean Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. URL `http://cs.gmu.edu/~sean/book/metaheuristics/`. Available for free.

Subhransu Maji. Large scale image annotations on amazon mechanical turk. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2011-79*, 2011.

Liviu Panait and Sean Luke. A comparison of two competitive fitness functions. In W. B. Langdon et al, editor, *Genetic and Evolutionary Computation Conference*, pages 503–511. Morgan Kaufmann, 2002.

Elena Popovici. *An Analysis of Two-Population Coevolutionary Computation*. PhD thesis, George Mason University, USA, 2006.

Elena Popovici. Co-optimization free lunches: Tractability of optimal black-box algorithms for maximizing expected utility. *Evolutionary Computation (accepted for publication March 21st)*, 2017.

Elena Popovici and Kenneth A. De Jong. Relationships between internal and external metrics in co-evolution. In *Congress on Evolutionary Computation*, pages 2800–2807. IEEE Press, 2005.

Elena Popovici and Kenneth A. De Jong. Monotonicity versus performance in co-optimization. In *Foundations of Genetic Algorithms*, pages 151–170. ACM Press, 2009.

Elena Popovici and Ezra Winston. A framework for co-optimization algorithm performance and its application to worst-case optimization. *Theoretical Computer Science*, 567:46 – 73, 2015.

Elena Popovici, Oliver Bandte, and Paolo Gaudiano. New approaches for the automated performance evaluation and design of control systems: a firemain example. In *Automation and Controls Symposium*. American Society of Naval Engineers, 2007.

Elena Popovici, Anthony Bucci, R. Paul Wiegand, and Edwin D. de Jong. Coevolutionary principles. In Grzegorz Rozenberg, Thomas Back, and Joost N. Kok, editors, *Handbook of Natural Computing*, Theoretical Computer Science, pages 987–1033. Springer-Verlag, 2010.

Elena Popovici, Ezra Winston, and Anthony Bucci. On the practicality of optimal output mechanisms for co-optimization algorithms. In *Foundations of Genetic Algorithms*, pages 43–60. ACM Press, 2011.

Mitchell A. Potter. *The Design and Analysis of a Computational Model of Cooperative Coevolution*. PhD thesis, George Mason University, USA, 1997.

Payam Refaeilzadeh, Lei Tang, and Huan Liu. Cross-validation. In *Encyclopedia of database systems*, pages 532–538. Springer, 2009.

Thomas Reinartz. A unifying view on instance selection. *Data Mining and Knowledge Discovery*, 6(2):191–210, 2002.

Lior Rokach and Oded Maimon. *Data Mining with Decision Trees: Theory and Applications*. World Scientific, 2014.

Christopher D Rosin and Richard K Belew. New methods for competitive coevolution. *Evolutionary Computation*, 5(1):1–29, 1997.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

Jerome Sacks, William J. Welch, Toby J. Mitchell, and Henry P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4(4):409–423, 1989.

Travis C. Service. Unbiased coevolutionary solution concepts. In *Foundations of Genetic Algorithms*, pages 121–130. ACM Press, 2009a.

Travis C. Service. Free lunches in pareto coevolution. In *Genetic and Evolutionary Computation Conference*, pages 1721–1728. ACM Press, 2009b.

Travis C. Service and Daniel R. Tauritz. Co-optimization algorithms. In *Genetic and Evolutionary Computation Conference*, pages 387–388. ACM, 2008a.

Travis C. Service and Daniel R. Tauritz. A no-free-lunch framework for coevolution. In *Genetic and Evolutionary Computation Conference*, pages 371–378. ACM, 2008b.

Burr Settles. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–114, 2012.

Kent A. Spackman. Signal detection theory: Valuable tools for evaluating inductive learning. In *Sixth International Workshop on Machine Learning*, pages 160–163. Morgan Kaufmann Publishers Inc., 1989. ISBN 1-55860-036-1.

Mervyn Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):111–147, 1974.

David H. Wolpert. On the connection between in-sample testing and generalization error. *Complex Systems*, 6(1):47, 1992.

David H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.

David H. Wolpert and William G. Macready. Coevolutionary free lunches. *IEEE Transactions on Evolutionary Computation*, 9(6):721–735, December 2005.