Bayesian Optimization with Inequality Constraints

Jacob R. Gardner[1] GARDNER.JAKE@WUSTL.EDU
Matt J. Kusner[1] MKUSNER@WUSTL.EDU
Zhixiang (Eddie) Xu[1] XUZX@CSE.WUSTL.EDU
Kilian Q. Weinberger[1] KILIAN@WUSTL.EDU
John P. Cunningham[2] JPC2181@COLUMBIA.EDU
[1]Washington University in St. Louis, 1 Brookings Dr., St. Louis, MO 63130

[2]Columbia University, 116th St and Broadway, New York, NY 10027

Abstract

Bayesian optimization is a powerful framework for minimizing expensive objective functions while using very few function evaluations. It has been successfully applied to a variety of problems, including hyperparameter tuning and experimental design. However, this framework has not been extended to the inequality-constrained optimization setting, particularly the setting in which evaluating feasibility is just as expensive as evaluating the objective. Here we present constrained Bayesian optimization, which places a prior distribution on both the objective and the constraint functions. We evaluate our method on simulated and real data, demonstrating that constrained Bayesian optimization can quickly find optimal and feasible points, even when small feasible regions cause standard methods to fail.

1. Introduction

Bayesian optimization has become a popular tool to solve a variety of optimization problems where traditional numerical methods are insufficient. For many optimization problems, traditional global optimizers will effectively find minima (Liberti & Maculan, 2006). However, these methods require evaluating the objective function many times. Bayesian optimization is designed to deal specifically with objective functions that are prohibitively expensive to compute repeatedly, and therefore must be evaluated as few times as possible. A popular application is hyperparameter tuning, where the task is to minimize the validation error of a machine learning algorithm as a function of its hyperparameters (Snoek et al., 2012; Bardenet et al., 2013; Swersky et al., 2013). In this setting, evaluating the objective function (validation error) requires training the machine learning algorithm and evaluating it on validation data. Another application is in experimental design, where the goal is to optimize the outcome of some laboratory experiment as a function of tunable parameters (Azimi et al., 2010b). In this setting, evaluating a specific parameter setting incurs resource costs–materials, money, time, etc.–required to run the experiment.

In addition to expensive evaluations of the objective function, many optimization programs have similarly expensive evaluations of constraint functions. For example, to speed up k-Nearest Neighbor classification (Cover & Hart, 1967), one may deploy data structures for approximate nearest neighbor search. The parameters of such data structures,*e.g.* locality sensitive hashing (LSH) (Gionis et al., 1999; Andoni & Indyk, 2006), represent a trade-off between test time and test accuracy. The goal of optimizing these hyperparameters is to minimize test time, while constraining test accuracy: a parameter setting is only feasible if it achieves the same accuracy as the exact model. Similarly, in the experimental design setting, one may wish to maximize the yield of a chemical process, subject to the constraint that the amount of some unwanted byproduct produced is below a specific threshold. In computer micro-architecture, fine-tuning the particular specifications of a CPU (*e.g.* L1-Cache size, branch predictor range, cycle time) needs to be carefully balanced to optimize CPU speed, while keeping the power usage strictly within a pre-specified budget. The speed and power usage of a particular configuration can only be evaluated with expensive simulation of typical workloads (Azizi et al., 2010). In all of these examples, the feasibility of an experiment is not known until after the experiment had been completed, and thus feasibility can not always be determined in advance. In the context of Bayesian optimization, we say that evaluating

Proceedings of the 31^{st} International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32. Copyright 2014 by the author(s).

feasibility in these cases is also prohibitively expensive, often on the same order of expense as evaluating the objective function. These problems are particularly difficult when the feasible region is relatively small, and it may be prohibitive to even find a feasible experiment, much less an optimal one.

In this paper, we extend the Bayesian optimization framework naturally to scenarios of optimizing an expensive-to-evaluate function under equally expensive-to-evaluate constraints. We evaluate our proposed framework on two simulation studies and two real world learning tasks, based on LSH hyperparameter tuning (Gionis et al., 1999) and SVM model compression (Bucilu et al., 2006; Burges & Schölkopf, 1997).

Across all experiments, we outperform uniform sampling (Bergstra & Bengio, 2012) on 13 out of 14 datasets—including cases where uniform sampling fails to find even a single feasible experiment.

2. Background

To motivate constrained Bayesian optimization, we begin by presenting Bayesian optimization and the key object on which it relies, the Gaussian process.

2.1. Gaussian Processes

A Gaussian process is an uncountable collection of random variables, any finite subset of which have a joint Gaussian distribution. A Gaussian process thus provides a distribution over functions $\ell(\cdot) \sim \mathcal{GP}\left(\mu(\cdot), k(\cdot,\cdot)\right)$, parameterized by mean function $\mu(\cdot)$ and covariance kernel $k(\cdot,\cdot)$, which are defined such that, for any pairs of input points $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$, we have:

$$\begin{aligned} \mu(\mathbf{x}) &= \mathbb{E}[\ell(\mathbf{x})] \\ k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[(\ell(\mathbf{x}) - \mu(\mathbf{x}))(\ell(\mathbf{x}') - \mu(\mathbf{x}'))]. \end{aligned}$$

Given a set of input points $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$, the corresponding function evaluations $\ell(\mathbf{X}) = \{\ell(\mathbf{x}_1), ..., \ell(\mathbf{x}_n)\}$, and some query point $\hat{\mathbf{x}}$, the joint Gaussianity of all finite subsets implies:

$$\begin{bmatrix} \ell(\mathbf{X}) \\ \ell(\hat{\mathbf{x}}) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(\mathbf{X}) \\ \mu(\hat{\mathbf{x}}) \end{bmatrix}, \begin{bmatrix} k(\mathbf{X}, \mathbf{X}) & k(\mathbf{X}, \hat{\mathbf{x}}) \\ k(\hat{\mathbf{x}}, \mathbf{X}) & k(\hat{\mathbf{x}}, \hat{\mathbf{x}}) \end{bmatrix} \right),$$

where we have (in the standard way) overloaded the functions $\ell(\cdot)$, $\mu(\cdot)$, and $k(\cdot,\cdot)$ to include elementwise-operation across their inputs. We then can calculate the posterior distribution of $\ell(\cdot)$ at the query point $\hat{\mathbf{x}}$, which we denote $\tilde{\ell}(\hat{\mathbf{x}}) \sim p\left(\ell(\hat{\mathbf{x}}) | \hat{\mathbf{x}}, \mathbf{X}, \ell(\mathbf{X})\right)$. Using the standard conditioning rules for Gaussian random variables, we see $\tilde{\ell}(\hat{\mathbf{x}}) \sim \mathcal{N}\left(\tilde{\mu}_\ell(\hat{\mathbf{x}}), \tilde{\Sigma}^2_\ell(\hat{\mathbf{x}})\right)$, where:

$$\begin{aligned} \tilde{\mu}_\ell(\hat{\mathbf{x}}) &= \mu(\hat{\mathbf{x}}) + k(\hat{\mathbf{x}}, \mathbf{X}) k(\mathbf{X}, \mathbf{X})^{-1} (\ell(\mathbf{X}) - \mu(\mathbf{X})) \\ \tilde{\Sigma}^2_\ell(\hat{\mathbf{x}}) &= k(\hat{\mathbf{x}}, \hat{\mathbf{x}}) - k(\hat{\mathbf{x}}, \mathbf{X}) k(\mathbf{X}, \mathbf{X})^{-1} k(\mathbf{X}, \hat{\mathbf{x}}). \end{aligned}$$

A full treatment of the use of Gaussian processes for machine learning is Rasmussen (2006). In the context of this work, the critical takeaway is that, given observed function values $\ell(\mathbf{X}) = \{\ell(\mathbf{x}_1), ..., \ell(\mathbf{x}_n)\}$, we are able to update our posterior belief $\tilde{\ell}(\hat{\mathbf{x}})$ of the function $\ell(\cdot)$ at any query point, with simple linear algebra.

2.2. Bayesian optimization

Bayesian optimization is a framework to solve programs:

$$\min_{\mathbf{x}} \ell(\mathbf{x}),$$

where the objective function $\ell(\mathbf{x})$ is considered prohibitively expensive to evaluate over a large set of values. Given this prohibitive expense, in the Bayesian formalism, the uncertainty of the objective $\ell(\cdot)$ across not-yet-evaluated input points is modeled as a probability distribution. Bayesian optimization models $\ell(\cdot)$ as a Gaussian process, which can be evaluated relatively cheaply and often (Brochu et al., 2010). At each iteration the Gaussian process model is used to select the most promising candidate $\mathbf{x}^*$ for evaluation. The costly function ℓ is then only evaluated at $\ell(\mathbf{x}^*)$ in this iteration. Subsequently, the Gaussian process naturally updates its posterior belief $\tilde{\ell}(\cdot)$ with the new data pair $(\mathbf{x}^*, \ell(\mathbf{x}^*))$, and that pair is added to the known experiment set $\mathcal{T}_\ell = \{(\mathbf{x}_1, \ell(\mathbf{x}_1)), ..., (\mathbf{x}_n, \ell(\mathbf{x}_n))\}$. This iteration can be repeated to iterate to an optimum.

The critical step is the selection of the candidate point $\mathbf{x}^*$, which is done via an *acquisition function* that enables active learning of the objective $\ell(\cdot)$ (Settles, 2010). The performance of Bayesian optimization depends critically on the choice of acquisition function. A popular choice is the *Expected improvement* of a candidate point (Jones et al., 1998; Mockus et al., 1978). Let $\hat{\mathbf{x}}$ be some candidate point, and let $\tilde{\ell}(\hat{\mathbf{x}})$ be the Gaussian process posterior random variable for $\ell(\hat{\mathbf{x}})$. Let $\mathbf{x}^+$ be the best point in $\mathcal{T}_\ell$ (evaluated thus far), namely:

$$\mathbf{x}^+ = \min_{\mathbf{x} \in \mathcal{T}_\ell} \ell(\mathbf{x}).$$

Following Mockus et al. (1978), we then define the *improvement* of the candidate point $\hat{\mathbf{x}}$ as the decrease of $\ell(\hat{\mathbf{x}})$ against $\ell(\mathbf{x}^+)$, which due to our Gaussian process model is itself a random quantity:

$$\tilde{I}(\hat{\mathbf{x}}) = \max\left\{0, \ell(\mathbf{x}^+) - \tilde{\ell}(\hat{\mathbf{x}}))\right\}, \quad (1)$$

and thus the expected improvement (EI) acquisition function is the expectation over this truncated Gaussian variable:

$$EI(\hat{\mathbf{x}}) = \mathbb{E}\left[\tilde{I}(\hat{\mathbf{x}}) | \hat{\mathbf{x}}\right]. \quad (2)$$

Mockus et al. (1978); Jones et al. (1998) derive an easy-to-compute closed form for the EI acquisition function:

$$\begin{aligned} EI(\hat{\mathbf{x}}) &= \tilde{\Sigma}_\ell(\hat{\mathbf{x}})\left(Z\Phi(Z)+\phi(Z)\right) \\ &\text{with:}\quad Z=\frac{\tilde{\mu}_\ell(\hat{\mathbf{x}})-\ell(\mathbf{x}^+)}{\tilde{\Sigma}_\ell(\hat{\mathbf{x}})}, \end{aligned}$$

where Φ is the standard normal cumulative distribution function, and ϕ is the standard normal probability density function. In summary, the Gaussian process model within Bayesian optimization leads to the simple acquisition function $EI(\hat{\mathbf{x}})$ that can be used to actively select candidate points.

3. Method

In this paper we extend Bayesian Optimization to incorporate inequality constraints, allowing problems of the form

$$\min_{c(\mathbf{x})\leq\lambda} \ell(\mathbf{x}). \tag{3}$$

where both $\ell(\mathbf{x})$ and $c(\mathbf{x})$ are the results of some expensive experiment. These values may often be the result of the *same* experiment, and so when we conduct the experiment, we compute both the value of $\ell(\mathbf{x})$ and that of $c(\mathbf{x})$.

3.1. Constrained Acquisition Function

Adding inequality constraints to Bayesian optimization is most directly done via the EI acquisition function, which needs to be modified in two ways. First, we augment our definition of $\mathbf{x}^+$ to be the feasible point with the lowest function value observed in $\mathcal{T}$. Second, we assign zero improvement to all infeasible point. This leads to the following *constrained improvement* function for a candidate $\hat{\mathbf{x}}$:

$$I_C(\hat{\mathbf{x}})=\Delta(\hat{\mathbf{x}})\max\left\{0,\ell(\mathbf{x}^+)-\ell(\hat{\mathbf{x}})\right\}=\Delta(\hat{\mathbf{x}})I(\hat{\mathbf{x}})$$

where $\Delta(\hat{\mathbf{x}})\in\{0,1\}$ is a feasibility indicator function that is 1 if $c(\hat{\mathbf{x}})\leq\lambda$, and 0 otherwise.

Because $c(\mathbf{x})$ and $\ell(\mathbf{x})$ are both expensive to compute, we again use the Bayesian formalism to model each with a conditionally independent Gaussian process, given $\mathbf{x}$. During Bayesian optimization, after we have picked a candidate $\hat{\mathbf{x}}$ to run, we evaluate $\ell(\hat{\mathbf{x}})$ and add $(\hat{\mathbf{x}},\ell(\hat{\mathbf{x}}))$ to the set $\mathcal{T}_\ell$ as previously, and we also now evaluate $c(\hat{\mathbf{x}})$ and add $(\hat{\mathbf{x}},c(\hat{\mathbf{x}}))$ to the set $\mathcal{T}_c$, which is then used to update the Gaussian process posterior $\tilde{c}(\mathbf{x})\sim\mathcal{N}(\tilde{\mu}_c(\mathbf{x}),\tilde{\Sigma}_c^2(\mathbf{x}))$ as above.

With this model, our Gaussian process models the constrained acquisition function as the random quantity:

$$\tilde{I}_C(\mathbf{x})=\tilde{\Delta}(\mathbf{x})\max\left\{0,\ell(\mathbf{x}^+)-\tilde{\ell}(\mathbf{x}))\right\}=\tilde{\Delta}(\mathbf{x})\tilde{I}(\mathbf{x}),$$

where the quantity $\tilde{\Delta}(\mathbf{x})$ is a Bernoulli random variable with parameter:

$$PF(\hat{\mathbf{x}}):=Pr\left[\tilde{c}(\mathbf{x})\leq\lambda\right]=\int_{-\infty}^{\lambda}p(c(\hat{\mathbf{x}})|\hat{\mathbf{x}},\mathcal{T}_c)dc(\hat{\mathbf{x}})$$

Conveniently, due to the marginal Gaussianity of $\tilde{c}(\hat{\mathbf{x}})$, the quantity $PF(\hat{\mathbf{x}})$ is a simple univariate Gaussian cumulative distribution function.

These steps lead to the *expected constrained improvement* acquisition function:

$$\begin{aligned} EI_C(\hat{\mathbf{x}}) &= \mathbb{E}\left[\tilde{I}_C(\hat{\mathbf{x}})|\hat{\mathbf{x}}\right] \\ &= \mathbb{E}\left[\tilde{\Delta}(\hat{\mathbf{x}})\tilde{I}(\hat{\mathbf{x}})|\hat{\mathbf{x}}\right] \\ &= \mathbb{E}\left[\tilde{\Delta}(\hat{\mathbf{x}})|\hat{\mathbf{x}}\right]\mathbb{E}\left[\tilde{I}(\hat{\mathbf{x}})|\hat{\mathbf{x}}\right] \\ &= PF(\hat{\mathbf{x}})EI(\hat{\mathbf{x}}), \end{aligned}$$

where the third equality comes from the conditional independence of $c(\mathbf{x})$ and $\ell(\mathbf{x})$, given $\mathbf{x}$.

Thus the expected constrained improvement acquisition function $EI_C(\hat{\mathbf{x}})$ is precisely the standard expected improvement of $\hat{\mathbf{x}}$ over the best *feasible* point so far weighted by the probability that $\hat{\mathbf{x}}$ is feasible.

It is worth noting that, while infeasible points are never considered our best experiment, they are still useful to add to $\mathcal{T}_\ell$ and $\mathcal{T}_c$ to improve the Gaussian process posteriors. Practically speaking, infeasible samples help to determine the shape and descent directions of $c(\mathbf{x})$, allowing the Gaussian process to discern which regions are more likely to be feasible without actually sampling there. This property–that we do not need to sample in feasible regions to find them–will prove highly useful in cases where the feasible region is relatively small, and uniform sampling would have difficulty finding these regions.

3.2. Multiple Inequality Constraints

It is possible to extend the above derivation to perform Bayesian optimization with multiple inequality constraints, $\mathbf{c}(\mathbf{x})\leq\mathbf{\Lambda}$, where $\mathbf{c}(\mathbf{x})=[c_1(\mathbf{x}),...,c_k(\mathbf{x})]$ and $\mathbf{\Lambda}=[\lambda_1,...,\lambda_k]$. We simply redefine $\tilde{\Delta}(\mathbf{x})$ as the Bernoulli random variable with $\mathbb{E}\left[\tilde{\Delta}(\mathbf{x})\right]=p(\tilde{c}_1(\mathbf{x})\leq\lambda_1,...,\tilde{c}_k(\mathbf{x})\leq\lambda_k)$, and the remainder of the $EI_c(\hat{\mathbf{x}})$ constrained acquisition function is unchanged.

Note that $p(\tilde{c}_1(\mathbf{x})\leq\lambda_1,...,\tilde{c}_k(\mathbf{x})\leq\lambda_k)$ is a multivariate Gaussian probability. In the simplest case, we assume the constraints are conditionally independent given $\mathbf{x}$, which conveniently factorizes the probability as $\prod_{i=1}^k p(\tilde{c}_i(\mathbf{x})\leq\lambda_i)$, a product of univariate Gaussian cumulative distribution functions. In the case of dependent constraints, this multivariate Gaussian probability can be calculated with available numerical methods (Cunningham et al., 2011).

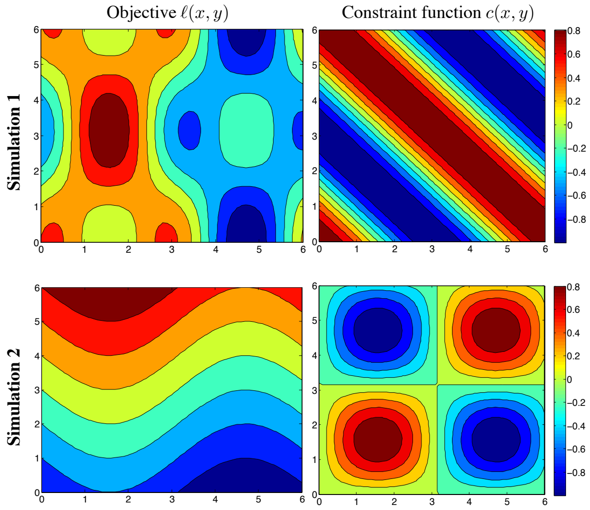

Figure 1. Objective functions $\ell(x, y)$ and constraint functions $c(x, y)$ used for simulations 1 and 2.

4. Results

We evaluate our method, which we call constrained Bayesian Optimization (cBO) on two synthetic tasks and two real world applications. In all cases we compare cBO with function minimization by uniform sampling, an approach that is generally considered competitive (Bergstra & Bengio, 2012) and typically more efficient than grid-searching (Bishop, 2006). Our implementation is written in MATLAB^{TM}. All GP hyperparameters were selected by maximizing the marginal likelihood. We will release our code and all scripts to reproduce the results in this section at http://tinyurl.com/kgj56vy.

4.1. Simulation Function

For the purpose of visualizing our method, we first evaluate it on two simulations with 2D objective and constraint functions. We compare cBO to standard Bayesian optimization and uniform sampling. All methods are allowed 30 evaluations of $\ell(\cdot)$ and $c(\cdot)$.

Simulation 1. For the first simulation, the objective function is

$$\ell(x, y) = \cos(2x)\cos(y) + \sin(x),$$

which we want to minimize subject to the constraint

$$c(x, y) = \cos(x)\cos(y) - \sin(x)\sin(y) \leq 0.5.$$

The top row of figure 1 depicts the contour plots of these functions, and the top row of figure 2 depicts the function evaluations initiated by all three methods during optimization. The infeasible regions are made opaque in figure 2. Black × symbols indicate *infeasible* locations at which $\ell(\cdot)$ and $c(\cdot)$ were evaluated. Circles (black with white filling) indicate *feasible* evaluations.

After a short amount of time, cBO narrows in on the global minimum of the constrained objective (the dark blue spot in the top right corner). In contrast, uniform sampling misses the optimum and wastes a lot of evaluations (22/30) *outside* the feasible region. It is noteworthy that cBO also initiates multiple evaluations outside the feasible regions (14/30), however these are very close to the global minimum (top right) or at the infeasible second minimum (dark blue spot at the bottom right), thus exploring the edge of feasibility where it matters the most. BO without constraints myopically optimizes to the *infeasible* global minimum (the bottom right corner), because it has no knowledge of the constraints.

Simulation 2. In the second simulation, we demonstrate how cBO can quickly find the minimum feasible value of a function even when this feasible region is very small. Here, the objective function (to be minimized) is

$$\ell(x, y) = \sin(x) + y,$$

subject to the constraint

$$c(x, y) = \sin(x)\sin(y) \leq -0.95.$$

The contour plots of these functions are in the bottom row of figure 1. The results of this simulation are displayed in the lower row of figure 2. The feasible regions are small enough that uniform sampling might take some time to sample a feasible point, and none of the 30 samples are feasible. BO without constraints manages to sample two feasible points, but without knowledge of the constraints, these were sampled by chance with the ultimate goal of BO being the global optimum in the lower right. By contrast, cBO is quickly able to use infeasible samples to sufficiently learn the constraint function $c(x, y)$ to locate the feasible regions.

4.2. Locality Sensitive Hashing

As a first real world task, we evaluate cBO by selecting parameters for locality-sensitive hashing (LSH) (Gionis et al., 1999; Andoni & Indyk, 2006) for approximate k-nearest neighbors (kNN) (Cover & Hart, 1967). We begin with a short description of LSH and the constrained optimization problem. We then present the performance of cBO alongside the uniform baseline. We do not compare against standard BO, as without knowledge of the constraints, BO only samples feasible points by chance.

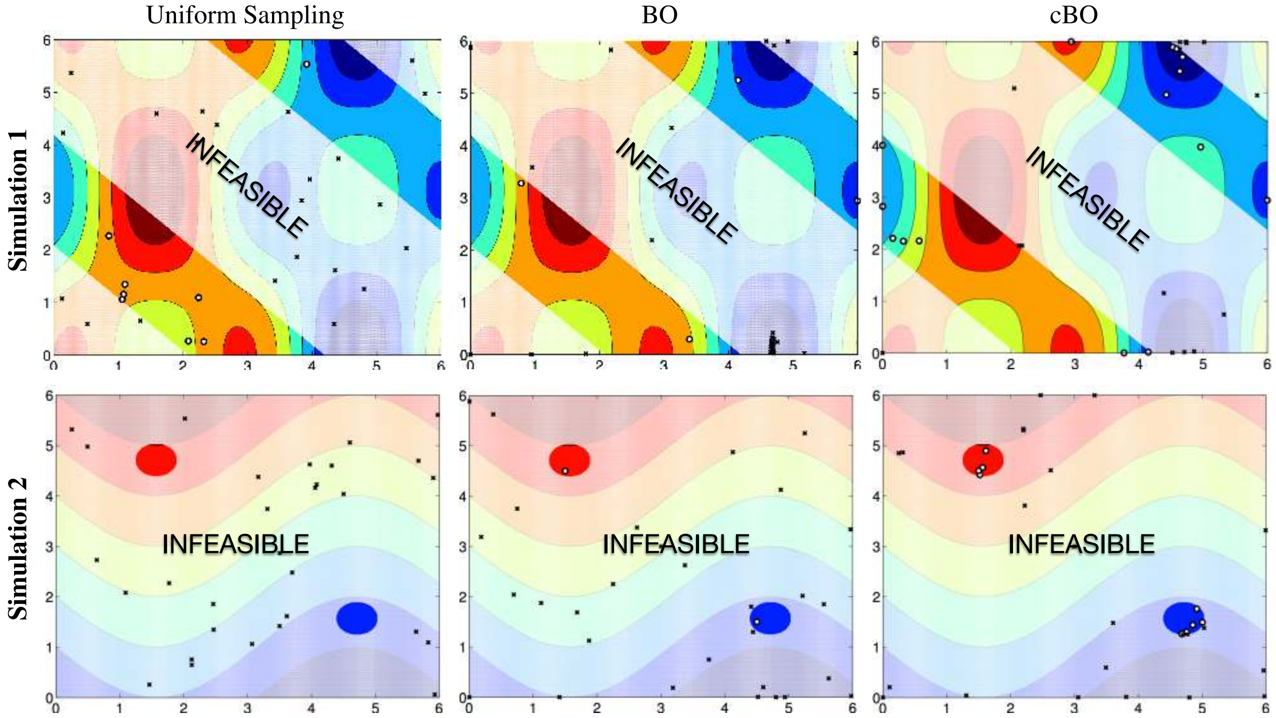

Figure 2. Evaluation of Uniform sampling, standard Bayesian Optimization (BO), and constrained Bayesian Optimization on the simulation problems. Areas shaded in white are infeasible regions. White circle indicate feasible points, and black crosses indicate infeasible points.

Locality-sensitive hashing (LSH) is an approximate method for nearest neighbor search based on random projections. The overall intuition is that nearest neighbors always stay close after projections. LSH defines j random projections, or *hash functions*, $h_1, \ldots, h_j$. This 'hashing' is performed multiple times, in sets of j hash functions, and each set is called a *hash table*.

For further details we refer the interested reader to a review by Slaney & Casey (2008).[1] The key idea is that these hyperparameters of LSH (the number of hash functions j and the number of hash tables L) create a tradeoff between speed and accuracy.

Ideally, one wants to search for the fastest setting that does not impact the classification error. Formally, the constraint function $c(j, L)$ is the *leave-one-out (LOO) classification error* obtained with the LSH data structure with j hash functions and L hash tables. Let ϵ denote the LOO classification error *without* LSH. Then our constraint is $c(j, L) \leq \epsilon$. Our objective, $\ell(j, L)$, is the *time* required to compute the LOO kNN classification error on the training set, which we aim to minimize.

Table 1. Mean LSH results with standard deviations over 10 runs for selecting the number of hash tables and functions for approximate kNN search. We show speedup over kNN and the percentage of infeasible points sampled.

	LSH			
	SPEEDUP (ℓ)		% INFEASIBLE	
DATASET	CBO	UNIFORM	CBO	UNIFORM
YALEFACES	$\mathbf{3.33} \pm 1.53\times$	$2.61 \pm 0.54\times$	$89 \pm 7.0\%$	$70 \pm 3.3\%$
COIL	$\mathbf{18.6} \pm 13.6\times$	$9.69 \pm 1.77\times$	$84 \pm 8.4\%$	$74 \pm 3.4\%$
ISOLET	$\mathbf{6.97} \pm 1.21\times$	$5.49 \pm 0.87\times$	$67 \pm 16\%$	$48 \pm 4.2\%$
USPS	$\mathbf{3.58} \pm 0.89\times$	$3.33 \pm 0.52\times$	$81 \pm 14\%$	$69 \pm 2.0\%$
LETTERS	$\mathbf{1.64} \pm 0.70\times$	$1.56 \pm 0.71\times$	$70 \pm 14\%$	$93 \pm 2.4\%$
ADULT*	$\mathbf{2.80} \pm 2.13\times$	$2.47 \pm 1.63\times$	$97 \pm 3.5\%$	$96 \pm 2.5\%$
W8A*	$\mathbf{3.01} \pm 0.30\times$	$2.32 \pm 0.12\times$	$54 \pm 15\%$	$54 \pm 1.4\%$
MNIST*	$\mathbf{1.69} \pm 0.59\times$	$1.37 \pm 0.28\times$	$71 \pm 16\%$	$64 \pm 1.4\%$

We allow both cBO and uniform sampling to perform 100 function evaluations to find feasible settings of j and L.

Evaluation. Table 1 shows results for learning these LSH parameters under the LOO constraint on 8 popular datasets for face detection (*YaleFaces*) (Georghiades et al., 2001), insurance policy prediction (*COIL*), letter recognition from audio and font-specific features (*Isolet* and *Letters*), income and webpage classification (*Adult* and *W8a*), and optical character recognition (*USPS, MNIST*). We subsampled the training data of three of the larger datasets to 10% (marked in the table with an asterisk). We compare cBO with uniform sampling of the LSH parameters (both optimized over the same range). The table shows the speedup obtained

[1] We use the LSH implementation from the Caltech Image Search Toolbox, http://tinyurl.com/caltechLSH.

with the final LSH model over standard Euclidean kNN search. In all cases the cBO-selected model is, on average, faster than the one obtained with uniform sampling.

Uniform sampling sometimes finds more feasible points than cBO. This is likely because the objective function is often decreasing at the boundary of the feasible region, for example, see the lower left corner of figure 3. This is because the boundary represents the region where LSH becomes *too* approximate, and sacrifices accuracy. cBO, in an effort to minimize the objective as much as possible, must explore this boundary to find its edge, resulting in more infeasible points sampled.

Figure 4 shows the traceplots of the fastest feasible LSH kNN time as a function of sample iterations on the *Coil* and *Adult* data sets. The red and blue dots depict iterations in which *feasible* points are selected. On the *Coil* dataset, after only 13 iterations, cBO finds a feasible setting of j and L that has a lower evaluation time than any setting discovered by uniform sampling. On *Adult*, it is able to further decrease the evaluation time from one that is similar to a setting eventually found by uniform sampling.

Figure 3 shows a contour plot of the 2D objective surface on the *USPS* handwritten digits data set. The infeasible region is masked out in light blue. Feasible evaluation points are marked as white circles, whereas infeasible evaluations are denoted as black crosses. cBO queries only a few infeasible parameter settings and narrows in on the fastest model settings (dark blue feasible region). The majority of infeasible points sampled are near the feasibility border (bottom left). These points are nearly feasible and likely have low objective. Because of this and the thin regions of feasibility, cBO explores this region with the hopes of further minimizing $\ell(\cdot)$. Although uniform sampling does evaluate parameters near the optimum, the final model only obtains a speedup of $3.03\times$ whereas cBO returns a model with speedup $4.1\times$ (see Table 1).

4.3. SVM Compression

Our second real-world application is speeding up support vector machines (SVM) (Cortes & Vapnik, 1995) through hyperparameter search and support-vector "compression" (Burges & Schölkopf, 1997). In this work, Burges & Schölkopf (1997) describe a method for reducing the number of SVM support vectors used for the kernel support vector machine. Their approach is to first train a kernel SVM and record the learned model and its predictions on the training set. Then, one selects an initial small subset of m support vectors and re-optimizes them so that an SVM with only m support vectors matches the predictions of the original model. This re-optimization can be done efficiently with conjugate gradient descent[2] and can be very effective at speeding up SVMs during test-time—however it is highly dependent on several hyperparameters and has the potential to degrade a classifier's performance.

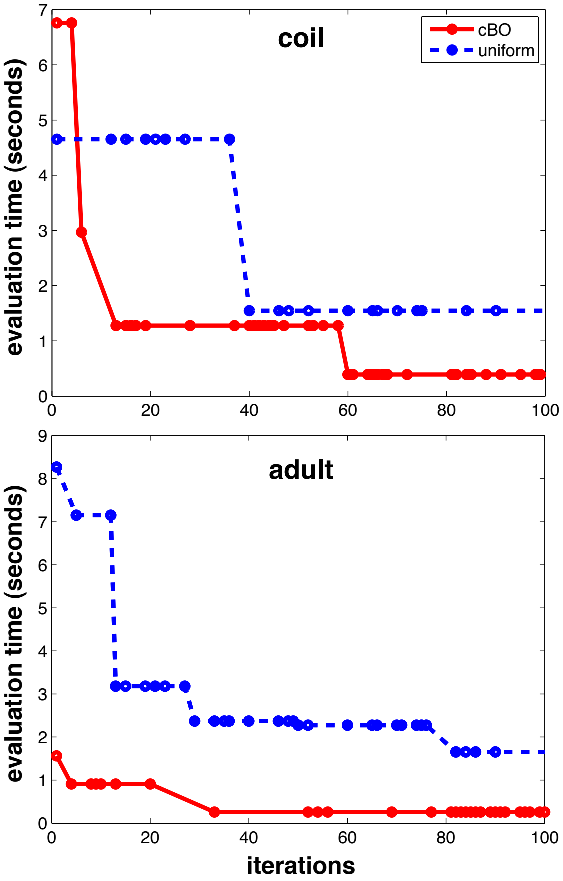

Figure 4. Plot of the best LSH nearest neighbor search evaluation time (ℓ) found so far versus iteration for cBO and uniform sampling over one run on the *Coil* and *Adult* datasets.

We restrict our setting to the popular radial basis function (RBF) kernel (Schölkopf & Smola, 2001),

$$k(\mathbf{x}, \mathbf{z}) = \exp\left(\gamma^2 \|\mathbf{x} - \mathbf{z}\|_2^2\right), \tag{4}$$

which is sensitive to a width parameter γ^2. To speed up SVM evaluation we need to select values for γ^2, the SVM cost parameter C, and the number of support vectors m that minimize the validation evaluation time. However, to avoid degrading the performance of our classifier by using fewer support vectors, we need to constrain the validation error to increase by no more than $s\%$ over the original SVM model.

To be precise, we first train an SVM on a particular data set (all hyperparameters are tuned with standard Bayesian optimization). We then compress this model to minimize validation evaluation time, while only minimally affecting its validation error (up to a relative increase of $s\%$). For

[2] http://tinyurl.com/minimize-m

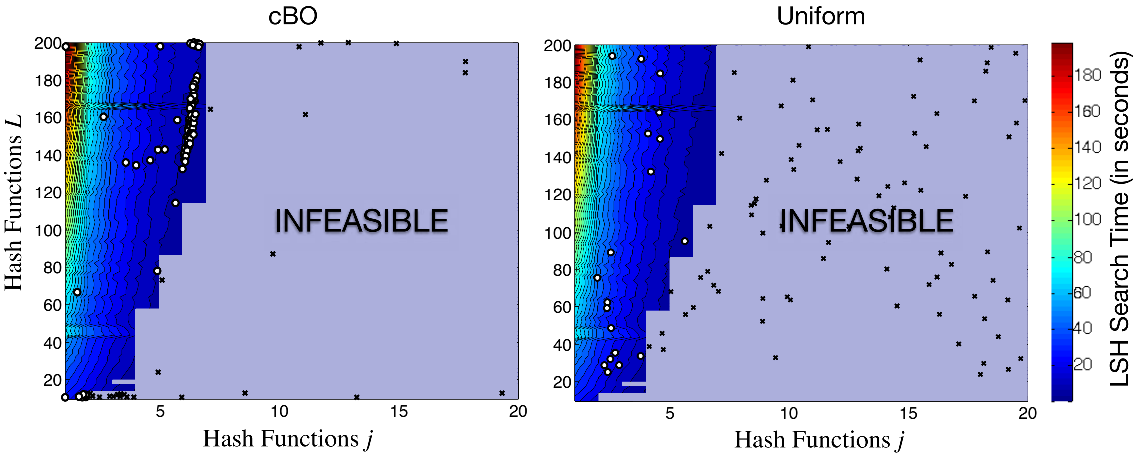

Figure 3. Contour plots of the kNN evaluation time on the *USPS* dataset using LSH with different number of hash tables L and hash functions j. The pale shaded region represents infeasible settings (based on the kNN error). Black crosses indicate infeasible points, white circles indicate feasible points. *Left:* Parameter settings evaluated by cBO. *Right:* Parameter settings evaluated by uniformly sampling.

a particular parameter setting $\{\gamma^2, C, m\}$, an evaluation of $\ell()$ and $c()$ involves first compressing an SVM with parameters $\{\gamma^2, C\}$ down to m support vectors following Burges & Schölkopf (1997), and then evaluating the resulting classifier on the validation set. The value of $\ell(\gamma^2, C, m)$ is the time required for the evaluation (not the compression), and the value of $c(\gamma^2, C, m)$ is the validation error. This error is constrained to be no more than $s\%$ larger than the validation error of the original SVM. As in the LSH task, we allow both cBO and uniform sampling to perform 100 evaluations.

Comparison. Table 2 shows results for learning γ^2, C and m on six medium scale UCI datasets[3] including spam classification (*Spam*), gamma particle and engine output detection (*Magic* and *IJCNN1*), and tree type identification (*Forest*). We also evaluate on *Adult* and *W8a*, as with LSH. Similar to LSH, we subsampled the training data of five of the larger datasets to 10% (marked in the table with an asterisk, the table shows the data set size after subsampling). We consider the two cases of $s = 1\%$ and $s = 10\%$ relative validation error increase. The table presents the best speedups found by cBO and uniform sampling, the corresponding number of support vectors (SVs), as well as the percent of parameter settings that turned out to be infeasible. cBO outperforms uniform sampling on all datasets in speedup. In the most extreme case (*Adult*), the compressed SVM model was $551\times$ faster than the original with only 1% relative increase in validation error. On two data sets (*IJCNN1* and *Forest*), uniform subsampling does not find a single compressed model that guarantees a validation error increase below 1% (as well as 10% for *IJCNN1*). The table also shows the number of support vectors m, to which the SVM is compressed. In all cases is the cBO model substantially smaller than the one obtained with uniform sampling.

One interesting observation is that uniform sampling finds more feasible points for *Adult* and *W8a* datasets. A possible explanation for this is that a very fast parameter setting is right near the feasibility border. Indeed, it is likely for only $m = 3$ support vectors many settings of γ^2 and C will be infeasible.

5. Related Work

There has been a large amount of recent work on using sampling methods for blackbox optimization in machine learning. A popular application of these methods is hyperparameter tuning for machine learning algorithms, or optimizing the validation performance of a machine learning algorithm as a function of its hyperparameters. Bergstra & Bengio (2012) demonstrates that uniform sampling performs significantly better than the common grid search approach. They propose that the use of Bayesian optimization for this task is promising, and uniform sampling serves as a baseline for Bayesian optimization papers (Snoek et al., 2012).

A large number of relevant papers have been published on the topic of hyperparameter tuning as well Hutter et al. (2011); Bergstra et al. (2011). Most similar to our work is Bernardo et al. (2011) and Snoek (2013). Constraints are considered in these, but only feasibility is observed. As a result, it is difficult to predict where feasible points will be before observing them. The method in these works is therefore less applicable in the less general scenario that we consider, where the constraint function is actually computable. Snoek et al. (2012) introduces *Spearmint*, a popular tool for this application. *Spearmint* marginalizes over

[3] http://tinyurl.com/ucidatasets

Table 2. SVM compression results with standard deviations over 10 runs for selecting γ^2, C, and the number of support vectors m.

		1% RELATIVE ERROR INCREASE						10% RELATIVE ERROR INCREASE					
	NUMBER OF	SPEEDUP (ℓ)		% INFEASIBLE		SVS		SPEEDUP ℓ		% INFEASIBLE		SVS	
DATASET	SAMPLES	CBO	UNIFORM	CBO	UNIFORM	CBO	UNIFORM	CBO	UNIFORM	CBO	UNIFORM	CBO	UNIFORM
SPAM	3681	**50** ± 24×	22 ± 8.2×	96 ± 3.5%	99 ± 0.8%	539 ± 643	746 ± 263	**294** ± 43×	123 ± 48×	87 ± 5.2%	82 ± 4.5%	7.3 ± 3.2	110 ± 105
MAGIC*	1522	**273** ± 84×	43 ± 21×	94 ± 4.1%	99.4 ± 0.7%	62 ± 89	348 ± 173	**361** ± 32×	73 ± 58×	91 ± 6.0%	99 ± 1.2%	23 ± 4.1	239 ± 122
ADULT*	3256	**1248** ± 244×	1007 ± 185×	24 ± 1.0%	6.6 ± 2.3%	10 ± 9.7	20 ± 16	**1371** ± 34×	1007 ± 185×	22 ± 1.2%	1.3 ± 0.8%	3.8 ± 0.7	20 ± 16
W8A*	4975	**555** ± 142×	463 ± 77×	29 ± 9.5%	19 ± 2.8%	236 ± 716	28 ± 24	**625** ± 156×	494 ± 80×	25 ± 8.1%	13 ± 2.2%	227 ± 701	22 ± 26
IJCNN1*	4999	**8.7** ± 0.96×	–	99.6 ± 0.5%	100 ± 0.0%	1099 ± 667	–	**9.2** ± 0.47×	7.9 ± 0.0×	99 ± 1.4%	99.9 ± 0.3%	909 ± 672	1946 ± 0
FOREST*	5229	**79** ± 42×	38 ± 17×	95 ± 4.3%	99 ± 1.0%	819 ± 841	1195 ± 596	**179** ± 58×	66 ± 42×	96 ± 2.5%	96 ± 2.5%	178 ± 126	910 ± 744

the Gaussian process hyperparameters using slice sampling rather than finding the maximum likelihood hyperparameters. Spearmint also introduces the *EI per cost* acquisition function, which—in addition to its applications with costs other than time—often allows for faster optimization when some parameters affect the running time of an experiment.

There has been other work on the hyperparameter tuning problem as well. A few papers have also been published dealing with multi task validation Bardenet et al. (2013); Swersky et al. (2013), where the goal is either to optimize multiple datasets simultaneously, or use the knowledge gained from tuning previous datasets to provide a warm start to the optimization of new datasets. Parallelizing Bayesian optimization is an active research area (Azimi et al., 2010a; 2012; Snoek et al., 2012). Wang et al. (2013) adapts Bayesian optimization to very high dimensional settings.

A number of other extensions to and applications of Bayesian optimization exist as well. Azimi et al. (2010b) extends Bayesian optimization to the case where one cannot control the precise value of some parameters in an experiment. Mahendran et al. (2012) applies Bayesian optimization to perform adaptive MCMC. Finally, Hoffman et al. (2013) introduce constraints on the number of function evaluations, rather than expensive-to-compute constraints, which we model with cBO.

6. Discussion

In conclusion, in this paper we extended Bayesian Optimization to incorporate expensive to evaluate inequality constraints. We believe this algorithm has the potential to gain traction in the machine learning community and become a practical and valuable tool. Classical Bayesian optimization provides an excellent means to get the most out of many machine learning algorithms. However, there are many algorithms–particularly approximate algorithms with the goal of speed–that the standard Bayesian optimization framework is ill-suited to optimize. This is because it has no way of dealing with the tradeoff between speed and accuracy that these algorithms present.

We extend the Bayesian optimization framework to deal with these tradeoffs via constrained optimization, and present two applications of our method that yield substantial speedups at little to no loss in accuracy for two of the most popular machine learning algorithms, kernel Support Vector Machines and k-Nearest Neighbors.

Although not the primary focus of this paper, the strong results of our model-compression applications (Burges & Schölkopf, 1997; Bucilu et al., 2006) demonstrate the high impact potential of cBO. The use of cBO eliminates all hyperparameters from the compression algorithm and guarantees that any output model matches the validation accuracy of the original classifier. In our experiments we obtain speedups of several order of magnitudes with kernel SVM, making the algorithm by Burges & Schölkopf (1997) (with cBO) suddenly a compelling option for many practitioners who care about test-time performance (Xu et al., 2012).

In addition, we believe that our method will find use in areas beyond machine learning as well. In particular, many industrial applications may have adjustable processes that produce unwanted byproducts—such as carbon emissions in manufacturing, side reactions in drug synthesis, or heat in computing infrastructures (Azizi et al., 2010)—that must be kept under certain levels. Our algorithm provides a way to quickly and cheaply tune these processes so that output is maximized while maintaining acceptable levels of byproduct.

7. Acknowledgements

JRG, MJK, ZX, and KQW are supported by NSF grants 1149882 and 1137211. JPC is supported by the Grossman Center for Statistics of Mind at Columbia University.Computations were performed via the Washington University Center for High Performance Computing, partially provided by grant NCRR 1S10RR022984-01A1.

References

Andoni, A. and Indyk, P. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS 2006*, pp. 459–468. IEEE, 2006.

Azimi, J., Fern, A., and Fern, X. Z. Batch Bayesian optimization via simulation matching. In *NIPS 2010*, pp. 109–117. 2010a.

Azimi, J., Fern, X., Fern, A., Burrows, E., Chaplen, F., Fan, Y., Liu, H., Jaio, J., and Schaller, R. Myopic policies for budgeted optimization with constrained experiments. In *AAAI*, 2010b.

Azimi, J., Jalali, A., and Fern, X. Z. Hybrid batch bayesian optimization. In *ICML 2012*, pp. 1215–1222, New York, NY, USA, 2012. ACM.

Azizi, O., Mahesri, A., Lee, B. C., Patel, Sanjay J., and Horowitz, M. Energy-performance tradeoffs in processor architecture and circuit design: a marginal cost analysis. In *ACM SIGARCH Computer Architecture News*, number 3, pp. 26–36. ACM, 2010.

Bardenet, R., M., Brendel, Kegl, B., and Sebag, M. Collaborative hyperparameter tuning. In *ICML*, 2013.

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *JMLR*, 13:281–305, 2012.

Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B., et al. Algorithms for hyper-parameter optimization. In *NIPS*, volume 24, pp. 2546–2554, 2011.

Bernardo, JM, Bayarri, MJ, Berger, JO, Dawid, AP, Heckerman, D, Smith, AFM, and West, M. Optimization under unknown constraints. *Bayesian Statistics 9*, 9:229, 2011.

Bishop, C.M. *Pattern recognition and machine learning*. Springer New York, 2006.

Brochu, E., Cora, V. M., and De Freitas, N. 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.

Bucilu, Cristian, Caruana, Rich, and Niculescu-Mizil, Alexandru. Model compression. In *KDD 2006*, pp. 535–541. ACM, 2006.

Burges, Chris J.C. and Schölkopf, Bernhard. Improving the accuracy and speed of support vector machines. *NIPS 1997*, 9: 375–381, 1997.

Cortes, C. and Vapnik, V. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Cover, T. and Hart, P. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 1967.

Cunningham, J. P., Hennig, P., and Simon, L. Gaussian probabilities and expectation propagation. *arXiv preprint arXiv:1111.6832*, 2011.

Georghiades, A.S., Belhumeur, P.N., and Kriegman, D.J. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Trans. Pattern Anal. Mach. Intelligence*, 23(6):643–660, 2001.

Gionis, A., Indyk, P., Motwani, R., et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pp. 518–529, 1999.

Hoffman, M. W., Shahriari, B., and de Freitas, N. Exploiting correlation and budget constraints in Bayesian multi-armed bandit optimization. *stat*, 1050:11, 2013.

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

Jones, D. R., Schonlau, M., and Welch, W. J. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

Liberti, L. and Maculan, N. *Global Optimization: Volume 84, From Theory to Implementation*, volume 84. Springer, 2006.

Mahendran, N., Wang, Z., Hamze, F., and Freitas, N. D. Adaptive mcmc with bayesian optimization. In *AISTATS 2012*, pp. 751–760, 2012.

Mockus, J., Tiesis, V., and Zilinskas, A. The application of bayesian methods for seeking the extremum. *Towards Global Optimization*, 2:117–129, 1978.

Rasmussen, C. E. Gaussian processes for machine learning. MIT Press, 2006.

Schölkopf, B. and Smola, A.J. *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

Settles, B. Active learning literature survey. *University of Wisconsin, Madison*, 2010.

Slaney, M. and Casey, M. Locality-sensitive hashing for finding nearest neighbors [lecture notes]. *Signal Processing Magazine, IEEE*, (2):128–131, 2008.

Snoek, J. *Bayesian Optimization and Semiparametric Models with Applications to Assistive Technology*. PhD thesis, University of Toronto, 2013.

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. In *NIPS 2012*, pp. 2960–2968. 2012.

Swersky, K., Snoek, J., and Adams, R. P. Multi-task bayesian optimization. In *NIPS 2013*, pp. 2004–2012. 2013.

Wang, Z., Zoghi, M., Hutter, F., Matheson, D., and de Freitas, N. Bayesian optimization in a billion dimensions via random embeddings. *arXiv preprint arXiv:1301.1942*, 2013.

Xu, Z., Weinberger, K., and Chapelle, O. The greedy miser: Learning under test-time budgets. In *ICML*, pp. 1175–1182, 2012.