The Supervised Learning No-Free-Lunch Theorems

Conference Paper · January 2001

DOI: 10.1007/978-1-4471-0123-9_3

1 author:

David H. Wolpert
Santa Fe Institute
305 PUBLICATIONS **28,392** CITATIONS

The Supervised Learning No-Free-Lunch Theorems

David H. Wolpert
MS 269-1
NASA Ames Research Center
Moffett Field, CA 94035
dhw@ptolemy.arc.nasa.gov
http://ic.arc.nasa.gov/ic/projects/bayes-group/people/dhw/

June 22, 2001

Abstract

This paper reviews the supervised learning versions of the no-free-lunch theorems in a simplified form. It also discusses the significance of those theorems, and their relation to other aspects of supervised learning.

1 Introduction

1.1 Off-Training-Set Error

Many introductory supervised learning texts take the view that "the overall objective. . .is to learn from samples and to generalize to new, *as yet unseen* cases" (italics mine—see [7] for example). Similarly, it is common practice to try to avoid fitting the training set exactly, i.e., to try to avoid "overtraining." One of the major rationales given for this is that if one overtrains, "the resulting (system) is unlikely to classify *additional points* (in the input space) correctly" (italics mine—see [2]).

Such language implies—correctly—that one of the major topics of interest in supervised learning is behavior off the training set. There are many reasons for concerning oneself with such behavior:

i) In the low-noise regime, optimal behavior on the training set is determined by look-up table memorization, and the only interesting issues concern off-training-set behavior.

ii) In particular, in that low-noise regime, if one uses a memorizing learning algorithm, then for test sets overlapping with training sets the upper bound on test set error shrinks as m, the size of the training set, grows. If one does not correct for this when comparing behavior at different m values (as when investigating learning curves), one is comparing apples and oranges. When there is no noise, correcting for this effect by renormalizing the range of possible errors is equivalent to requiring that test sets and training sets be distinct. (See below.)

iii) In artificial intelligence—one of the primary fields concerned with supervised learning—the emphasis is often exclusively on generalizing to as-yet-unseen examples.

iv) Very often the stochastic process generating the training set is not the same as that governing testing. In such scenarios, the usual justification for testing with the same process that generated the training set (and with it the necessity that test sets are allowed to overlap with training sets) does not apply. An example is provided in the problem of protein tertiary structure prediction. Say that to do drug design we wish to use supervised learning to ascertain the mapping taking protein primary structure to the corresponding tertiary structure (i.e., the mapping from the protein's amino acid sequence to its three-dimensional conformation). In doing this we *already know* what tertiary structure corresponds to the primary structures in the training set. So to do design of new drugs we will never have those structures in the "test set" (i.e., in the set of amino acid sequences whose tertiary structure we wish to predict by using the training data). So we will only be interested in off-training-set error.

iv) Distinguishing the regime where test examples coincide with the training set from the one where there is no overlap amounts to splitting supervised learning along its natural "cleavage plane." Since behavior can be radically different in the two regimes, it is hard to see why one would not want to distinguish them.

v) There is a popular heuristic argument that the value of error when test sets are generated by the same process that generated training sets must become identical to off-training-set testing error when the size of the input space grows very large. If true, this would mean that one can ignore the distinction between the two kinds of error. However, this heuristic is often wrong. See, for example, the discussion on the "statistical physics supervised learning framework" in [11].

None of this means that one should never allow test sets to overlap with training sets. Rather it means that off-training-set testing is an issue of major importance which warrants scrutiny. However, the common mathematical frameworks for analyzing supervised learning — standard Bayesian analysis, sampling theory statistics, and the various computational learning theory approaches— all *require* that the test set be allowed to overlap with the training set. (They do this by having testing governed by an independent identically distributed (IID) rerunning of the process that generated the training set. See [11].) Therefore they mix together behavior on already-seen examples with that for not-already-seen examples, and accordingly cannot distinguish the two kinds of behavior.

The primary reason that the conventional frameworks allow the test set to overlap with the training set is that much of their research has been driven by the mathematical tools their practitioners are well-versed in rather than by consideration of what the most important issues in supervised learning are. Unfortunately, those tools are ill-suited for investigating off-training-set behavior. In fact, often the four frameworks use language that implies that their goal is understanding off-training-set behavior, even when they use a test set that can overlap with the training set. For example, in a paper by Blumer et al.,[4] in the context of noise-free supervised learning, we read that "the real value of a scientific explanation lies not in its ability to explain [what one has already seen], but in predicting events that have yet to occur," despite the fact that the subsequent analysis allows test sets to overlap with training sets.

This reflects the fact that the conventional frameworks are, to a degree, applications to supervised learning of paradigms *developed for different fields.* As such, they carry with them the entire cultural baggage—implicit assumptions and all—of the fields in which they originated. Such assumptions tend to be "background" in those originating fields, so it is natural that they are also background—though perhaps no longer so innocuous—when practitioners of those fields cross over to supervised learning.[1] Clearly then to investigate the crucially important topic of off-training-set error we need a different framework.

[1]As an aside, this tool-driven approach to theoretical learning has applied learning analogues. An important example is the manipulation of parametrizations of an object of interest with little concern for how those manipulations affect the object of interest itself. For example, many researchers impose priors on neural net weights, implement a bias in favor of fewer hidden neurons, etc. Often, for lack of an alternative, they do this without taking into account the ultimate effect on the direct object of interest, the input-output functions parametrized by those weights [10, 6]. One advantage of the framework presented below is that it is nonparametric and, therefore, helps focus attention directly on the object of interest rather than on parametrizations of that object.

1.2 Overview

In Section 2 of this paper I present a framework capable of addressing off-training-set error, the "Extended Bayesian Framework" (EBF — [9, 10, 14]). This framework has the other major advantage that it encompasses the conventional frameworks, illustrating the subtleties of how they are related, and suggesting variants of them. In Section 3 I present the "no-free-lunch theorems" based on the EBF. These are theorems that bound how much one can infer concerning the (off-training-set) generalization error probability distribution without making relatively strong assumptions concerning the real world. They serve as a broad context in which one should view the claims of any supervised learning framework.

2 The Extended Bayesian Formalism

2.1 Introduction

Intuitively, the EBF is just the conventional Bayesian supervised learning framework, extended to add one extra random variable. In addition to "costs" or "generalization errors" c, and "target" input-output relationships f, from which are produced m-element "training sets" d, there are also hypotheses h. These are the outputs of one's learning algorithm, made in response to d. (Loosely speaking, h can be viewed as the algorithm's guess for f.) The EBF[9, 10] is conventional probability theory applied to the space of quadruples $\{h, f, d, c\}$.

It is the inclusion of h in the space that allows the EBF to go beyond the conventional Bayesian supervised learning framework; without h, the EBF could not encompass the non-Bayesian frameworks like computational learning theory (e.g., PAC and the VC framework) and sampling theory statistics (e.g., confidence intervals). (See [13] for an overview of those frameworks.) To understand this, note that one's learning algorithm (or "generalizer") is given by the conditional probability distribution $P(h|d)$. There is no direct analogue to $P(h|d)$ in the conventional Bayesian framework. In particular, the "likelihood function" of the Bayesian framework, which gives the data generation process, is $P(d|f)$; the "posterior distribution" referred to in that framework usually means $P(f|d)$; and the "prior" referred to in that framework usually means $P(f)$. (More generally, the terms "prior" and "posterior" mean not-conditioned and data-conditioned, respectively.) Viewed another way, the conventional Bayesian framework has $P(h|d)$ pre-fixed, to be the "Bayes-optimal" $P(h|d)$ associated with $P(f|d)$ (see Section 5 of [13] and references therein). No allowance is made for $P(h|d)$'s like those considered in the other frameworks (and found in the real world) that are not Bayes-optimal for any $P(f|d)$.

The EBF itself imposes no restrictions on h and f; all such restrictions are imposed by $P(f, h, d, c)$. For example, if one's algorithm always makes guesses lying in a class H, then $P(h) = 0$ for all $h \notin H$. Similarly, if that the truths come from some class F, then $P(f) = 0$ for all $f \notin F$. If instead one's learning algorithm merely assumes that, and accordingly never makes a guess in F, then this means that $P(h) = 0$ for $h \notin F.P(h|d)$ can be "deterministic"—i.e., always guess the same h for the same d (as in a nearest neighbor algorithm)—or "stochastic"—i.e., potentially guess different h's even when d is fixed (as in backpropagation with a random initial weight).

The rest of this section presents a formal definition of the EBF. Those whose eyes turn glassy at such formal text are encouraged to skip to Section 3, referring back to the synopsis at the end of this section as needed.

2.2 Definition of the EBF

This subsection synopsizes the EBF; see [14] for a fully formal exposition.

- In this paper, random variables are denoted by capital letters, and instances of random variable with corresponding lower-case letters. For the purposes of this paper, there is no reason to be concerned with quasi-philosophical distinctions between random variables and "parameters." (See the discussion in Wolpert[15] on prior information.)

Whenever possible, "P" notation will be used: the arguments of a "P" indicates if it is a probability or a density or a mixture of the two and, if it involves densities, what random variables they are over. When more precision is required, "Pr(A)" will be used to indicate the probability of the event A, and lowercase "p" will be used to indicate a probability density. (So $P(z) = p_Z(z)$ is the probability density of random variable Z, evaluated at value z.) The notation "$E(Z|a)$" is defined to mean the expectation value $E(Z|A = a) = \int dz\, z\, P(z|a)$ (the integral being replaced by a sum if that is appropriate).

- The input space X has n elements, and the output space Y has r elements, where both n and r are countable (though perhaps infinite). Such discreteness of the spaces does not amount to an undue restriction, since it always holds in the real world, where measuring devices have finite precision and where the computers used to emulate learning algorithms are finite state machines. (Note also that our training and test sets will always be finite, and often we can restrict X to be just the input space values found in those sets.)

- The training set d consists of m ordered pairs of inputs and output values, $\{d_X(i), d_Y(i)\} : 1 \leq i \leq m$. The number of distinct values in d_X is indicated by m'.

- Let "f" be a function giving the probability of $y \in Y$ conditioned on $x \in X$. This is indicated by writing $P(y|x, f) = f_{x,y}$. (Note that f is a vector of real numbers, with components $f_{x,y}$.) When extra precision is required, "F" will indicate the random variable of which "f" is an instantiation. So, for example, $p_{Y|X,F}(y|x, f) = f_{x,y}$.

The variable "f" labels the "true" or "target" conditional distribution of y given x, in that training set output values are generated according to f. For the purposes of this paper, this means that $P(d_Y|d_X, f) = \Pi_{i=1}^{m} f_{d_{X(i)}, d_{Y(i)}}$, where d_X and d_Y are the inputs and outputs of the training set d. (Note that this equation need not fix anything concerning test set generation.)

- Let "g" be a function giving the probability of $x \in X$. This is indicated by writing $P(x|g) = g_x$. (Note that g is a vector of real numbers, with components g_x.) Training set input values are generated according to g: $P(d_X|g) = \Pi_i g_{d_{X(i)}}$. (This means that repeats are allowed in d_X.) In this paper it is stipulated that $P(d|f, g)$ (which equals $P(d_Y|d_X, f, g)P(d_X|f, g)$) is equivalent to $P(d_Y|d_X, f)P(d_X|g)$. This can be viewed as part of the definition of f and g.

In addition to such definitional requirements, here it will be convenient if certain other properties of g are assumed. In particular, it is assumed that g and f are statistically independent. This means that

$$\begin{aligned} P(d|f) = & \int dg P(d|f, g)P(g|f) = && \int dg P(d|f, g)P(g) \\ = & \; P(d_Y|d_X, f) \int dg P(d_X|g)P(g) = && P(d_Y|d_X, f) \int dg \; [\Pi_i g_{d_{X(i)}} P(g)]. \end{aligned}$$

It is also assumed that $P(g)$ is a delta function about some "sampling" distribution $\pi(x)$. Taken together these assumptions mean that $P(d|f) = \Pi_i[\pi(d_X(i)) f_{d_X(i), d_Y(i)}]$. For current purposes, this means that the variable g can henceforth be ignored.

Note that these assumptions concerning g are often unrealistic. Usually there is *some* coupling between f and g, and sometimes there is a lot. (This is the case in the drug design problem outlined above, for example.) Moreover, any techniques that try to use unsupervised learning to aid supervised learning (e.g., decision-directed learning [3]) implicitly assume that g and f are coupled. However, the vast majority of the work in the conventional frameworks implicitly assumes that g and f are independent. That is why that assumption is adopted here.[2]

- The fact that they themselves parameterize distributions does not forbid

[2] See Wolpert[8] for a discussion of the ramifications of the assumption that g is independent

either f or g from being arguments of probability distributions. For example, it is perfectly meaningful to write $P(f|d) = P(d|f)P(f)/P(d)$. In this way new data can update our estimation of what distribution generated that data.

- Let $h_{x,y}$ be the x-conditioned probability distribution over values y which is produced by our learning algorithm in response to d. Sometimes our algorithm's "output" is a quantity based on h (e.g., our algorithm might produce a decision of some sort based on h), and sometimes h itself is the output of our algorithm. $P(h|d)$ is the rule for how hypotheses are produced from training sets and is known as a "generalizer." Examples are back-propagation applied to neural nets and memory-based (i.e., nearest neighbor) reasoners.

If our algorithm's output is a guessed function from X to Y (rather than a guessed distribution), then we can view that output as an h where $h_{x,y}$ is of the form $\delta(y, \eta(x))$ for some function $\eta(.)$ parametrized by d ($\delta(.,.)$ being the Kronecker delta function). Such an h is "single valued." As examples, schemes like nearest-neighbor classifiers, and "Bayes-optimal" classifiers are deterministic and produce single-valued h. This should be contrasted with schemes like softmax applied to neural nets, in which the net gives a mapping from X to a distribution over Y, rather than from X to Y. (See Appendix 2 in [10].) If a generalizer produces a single-valued h that goes through all the elements of d, that generalizer is said to "reproduce" d.

- To simplify the exposition, unless explicitly stated otherwise, it will be assumed in this paper that our algorithm produces single-valued h's. (However, it is *not* necessarily assumed that the algorithm is deterministic.) It is similarly assumed that any f is of the form $\delta(y, \phi(x))$ for some single-valued function $\phi(x)$ from X to Y. (Or equivalently, it is assumed that $P(f)$ equals zero for any other f.) Note that due to the form of $P(d|f)$, this assumption is equivalent to requiring that the training set is generated without any noise. These two assumptions mean that, as restricted in this paper, the EBF is not capable of addressing problems where the "target" is a (non-delta function) distribution, and you wish to guess that target, so h, too, is a distribution. It is straightforward to use the EBF when none of these assumptions hold [12]. However doing so introduces extra mathematics which obscures the underlying issues.

To simplify notation, without loss of generality, from now on I will use the symbols "h" and "f" to refer to single-valued functions (i.e., I will rewrite $\eta(x)$ as $h(x)$, and $\phi(x)$as $f(x)$). It will be convenient to write $P(d_Y|d_X, f)$ as $\delta(d \subset f)$,

of f. For the views of conventional statistics on this issue, see also Titterington[5] and, in particular, the Dawid references therein concerning the "predictive" paradigm and the "diagnostic" paradigm.

i.e., $\delta(d \subset f) = 1$ if d lies on f, 0 otherwise. The notation is motivated by viewing f as a set of X–Y pairs, just like d. (Note though that repeats are allowed in d but not in f. So $\delta(d \subset f)$ can equal 1 even if the set d is not, formally speaking, contained in the set f.)

- One crucial stipulation—again, one that is adhered to in all if the conventional supevised learning frameworks—is that the guess the learning algorithm makes depends only on d. This means that $P(h|f, d) = P(h|d)$; if d is held fixed but f changes, the learning algorithm behaves the same. (Note that the learning algorithm can be based on *assumptions* concerning f. But those are embodied in $P(h|d)$, and do not change if f changes.) An immediate corollary is that $P(f|h, d) = P(f|d)$. Note the symmetry between h and f.

- In this paper only algorithms that work exclusively with full input-output (i-o) pairs are considered. As an alternative, one could have an algorithm that can try to learn even if some of the data is unlabeled or label-only (i.e., if we have a $d_X(i)$ without a corresponding $d_Y(i)$ or vice versa). For such cases, one must introduce a new random variable that specifies which of the elements of the data set consist of a full i-o pair, just an input, or just an output.[3]

- Now define a real-world random variable C which represents the "loss" (or what in some circumstances is called "cost" or "utility" or "value") associated with a particular f and h. Intuitively, C represents the real-world implications of a particular use of a learning algorithm. Formally, its meaning is set by the distribution $P(c|h, f, d)$. That distribution reflects how "test sets" are generated from f (and in particular whether it is in the same manner that training sets are generated), how big test sets are, how h is mapped (stochastically or otherwise) to a "decision" or an "action," how such a decision is combined with a test set to generate a real-world loss, etc. For current purposes, all of these kinds of details are irrelevant—only the final distribution $P(c|h, f, d)$ matters. This is similar to the fact that the only aspect of the learning algorithm that matters is the distribution $P(h|d)$. (The way a particular $P(h|d)$ is implemented—through a gradient descent, stochastic sampling, nearest neighbor rule, or whatever—is irrelevant as far as an investigation of generalization error is concerned.)

The "generalization error function" much discussed in the four frameworks is the expectation value $E(C|h, f, d)$. For example, when one is interested in

[3]Without such a variable, if we did not make our restriction there would be ambiguity in the notation: $P(h|d_X)$ could either mean the distribution over h's when the generalizer tries to learn from the inputs-only values in d_X, or it could mean the average over d_Y of the distribution when the generalizer tries to generalize from a set of input-output pairs, $\{d_X, d_Y\}$ (i.e., $\sum_{d_Y} P(d_Y|d_X)P(h|d_X, d_Y)$).

"average misclassification rate error," one might have $E(C|h,f,d) = E(C|h,f) = \sum_x \pi(x)[1 - \delta(f(x), h(x))]$. This is the average (according to $\pi(x)$) number of times across X that h and f differ.

In this paper it is assumed that we are interested in misclassification rates, and that $P(c|h,f,d)$ takes one of two forms. Formally, with $C = Er(F,H,D)$ (so that $P(c|f,h,d) = \delta[c, Er(f,h,d)]$), either C is independent of D and is given by

$$C = Er(F,H,D) = \textstyle\sum_x \pi(x)[1 - \delta(F(x), H(x))] \quad \text{("IID error")},$$

or C depends on D and is given by

$$C = Er(F,H,D) = \frac{\sum_{X \notin D_X} \pi(x)[1-\delta(F(x),H(x))]}{\sum_{X \notin D_X} \pi(x)} \quad \text{("off-training-set error")}.$$

Since $C = Er(F,H,D)$, we can write $E(C|f,h,d) = E(C|F = f, H = h, D = d) = Er(f,h,d)$, and in general $P(c \text{ obeys } \mathit{property}|\mathit{stuff}) = P(f,h,d \text{ such that } Er(f,h,d) \text{ obeys } \mathit{property}|\mathit{stuff})$. $Er(f,h,d)$ is called the "error function." Note that for any generalizer that reproduces the training set the off-training-set C is simply the IID C, renormalized so that the maximal value (over all f and h such that both $P(f|d)$ and $P(h|d)$ are nonzero) is always 1.

One should not confuse the error function with the "error surface" found in techniques like backpropagation. In the standard Bayesian formulation of backpropagation,[1, 10] the error surface is (the log of) $P(\mathbf{w}|d)$, where $\mathbf{w}$ is the weight vector parametrizing f. So, for example, the term in that error surface that equals the squared error on the training set simply reflects the assumption that $P(d|f)$ is created with Gaussian noise. That squared error need have nothing to do with $Er(f,h,d)$, even if $Er(f,h,d)$ is quadratic in $(f - h)$ (unlike the misclassification rate error functions analyzed in this paper).

- Although this paper restricts itself to the noise-free scenario, it is worth briefly pointing out some of the subtleties involved with noise. Usually the best way to allow for noise is to have f be a non-delta-function distribution over Y. However, it is common to instead adopt a "function + noise" scenario. In this scenario, f is still a single-valued function, but now $P(d|f)$ reflects the process of adding noise to f to create d (i.e., $P(d|f)$ can be nonzero even if d does not lie on f). If one uses this scenario, care should be exercised in the choice of the error function. In particular, often we are interested in whether h agrees with a sample of f where that sample is created *with the noise process.* (This is usually the case when we measure performance with a "test set," for example.) In general, this differs from whether h agrees with a noise-free sample of f, and the definitions given above for C should be adjusted accordingly.

- All that is necessary for the EBF to be an appropriate formalism for a particular problem is that the cost random variable only depends on the random variables f, h, and/or d. In some scenarios—none of which are considered in this paper—this is not the case, and it is appropriate to modify the EBF by adding some other variables to the space (e.g., g, or a hyperparameter). In other scenarios, although the standard EBF might suffice, a slight modification is more appropriate. (For example, if one is investigating the use of cross-validation over a fixed set of generalizers it makes sense to have one of the random variables in the EBF be the generalizer one chooses[9].) The underlying feature that unites all such variations of the EBF is that the space being analyzed includes hypotheses as well as targets.

• All of Bayesian supervised learning, computational learning theory, and statistical sampling theory can be cast in terms of the EBF, or slight variants of it. The converse does not hold. In particular, the "vanilla" versions of such frameworkscan usually can be defined in terms of the following abridged version of the EBF:

- n and r are the number of elements in the input and output spaces, X and Y, respectively.

- m is the number of elements in the (ordered) training set d. $\{d_X(i), d_Y(i)\}$ is the corresponding set of m input and output values. m' is the number of distinct values in d_X.

- Outputs h of the learning algorithm are always assumed to be of the form of a function from X to Y, indicated by $h(x \in X)$. Any restrictions on h are imposed by $P(f, h, d, c)$.

- The learning algorithm is given by $P(h|d)$. It is "deterministic" if the same d always gives the same h.

- "Targets" f are always assumed to be of the form of a function from X to Y, indicated by $f(x \in X)$. Any restrictions on f are imposed by $P(f, h, d, c)$.

- The "likelihood" is $P(d|f) = \delta(d \subset f)\Pi_i \pi(d_X(i))$, where "$\delta(d \subset f)$" equals 1 if d lies completely on f, 0 otherwise, and $\pi(x)$ is the "sampling distribution."

- The "posterior" is $P(f|d)$. In this paper, probability is *not* restricted to mean "degree of personal belief," as some conventional Bayesians define it. Accordingly, it is not true that the researcher automatically knows $P(f|d)$[15].

- $P(h|f,d) = P(h|d)$, $P(f|h,d) = P(f|d)$, and therefore $P(h,f|d) = P(h|d)P(f|d)$.
- The cost c associated with a particular h and f is either given by $Er(f,h,d) = \sum_x \pi(x)[1 - \delta(f(x), h(x))]$ ("IID error function"), or by the "off-training-set error" function, $Er(h,f,d) = \sum_{x \notin d_X} \pi(x)[1 - \delta(f(x), h(x))] / \sum_{x \notin d_X} \pi(x)$.
- The empirical misclassification rate $s \equiv \sum_{i=1}^{m} \{1 - \delta[h(d_X(i)), d_Y(i)]\}/m$.

Figure 1. Synoposis of the EBF.

In what follows, it is implicitly understood that assumptions like the exact absence of noise are replaced by the presence of infinitesimal noise. More precisely, it is implicit that no distributions ever equal zero *exactly*, although they might be arbitrarily close to zero. This is to ensure that we will never divide by zero in evaluating conditional probabilities.

3 The No-Free-Lunch Theorems

3.1 Presentation of the theorems

The theorems presented in this section bound how well a learning algorithm can be assured of performing in the absence of assumptions concerning the real world. For the sake of space, no proofs that appear in other papers are presented. The interested reader is referred[9, 12].

Theorem 1: $E(C|d)$ can be written as a (non-Euclidean) inner product between the distributions $P(h|d)$ and $P(f|d)$: $E(C|d) = \sum_{h,f} Er(h,f,d)P(h|d)P(f|d)$.

(Similar results hold for $E(C|m)$, etc.)

Theorem 1 says that how well you do is determined by how "aligned" your learning algorithm $P(h|d)$ is with the actual posterior, $P(f|d)$. This allows one to ask questions like "for what set of posteriors is algorithm G_1 better than algorithm G_2?" It also means that, unless one can somehow prove (!), from first principles, that $P(f|d)$ has a certain form, one cannot prove that a particular $P(h|d)$ will be aligned with $P(f|d)$ and, therefore, one cannot prove anything concerning how well that learning algorithm generalizes.

This impossibility of first-principles proofs can be formalized in a number of ways. One of them is as follows:

Theorem 2: Consider the off-training-set error function. Let "$E_i(.)$" indicate an expectation value evaluated using learning algorithm i. Then for any two learning algorithms $P_1(h|d)$ and $P_2(h|d)$, independent of the sampling distribution,

- Uniformly averaged over all f, $E_1(C|f,m) - E_2(C|f,m) = 0$;
- Uniformly averaged over all f, for any training set d, $E_1(C|f,d) - E_2(C|f,d) = 0$;
- Uniformly averaged over all $P(f)$, $E_1(C|m) - E_2(C|m) = 0$;
- Uniformly averaged over all $P(f)$, for any training set d, $E_1(C|d) - E_2(C|d) = 0$.

In other words, by any of the measures $E(C|d)$, $E(C|m)$, $E(C|f,d)$, or $E(C|f,m)$ (all generically known as "risks"), all algorithms are equivalent, on average.[4] Or to put it another way, for any two learning algorithms, there are just as many situations (appropriately weighted) in which algorithm one is superior to algorithm two as vice versa, according to any of the measures of "superiority" in Theorem 2.

[4]Note that one could argue that 2.i, for example, is misleading; different f's will have different probabilities in the real world, so a flat average over all f's is in some sense inappropriate. To correct this "misleading" nature of 2.i we are lead to consider averaging over all f according to a $P(f)$ which need not be uniform. Such an average equals $E(C|m)$. Now one can always construct a $P(f)$ to argue in favor of any particular learning algorithm. However, in almost all real-world supervised learning, we do not know $P(f)$. So we are led to ask if there is a $P(f)$ such that $E(C|m)$ is lower for algorithm one, and/or if there is also a $P(f)$ such that according to $E(C|m)$ algorithm two is superior. But this question has already been answered—in the affirmative—by 2.iii. In fact, given that we do not know $P(f)$, the obvious thing to do, if one wishes to compare two algorithms with the measure $E(C|m)$, is compare their averages over all $P(f)$—which 2.iii tells us makes all learning algorithms just as good as one another. Now one could try to "jump a level" yet again, and argue that some $P(f)$ are "more likely" than others, so one should not perform a flat average over all $P(f)$, etc. But the math responds the same way as it did to the objection to 2.i—in response to this new objection, one constructs new questions concerning probability distributions across the set of $P(f)$'s, questions whose answers again state that all algorithms perform the same, in the absence of information about the problem suggesting otherwise.

3.2 Examples

As an example, an algorithm that uses cross validation to choose amongst a pre-fixed set of learning algorithms does no better on average than one that does not. (However, since cross validation can only be viewed as a $P(h|d)$ if it is used to choose amongst a pre-fixed set of learning algorithms, Theorem 2 says nothing about cross validation "in general," when the set of generalizers is not pre-fixed. As another example of the no-free-lunch theorems, assume you are a Bayesian, and calculate the Bayes-optimal guess assuming a particular $P(f)$. (For example, you use the $P(h|d)$ that minimizes the data-conditioned risk $E(C|d)$, given your assumed $P(f)$.) You now compare your guess to that made by someone who uses a non-Bayesian method. Then 2.iv means (loosely speaking) that there are as many actual priors in which the other person has a lower data-conditioned risk as there are for which your risk is lower. Another set of examples is provided by all the heuristics that people have come up with for supervised learning: avoid "over-fitting," prefer "simpler" to more "complex" models, etc. Theorem 2 says that all such heuristics fail as often as they succeed.

Another example of Theorem 2 is given by the case where our learning algorithm is deterministic, and we have a particular training set d so the risk of interest is $E(C|d)$. The empirical misclassification rate s is fixed by d, since our algorithm takes d and gives h, which together with d gives s. Accordingly, $E(C|s, d) = E(C|d)$. Now assume that for our d, s happens to be very small (i.e., h and f agree almost always across the elements d_X). Assume further that our learning algorithm has a very low VC dimension. Since s is low, we might hope that that low VC dimension confers some assurance that our generalization error will be low. (This is one common way people try to interpret the VC theorems.) However, according to 2.iv, low s and low VC dimension provide no such assurances concerning off-training-set error. Either given d or (equivalently) given both d and s, no advantage is conferred as far as off-training-set behavior is concerned if s is low and one's algorithm happens to have low VC dimension.[5]

So all learning algorithms are the same in that: (1) by several definitions of "average," all algorithms have the same average off-training-set misclassification

[5]This result is reconciled with the usual VC theorems in the VC section of [11]. As an aside, it should be mentioned that the *only* reason this result might appear to be at odds with the VC theorems is because the usual statements of those theorems are guilty of sin number one from the introduction—the conditioning event is not specified. Accordingly, it is not immediately clear to the first-time reader of the VC theorems that they do not concern any of the conditioning events discussed in Theorem 2. (It should also be noted that the VC theorems concern IID error, not off-training-set error.)

risk and, therefore, (2) no learning algorithm can have lower risk than another one for all f, for all $P(f)$, for all f and d, or for all $P(f)$ and d. However, learning algorithms can differ in that: (1) for particular (nonuniform) $P(f)$, different algorithms have different data-conditioned risk (and similarly for other kinds of risk), and (2) for some algorithms there is a distribution-conditioning quantity (e.g., f) for which that algorithm is optimal (i.e., for which that algorithm beats all other algorithms), but some algorithms are not optimal for any value of such a quantity; and, more generally, (3) for some pairs of algorithms the no-free-lunch theorems are met by having comparitively many cases in which algorithm A is just slightly worse than algorithm B, and a few cases in which algorithm A beats algorithm B by a lot.

It is interesting to speculate about the possible implications of point (3) for cross validation. Consider two algorithms α and β. α is identical to algorithm A, and β works by using cross validation to choose between A and B. α and β must have the same expected error, on average. However, the following might be the case for many choices of A, B, $\pi(x)$, etc.: For most situations (i.e., most f or $P(f)$, depending on which of Theorem 2's averages is being examined) A and B have approximately the same expected off-training-set error, but β usually chooses the worse of the two, so in such situations the expected cost of β is (slighly) worse than that of α. In those comparitively few situations where A and B have significantly different expected off-training-set error, β might correctly choose between them, so the expected cost of β is significantly better than that of α for such situations. In other words, it might be a common case that when asked to choose between two generalizers in a situation where they have comparable expected cost, cross validation usually fails, but in those situations where the generalizers have significantly different costs, cross validation successfully chooses the better of the two. In such a case, cross validation still has the same average off-training-set behavior as any other algorithm. And there are actually more situations in which it fails than in which it succeeds. However, in such a case, cross validation has desirable minimax behavior. (It's important to note though that one can explicitly construct cases where cross validation does not have this desirable minimax behavior. See Section 8 of [11].)

3.3 Variants of Theorem 2

All of this applies to more than just off-training-set error. In general IID error can be expressed as a linear combination of off-training-set error plus on-training set error, where the combination coefficients depend only on d_X and $\pi(x \in d_X)$. So generically, if two algorithms have the same on-training-set behavior (e.g., they reproduce d exactly), the no-free-lunch theorems apply to their IID errors as well as their off-training-set errors.

In addition, there are a number of variants of Theorem 2, for example, dealing with noisy training sets, other conditional distributions (like $E(C|s, \{\text{other quantities}\})$), etc. For the sake of space they are not detailed here; the purpose of this section is only to present a sample of the no-free-lunch theorems, sufficient to provide a context for scrutinizing the results of the four frameworks.

3.4 Implications of Theorem 2 for the use of "test sets"

Theorem 2 also has implications for the (very common) use of a single test set to estimate c. Consider splitting d into two parts, d_1 and d_2. Training is done on d_1, and d_2 is a "test set" used to measure the resultant performance. (Note that since there can be duplicates in d, d_1 and d_2 might share input-output pairs.) The simplest situation to set up is where our error function runs over both $(d_1)_X$ and $(d_2)_X$—in this simple-minded version of things, no attention is being paid to off-training-set considerations, and the no-free-lunch theorems do not apply.

As an alternative, consider the off-training-set $Er(f, h, d)$, where "off-training-set" means off all of d. Now the no-free-lunch theorems apply; they tell us that behavior on d, which includes behavior on d_2, can tell us nothing about c, on average. (If this were not the case, behavior on d could be used to successfuly choose between competing algorithms.) The implication is that as far as off-d behavior is concerned, the most common procedure used for evaluating algorithms—examining their behavior on test sets—fails as often as it succeeds, on average. (Although, in general the minimax properties of this procedure might not be so poor—see the discussion above on cross validation's minimax behavior.)

On the other hand, assume that once d_1 is fixed d_2 is set to be the remaining pairs in d that are not also found in d_1. One might view such a d_2 as "off-training-set" in the sense of having no overlap with d_1. (Which definition of off-training-set is appropriate depends on which of the reasons listed in the introduction for being interested in off-training-set behavior applies.) If we adopt this definition, then it makes sense to redefine the error function to be the off-training-set error function for this training set d_1 (rather than for all of d). With this redefinition the error function runs over $(d_2)_X$ as well as $X - d_X$, and the no-free-lunch theorems do not apply; behavior on d_2 now can tell us something about the likely c value.

Indeed, for this scenario we might have d_2 be an IID sample of a process which, if infinitely repeated, gives our error function. We could then apply the usual variants of the central limit theorem to derive a confidence interval bounding the likely difference between empirical error on d_2 and the value of c. As with all confidence intervals though, this one comes with the major caveat that it does not directly give us what we want, which in this case is $P(\text{error on } d_2)$. See Sections 8 and 10 of [11]. (These points concerning test sets grew out of a conversation with Manny Knill.)

3.5 Intuition behind Theorem 2

The results presented above do not mean that the technique of cross validation does not work, or that the technique of using test sets to estimate error off of test sets does not work, or the like. Rather they mean that one can not formally justify these techniques (as far as expected off-training set error is concerned) without making assumptions. More practically, the results mean that if you are interested in off-training-set behavior, then using such a technique amounts to an assumption that $P(f)$ is not "typical," as measured by a uniform distribution over $P(f)$ (or that f is not typical as measured by a uniform distribution over f, or what have you.) As with all other assumptions, the validity of this one will vary from case to case.

Intuitively, it is not hard to see why an assumption must be implicit in techniques like cross validation. Consider the case where $P(f)$ is fixed and uniform over all f, and we are concerned with $E(C|d)$ for some particular d. Since $P(f)$ is uniform over all f, all f agreeing with d are equally probable. Accordingly, all possible patterns of f values outside of the training set are equally probable; the off-training-set world is essentially random. This means that building into a learning algorithm a preference for some particular outside-the-training-set pattern will not gain you anything; all algorithms are equal as far as off-training-set behavior is concerned when $P(f)$ is uniform.

To complete the intuitive justification for Theorem 2, note that since X and Y are finite, so is the set of all f's, and therefore $P(f)$ is a finite-dimensional real-valued vector living on the unit simplex. Accordingly, uniformly averaging all $P(f)$ results in a vector on that simplex all of whose components are equal—the uniform $P(f)$. So uniformly averaging all $P(f)$ (or uniformly averaging all f) is essentially equivalent to having a uniform $P(f)$. (The actual proof of Theorem 2 is a bit more complicated than this because we're not interested directly in the average of $P(f)$ but rather in the average of a distribution conditioned on $P(f)$ together with some quantity statistically coupled to $P(f)$. But the basic idea is the same.)

When considering things from the perspective of this particular argument, one should bear in mind that for $Y = \{0, 1\}$ and n large, uniform $P(f)$ (or equivalently a uniform prior over $P(f)$) means you are unlikely to find an f for which the proportion of all x such that $f(x) = 1$ differs much from .5. One might wish to instead have something like a uniform probability over the proportion of 1's in f (rather than over f's directly). For such a case, the f of all 1's is more probable than any particular f having both 1's and 0's in its outputs. A direct result is that if the training set has all its output values equal to 1, then the posterior favors the f having all 1's off the training set over the one having all 0's off the training set. (More generally, this is a situation that favors having more 1's off the training set than 0's.) In other words, in such a case we would have an automatic coupling between on- and off-training-set behavior. (Note though that Theorem 2 means that there also cases in which we have an automatic "anti-coupling" between on- and off-training-set behavior.) Similar arguments follow from the fact that for any fixed h, you are unlikely to find an f for which the number of x such that $f(x) \neq h(x)$ differs much from .5.

Another intuitive justification for the no-free-lunch theorems is based on viewing supervised learning "in reverse." Conventionally one views supervised learning as a process whereby $P(f)$ is sampled to set f, which is then sampled to get d, which is then used to get h. Viewed this way, it might seem odd that an h that agrees with d has no *a priori* correlation with f off of d. However, instead view the process as starting with a d, fitting h to it, and then considering all f going through those d. From this point of view, there is no reason at all to believe that h and f agree off of d.

As a final example of intuitive arguments supporting Theorem 2, simply note that it's very difficult to see how you could infer anything substantive about the likely c that accompanies use of a particular learning algorithm, unless you make an assumption for $P(c, f, d)$. And if you do make an assumption for $P(c, f, d)$, but it only concerns on-training set behavior (e.g., a noise-model), it's very difficult to see how you could infer anything substantive about the likely off-training-set c that accompanies use of a particular learning algorithm.

As these intuitive arguments suggest, there are many other aspects of off-training-set error which, although not actually no-free-lunch theorems, can nonetheless be surprising to those used to IID error. An example is the proof in Appendix 1 that in certain situations the expected off-training-set error grows as the size of the training set increases, even if one uses the best possible learning algorithm, the Bayes-optimal learning algorithm (i.e., the learning algorithm that minimizes $E(C|d)$—see the section in [11] on the Bayesian framework). In other words, sometimes the more data you have, the less you know about the off-training-set behavior of f, on average.

3.6 Error functions other than the misclassification rate

Finally, it should be pointed out that things are a bit messier when error functions other than the misclassification rate are considered. (In that the vanilla versions of the four frameworks do not consider such error functions, such error functions are not considered in this paper.) In particular, if the error function induces a geometrical structure over Y, then we can have *a priori* distinctions between learning algorithms.

An example is the case of quadratic error functions. For such functions, everything else being equal, an algorithm whose guessed Y values lie away from the boundaries of Y is to be preferred over an algorithm that guesses near the boundaries.[6] In addition, for such an error function, guessing an h that equals the Y-average of a stochastic algorithm's guess can never increase expected error beyond that of the original algorithm. Phrased differently, for a quadratic error function, given a series of experiments and a set of deterministic generalizers G_i, it is always preferable to use the average generalizer $G' \equiv \sum_i G_i / \sum_i 1$ for all such experiments rather than randomly to choose a new member of the G_i to use for each experiment. Intuitively, this is because such an average reduces variance without changing bias — see [12]. (Note though that this in no way implies that using G' for all the experiments is better than using any particular single $G \in \{G_i\}$ for all the experiments.)

Though important, such geometry-based distinctions do not say much about generalization once their strictures are met. In essence they serve as a zero-point or a baseline to generalization.

3.7 Summary

- Theorem 1 tell us that $E(C|d)$ is given by an inner product between the generalizer and the posterior.

- Theorem 2 tells us that if one is interested in off-training-set error, then any

[6]As an aside, note that in certain circumstances, this kind of effect will mean that, everything else being equal, one should prefer an h that stays away from the borders of Y, and therefore one should prefer an h that is relatively smooth. This is an example of how the choice of error function can affect how one regularizes (i.e., can affect the "bias" one imposes that competes with fitting the training set in determining h). This issue is addressed in Section 5 of [11].

pair of generalizers perform the same on average, where performance is measured by one of the distributions $E(C|d), E(C|f,d), E(C|m), E(C|f,m)$, or $E(C|s,d)$, and the averaging is over all f or all $P(f)$ as is appropriate.

- Some of the implications of Theorem 2 are that as far as off-training-set behavior is concerned, even techniques like cross validation and the use of test sets to estimate generalization error are unjustifiable unless one makes assumptions. They fail in as many scenarios as they succeed, loosely speaking.

- Appendix 1 of [11] demonstrates that even for the Bayes-optimal algorithm, $E(C|m)$ can rise with m for off-training-set error. So can $E(C|f,m)$.

- These results must be modified when there is a natural metric structure in the error function, since that structure allows for the *a priori* superiority of one algorithm over another.

4 Acknowledgments

This work was supported by NASA Ames Research Center.

References

[1] W. Buntine and A. Weigend. Bayesian back-propagation. *Complex Systems*, 5:603–643, 1991.

[2] T. Dietterich. Machine learning. *Ann. Rev. Comp. Sci*, 4:255–306, 1990.

[3] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2nd ed.)*. Wiley and Sons, 2000.

[4] A. Blumer et al. Occam's razor. *Info. Proc. Lett.*, 24:377–380, 1987.

[5] Titterington et al. *Statistical Analysis of Finite Mixture Distributions.* Wiley and sons, 1985.

[6] R. Neal. Priors for infinite networks. Department of Computer Science, University of Toronto, Technical Report, 1994.

[7] S.M. Weiss and C. A. Kulikowski. *Computer Systems that Learn.* Morgan Kauffman, 1991.

[8] D. Wolpert. Filter likelihoods and exhaustive learning. In *Computational Learning Theory and Natural Learning Systems Volume II: Natural Learning Systems*. MIT Press, 1994.

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

[10] D. H. Wolpert. Bayesian backpropagation over i-o functions rather than weights. In S. Hanson et al., editor, *Neural Information Processing Systems 6*. Morgan-Kauffman, San Mateo, CA, 1994.

[11] D. H. Wolpert. The relationship between pac, the statistical physics framework, the bayesian framework, and the vc framework. In *The Mathematics of Generalization* [13].

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

[13] D. H. Wolpert, editor. *The Mathematics of Generalization*. Addison-Wesley, New York, 1996.

[14] D. H. Wolpert. On bias plus variance. *Neural Computation*, 9:1211–1244, 1996.

[15] D. H. Wolpert. Reconciling bayesian and non-bayesian analysis. In *Maximum Entropy and Bayesian Methods 1993*. Kluwer Academic Press, 1996.