Lazy Paired Hyper-Parameter Tuning

Alice X. Zheng and **Mikhail Bilenko**
Microsoft Research
1 Microsoft Way
Redmond, WA 98052, U.S.A.
{alicez,mbilenko}@microsoft.com

Abstract

In virtually all machine learning applications, hyper-parameter tuning is required to maximize predictive accuracy. Such tuning is computationally expensive, and the cost is further exacerbated by the need for multiple evaluations (via cross-validation or bootstrap) at each configuration setting to guarantee statistically significant results. This paper presents a simple, general technique for improving the efficiency of hyper-parameter tuning by minimizing the number of resampled evaluations at each configuration. We exploit the fact that train-test samples can easily be *matched* across candidate hyper-parameter configurations. This permits the use of paired hypothesis tests and power analysis that allow for statistically sound early elimination of suboptimal candidates to minimize the number of evaluations. Results on synthetic and real-world datasets demonstrate that our method improves over competitors for discrete parameter settings, and enhances state-of-the-art techniques for continuous parameter settings.

1 Introduction

Supervised machine learning techniques are increasingly being used as black-box components by engineers who expect them to produce high-accuracy predictions automatically. A key obstacle to this is the sensitivity of learning algorithms to hyper-parameter settings, also known as the model selection problem. To maximize accuracy on a particular dataset, one typically needs to identify optimal, domain-specific values for some set of algorithm-specific variables. Learning rates, regularization coefficients and pre-processing options are all examples of parameters that have significant impact on accuracy, and hence must be tuned by practitioners by performing repeated experiments on validation sets.

However, the variance inherent in machine learning experiments present a challenge to efficient hyper-parameter search. Due to the stochastic nature of training and testing on finite datasets, each candidate hyper-parameter configuration must be evaluated on multiple samples from the source distribution in order to identify a statistically meaningful optimum. Cross-validation and bootstrap are the most popular sampling-based approaches for estimating statistical significance. These methodologies inevitably increase the computational cost of experiments, amplifying the number of evaluations for each considered hyper-parameter configuration. This amplification persists with any of the many techniques for global optimization that can be employed to intelligently search the space of potential configurations, such as sequential model-based (e.g., Bayesian) optimization, and directed, random or evolutionary search methods. Hence, all of these approaches can benefit from eliminating unnecessary resampled evaluations at every candidate point.

Two lines of prior research are relevant for developing solutions for this problem when there is a finite set of candidate hyper-parameter configurations. Maron and Moore [1994] proposed Hoeffding Races for early elimination of underperforming candidates in a leave-one-out validation setting. An alternative approach has been pursued by multi-armed bandit algorithms, where upper bounds are employed to favor promising candidates and account for confidence in accuracy estimates from repeated evaluations [Lai and Robbins, 1985, Auer *et al.*, 2002a,b, Gittins *et al.*, 2011].

However, these approaches assume that each sample for every configuration is constructed either independently or adversarially. In the context of hyper-parameter optimization, such rigid assumptions are unnecessary if samples chosen for cross-validation or bootstrap folds are consistent across configurations. Then, evaluation results can be compared using stronger statistical tests for matched samples, leading to significant efficiency improvements.

We demonstrate that these improvements can be reaped by incorporating paired hypothesis tests with hyper-parameter search techniques for both discrete and continuous parameter spaces. For the discrete case, we show that paired tests allow us to identify winners much more efficiently than the standard racing or multi-armed bandit strategies that do not take matched-pairing into account. For continuous hyper-parameter spaces, paired tests can be organically integrated inside search techniques such as Nelder-Mead, a classic simplex-search algorithm, resulting in reaching the optimum

with much fewer evaluations than full cross-validation, and actually improving convergence of the overall algorithm. The efficiency gains achieved with our algorithm are illustrated by experimental results on real and synthetic datasets.

2 Problem Definition

Hyper-parameter tuning for machine learning is a stochastic optimization problem. The stochasticity arises from the fact that train and test sets are finite samples (typically assumed to be i.i.d.) from an underlying joint distribution of data points and prediction targets, as well as from possible randomness in the learning algorithm itself. Let θ denote the hyper-parameters of the learning algorithm, X a random variable representing the source(s) of stochasticity, and $g(\cdot)$ a performance metric chosen by the user for the prediction task at hand. For binary classification, $g(\cdot)$ may be the area under ROC curve (AUC), and for regression $g(\cdot)$ may be the negative mean squared loss. The problem of hyper-parameter tuning can then be defined as:

$$\theta^* = \max_{\theta} \mathbf{E}\left[g(\theta, X)\right]. \tag{1}$$

For the moment, assume that the only source of stochasticity is the data itself, so that X represents simply the data distribution. Since the true mean for this unknown distribution cannot be computed directly, the empirical mean is optimized over random samples $\{X^{(1)}, \ldots, X^{(n)}\}$, an approach known as sample average approximation [Srebro and Tewari, 2010]:

$$\max_{\theta} \frac{1}{n} \sum_{i} g(\theta, X^{(i)}). \tag{2}$$

In the hyper-parameter tuning process, the i-th sample $X^{(i)}$ can be made identical for all candidate values of θ. For example, this is true in the case of cross-validation or bootstrap starting from the same initial random seed, generating the same sequence of training and validation datasets from an initial training set. In this work, we demonstrate that such tying of evaluation subsets enables *lazy* hyper-parameter tuning, where the optimal θ^* can be found without performing all n evaluations for every candidate, resulting in significant computational savings.

We note that the evaluation of $g(\cdot)$ is carried out in two steps. First, the learning algorithm determines the model parameters (let's call them λ) via a modeling objective function $f(\cdot)$ evaluated on the trainig data X_T . The learned model is then evaluated on a validation set X_V under the metric $g(\cdot)$.

$$\lambda^*_\theta = \arg\max_{\lambda} \mathbf{E}\left[f(\lambda, X_T; \theta)\right] \tag{3}$$

$$\theta^* = \arg\max_{\theta} \mathbf{E}\left[g(\theta, \lambda^*_\theta, X_V)\right]. \tag{4}$$

Here, X_T, X_V, as well as the optimization of Eq. (3) may all be stochastic. Hence the empirical average of $g(\cdot)$ should be computed over samples of all three. In practice, people often forgo cross-validation inside the grid search loop because it is computationally expensive. This may be justifiable if the variance of the evaluations is low—a possibility when the validation or hold-out set is large, $g(\cdot)$ is itself an empirical average over the validation set, and there is no stochasticity within the learning procedure (Eq. (3)). In general, however, one should perform cross-validation or bootstrapping to account for the variance of the estimates of $g(\cdot)$.

Another notable challenge in this optimization problem is that there is no gradient information: $f(\cdot)$ and $g(\cdot)$ represent black-box learning procedure whose output cannot be expressed as a closed-form function of hyper-parameters. Hence, existing approaches to hyper-parameter optimization are "gradient-free" algorithms that either approximate the uphill direction via simplex search (e.g., Nelder-Mead), or model the evaluation (response) surface with a surrogate function (e.g., Gaussian Process optimization) [Conn *et al.*, 2009]. Thus, finite-difference variants of stochastic gradient methods are not competitive for this problem, as they require many iterations to converge due to noise in the gradient approximations.

3 Lazy Paired Hyper-Parameter Tuning (LaPPT)

In this section, we demonstrate that performing full cross-validation on all $X^{(i)}$'s is not necessary if hypothesis testing is embedded in the search algorithm, so that the only evaluations performed are those that are statistically necessary to identify leading configurations.

3.1 Background: Statistical Hypothesis Testing

The classic example of an application for statistical hypothesis testing is the two-sample location test, where one observes two samples X and Y, each of size n and drawn from two distributions from the same family with possibly different locations (medians or means). Let μ_X and μ_Y denote the true means and $\tau = \mu_X - \mu_Y$ their difference. The null hypothesis $\mathcal{H}_0$ may assert that $\tau = 0$, while the alternative hypothesis $\mathcal{H}_1$ may test for $\tau > 0$, $\tau < 0$, or $\tau \neq 0$. If the distributions are Gaussian and the samples are matched, i.e., X and Y contain the same individuals under two different "treatments," then the *matched-pairs t-test* can decide whether to accept or reject the null hypothesis while guaranteeing a false positive rate. The test computes the t-statistic $T_{X-Y} = \frac{\hat{\mu}_{X-Y}}{\hat{\sigma}_{X-Y}/\sqrt{n}}$, where $\hat{\mu}_{X-Y} = \sum_i (x_i - y_i)/n$ and $\hat{\sigma}^2_{X-Y} = \sum_i ((x_i - y_i) - \hat{\mu}_{X-Y})^2/(n-1)$ are, respectively, the sample mean and variance of the matched differences. Under the null hypothesis, T_{X-Y} follows the Student's t-distribution with $n-1$ degrees of freedom. Let $\mathbb{T}_{n-1}$ denote the cdf of this distribution.

The statistical hypothesis testing procedure [Lehmann and Romano, 2005] explicitly controls for the probability of false positives (Type I error): it would only reject the null hypothesis when the probability of observing the statistic falls below a given threshold α (the significance level). Let t_α denote the α-th quantile of the Student's t-distribution, i.e., $\mathbb{T}(t_\alpha) = \alpha$. If the alternate hypothesis is $\{\tau > 0\}$, then we would reject $\mathcal{H}_0$ and accept $\mathcal{H}_1$ if $T_{X-Y} > t_{1-\alpha}$.

The false negative probability (Type II error) can be controlled by increasing the number of evaluations at each point in a procedure known as *power analysis*. The power of a statistical test is the probability that it correctly rejects $\mathcal{H}_0$ when

$\mathcal{H}_1$ is true. Suppose $\mathcal{H}_1 = \{\tau > 0\}$. Then the power of the matched-pairs t-test may be written as:

$$\mathbf{P}\left(\text{Reject } \mathcal{H}_0 \mid \mathcal{H}_1 \text{ is true}\right) = \mathbf{P}\left(T_{X-Y} > t_{1-\alpha} \mid \tau > 0\right)$$
$$= \mathbf{P}\left(T_{X-Y} - \frac{\tau}{\hat{\sigma}_{X-Y}/\sqrt{n}} > t_{1-\alpha} - \frac{\tau}{\hat{\sigma}_{X-Y}/\sqrt{n}} \mid \tau > 0\right)$$
$$= 1 - \mathbb{T}_{n-1}(t_{1-\alpha} - \frac{\tau}{\hat{\sigma}_{X-Y}/\sqrt{n}}). \quad (5)$$

Let β denote the acceptable false negative rate. Given β and a value for τ, we can compute the necessary sample size to guarantee that the probability of a false negative is below β when the test accepts the null hypothesis. This can be used to determine how many additional evaluations to make at each hyper-parameter configuration.

3.2 Lazy Paired Tuning for Discrete Hyper-parameters

Algorithm 1: Finite Lazy Paired Parameter Tuning

Input: $C = \{\theta_1 \dots \theta_m\}$: Set of candidate hyper-parameter configurations
Input: α: Significance level
Input: β: False negative rate
Input: n_o: Number of initial evaluations
Input: $(X_{T_1}, X_{V_1}), \dots, (X_{T_n}, X_{V_n})$: Sequence of training and validation sets
Output: θ^*: Best configuration

for $i = 1 \dots m$ **do**
  for $\ell = 1 \dots n_0$ **do**
    $R_{i\ell} = g(\theta_i, X_{T_\ell}, X_{V_\ell})$
while $|C| > 1$ **do**
  for $\theta_i \in C$ **do**
    for $\theta_j \neq \theta_i \in C$ **do**
      result = LazyPairedTCompare(R_i, R_j, α, β)
      if *result* < 0 **then** $C = C \backslash \theta_i$
      else if *result* > 0 **then** $C = C \backslash \theta_j$
return C

For hyper-parameter spaces that are discreet and hence have no structure to exploit directly, selecting the best configuration can be viewed as a race over a sequence of rounds corresponding to fold evaluations [Maron and Moore, 1994]. In each round, some candidates are eliminated, and further evaluations are scheduled for the others. At the beginning of a round, all current candidates are compared pairwise. If a test accepts the alternate hypothesis that candidate i performs worse than j, then i is eliminated.[1] Conversely, if the test accept the null hypothesis of equal performance, we compute the number of evaluations needed to guarantee that they are indeed equal under an acceptable false negative rate β.[2] At the end of the round, inferior configurations are eliminated, and more evaluations are scheduled in batch for the remaining candidates. Alternatively, evaluations can be added in mini-batches, with re-testing done at the end of every mini-batch. Alg. 1 contains an outline of the batch evaluation procedure.

Algorithm 2: LazyPairedTCompare

Input: X, Y: Paired observations
Input: α: Significance level
Input: β: False negative rate
Input: MaxN: Maximum number of observations
Output: 0 $\{\mu_X = \mu_Y\}$, 1 $\{\mu_X > \mu_Y\}$, -1 $\{\mu_X < \mu_Y\}$ w.s. α

$n := \min(|Y|, |X|)$
$T_{X-Y} := \frac{\hat{\mu}_{X-Y}}{\hat{\sigma}_{X-Y}/\sqrt{n}}$
$t_{\alpha/2}$:= $\alpha/2$-th quantile of Student's T distribution
$t_{1-\alpha/2}$:= $(1-\alpha/2)$-th quantile of Student's T distribution

if $T_{X-Y} < t_{\alpha/2}$ **then return** -1 // $X < Y$
else if $T_{X-Y} > t_{1-\alpha/2}$ **then return** 1 // $X > Y$
else
  if n = *MaxN* **then return** 0
  n' := PowerAnalysis (T_{X-Y}, β)
  if $n' > n$ **then**
    X', Y' := Get $(n' - n)$ more samples of X and Y
    return *LazyPairedTCompare(X', Y')*
  else return 0

Algorithm 3: Lazy Paired Nelder-Mead (LaPPT-NM)

Input: Initial simplex $\{\theta_0, \theta_1, \dots, \theta_d\}$, objective function $g(\theta)$, evaluations $\{g(\theta_0), g(\theta_1), \dots, g(\theta_d)\}$
Output: A local maximum of $g(\cdot)$

Define all comparisons of $g(\theta)$ using LazyPairedTCompare
Run Nelder-Mead algorithm

3.3 Lazy Paired Tuning for Infinite Hyper-parameters in Nelder-Mead Optimization

When the hyperparameter space is metric and continuous, we can combine lazy paired tuning with a global optimization routine such as direct search [Conn *et al.*, 2009]. As an illustration, we demonstrate such an integration with the Nelder-Mead algorithm [Nelder and Mead, 1965], a classic direct simplex search technique that continues to enjoy wide popularity for hyper-parameter tuning (e.g., see [Dror *et al.*, 2011]).

Nelder-Mead (NM) operates sequentially, starting out with a simplex of $d + 1$ initialization points (for d parameters).

[1] Since we are performing multiple hypothesis tests simultaneously, one may be concerned with the false discovery rate inherent in multiple testing. One possible remedy is to apply the standard Bonferroni correction [Shaffer, 1995] or one of its derivatives. This controls the number of false rejections of the null hypothesis, making it less likely that we eliminate promising candidates early on.

[2] The power analysis in Eq. (5) requires having an estimate of τ, the expected difference in performance. We use the observed difference in the current round.

Let $\{\theta_0, \theta_1, \ldots, \theta_d\}$ represent the simplex vertices, ranked from worst to best by their value on the response surface, $g(\theta_0), \ldots, g(\theta_d)$. At each step, NM tries to move θ_0, the worst performing point, toward or beyond the rest of the simplex. It does so via line search from θ_0 towards $\theta_c = \sum_{i=1}^{d} \theta_i/d$, the centroid of the rest of the simplex.

We incorporate lazy tuning into classic Nelder-Mead by replacing all the comparison steps with the LAZYPAIREDT-COMPARE algorithm detailed in Alg. 2. Evaluations are added until the test decision can guarantee the desired false positive and false negative probabilities.

4 Related Work

A number of methods have been proposed to address the search aspect of the hyper-parameter tuning problem as efficient alternatives to exhaustive grid search (which unfortunately continues to be routinely employed in practice). Representative technique families include random search [Bergstra and Bengio, 2012], sequential model-based optimization [Brochu *et al.*, 2010, Hutter *et al.*, 2011], direct search [Luersen *et al.*, 2004], evolutionary search [Hansen and Ostermeier, 1996] and structured search [Yang and Rahimi, 2011]. Use of these techniques assumes standard resampled evaluations at each potential configuration in the search space. Hence, all of these methods can benefit from eliminating unnecessary resampling steps. Conn *et al.* [2009], Brochu *et al.* [2010], Spall [2003] all present comprehensive overviews of solutions from adjacent communities that studied the problem. More generally, the problem of searching for optima of a function with probabilistic output falls under the category of stochastic optimization, where previously proposed approaches include, e.g., probabilistic hill climbing [Greiner, 1992].

On the cost-saving front, Krueger *et al.* [2012] proposed to speed up cross-validation by sequentially evaluating the hyper-parameter candidates on a growing sequence of datasets. At each step, a set of sub-optimal performers are eliminated. The statistical elimination tests are in the same spirit as our setup. But they approach the problem from the angle of fast evaluation on smaller datasets, and hence have to contend with the possibility of losing track of the true optimum configuration when starting from very small datasets.

Hoeffding Racing [Heidrich-Meisner and Igel, 2009, Maron and Moore, 1994] tackles the problem of model selection on a finite grid by comparing confidence intervals around the sample means of model performance. The race utilizes the Hoeffding or Bernstein inequalities to provide concentration bounds around the means. More performance samples are added until the bounds tighten to the point where a clear winner emerges. However, it is well-known that these concentration bound can be quite loose, hence many evaluations may be required before one hyperparameter setting may be declared the winner. The advantage of these bounds is that they are uniform against all alternatives, which is useful when each parameter setting much be compared against all other settings. This is applicable in the discrete hyper-parameter space setting, where Hoeffding Racing may be combined with grid search or random search methods.

The discrete parameter space setting is also closely related to the multi-armed bandits problem [Lai and Robbins, 1985, Auer *et al.*, 2002a,b, Gittins *et al.*, 2011]. A variety of sampling strategies exist with different assumptions on the structure of the payoffs and relationship between the arms. We compare against two classic heuristics: UCB and EXP3. Recent work on dueling bandits [Yue *et al.*, 2012] examines the setting where the bandits are pitched against one another, a pair at a time. This similates the testing scenario of production machine learning systems such as a search engine ranker, but does not fit as well to the hyper-parameter tuning and cross-validation setting.

Strens and Moore [2002] proposed using paired comparisons in policy search for reinforcement learning. The search procedure evaluates a number of candidate policies where paired samples may be obtained by, for example, fixing the random number sequence of the simulations. While closely related to the scenario of hyper-parameter search, they do not take advantage of power analysis to determine the number of necessary evaluations.

5 Evaluation

In this section, we evaluate the efficiency of LaPPT on real and synthetic problems in both discrete and continuous hyper-parameter spaces.

5.1 Discrete hyper-parameter spaces

We compare LaPPT to standard approaches for discrete-space search: Hoeffding Racing and most popular multi-armed bandit strategies, UCB1 and EXP3. In this section, we shift to terminology used in multi-armed bandit literature: each hyper-parameter configuration corresponds to an arm, pulling which is synonymous with evaluating the learner on the next fold. Let the random varaible $R \in [0, 1]$ denote reward (the learner's predictive performance), and $\hat{R}_n$ its empirical mean after n observations.

Random: select an arm uniformly at random at each pull.

Hoeffding Race: The Hoeffding inequality states that $\mathbf{P}\left(\hat{R}_n - \mathbf{E}[R] > \epsilon\right) < \exp(-2\epsilon^2/n)$, implying that, with probability greater than $1-\epsilon$, the true mean of the learner's performance lies in the interval $\hat{R}_n \pm \sqrt{-\log(\epsilon)/2n}$. At each round, the race eliminates candidates whose performance upper bound is below another candidate's lower bound. Each of the remaining candidates obtains another evaluation, and the best configuration for the round is taken to be the one yielding the highest empirical mean. In our experiments, $\epsilon = 0.01$.

UCB1: The classic UCB1 policy evaluates the configuration with the maximum Hoeffding upper bound, where ϵ is set to decrease over time. Namely, at time t, UCB1 plays the arm $\arg\max_k \hat{R}_{k,t-1} + \sqrt{\rho \log t / n_{k,t-1}}$, where $R_{k,t-1}$ denotes the empirical average reward for the k-th arm at time $t-1$, $n_{k,t-1}$ the number of times this arm has been played thus far, and ρ is a parameter set to 0.2 in our experiments. This corresponds to setting

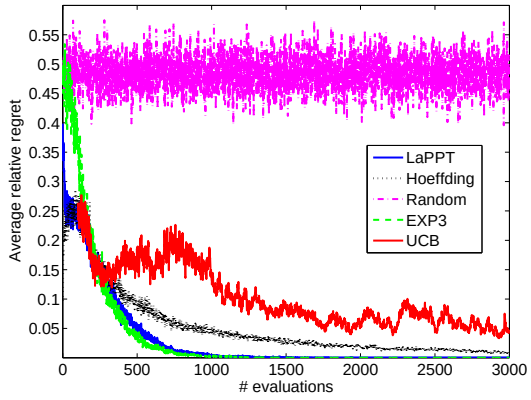


Figure 2: Average regret of LaPPT vs. other sampling strategies on Bernoulli bandits with tied rewards.

$\epsilon_t = O(1/t^4)$ in the Hoeffding inequality. Unlike the Hoeffding Race, UCB does not eliminate candidates, thereby reducing the risk of falsely eliminating a good candidate.

EXP3: The EXP3 policy maintains a probability distribution based on the cumulative reward accrued at each arm. At time t, it pulls the k-th arm with probability proportional to $p_{k,t} = \exp(\eta S_{i,t-1})$, where $S_{i,t-1} := \sum_{j=1}^{t-1} R_{i,j}$ is the cumulative reward for the i-th arm up to time $t-1$ and η is a free parameter set to 0.2 in our experiments. The estimated reward $\tilde{R}_{i,j}$ is taken to be 1 if the i-th arm is pulled on round j, and $1 - \frac{1-R_{i,j}}{p_{i,j}}$ otherwise (an unbiased estimate of the unobserved reward).

For all experiments on finite space tuning, we set $\alpha = 0.1$ and $\beta = 0.6$ for the paired-T testing and power analysis.

We first perform a sanity check using Bernoulli bandits. We construct 100 bandits, each generating binary rewards with success probability θ_k sampled uniformly from $(0, 1)$. To simulate the matched-pairs evaluation setting, the rewards of the bandits are tied so that the random seed for the i-th pull is the same across all arms. Formally, the reward for the i-th pull of the k-th arm is $\mathbb{1}(r_i < \theta_k)$, with r_i being sampled uniformly at random and held to be the same across all k. We repeat the trial 100 times, each time going up to a maximum of 3000 total evaluations/arm-pulls. Our primary performance measure is simple regret:

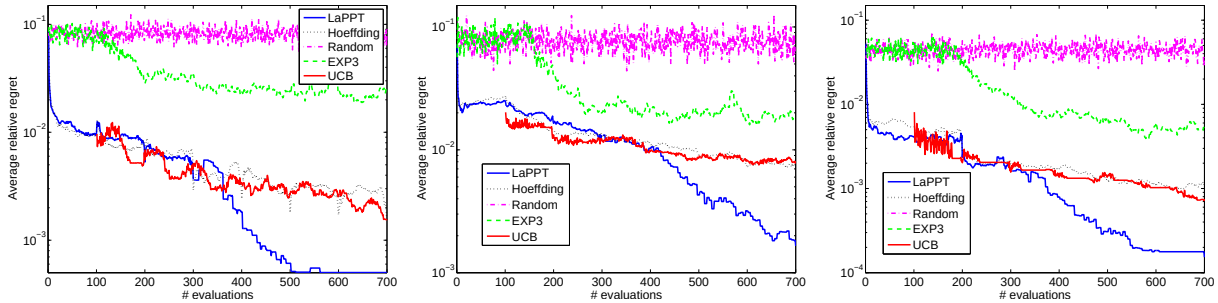
$$r_t = \frac{|\theta_t - \theta^*|}{|\theta^*|},$$

where θ_t is the (true) expected reward of the chosen arm at time t, and θ^* is the expected reward of the best arm.

Results of racing on Bernoulli bandits are shown in Fig. 2. Not surprisingly, the random strategy does not learn over time: the arm it chooses after 3000 evaluations is no more likely to be the best arm. UCB is the second-to-worst performer, bested by Hoeffding Racing, EXP3 and LaPPT. EXP3 and LaPPT are the overall winners with closely matching results. An in-depth inspection of the results reveals that EXP3 selects a non-optimal arm 5 times out of 100 trials, whereas LaPPT is only wrong once, but selects a wrong arm whose true reward is worse than the those of EXP3's mistakes. So LaPPT has larger mean and variance in terms of average regret, but is wrong fewer times.

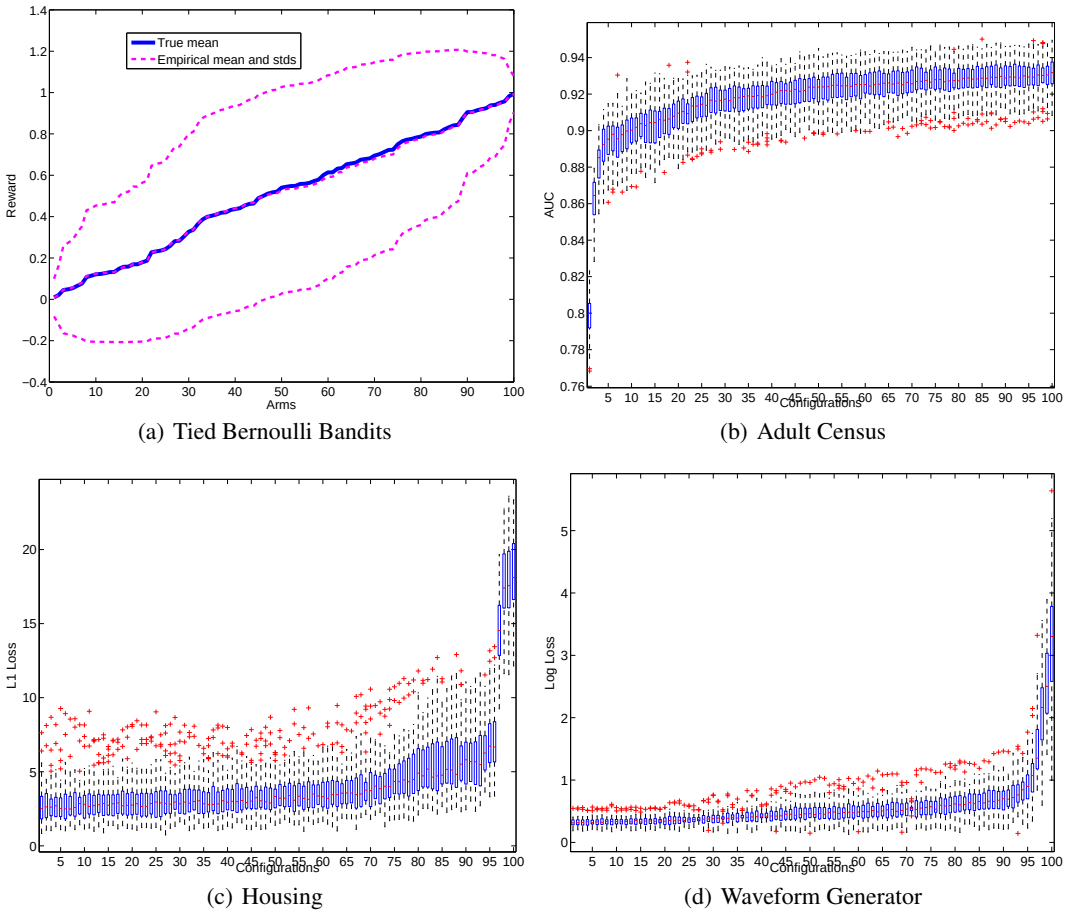
Another notable trend is that both Hoeffding Racing and LaPPT are much more aggressive than EXP3 at the beginning of the game, starting with much lower average regret given very few arm pulls. This is due to our greedy start-game strategy: we give all arms 3 evaluations to start with, before eliminating any candidates. During the initial phase, we select the arm with the best empirical average reward so far to compare against the optimal arm. This is the same as the simple strategy of performing 1, 2, or 3 evaluations of each configuration and selecting the empirical best.

Next, we evaluate the performance of the algorithms for actual hyper-parameter tuning. We utilize three popular UCI datasets chosen to span a variety of data characteristics and learning tasks, with corresponding predictive accuracy metrics: *Adult Census* (binary classification, AUC), *Housing* (regression, L1 error), and *Waveform* (multiclass classification, cross-entropy). Learning is performed with task-appropriate variants of gradient tree boosting [Friedman, 2001], varying four key hyper-parameters: gradient boosting learning rate, number of trees, maximum number of leaves per tree, and minimum number of instances per leaf. We also experimented with Logistic Regression as the base learner and observed analogous results, hence omitted.



(a) Adult Census (b) Housing (c) Waveform Generator

Figure 1: Average regret of LaPPT vs. other sampling strategies on a finite hyperparameter space.



(a) Tied Bernoulli Bandits (b) Adult Census

(c) Housing (d) Waveform Generator

Figure 3: Reward statistics for tied Bernoulli bandits vs. learning scenarios. (a) The solid blue line is the sorted true expected reward of the Bernoulli bandits. The pink dashed lines show the empirical means $\pm$ the standard deviations. (b–d) contain quantile box plots of the raw rewards at each hyper-parameter configuration, sorted by the mean performance over all folds. In each box-and-whiskers plot, the central mark indicates the median, the edges of the box indicate the 25th and 75th percentiles, and the whiskers extend to the minimum and maximum values.

To simulate the setting of the discrete hyper-parameter space, we randomly sample 100 parameter configurations corresponding to 100 arms, and perform 50-fold cross-validation at each configuration. We use the negative L1 error rate and cross-entropy, and we rescale all metrics to the $[0, 1]$ range, so that accuracy on the i-th fold corresponds to the reward from the i-th pull of an arm. The reward is only revealed when the configuration is selected for evaluation. If a selected configuration has no more remaining unevaluated cross-validation folds, we re-calculate the best arm using the current set of evaluations.

As shown in Fig. 3, the reward distributions on real experiments are very different from that of Bernoulli bandits. For Bernoulli bandits, the expected reward p is uniformly sampled from $(0, 1)$ with variance equal to $p(1 - p)$. Fig. 3(a) shows the sorted true rewards, the empirical means, and the standard deviation envelope. The elliptical shape of the envelope demonstrates that the variance is smallest for extreme values of p. As designed, the rewards are uniformly distributed between 0 and 1. In contrast, for cross-validation results from real learning experiments, a significant portion of settings produced accuracy that are close to optimal (i.e., about half of the configurations for *Adult Census* are within 5% of the optimal configuration). In addition, reward distribution over evaluations at each arm is more Gaussian than Bernoulli. As we shall see, the difference in reward distributions results in very different behaviors of the algorithms.

Hyper-parameter tuning results on cross-validation evaluations are presented in Fig. 1, with log-scale y-axis highlighting differences between the algorithms' final solutions. The performances are similar on all three datasets, and below we use the numbers from the *Adult Census* dataset when discussing the specifics. As before, the Random strategy does not learn. EXP3 behaves very differently from before and is now the second slowest learner, only better than Random. It starts out with a lot of explorations, presumably because there are a large number of close-to-optimal configurations. After about 200 evaluations, EXP3 starts to get closer to optimal, but much more slowly than the other strategies. UCB starts out with one evaluation per configuration (which we

do not count when evaluating regret), and thereafter performs on-par with Hoeffding Racing, both reaching an average regret of 0.002 by the end. LaPPT and Hoeffding Racing uses the same selection strategy during the initial warm-up phase, hence are indistinguishable from each other at the beginning. After 300 evaluations (3 per configuration), LaPPT starts to compare the candidates using matched pairs t-tests and discards the statistical losers. It does so very quickly and is able to find the best configuration in 90 out of 100 trials, taking on average fewer than 425 evaluations. In all but 6 trials, LaPPT narrows down to only one candidate. In comparison, Hoeffding Racing is unable to eliminate any candidate within 700 evaluations. This is due to the looseness of the Hoeffding bound and the fact that the mean rewards are closely clustered together. We note that when we run the experiments beyond 700 evaluations, Hoeffding Racing and UCB do eventually obtain the optimal candidate, and LaPPT sometimes pays the price for aggressive elimination when it eliminates the wrong one. However, given that evaluation of learning algorithms could be expensive, such long evaluation horizon is often not practical, and it is really the regret one achieves with few evaluations that is important. To that end, the greedy strategy of evaluating each arm once or twice and picking the one with the best empirical average is a reasonable strategy. But as we can see, the more sophisticated strategies do provide a gain when fine tuning is important.

5.2 Continuous hyper-parameter space

To investigate the predictive and computational performance of LaPPT in combination with Nelder-Mead, we demonstrate their use on *Adult Census* with the same four hyper-parameters: number of trees, learning rate, maximum number of leaves per tree, and minimum number of instances per leaf. Area under ROC curve (AUC) is the metric to optimize, and we report results on running experiments with bootstrap.

The central conjecture of the proposed approach is that lazy evaluation improves the efficiency of the Nelder-Mead algorithm for stochastic functions while maintaining predictive accuracy. To test it, we first compare the optimization quality of classic Nelder-Mead (with 20 bootstrap rounds) with that of LaPPT Nelder-Mead (with up to 20 bootstrap rounds, evaluated only if required). Fig. 4 shows a heatmap of the optimal true AUC over the learning rate and the number of leaves. (The AUC value at each point is the maximum over the other hyper-parameters.) The figure overlays the locations of optima found in 20 runs of LaPPT Nelder-Mead and classic Nelder-Mead, demonstrating that LaPPT-NM can find the same optima as NM, but, as we shall see, much faster.

In Fig. 5, we compare the total number of train-test evaluations, the number of iterations of Nelder-Mead, and the accuracy at the final optimum for Nelder-Mead and LaPPT Nelder-Mead with varying test power β. These results demonstrate that LaPPT Nelder-Mead significantly improves the efficiency of hyper-parameter search while still identifying the optimal hyper-parameter settings. Furthermore, we note that the algorithm not only reduce the number of evaluations, but, more importantly, also results in faster convergence due to faster onset of the inside contraction and shrinkage iterations, which occurs when the test is able to judge minor variations in $g(\cdot)$ to be statistically insignificant. Note that these results directly imply that LaPPT-NM is faster than NM in wall-clock run-time.

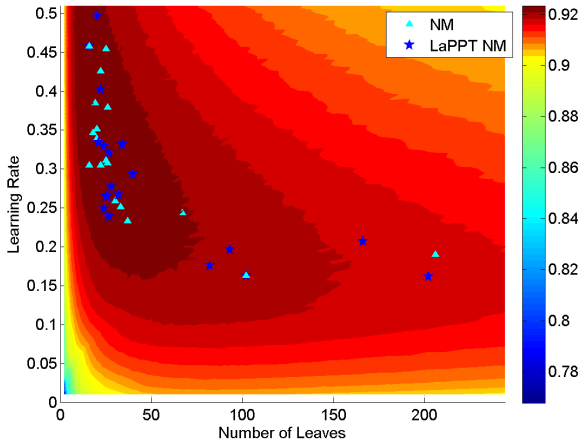


Figure 4: Locations of optima found by 20 runs of Nelder-Mead vs. LaPPT Nelder-Mead

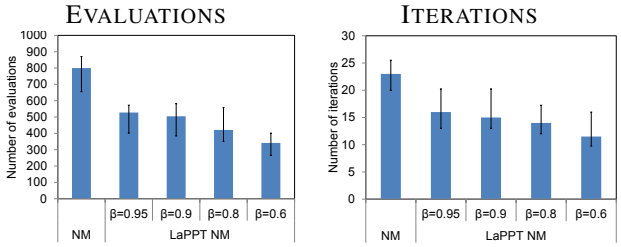


Figure 5: Efficiency of classic and LaPPT Nelder-Mead

6 Conclusions

In this paper, we present a simple and general technique for improving the efficiency of hyper-parameter search in supervised machine learning. Tying train-test resampling during bootstrap or cross-validation allows utilizing paired hypothesis tests and power analysis to limit the number of resampled evaluations. The approach is shown to be effective empirically on discrete hyper-parameter spaces where it outperforms bandit-based methods, and on continuous hyper-parameter spaces where it can be easily integrated with methods such as Nelder-Mead, improving their speed of convergence in addition to reducing the number of train-test evaluations.

References

P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002.

Peter Auer, Nicol Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, Feb 2012.

E. Brochu, V.M. Cora, and N. De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

A.R. Conn, K. Scheinberg, and L.N. Vicente. *Introduction to derivative-free optimization*. MPS-SIAM series on optimization. Society for Industrial and Applied Mathematics/Mathematical Programming Society, 2009.

G. Dror, N. Koenigstein, and Y. Koren. Yahoo! music recommendations: Modeling music ratings with temporal dynamics and item taxonomy. In *Proceedings of RecSys*, 2011.

J. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 25(5):1189–1232, 2001.

J. Gittins, K. Glazebrook, and R. Weber. *Multi-armed bandit allocation indices*. Wiley, 2011.

Russell Greiner. Probabilistic hill-climbing: Theory and applications. In *Proceedings of the Ninth Canadian Conference on Artificial Intelligence (CSCSI-92)*, pages 60–67. Morgan Kaufmann Publishers, Inc, 1992.

N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 312–317. IEEE, 1996.

Verena Heidrich-Meisner and Christian Igel. Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 401–408, New York, NY, USA, 2009. ACM.

F. Hutter, H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. *Learning and Intelligent Optimization*, pages 507–523, 2011.

Tammo Krueger, Danny Panknin, and Mikio Braun. Fast cross-validation via sequential testing. *arxiv CoRR, abs/1206.2248*, 2012.

T.L. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6:4–22, 1985.

E. L. Lehmann and Joseph P. Romano. *Testing statistical hypotheses*. Springer Texts in Statistics. Springer, New York, third edition, 2005.

M.A. Luersen, R. Le Riche, and F. Guyon. A constrained, globalized, and bounded Nelder–Mead method for engineering optimization. *Structural and Multidisciplinary Optimization*, 27(1):43–54, 2004.

Oded Maron and Andrew Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In *Advances in Neural Information Processing Systems*, volume 6, pages 59–66, 1994.

J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

Juliet Popper Shaffer. Multiple hypothesis testing. *Annual review of psychology*, 46(1):561–584, 1995.

J.C. Spall. *Introduction to stochastic search and optimization: estimation, simulation, and control*, volume 64. Wiley-Interscience, 2003.

Nathan Srebro and Ambuj Tewari. Stochastic Optimization: ICML 2010 Tutorial, 2010.

M.J.A. Strens and A.W. Moore. Policy search using paired comparisons. *Journal of Machine Learning Research*, 3(1):921–950, 2002.

Shulin Yang and Ali Rahimi. Structure learning for optimization. In J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 1044–1052. MIT Press, 2011.

Y. Yue, J. Broder, R. Kleinberg, and T. Joachims. The *K*-armed dueling bandits problem. *Journal of Computer and System Sciences*, 2012.