# Evaluating optimization algorithms: bounds on the performance of optimizers on unseen problems

David Corne
School of MACS, Heriot-Watt University
Edinburgh EH14 4AS
United Kingdom
dwcorne@gmail.com

Alan Reynolds
School of MACS, Heriot-Watt University
Edinburgh EH14 4AS
United Kingdom
A.Reynolds@hw.ac.uk

## ABSTRACT
In this extended abstract, we look at the common practice of using optimization problem test suites to develop and/or evaluate optimization algorithms, and bring to bear on this practice a number of results available from computational learning theory. This enables optimization algorithm developers to express principled quantitative bounds on the likely performance of their algorithms on unseen problem instances, on the basis of details of their experimental design and empirical results on training or test instances. We first recap some relevant results from computational learning theory, and then describe how optimization development practice can be suitably recast in a way that enables these results to be applied. We then briefly discuss some related implications. An updated version of this article and associated material, including statistical tables relating to generalization bounds, are provided at http://is.gd/evalopt.

## Categories and Subject Descriptors
I.2.6 Learning; I.2.8 Problem Solving, Control Methods and Search

## General Terms
Algorithms, Performance, Experimentation, Theory.

## Keywords
Computational learning theory, Optimization

## 1. INTRODUCTION
When new optimization algorithms are proposed, either for 'general' use or for a specific problem domain, it is rare that we are given guarantees or bounds concerning the performance of the new algorithms on unseen problems. Normally, an expectation of good performance is left as an implicit inference, based on performance over a test suite of problem instances. In a handful of cases, researchers have used a test set too. But we still release new optimisations with no principled expectation of their generalisation performance. Meanwhile, computational learning theory provides a growing collection of approaches to estimating generalisation performance in a variety of machine learning settings. Prominent in this line of work, for example, is Valiant's 'Probably Approximately Correct' (PAC) framework for estimating the generalisation performance of classifiers [4], while the more recent PAC-Bayes framework [3] is able to reason about estimated performance in the distributions over classifiers. Although the early work in this area tended towards delivering loose bounds on generalisation performance, there is considerable progress in recent years that has led to the derivation of tight bounds in a number of contexts, with practical value.

In this extended abstract, we introduce a way to map and adjust some basic frameworks from computational learning theory into a number of optimisation contexts. This leads to bounds on the performance of optimisation algorithms that depend on their derivation. This will provide, for example, ways to estimate the confidence we can have in whatever algorithm was found 'best' in a comparison study, as a function of various elements of the training regime applied. Results can be provided that are salient for various kinds of optimisation algorithm development study, ranging from the development of general optimisers (e.g. over standard test suites) or the development of an optimiser for a specific problem class (e.g. a space of vehicle routing problems associated with a particular delivery firm).

In the remainder of this extended abstract, we first recap the basic relevant elements of computational learning theory in section 2, then discuss their interpretation and use in the optimizer development context in section 3, and end with a brief discussion in section 4.

## 2. ELEMENTS FROM COMPUTATIONAL LEARNING THEORY
Computational learning theory aims, among other things, to provide estimates of how well a learned model or classifier will generalise over instances that were not seen during the classifier's development (i.e. during the training phase). There are a variety of learning models that vary in their basic assumptions and their applicability. Notable examples include the 'uniform convergence' model [5] and Probably-Approximately-Correct (PAC) Learning [4]. An accessible tutorial that surveys and discusses the main results is provided by Langford [2].

In the most straightforward results from computational learning theory, bounds on generalisation error are given as a function of the number of training examples, and the error on the test set. In the remainder of this section, we present some terminology and a number of results that we later refer to. Our terminology follows the formalism in [2], which in turn provided a compromise between varied formalisms in different branches of computational learning theory.

## 2.1 Terminology

We assume that training data comes in the form of ($x$, $y$) pairs, where $x \in X$ is an instance to be classified, and $y \in Y$ is the correct classification label. $D$ is an unknown distribution over $X \times Y$, from which a sequence $S$ of ($x$, $y$) pairs is drawn *independently* to form the training set. We assume $|S|=m$ - i.e. there are $m$ examples in the training set. A classifier, $c$, is a function that maps a member of $X$ to a member of $Y$. Primarily, we will be concerned with estimating the generalisation error of a classifier.

The 'True Error' of a classifier $c$ will be denoted $c_D$ - this is the actual (but typically unknown) error of the classifier evaluated over the distribution $D$ - in other words, this is the generalisation error that we usually aim to estimate. This is defined, again following [2] as:

$$c_D \equiv \Pr_{(x,y)\sim D}(c(x) \neq y) \qquad (1)$$

That is, this is the probability that any example $x$ drawn from $D$ will be mislabelled by the classifier $c$. Again, this is unknown, but we do have available to us the empirical error, $c_S$ or training error, calculated by testing our classifier on the sequence of examples $S$. We can define this as:

$$c_S \equiv \frac{1}{m} \sum_{(x,y)\in S} c(x) \neq y \qquad (2)$$

Now we are able to discuss the main results of interest in this abstract.

## 2.2 Bounds on Generalisation, Given Performance on a Test Set

The first, and central, result we consider is one that assumes we have a classifier $c$, that has been chosen or learned via some preliminary work. We then evaluate the performance of this classifier over the sequence of examples $S$. Importantly, the preliminary work that led to the derivation of $c$ must not have involved any of the examples in $S$. In context, we will see later that this is relevant to several types of study commonly reported in the optimisation literature, in which a previously published algorithm is evaluated over a set of test problems.



The central 'test set bound' is expressed in terms of the binomial distribution. Basically, given the assumed true error rate $c_D$, the number of errors we observe over a test set can be assumed to be drawn from a binomial distribution based on a 'success rate' of $c_D$. That is, we make much use of the quantity:

$$\text{Binomial}(m,k,c_D) = \sum_{i=0}^{k} \binom{m}{i} {c_D}^i (1-c_D)^{m-i} \qquad (3)$$

which gives the probability of up to $k$ errors in a series of $m$ trials, given that the probability of an error is $c_D$. This helps us see how a given true error rate can impact on the error rates we see on a test set, however it is more convenient to be able to reason in the other direction -- i.e. to learn something about the likely true error rate given the test set performance. So, we define the Binomial Tail Inversion [2], as:

$$\text{Btinv}(m,k,\delta) = \max_p \{ p : \text{Binomial}(m,k,p) \geq \delta \} \qquad (4)$$

That is, the binomial tail inversion tells us, for a given number of errors $k$ over a test set of size $m$, and for a given probability $\delta$, the highest true error rate that gives up to that many errors with probability at least $\delta$.

The test set upper bound itself can now be stated as follows:

$$\Pr_{S\sim D^m}(c_D \leq \text{Btinv}(m,c_S,\delta)) \geq 1-\delta \qquad (5)$$

In words, and valid for all distributions $D$, classifiers $c$, and for $\delta$ in (0, 1]: given that the classifier $c$ has recorded an error of $c_S$ on the sequence $S$ of examples drawn from $D$ (none of which was used in the training and/or choice of the classifier), the expression gives us probabilistic bounds on the true error rate. For example, we may choose to set $\delta$ at 0.1, and therefore can derive exactly the value $r$ such that $\Pr(c_D \leq r) \geq 0.9$.

A lower bound can be similarly derived, which is:

$$\Pr_{S\sim D^m}(c_D \geq \min_p \{ p : (1-\text{Binomial}(m,c_S,p)) \geq \delta \}) \geq 1-\delta \qquad (6)$$

Langford [2] shows how these can be approximated to the more tractable form:

$$\Pr_{S\sim D^m}\left( \left| c_D - \frac{c_S}{m} \right| \leq \sqrt{\frac{\ln\frac{2}{\delta}}{2m}} \right) \geq 1-\delta \qquad (7)$$

Meanwhile, a handy approximation for the special case when $c_S$ is 0 (i.e. we see zero error on the test set), gives us an upper bound for the true error in such a case, as follows:

$$\Pr_{S\sim D^m}\left( c_S = 0 \Rightarrow c_D \leq \frac{\ln\frac{1}{\delta}}{m} \right) \geq 1-\delta \qquad (8)$$

.

## 3. Optimizer Development as Classification

Consider a common scenario in the optimisation literature, whereby a proposed new optimisation algorithm is tested on a number of problems. Typically, we may test $k$ optimization algorithms on a set of $m$ test problems. $k$ might be in the region of 5, for example, comprising a new algorithm proposed by the authors, one or two variants thereof, and a handful of (ideally) 'state of the art' rival algorithms. $m$ may be around 20, comprising the functions from one or more standard test suites. Suppose further that the new algorithm $A$ outperforms the remaining $k-1$ algorithms on $q$ of the $m$ test problems.

Authors of such research papers tend to be at a loss for what can be claimed about algorithm $A$, other than the basic statement that $A$ was found best of the comparative set on $q$ of the $m$ problems. There is a vague implication, to the extent that $q$ is high and the comparative set is generally well-regarded, that this level of performance suggests strong relative performance for $A$ on other problems.

However, if we cast this scenario in terms of prediction theory, we can say much more than this. A way to cross the bridge between this scenario and computational learning theory is to posit an 'implicit classifier', $C$. This implicit classifier is a function that maps a problem instance (from a distribution of optimization problems $D$) to a label $T$ or $F$ (i.e. True or False), where $T$ indicates the prediction that algorithm $A$ will outperform the remainder of the comparative set on this problem. We can also see $C$ simply as the *claim* that we can label a new problem instance with $T$. The results described in section 2, and many similar, can be harnessed to quantitatively evaluate and bound that claim.

For example, suppose $m$=20 and $q$ =10. This is not an uncommon degree of outperformance of the comparative set in such papers, especially when the comparative set includes strongly performing rivals - the implicit claim towards generally good performance in such a paper is often supported by $A$'s achievement of more 'wins' than the rival algorithms, despite perhaps not outperforming them on a majority of the test problems. In this situation, we have $c_S$ = 0.5 - and the test set bounds, with $\delta$ set at 0.1, and using the more easily tractable approximation of equation (7), give us a "90% confidence interval" of [0.23, 0.77]. That is, given that $A$ outperformed the comparative set on 10 of the 20 test problems, we can have 90% confidence that its rate of outperforming the comparative set on new problems *from the same distribution* will be between 23% and 77%. I.e. it is quite unlikely that it will be best on fewer than 1 problem in 4, and similarly unlikely to be best on more than 3 problems in 4.

Besides providing quantitative bounds, we can also use equations (5) and (6) (or the approximations, (7) and (8)) to derive threshold performance values based on a given number of test problems. E.g., given $m$=20, the highest empirical error rate that yields a 90% upper bound of 0.5 is 0.23. So, when using a suite of 20 problems, algorithm $A$ needs to outperform its comparative set on 16 of the 20 problems before we can have 90% confidence that it will be better on "unseen instances" at least a little more than half of the time. A collection of such threshold performance values is provided at http://is.gd/evalopt.

### 3.1 Bounds Arising From Performance on a Training Set

Our deliberations so far apply to the situation when we are *evaluating* an optimiser, in the context of a set of other optimisers, over a given suite of test problems $S$, and where there was no bias or prejudice in the choice or development of the optimiser itself arising from the use of $S$. There are certainly cases where the scenario clearly applies. For example, an algorithm may be developed and tuned by first using a common test suite of function optimisation problems, designed to be difficult, rather than to reflect any particular distribution of real-world instances. Then, in a later experiment, the algorithm is compared with some others over an entirely different suite – e.g. a collection of real-world filter design problems. The scenario to which the discussion so far applies is that of evaluating the generalisation performance of our algorithms in the latter case.

However, in the latter case, it is also common to then perform tuning experiments, or other experiments that effectively modify the optimiser to yield one that performs as well as possible over this second set of real-world cases. More generally, what can we say about the optimizer-development scenario in which algorithm $A$ achieves error rate $c_S$ over problem set $S$, following modification and tuning that involved problem set $S$? I.e. this is the situation in which $S$ is used to 'train' $A$ – we can generally expect that $c_S$ will tend to be lower in this scenario (perhaps often zero), but our expectations for $c_D$ should be downgraded, to allow for overfitting on $S$. In general, the way to evaluate performance in such cases is to use an unseen test set, in which case the material discussed so far directly applies. However, this is often problematic in machine learning practice, where the set of available classified examples is already quite small, and splitting into training and test set excludes too much information from the training process, outweighing the benefits of a less biased estimate of performance. This is sometimes the case in optimizer development scenarios where, for example, runs on the available problem instances are highly time consuming. More often, the use of a test set is in fact quite feasible in optimiser development scenarios, but rarely done.

In either case, we can make use of the Occam's Razor bound [1], which is the same as equation [5] (we consider now only the upper bound on $c_D$, which is more practically salient), except for the replacement of $\delta$ by $\delta P(c)$, where $P(c)$ is the prior probability of our classifier $c$, and the qualification over all $c$.

$$\Pr_{S\sim D^m}(\forall c:\ c_D \le \mathrm{Btinv}(m, c_S, \delta P(c))) \ge 1-\delta \qquad (9)$$

In typical machine learning contexts in which this result is used, we would normally set $P(c)$ to $1/|H|$ where $H$ is a set of classifiers somehow under consideration, all deemed equally probable before we have knowledge of the set $S$. For example, $H$ may be the set of all decision trees of a given depth over a particular set of operators and leaves. Notice that this often leads to rather small values for $P(c)$ and consequently a rather loose upper bound (easier to see via doing the same substitution in equation (7)).

A way that we can use the Occam's razor bound in the optimization algorithm evaluation context is as follows. Suppose we have trained/tuned algorithm $A$ on our set $S$ of $m$ problems – the tuned $A$ achieves error rate of $c_S$ on these problems, in the

sense that, on a proportion $1-c_S$ of these problems, $A$ outperforms a comparative set of (including $A$) $k$ algorithms. Now, recall our 'implicit classifier', which labels a problem instance with $T$ or $F$ in respect of the claim "$A$ outperforms the comparative set on this instance". We can consider this classifier to be a choice from the set of $k$ similar classifiers, one for each algorithm in the comparative set. Its prior probability is therefore $1/|H|$.

Given any set of comparative algorithms, the researcher is of course free to use equation 2.4.1 and, for example take $|H|=2$ and specialize the claim to the relative performance of algorithm $A$ and just one from the comparative set (e.g. the state of the art competitor). But in such cases, further claims involving either $A$ or this single comparator cannot be made on the basis of the same set of experiments, without needing to apply a Bonferroni-style correction.

In both the cases of test set and training set bounds, we may wish to make different kinds of claims. E.g. rather than claims in terms of the comparative set, we may wish to evaluate the degree to which algorithm $A$ returns a result within 10% of a known optimum value, or achieves a certain fitness level within a given amount of time. In the test-set scenarios (where the test set of instances was not used at all in the development of the algorithm under study), the approach we have outlined is directly applicable. In the training scenario, applying these results in a principled way is less clear. However it is better in any case, and more often feasible in the optimization context, to evaluate the tuned or trained algorithms on an unseen test set, in which case the test set scenario applies without the necessity to have estimated priors over the implicit classifiers.

To support providing quantified bounds on the generalization performance of optimization algorithms, we provide a collection of tables at http://is.gd/evalopt, covering the test set and training set scenarios, providing bounds on performance that arise from each of several combinations of empirical error $c_S$ and number of problems $m$.

## 4. Discussion

Perhaps the most pressing point of discussion that we have not yet touched upon is the nature of the distribution of problems $D$. The generalization error bounds we present all need to be qualified with the statement that they apply over unseen problem instances that are drawn from the same distribution as the set $S$ of $m$ training or test instances. In some scenarios this is unproblematic – for example we may have a parameterized distribution of problems from a certain domain (*NK* problems, a space of quadratic assignment problems, a set of graphs to be treated as colouring problems, and so on). But in many scenarios, the test suite is a well known eclectic collection of problems, such as the CEC2005 functions. In such cases, it is entirely unclear what useful meaning the generalization statement has. However this observation cuts two ways – the computational learning theory context can be seen as emphasizing to us the impoverished nature of the common practice of reporting optimization algorithm performance on such a test suite. If such a test suite is useful at all (and they seem to be, since there is no doubt that there has been progress in optimizer development facilitated by their use), we can conclude that generalizing performance properties from empirical performance over the suite has some value, and in that context the results in this paper provide some quantitative bounds that can be used. However, clearer statements can be made on the basis of parameterized (or similar) distributions of problems, and/or on the basis of performance on an unseen problem set, and it is therefore strongly recommended that optimizer development practice adopts such approaches much more widely. Arguably, developing an optimizer for real-world problems by tuning on a standard contrived-problem test suite may be akin to developing a classifier for ovarian cancer by training it to classify species of iris.

An updated version of this document, with associated tables, is under maintenance at http://is.gd/evalopt.

## 5. REFERENCES


[1] Blumer, A., Ehrenfeucht, A, Haussler, D., Warmuth, M. (1987) Occam's Razor, *Information Processing Letters*, **24**:377—380.

[2] Langford, J. (2005) Tutorial on Practical Prediction Theory for Classification, *Journal of Machine Learning Research* **6** (2005) 273–306.

[3] McAllester, D. (1999) PAC-Bayesian Model Averaging, in Proc. *Annual Conf. on Computational Learning Theory* (COLT), pp. 164—170.

[4] Valiant, L.G. (1984) A theory of the learnable. *Communications of the ACM*, **27**(11):1134–1142.

[5] Vapnik, V. N., Chervonenkis, Y. (1971) On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, **16**(2):264–280.