What the No Free Lunch Theorems Really Mean; How to Improve Search Algorithms

David H. Wolpert

SFI WORKING PAPER: 2012-10-017

SANTA FE INSTITUTE

What the no free lunch theorems really mean; how to improve search algorithms

David H. Wolpert

Santa Fe Institute
1399 Hyde Park Road, Santa Fe, NM, 87501
and
Information Sciences Division
MS B256, Los Alamos National Laboratory, Los Alamos, NM, 87545
david.h.wolpert@gmail.com

May 22, 2012

1 Introducton

The first No Free Lunch (NFL) theorems were introduced in [9], in the context of supervised machine learning. These theorems were then popularized in [8], based on a preprint version of [9]. Loosely speaking, these original theorems can be viewed as a formalization and elaboration of concerns about the legitimacy of inductive inference, concerns that date back to David Hume (if not earlier). Shortly after these original theorems were published, additional NFL theorems that apply to search were introduced in [12].

The NFL theorems have stimulated lots of subsequent work, with over 2500 citations of [12] alone by spring 2012 according to Google Scholar. However arguably much of that research has missed the most important implications of the theorems. As stated in [12], the primary importance of the NFL theorems for search is what they tell us about "the underlying mathematical 'skeleton' of optimization theory *before the 'flesh' of the probability distributions of a particular context and set of optimization problems are imposed*". So in particular, while the NFL theorems have strong implications *if* one believes in a uniform distribution over optimization problems, in no sense should they be interpreted as advocating such a distribution.

In this short note I elaborate this perspective on what it is that is really important about the NFL theorems for search. I then discuss how the fact that there are NFL theorems for both search and for supervised learning is symptomatic of the deep formal relationship between those two fields. Once that relationship is disentangled, it suggests many ways that we can exploit practical techniques that were first developed in supervised learning to help us do search. I summarize some experiments that confirm the power of search algorithms developed in this way. I end by briefly discussing the various *free* lunch theorems that have been derived, and possible directions for future research.

2 The inner product at the heart of all search

Let X be a countable **search space**, and specify an **objective function** $f : X \to Y$ where $Y \subset \mathbb{R}$ is a countable set. Everything presented in this paper can be extended in a straightforward way to the case of a stochastic search algorithm, stochastic objective function, time-varying objective function, etc. Sometimes an objective function is instead called a "search problem", "fitness function", "cost function", etc.

In practice often one does not know f explicitly. This is the case whenever f is a "blackbox", or an "oracle", that one can sample at a particular x, but does not know in closed form. Moreover, often even if a practitioner does explicitly know f, they *act* as though they do not, for example when they choose what search algorithm to use on f. For example, often someone trying to solve a Traveling Salesman Problem (TSP) will use the same search algorithm for any TSP. In such a case, they are behaving exactly as they would if they only knew that the objective function is an TSP, without knowing specifically which one it is.

These kinds of uncertainty about the precise f being searched can be expressed as a distribution $P(f)$. Say we are given such a $P(f)$, along with a search algorithm, and a real-valued measure of the performance of that algorithm when it is run on any objective function f. Then we can solve for the probability that the algorithm results in a performance value ϕ. The result is an inner product of two real-valued vectors each indexed by f. (See Appendix.) The first of those vectors gives all the details of how the search algorithm operates, but nothing concerning the world in which one deploys that search algorithm. The second vector is $P(f)$. All the details of the world in which one deploys that search algorithm are specified in this vector, but nothing concerning the search algorithm itself.

This result tells us that at root, how well any search algorithm performs is determined by how well it is "aligned" with the distribution $P(f)$ that governs the problems on which that algorithm is run. For example, Eq. (2) means that the years of research into the traveling salesman problem (TSP) have (presumably) resulted in algorithms aligned with the implicit $P(f)$ describing traveling salesman problems of interest to TSP researchers.

3 The no free lunch theorems for search

The inner product result governs how well any particular search algorithm does in practice. Therefore, either explicitly or implicitly, it serves as the basis for any practitioner who chooses a search algorithm to use in a given scenario. More precisely, the designer of any search algorithm first specifies a $P(f)$ (usually implicitly, e.g., by restricting attention to a class of optimization problems). Then they specify a performance measure Φ (sometimes explicitly). Properly speaking, they should then solve for the search algorithm that the inner product result tells us will have the best distribution of values of that performance measure, for that $P(f)$. In practice though, instead informal arguments are often used to motivate the search algorithm.

In addition to governing both how a practitioner should design their search algorithm, and how well the actual algorithm they use performs, the inner product result can be used to make more general statements about search, results that hold for all $P(f)$'s. It does this by allowing us to compare the performance of a given search algorithm on different subsets of the set of all objective functions. The result is the no free lunch theorem for search (NFL). It tells us that if any search algorithm performs particularly well on one set of objective functions, it must perform correspondingly *poorly* on all other objective functions. This implication is the primary significance of the NFL theorem for search. To illustrate it, choose the first set to be the set of objective functions on which your favorite search algorithm performs better than the *purely random search algorithm* that chooses the next sample point randomly. Then the NFL for search theorem says that compared to random search, your favorite search algorithm "loses on as many" objective functions as it wins (if one weights wins / losses by the amount of the win / loss). This is true no matter what performance measure you use.

As another example, say that your performance measure prefers low values of the objective function to high values, i.e., that your goal is to find low values of the objective rather than high ones. Then we can use the no free lunch for search theorem to compare a hill-descending algorithm to a hill-*ascending* algorithm, i.e., to an algorithm that "tries" to do as poorly as possible according to the objective function. The conclusion is that the hill-descending algorithm "loses to the hill-ascending algorithm on as many" objective functions as it wins. The lesson is that without arguing for a particular $P(f)$ that is biased towards a set B of objective functions on which one's favorite search algorithm performs well, one has no formal justification that that algorithm has good performance.

A secondary implication of the NFL theorem for search is that *if* it so happens that you assume / believe that $P(f)$ is uniform, then the average over f's used in the NFL for search theorem is the same as $P(f)$. In this case, you must conclude that all search algorithms perform equally well for your assumed $P(f)$. This conclusion is only as legitimate as is the assumption for $P(f)$ it is based on. Once other $P(f)$'s are allowed, the conclusion need not hold.

An important point in this regard is that simply allowing $P(f)$ to be non-uniform, *by itself*, does not nullify the NFL theorem for search. Arguments that $P(f)$ is non-uniform in the real world do not, by themselves, establish anything whatsoever about what search algorithm to use in the real world.

In fact, allowing $P(f)$'s to vary provides us with a new NFL theorem. In this new theorem, rather than compare the performance of two search algorithms over all f's, we compare them overall $P(f)$'s. The result is what one might expect: If any given search algorithm performs better than another over a given set of $P(f)$'s, then it must perform corresponding worse on all other $P(f)$'s. (See appendix for proof.)

4 The supervised learning no free lunch theorems

The discussion above tells us that if we only knew and properly exploited $P(f)$, we would be able to design an associated search algorithm that performs better than random. This suggests that we try to use a search process itself to learn something about the real world's $P(f)$, or at least about how well one or more search algorithms perform on that $P(f)$. For example, we could do this by recording the results of running a particular search algorithm on a set of (randomly chosen) real-world search problems, and using those results as a "training set" for a supervised machine elearning algorithm that models how those algorithms compare to one another on such search problems. The hope would be that by doing this, we can give ourselves formal assurances that one search algorithm should be used rather than another, for the $P(f)$ that governs the real world.

The precise details of how well such an approach would perform depend on the precise way that it is formalized. However two broadly applicable restrictions on its performance are given by an inner product formula for supervised learning and an associated NFL theorem for supervised learning.

Just like search, supervised learning involves an input space X, an output space Y, a function f relating the two, and a data set of (x, y) pairs. The goal in supervised learning though is not to iteratively augment the data to find what x minimizes $f(x)$. Rather it is to take a fixed data set and estimate the entire function f. Such a function mapping a data set to an estimate of f (or more generally an estimate of a distribution over f's) is called a **learning algorithm**. We then refer to the accuracy of the estimate for x's that do not occur in the data set as **off-training set error**.

The supervised learning inner product formula tells us that the performance of any supervised learning algorithm is governed by an inner product between two vectors both indexed by the set of all target functions. More precisely, it tells us that as long as the loss function is symmetric, how "aligned" the supervised learning algorithm is with the real world (i.e., with the posterior distribution of target functions conditioned on a training set) determines how well that algorithm will generalize from any training set to a separate test set. (See appendix.)

This supervised learning inner product formula results in a set of NFL theorems for supervised learning, applicable when some additional common conditions concerning the loss function hold. The implications of these theorems for the entire scientific enterprise (and for trying to design good search algorithms in particular) are wide-ranging. In particular, we can let X be the specification of how to configure an experimental apparatus, and Y the outcome of the associated experiment. So f is the relevant physical laws determining the results of any such experiment, i.e., they are a specification of a universe. In addition, d is a set of such experiments, and h is a theory that tries to explain that experimental data ($P(h \mid d)$ being the distribution that embodies the scientist who generates that theory). Under this interpretation, off-training set error quantifies how well any theory produced by a particular scientist predicts the results of experiments not yet conducted. So roughly speaking, according to the NFL theorems for search, if scientist $\mathscr{A}$ does a better job than scientist $\mathscr{B}$ of producing accurate theories from data for one set of universes, scientist $\mathscr{B}$ will do a better job on the remaining set of universes. This is true even if both universes produced the exact same set of scientific data that the scientists use to construct their theories — in which case it is theoretically impossible for the scientists to use any of the experimental data they have ever seen *in any way whatsoever* to determine which set of universes they are in.

As another implication of NFL for supervised learning, take $x \in X$ to be the specification of an objective function, and say we have two professors, Smith and Jones, each of whom when given any such x will produce a search algorithm to run on x. Let $y \in Y$ be the bit that equals 1 iff the performance of the search algorithm produced by Prof. Smith is better than the performance of the search algorithm produced by Prof. Jones.[1] So any training set d is a set of objective functions, together with the bit of which of (the search algorithms produced by) the two professors on those objective functions performed better.

[1]Note that as a special case, we could have each of the two professors always produce the exact same search algorithm for any objective function they are presented. In this case comparing the performance of the two professors just amounts to comparing the performance of the two associated search algorithms.

Next, let the learning algorithm $\mathscr{C}$ be the simple rule that we predict y for all $x \notin d_X^m$ to be 1 iff the majority of the values in d_Y^m is 1, and the learning algorithm $\mathscr{D}$ to be the rule that we predict y to be -1 iff the majority of the values in d_Y^m is 1. So $\mathscr{C}$ is saying that if Professor Smith's choice of search algorithm outperformed the choice by Professor Jones the majority of times in the past, predict that they will continue to do so in the future. In contrast, $\mathscr{D}$ is saying that there will be a magical flipping of relative performance, in which suddenly Professor Jones is doing better in the future, if and only if they did worse in the past.

The NFL for supervised learning theorem tells us that there are as many universes in which algorithm $\mathscr{C}$ will perform worse than algorithm $\mathscr{D}$ — so that Professor Jones magically starts performing worse than Professor Smith — as there are universes the other way around. This is true even if Professor Jones produces the random search algorithm no matter what the value of x (i.e., no matter what objective function they are searching). In other words, just because Professor Smith produces search algorithms that outperform random search in the past, without making some assumption about the probability distribution over universes, we cannot conclude that they are likely to continue to do so in the future.

5 Exploiting the relation between supervised learning and search to improve search

Given the preceding discussion, it seems that supervised learning is closely analogous to search, if one replaces the "search algorithm" with a "learning algorithm" and the "objective function" with a "target function". So it should not be too surprising that the inner product formula and NFL theorem for search have analogs in supervised learning. This close formal relationship between search and supervised learning means that techniques developed in one field can often be "translated" to apply directly to the other field.

A particularly pronounced example of this occurs in the simplest (greedy) form of the Monte Carlo Optimization (MCO) approach to search [3]. In that form of MCO, one uses a data set d to form a distribution $q(x \in X)$ rather than (as in most conventional search algorithms) directly form a new x. That q is chosen so that that one expects the expected value of the objective function, $\sum_x q(x)f(x)$ to have a low value, i.e., so that one expects a sample of $q(.)$ to produce an x with a good value of the objective function. One then forms a sample x of that $q(.)$, and evaluates $f(x)$. This provides a new pair $(x, f(x))$ that gets added to the data set d, and the process repeats.

MCO algorithms can be viewed as variants of random search algorithms like genetic algorithms and simulated annealing, in which the random distribution governing which point to sample next is explicitly expressed and controlled, rather than be implicit and only manipulated indirectly. Several other algorithms can be cast as forms of MCO (e.g., the cross-entropy method [7], the MIMIC algorithm [1]). MCO algorithms differ from one another in how they form the distribution q for what point next to sample, with some not trying directly to optimize $\sum_x q(x)f(x)$ but instead using some other optimization goal.

It turns out that the problem of how best to choose a next q in MCO is formally identical to the supervised learning problem of how best to choose a hypothesis h based on a training set d [13, 5, 6]. If one simply re-interprets all MCO variables as appropriate supervised learning variables, one transforms any MCO problem into a supervised learning problem (and vice-versa). The rule for this re-interpretation is effectively a dictionary that allows us to transform any technique that has been developed for supervised learning into a technique for (MCO-based) search. Regularization, bagging, boosting, cross-validation, stacking, etc., can all be transformed this way into techniques to improve search.

As an illustration, cross-validation in supervised learning is a technique for using an existing training set d to choose a value for a hyperparameter arising in a given supervised learning algorithm. We can use the dictionary to translate this use of cross-validation from the domain of supervised learning into the domain of search. Training sets become data sets, and the hyperparameters of a supervised learning algorithm become the parameters of an MCO-based search algorithm. For example, a regularization constant in supervised learning gets transformed into the temperature parameter of a form of MCO that is very similar to simulated annealing. In this way using the dictionary to translate cross-validation into the search domain shows us how to use it on one's data set in search to dynamically update the temperature in the temperature-based MCO search algorithm. That updating proceeds by running the MCO algorithm repeatedly on subsets of one's *already existing* data set d. (No new samples of the objective function f beyond those already in d are involved in this use of cross-validation for search, just like no new samples are involved in the use of cross-validation in supervised learning.)

Experimental tests of MCO search algorithms designed by using the dictionary have established that they work quite well in practice [13, 5, 6]. Applying bagging and and stacking, in addition to cross-validation, have all been found to transform an initially powerful search algorithm into a new one with improved search performance.

Of course, these experimental results do not mean there is any formal justification for these kinds of MCO search algorithms; NFL for search cannot be circumvented. To understand in more detail why one cannot provide a formal justification for a technique like cross-validation in search, it is worth elaborating why there is not even a formal justification for using cross-validation in supervised learning. Let Θ be a set of learning algorithms. Then given a training set d, let scientist $\mathscr{A}$ estimate what f produced their training set the following way. First they run cross-validation on d to compare the algorithms in Θ. They then choose the algorithm $\theta \in \Theta$ with lowest such cross-validation error. As a final step, they run that algorithm on all of d. In this way $\mathscr{A}$ generates their final hypothesis h to generalize from d. Next let scientist $\mathscr{B}$ do the exact same thing, except that they use *anti*-cross-validation, i.e., the algorithm they choose to train on all of d is the element of Θ with *greatest* cross-validation error on d, not smallest error. By the NFL theorems for supervised learning, we have no *a priori* basis for preferring scientist $\mathscr{A}$'s hypothesis to scientist $\mathscr{B}$'s. Although it is difficult to produce f's in which $\mathscr{B}$ beats $\mathscr{A}$, by the NFL for supervised learning theorem we know that there must be "as many" of them (weighted by performance) as there are f's for which $\mathscr{A}$ beats B.

Despite this lack of formal guarantees behind cross-validation in supervised learning, it is hard to imagine any scientist who would not prefer to use it to using anti-cross-validation. Indeed, one can view cross-validation (or more generally "out of sample" techniques) as a formalization of the scientific method: choose among theories according to which better fits experimental data that was generated after the theory was formulated, and then use that theory to make predictions for new experiments. By the inner product formula for supervised learning, this bias of the scientific community in favor of using out-of-sample techniques in general, and cross-validation in particular, must correspond somehow to a bias in favor of a particular $P(f)$. This implicit prior $P(f)$ is quite difficult to express mathematically. Yet almost every conventional supervised learning prior (e.g., in favor of smooth targets) or non-Bayesian bias favoring some learning algorithms over others (e.g., a bias in favor of having few degrees of freedom in a hypothesis class, in favor of generating a hypothesis with low algorithmic complexity, etc.) is often debated by members of the scientific community. In contrast, nobody debates the "prior" implicit in out-of-sample techniques. Indeed, it is exactly this prior which justifies the ubiquitous use of contests involving hidden test data sets to judge which of a set of learning algorithms are best.

6 Free lunches and future research

There are many avenues of research related to the NFL theorems which have not yet been properly explored. Some of these involve *free lunch* theorems which concern fields closely related to search, e.g., co-evolution [11]. Other free lunches arise in supervised learning, e.g., when the loss function does not obey the conditions that were alluded to above [10].

However it is important to realize that none of these (no) free lunch theorems concern the *covariational* behavior of search and / or learning algorithms. For example, despite the NFL for search theorems, there are scenarios where, for some f's, $\mathbb{E}(\Phi \mid f, m, \mathscr{A}) - \mathbb{E}(\Phi \mid f, m, \mathscr{B}) = k$ (using the notation of the appendix), but there are no f's for which the reverse it true, i.e., for which the difference $\mathbb{E}(\Phi \mid f, m, \mathscr{B}) - \mathbb{E}(\Phi \mid f, m, \mathscr{A}) = k$. It is interesting to speculate that such "head-to-head" distinctions might ultimately provide a rationale for using many almost universally applied heuristics, in particular for using cross-validation rather than anti-cross-validation in both supervised learning and search.

There are other results where, in contrast to the NFL for search theorem, one does not consider fixed search algorithms and averages over f's, but rather fixes f and averages over algorithms. These results allow us to compare how intrinsically hard it is to search over a particular f. They do this by allowing us to compare two f's based on the sizes of the sets of algorithms that do better than the random algorithm does on those f's [4]. While there are presumably analogous results for supervised learning, which would allow us to measure how intrinsically hard it is to learn a given f, nobody currently knows. All of these issues are the subject of future research.

A Appendix of mathematical derivations

A.1 NFL and inner product formulas for search

To formalize the discussion in the text, it will be useful to use the following notation. Let a **data set** $d^m = \{d^m_X, d^m_Y\}$ be any set of m separate pairs $(x \in X, f(x))$. A **search algorithm** is a function $\mathscr{A}$ that maps any d^m for any $m \in \{0, 1, \ldots\}$ to an $x \notin d^m_X$. (Note that to "normalize" different search algorithms, we only consider their behavior in terms of generating new points to search that have not yet been sampled.) By iterating a search algorithm we can build successively larger data sets: $d^{m+1} = d^m \cup (A(d^m_X), f[A(d^m_X)])$ for all $m \geq 0$. So if we are given a **performance measure** $\Phi : d^m_Y \to \mathbb{R}$, then we can evaluate how the performance of a given search algorithm on a given objective function changes as it is run on that function.

This allows us to establish the inner product for search result, mentioned in the text. To begin expand

$$\begin{aligned} P(\phi \mid \mathscr{A}, m) &= \sum_{d^m_Y} P(d^m_Y \mid \mathscr{A}, m) P(\phi \mid d^m_Y, \mathscr{A}, m) \\ &= \sum_{d^m_Y} P(d^m_Y \mid \mathscr{A}, m) \delta(\phi, \Phi(d^Y_m)) \end{aligned} \tag{1}$$

where the delta function equals 1 if its two arguments are equal, zero otherwise.

Eq. (1) shows that the choice of search algorithm affects performance only through the term $P(d^m_Y \mid \mathscr{A}, m)$. In turn, this probability of d^m_Y under $\mathscr{A}$ is given by

$$\begin{aligned} P(d^m_Y \mid \mathscr{A}, m) &= \sum_f P(d^m_Y \mid f, m, \mathscr{A}) P(f \mid m, \mathscr{A}) \\ &= \sum_f P(d^m_Y \mid f, m, \mathscr{A}) P(f). \end{aligned} \tag{2}$$

As claimed, this is an inner product of two real-valued vectors each indexed by f: $P(d^m_Y \mid f, m, \mathscr{A})$ and $P(f)$. Note that the first of those gives all the details of how the search algorithm operates.

This notation also allows us to state the NFL for search theorem formally. Let B be any subset of the set of all objective functions, Y^X. Also let $\mathscr{A}$ be any search algorithm, and let Φ be any performance measure. Then Eq. (2) can be used to prove that

$$\sum_{f \in B} \mathbb{E}(\Phi \mid f, m, \mathscr{A}) = constant - \sum_{f \in Y^X \setminus B} \mathbb{E}(\Phi \mid f, m, \mathscr{A}) \tag{3}$$

where the symbol "\" indicates set subtraction and the constant on the right-hand side depends on Φ, but is independent of both $\mathscr{A}$ and B [12]. Expressed differently, Eq. (3) says that $\sum_f E(\Phi \mid f, m, \mathscr{A})$ is independent of $\mathscr{A}$. This is the NFL for search.

A.2 NFL for search when we average over $P(f)$'s

To derive the NFL theorem that applies when we vary over $P(f)$'s, first recall our simplifying assumption that both X and Y are finite (as they will be when doing search on any digital computer). Due to this, any $P(f)$ is a finite dimensional real-valued vector living on a simplex Ω. Let π refer to a generic element of Ω. So $\int_\Omega d\pi \; P(f \mid \pi)$ is the average probability of any one particular f, if one uniformly averages over all distributions on f's. By symmetry, this integral must be a constant, independent of f. In addition, as mentioned above, Eq. (3) tells us that $\sum_{f \in B} \mathbb{E}(\Phi \mid f, m, \mathscr{A})$ is independent of $\mathscr{A}$. Therefore for any two search algorithms $\mathscr{A}$ and $\mathscr{B}$,

$$\begin{aligned}
\sum_f \mathbb{E}(\Phi \mid f, m, \mathscr{A}) &= \sum_f \mathbb{E}(\Phi \mid f, m, \mathscr{B}), \\
\sum_f \mathbb{E}(\Phi \mid f, m, \mathscr{A})\Big[\int_\Omega d\pi \; P(f \mid \pi)\Big] &= \sum_f \mathbb{E}(\Phi \mid f, m, \mathscr{B})\Big[\int_\Omega d\pi \; P(f \mid \pi)\Big], \\
\sum_f \mathbb{E}(\Phi \mid f, m, \mathscr{A})\Big[\int_\Omega d\pi \; \pi(f)\Big] &= \sum_f \mathbb{E}(\Phi \mid f, m, \mathscr{B})\Big[\int_\Omega d\pi \; \pi(f)\Big], \\
\int_\Omega d\pi \; \sum_f \mathbb{E}(\Phi \mid f, m, \mathscr{A})\pi(f) &= \int_\Omega d\pi \; \sum_f \mathbb{E}(\Phi \mid f, m, \mathscr{B})\pi(f), \qquad (4)
\end{aligned}$$

i.e.,

$$\int_\Omega d\pi \; \mathbb{E}_\pi(\Phi \mid m, \mathscr{A}) = \int_\Omega d\pi \; \mathbb{E}_\pi(\Phi \mid m, \mathscr{B}). \qquad (5)$$

We can re-express this result as the statement that $\int_\Omega d\pi \; \mathbb{E}_\pi(\Phi \mid m, \mathscr{A})$ is independent of $\mathscr{A}$.

Next, let Π be any subset of Ω. Then our result that $\int_\Omega d\pi \; \mathbb{E}_\pi(\Phi \mid m, \mathscr{A})$ is independent of $\mathscr{A}$ implies

$$\int_{\pi \in \Pi} d\pi \; \mathbb{E}_\pi(\Phi \mid m, \mathscr{A}) = constant - \int_{\pi \in \Omega \setminus \Pi} d\pi \; \mathbb{E}_\pi(\Phi \mid m, \mathscr{A}) \qquad (6)$$

where the constant depends on Φ, but is independent of both $\mathscr{A}$ and Π. So if any search algorithm performs particularly well for one set of $P(f)$'s, Π, it must perform correspondingly *poorly* on all other $P(f)$'s. This is the NFL theorem for search when $P(f)$'s vary.

A.3 NFL and inner product formulas for supervised learning

To state the supervised learning inner product and NFL theorems requires introducing some more notation. Conventionally, these theorems are presented in the version where both the the learning algorithm and target function are stochastic. (In contrast, the restrictions for search — presented above — conventionally involve a deterministic search algorithm and deterministic objective function.) This makes the statement of the restrictions for supervised learning intrinsically more complicated.

Let X be a finite **input** space, Y a finite **output** space, and say we have a **target distribution** $f(y_f \in Y \mid x \in X)$, along with a **training set** $d = (d^m_X, d^m_Y)$ of m pairs $\{(d^m_X(i) \in X, d^m_Y(i) \in Y)\}$, that is stochastically generated according to a distribution $P(d \mid f)$ (conventionally called a **likelihood**, or "data-generation process"). Assume that based on d we have a **hypothesis distribution** $h(y_h \in Y \mid x \in X)$. (The creation of h from d — specified *in toto* by the distribution $P(h \mid d)$ — is conventionally called the **learning algorithm**.) In addition, let $L(y_h, y_f)$ be a **loss function** taking $Y \times Y \to \mathbb{R}$. Finally, let $C(f, h, d)$ be an **off-training set cost function**[2],

$$C(f,h,d) \propto \sum_{y_f \in Y, y_h \in Y} \sum_{q \in X \setminus d_X^m} P(q) L(y_f, y_h) f(y_f \mid q) h(y_h \mid q) \tag{7}$$

where $P(q)$ is some probability distribution over X assigning non-zero measure to $X \setminus d_X^m$.

All aspects of any supervised learning scenario — including the prior, the learning algorithm, the data likelihood function, etc. — are given by the joint distribution $P(f, h, d, c)$ (where c is values of the cost function) and its marginals. In particular, in [2] it is proven that the probability of a particular cost value c is given by

$$P(c \mid d) \;=\; \int df dh \; P(h \mid d) P(f \mid d) M_{c,d}(f, h) \tag{8}$$

for a matrix $M_{c,d}$ that is symmetric in its arguments so long as the loss function is. $P(f \mid d) \propto P(d \mid f) P(f)$ is the posterior probability that the real world has produced a target f for you to try to learn, given that you only know d. It has nothing to do with your learning algorithm. In contrast, $P(h \mid d)$ is the specification of your learning algorithm. It has nothing to do with the distribution of targets f in the real world. So Eq. (8)

References

[1] J.S. De Bonet, C.L. Isbell Jr., and P. Viola, *Mimic: Finding optima by estimating probability densities*, Advances in Neural Information Processing Systems - 9, MIT Press, 1997.

[2] Wolpert. D.H., *On the connection between in-sample testing and generalization error*, Complex Systems **6** (1992), 47–94.

[2]The choice to use an off-training set cost function for the analysis of supervised learning is the analog of the choice in the analysis of search to use a search algorithm that only searches over points not yet sampled. In both the cases, the goal is to "mod out" aspects of the problem that are typically not of interest and might result in misleading results: ability of the learning algorithm to reproduce a training set in the case of supervised learning, and ability to revisit points already sampled with a good objective value in the case of search.

[3] Y. M. Ermoliev and V. I. Norkin, *Monte carlo optimization and path dependent nonstationary laws of large numbers*, Tech. Report IR-98-009, International Institute for Applied Systems Analysis, March 1998.

[4] W. G. Macready and D. H. Wolpert, *What makes an optimization problem hard?*, Complexity **1** (1995), 40–46.

[5] D. Rajnarayan and David H. Wolpert, *Exploiting parametric learning to improve black-box optimization*, Proceedings of ECCS 2007 (J. Jost, ed.), 2007.

[6] ______, *Bias-variance techniques for monte carlo optimization: Cross-validation for the ce method*, arXiv:0810.0877v1, 2008.

[7] R. Rubinstein and D. Kroese, *The cross-entropy method*, Springer, 2004.

[8] C. Schaffer, *A conservation law for generalization performance*, International Conference on Machine Learning, Morgan Kaufmann, 1994, pp. 295–265.

[9] D. H. Wolpert, *The lack of a prior distinctions between learning algorithms and the existence of a priori distinctions between learning algorithms*, Neural Computation **8** (1996), 1341–1390,1391–1421.

[10] ______, *On bias plus variance*, Neural Computation **9** (1997), 1211–1244.

[11] D. H. Wolpert and W. Macready, *Coevolutionary free lunches*, Transactions on Evolutionary Computation **9** (2005), 721–735.

[12] D. H. Wolpert and W. G. Macready, *No free lunch theorems for optimization*, IEEE Transactions on Evolutionary Computation **1** (1997), no. 1, 67–82.

[13] D. H. Wolpert, D. Rajnarayan, and Bieniawski S., *Probability collectives in optimization*, Encyclopedia of Stastistics, 2012.