Bayesian Optimization for Conditional Hyperparameter Spaces

Julien-Charles Lévesque
Université Laval
julien-charles.levesque.1@ulaval.ca

Audrey Durand
Université Laval
audrey.durand.2@ulaval.ca

Christian Gagné
Université Laval
christian.gagne@gel.ulaval.ca

Robert Sabourin
École de Technologie Supérieure
robert.sabourin@etsmtl.ca

***Abstract*—Hyperparameter optimization is now widely applied to tune the hyperparameters of learning algorithms. The hyperparameters can have structure, resulting in hyperparameters depending on conditions, or on the values of other hyperparameters. We target the problem of combined algorithm selection and hyperparameter optimization, which includes at least one conditional hyperparameter: the choice of the learning algorithm. In this work, we show that Bayesian optimization with Gaussian processes can be used for the optimization of conditional spaces with the injection of knowledge concerning conditions in the kernel. We propose and examine the behavior of two kernels, a conditional kernel which forces the similarity of two samples from different condition branches to be zero, and the Laplace kernel, based on similarities with Mondrian processes and random forests. We show the benefit of using such kernels, as well as proper imputation of inactive hyperparameters, on a benchmark of scikit-learn models.**

I. Introduction

In order to correctly apply machine learning algorithms, a practitioner must often specify some free parameters, known as hyperparameters. An example of hyperparameter is the strength of the regularization applied to model weights or the number of layers and neurons in a neural network. Executing an external optimization procedure to select those hyperparameters is known as hyperparameter optimization [1], and can lead to substantial benefits given that proper cross-validation strategies are used.

Rather than concerning oneself only with the hyperparameters of a single learning algorithm chosen beforehand, one could extend the scope of the hyperparameter optimization problem to include the choice of the learning algorithm. The resulting problem has been called a Combined Algorithm Selection and Hyperparameter optimization (CASH) problem, examples of which are solved by the Auto-WEKA [2] and Auto-Sklearn frameworks [3]. For instance, given a classification or regression dataset, the Auto-WEKA platform attempts to identify the best algorithm and the best hyperparameters amongst a wide selection of algorithms available in the WEKA library (the same goes for the Auto-Sklearn framework with the scikit-learn library).

One challenge with combined algorithm and hyperparameter selection is the *conditionality* of some hyperparameters. Indeed, some parameters need only be specified if another parameter is active or has a certain value. For example, given a joint search space over multiple classifiers, once the classifier choice is known, only the parameters specific to this classifier are relevant. The Sequential Model-based Algorithm Configuration (SMAC) [4] and the Tree Parzen Estimators (TPE) [1], both handling conditional or hierarchical hyperparameters, were compared on the Auto-WEKA problem, with SMAC outperforming TPEs. Bayesian optimization with Gaussian Processes (GPs) and conditional hyperparameters have been considered in [1, 5], but appear to be inferior to TPE and SMAC with random forests as a surrogate model on the Auto-WEKA benchmark [6].

In this work, we show that Bayesian optimization with GPs can perform well in conditional spaces, provided some adjustments are made. The traditional Bayesian optimization pipeline itself is not going to change, and we will mostly concern ourselves with the underlying probabilistic model used, that is the GP and its kernel. We show that by integrating knowledge concerning the problem in the covariance function of the GP, the performance of a Bayesian optimization procedure can be greatly improved, comparing favorably with other sequential model-based optimization methods in the literature. We evaluate our suggested methods on a CASH problem, but it can be applied for any black-box optimization problem which includes some form of conditionality.

In the next section, we briefly describe some previous work on hyperparameter optimization. In Section III, we present Bayesian optimization, define the concepts of conditionality and imputation, and state how they can be applied to hyperparameter optimization in conditional spaces. In Section IV, our proposed kernels for the optimization of conditional spaces are presented, which are the conditional kernel and the Laplace kernel. Section V presents experiments on a CASH problem with models from scikit-learn, and following discussions.

II. Related Works

The Spearmint toolbox was proposed to perform Bayesian optimization of hyperparameters with GPs as a prior over functions [7]. It makes use of slice sampling to tune the hyperparameters of the GPs [8], proposes a technique for the evaluation of many solutions in parallel, and uses LBFGS for the optimization of the acquisition function. TPEs define a custom hierarchy for Bayesian optimization with tree-structured Parzen density estimators [1], which supports conditions in the hyperparameter space. SMAC is another toolbox for the Bayesian optimization of hyperparameters for learning algorithms and SAT solvers, with support for conditional

hyperparameters [4]. Key features of SMAC are incumbent challenging mechanisms and the use of random forests as a surrogate model for the objective function. SMAC has been shown to outperform Spearmint and TPEs on the combined algorithm selection and hyperparameter optimization on all classifiers available in the Weka toolbox [2, 6]. Another toolbox called Auto-Sklearn allows for the optimization and selection of classifier and preprocessing methods amongst a wide selection of algorithms available in the scikit-learn library [3].

Amongst more recent contributions, dataset subsampling has been exploited to speed up hyperparameter optimization with great success. Li *et al.* [9] iteratively increase the size of the training and prune unpromising solution with the successive halving strategy. Klein *et al.* [10] model the function of validation error with regards to dataset size and let the Bayesian optimization procedure choose the subset size with relevant acquisition functions. Our work does not prevent the application of any of these methods, and they should be considered by any practitioner looking to improve the efficiency of their Bayesian hyperparameter optimization pipeline.

III. Bayesian Optimization

When considering a single learning algorithm A, Bayesian optimization of hyperparameters is performed by posing a GP prior on the loss function $f(\gamma)$ in the space of hyperparameters $\gamma \in \Gamma$. This loss function can be observed with some noise through a validation dataset X_V:

$$L(h_\gamma, X_V) = \frac{1}{|X_V|} \sum_{(x_i,y_i)\in X_V} l(h_\gamma(x_i), y_i), \tag{1}$$

$$f(\gamma) = L(h_\gamma, X_V) + \varepsilon, \tag{2}$$

where h_γ is the prediction rule obtained by running learning algorithm A with hyperparameters γ on a given training dataset X_T, and $l(\cdot, y)$ is a target loss function (e.g., the zero-one loss for classification). The true function $f(\gamma)$ is unknown and we are only glimpsing it through noisy observations computed on the validation dataset.

A GP can be defined as a generalization of the Gaussian probability distribution, where a stochastic process governs the properties of Gaussian distributions at every point of a space. A GP(μ, k) is completely described by its mean function $\mu : \Gamma \to \mathbb{R}$, $\mu(\gamma) = \mathbb{E}[f(\gamma)]$ and covariance (kernel) function $k : \Gamma \times \Gamma \to \mathbb{R}$, $k(\gamma, \gamma') = \mathbb{E}[(f(\gamma) - \mu(\gamma))(f(\gamma') - \mu(\gamma'))]$. Suppose we condition a GP(μ, k) on observed outputs $\mathbf{y} = \{L(h_{\gamma_i}, X_V)\}_{i=1}^t$ associated with inputs $G = \{\gamma_1, \ldots, \gamma_t\}$, where $y_n = f(\gamma_n) + \varepsilon$ with i.i.d. Gaussian noise $\varepsilon \sim \mathcal{N}(0, \sigma_n^2)$. The predictive distribution at test point γ_* is estimated by

$$\hat{f}_* = \mathbf{k}_*^T (K + \sigma_n^2 I)^{-1} \mathbf{y}, \tag{3}$$

$$\mathbb{V}[f_*] = k(\gamma_*, \gamma_*) - \mathbf{k}_*^T (K + \sigma_n^2 I)^{-1} \mathbf{k}_*, \tag{4}$$

where $\mathbf{k}_* = [k(\gamma_1, \gamma_*), \ldots, k(\gamma_N, \gamma_*)]^T$ and K is the positive semi-definite kernel matrix $[k(\gamma, \gamma')]_{\forall(\gamma,\gamma')\in(G\times G)}$. On each trial t, the GP is conditioned on the full history of observations $\mathcal{H} = \{\gamma_i, L(h_{\gamma_i}, X_V)\}_{i=1}^{t-1}$. One can then use the posterior mean and variance to select the next hyperparameters to evaluate with an *acquisition function* balancing exploration and exploitation.

Algorithm 1 Bayesian Optimization Procedure

Input: X_T and X_V, some training and validation datasets, A the learning algorithm (or a wrapper around many learning algorithms), and L the loss function

$\mathcal{H}_0 \leftarrow \varnothing$
for $t \in 1, \ldots, T$ **do**
  $f(\gamma) \leftarrow GP(\mathcal{H}_{t-1})$ // Fit GP on observations
  $\gamma_t \leftarrow \arg\max_\gamma a(\gamma | f(\cdot))$ // Choose next hypers
  $h_{\gamma_t} \leftarrow A(\gamma_t, X_T)$ // Train model
  $l_{\gamma_t} \leftarrow L(h_{\gamma_t}, X_V)$ // Compute validation loss
  $\mathcal{H}_t \leftarrow \mathcal{H}_{t-1} \cup \{(\gamma_t, l_{\gamma_t})\}$ // Update observations
end for
return $\arg\min_{h\in\mathcal{H}_T} L(h, X_V)$ // Return best model

A. Conditional Hyperparameters

A straightforward way to solve CASH problems with Bayesian optimization is to produce a single posterior on the joint space of all classifiers hyperparameters, adding an extra algorithm selection parameter. Given a set of m algorithms from which to choose from $\mathcal{A} = \{A_1, A_2, \ldots, A_m\}$ and their respective hyperparameter spaces $\Gamma_1, \Gamma_2, \ldots, \Gamma_m$, the resulting search space is $\mathcal{G} = \mathcal{A} \cup \Gamma_1 \cup \Gamma_2, \ldots, \Gamma_m$, where the first hyperparameter $a \in \mathcal{A}$ is a new hyperparameter representing the choice of the learning algorithm. The dimensionality of this joint space is $1 + \sum_{i=1}^{C} |\Gamma_i|$. On each trial t, the model is conditioned on the full history of observations $\mathcal{H} = \{\boldsymbol{\kappa}_i, L_i\}_{i=1}^{t-1}$, where $\boldsymbol{\kappa}_i = \{a_i, \gamma_{1,i}, \ldots, \gamma_{m,i}\}$ and L_i is the empirical loss of algorithm A_{c_i} using hyperparameters $\gamma_{c_i,i}$. This formulation is the equivalent of assuming that there is information shared between the hyperparameters of different learning algorithms. A GP with a regular distance-based kernel would not be able to distinguish between active and inactive parameters in a dense input $\boldsymbol{\kappa_i}$, generating a confusion concerning the responsibility of hyperparameters for model performance, or, in other words, making an invalid credit assignment.

Combined algorithm selection and hyperparameter optimization is just one example where conditions are present. Single classifiers can also generate levels of conditionality, so rather than referring to classifier choices, we will be referring to conditions for the remainder of this work. The *effective* dimensionality of a space with conditionality is not given by the concatenation of all spaces; in fact it depends upon the currently activated conditions and hyperparameters. Figure 1 shows a generic configuration space with hierarchical structure. Given hyperparameters γ_i, the conditionality is defined vertically in the graph, with subgroups separated by different values for parent hyperparameters, recursively. For example, hyperparameter γ_5 would only be relevant if γ_2 takes the value c, and γ_1 takes the value a (otherwise parameter γ_2 would not

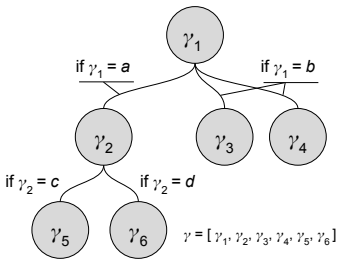

Fig. 1. Generic hyperparameter space with conditions activating hyperparameters

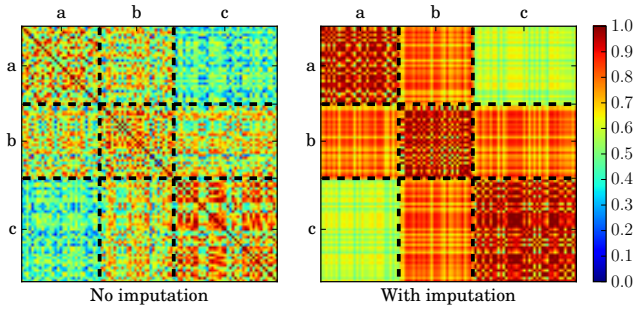

Fig. 2. Two covariance matrices for the same sample. On the left is the covariance for densely sampled hyperparameters, with inactive values left set to random value, and on the right side the inactive values have been replaced with a default value. The three blocks correspond to three different classifiers (samples are sorted along the value of the classifier choice parameter to highlight differences).

be active, in which case its child parameters could not be active either). At any given time, only a subset of the parameters are active, and the other parameters do not need to have a defined value. Formally, a hyperparameter γ_i which depends on parent hyperparameter γ_j will be referred to as active if $\gamma_j \in V_{j,i}$, where $V_{j,i} \subset \Gamma_j$ is the set of values for γ_j which should result in the activation of γ_i.

SMAC handles conditional parameters in the way described above [4]. However, in order to train a surrogate model of $f(\gamma)$ some values need to be given even for inactive parameters. In this case, an *imputation* strategy is defined to give a value to those inactive parameters. A common strategy is to give a default value to the inactive parameters, and let the surrogate model ignore those default values (either by generating a split for a random forest, or by having a distance of 0 for a kernel GP), as was done by Feurer *et al.* [3].

Even such a simple imputation scheme is vital to obtain properly behaving surrogate models. This can be illustrated with a simple example taking advantage of Gaussian Processes. With GPs, the impact of each sample over others can be measured through the covariance function, which can be drawn in 2D. The reader should note that this type of similarity analysis is not possible with other surrogate models such as the random forest.

Let us define a dummy search space with four parameters, one being a categorical classifier choice parameter and the other three being single parameters for each of those three classifiers (with values between 0 and 1). We can analyze the covariance of random samples from this space in order to observe the impact of imputation. A standard squared exponential kernel is used, giving the following covariance between samples:

$$k(x, x') = \exp\left(\frac{-\|x - x'\|_2^2}{\ell}\right), \tag{5}$$

where ℓ is a length-scale parameter manually set to 0.75 in this example. Figure 2 shows the resulting covariance matrix for two cases, the case where the evaluated hyperparameters were densely sampled in the unit cube $[0, 1]^4$, and the case where inactive parameters were imputed with a default value (in this case the central value of the space 0.5). It can be seen that the imputation has a positive impact on discrimination between different choices (labeled *a*, *b*, and *c* on the figure), however it can also be seen that even with imputation there is still confusion between the hyperparameters that are close in the search space, e.g. the pairs $a - b$ and $b - c$.

B. Surrogate Model Optimization

At each iteration t during a model-based optimization procedure, a so-called *acquisition function* is used to sample the next points to evaluate. This acquisition function uses a surrogate model trained on observations of $f(\gamma)$, and lends a value to each given potential hyperparameter to explore $a(\gamma|f(\cdot))$. The acquisition function balances between exploration and exploitation, in order to maximize the chances of finding the global optimum, or in the case of greedier acquisition functions some local optima. The next hyperparameters to evaluate are chosen through a proxy optimization $\gamma_t = \arg\max_\gamma a(\gamma|f(\cdot))$. Popular choices of acquisition functions include the Expected Improvement $a_{EI}(\gamma|\mu, \sigma) = \mathbb{E}\left[\max\left(0, f(\gamma) - f(\gamma_{best})\right)\right]$, where γ_{best} is the best found solution so far, and the Predictive Entropy Search, which entails maximizing the information gain about the location of the global optima [11].

One way to optimize the acquisition function is through random sampling, and then selecting hyperparameters which maximize the acquisition function. Another way is to directly optimize it, like any regular function, given that the surrogate model is much cheaper to evaluate than the true underlying function. Snoek *et al.* [7] use LBFGS to find the minimum of the acquisition function, while in [1] the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is used. These methods are suitable for dense spaces, but the conditionality of the search spaces considered in this work might be problematic for such optimization procedures. Constrained optimization techniques would hardly apply in this case given that the conditions are complex and cannot be formulated explicitly. For this reason, the SMAC toolbox [3] uses a technique called local search, which is similar to coordinate ascent, performing steps in single coordinates one at a time. The value of the acquisition function for a hyperparameter configuration γ is evaluated and compared with that of its neighboring configurations, by alternatively taking steps in each direction

for each *active parameter*. Given a starting point x, the local search updates the candidate hyperparameter tuple to evaluate for a number of iterations with the following rule:

$$\gamma_{i+1} = \arg\max_{\gamma \in S(\gamma_i)} a(\gamma | f(\cdot)), \tag{6}$$

$$S(\gamma_i) = \{\gamma_i\} \cup \{N_j(\gamma_i)\}_{j=1}^{|\Gamma|}, \tag{7}$$

where the function N_j returns the neighbors of a configuration according to hyperparameter or coordinate j. The neighbors vary according to the type of the hyperparameter: for a real number they are a step in both directions (with fixed step size), for a categorical hyperparameter the neighbors are every other value possible, and for integer values they are a step with plus or minus one.

IV. Kernels for Conditional Spaces

The main contribution of this paper is to highlight the need for better kernels in handling the structure of hierarchical hyperparameter spaces with Gaussian Processes. Staying within the framework of kernel-based methods is desirable because it allows us to inject knowledge concerning the problem through the kernel. Our first proposed kernel forces zero values on the covariance of hyperparameters in different branches of the hyperparameter hierarchy. Secondly, we propose to use an already existing kernel, the Laplace kernel, which has strong ties to random and Mondrian forests [12, 13]. The interest of the Laplace kernel is to have a kernel that behaves similarly to a random forest. These proposed kernels, combined with proper imputation of inactive values when it applies, allow for Bayesian optimization on problems such as combined algorithm selection and hyperparameter selection.

A. Conditional Kernel

The conditional kernel is based on the idea that observations from one condition branch should not influence the surrogate model on other condition branches. This can be encoded through a kernel that observes the conditional structure or hierarchy of the search space. Given two hyperparameter instances γ and γ', the indexes of active conditional hyperparameters C, and an underlying kernel k, the conditional kernel k_C is expressed as a wrapper around k:

$$k_C(\gamma, \gamma') = \begin{cases} k(\gamma, \gamma') & \text{if } \gamma_c = \gamma'_c \quad \forall c \in C \\ 0 & \text{otherwise} \end{cases}. \tag{8}$$

Here the conditions are stated as equalities for simplicity, but the essence is to check that the conditions activate the same children. Thus, if all active conditions are the same, it means that the two samples γ and γ' are directly comparable with the chosen kernel function k. The active conditions C are determined by descending a graph posed solely on the conditional hyperparameters. Let us take the same space as shown in Figure 1, where the graph to parse contains two conditions, θ_1 which would always be active, and θ_2 which would only be active when $\theta_1 = a$. This graph is descended and the descent stops once hyperparameters have no children. This is required to prevent the evaluation of inactive conditions, which could force a zero kernel value. The comparison of an active condition with an inactive condition is defined as being false, returning a zero kernel value (hence no shared information). It should be noted that forcing the length scale of the conditional variables to very low values should have the same effect as posing equality conditions, resulting in zero covariance between samples from different conditions. However, this method would not allow the definition of more complex conditions, such as conditions depending on more than one variable, and range conditions.

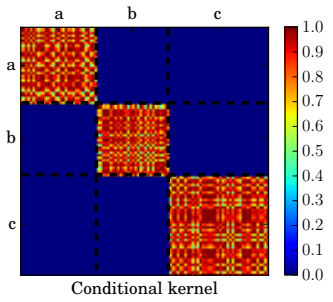

Fig. 3. Covariance matrix for the same sample and space used defined in Section III-A. The conditional kernel enforces a null covariance if samples have different choices for activated hyperparameters. The three blocks correspond to three different conditions (samples are sorted along the value of the conditional parameter to highlight differences).

This kernel has the attractive property of completely isolating otherwise unrelated hyperparameters, by returning a null covariance between hyperparameters from different branches. It is usable with a regular Gaussian Process as is. The definition of a derivative for such a covariance function is slightly more involved, however in our case it is not required. Indeed, the derivative could be used for gradient descent optimization of the acquisition function, but since the hyperparameter space is filled with holes, this might not lead to valid solutions.

The conditional kernel also produces a covariance matrix that is highly sparse. If the evaluated samples are sorted by groups of identical conditions, i.e. groups of values for which the condition of Equation 8 is true, then the covariance matrix will be filled with small dense squares of non zero-values centered on a 1-valued diagonal (since $k(x, x) = 1$). Figure 3 shows the values of the covariance matrix evaluated for the same example proposed for Figure 2.

The structure induced by the conditional kernel can be exploited to compute the GP posterior faster, an operation which involves the inversion of the covariance matrix K (or $K + \sigma_n^2 I$). Supposing a space with three conditions, we can express the covariance matrix of the conditional kernel as:

$$K_C = \begin{bmatrix} K_{cond_1} & 0 & 0 \\ 0 & K_{cond_2} & 0 \\ 0 & 0 & K_{cond_3} \end{bmatrix}, \tag{9}$$

where K_{cond_i} is a dense matrix of covariance for terms that fall under the same condition $cond_i$. The invert of this

matrix can be computed blockwise, which can be shown with straightforward manipulations (omitted here for space):

$$K_C^{-1} = \begin{bmatrix} K_{cond_1}^{-1} & 0 & 0 \\ 0 & K_{cond_2}^{-1} & 0 \\ 0 & 0 & K_{cond_3}^{-1} \end{bmatrix}. \quad (10)$$

This can represent a significant speedup on the longest operation present in the GP training procedure. The original complexity is roughly $\mathcal{O}(n^3)$, where n is the number of samples observed so far, and the new complexity becomes $\mathcal{O}(m_1^3 + \cdots + m_c^3)$, where m_i is the number of samples present in a subspace, with $n = \sum_{i=1}^{c} m_i$. The Cholesky decomposition is often used to solve the system of equations resulting from the GP problem. This decomposition can be segmented into a series of sub-problems, resulting in a similar speedup. This process is very similar to training a different GP for each conditional subspace (keeping all other factors constant).

B. Laplace Kernel

Lastly, we propose the use of the Laplace kernel, which has been shown to have ties with random forests. Balog *et al.* [13] have recently proposed the Mondrian kernel, a kernel which proceeds by drawing multiple samples of Mondrian processes, and encoding an explicit feature mapping from it. They show that this kernel acts as a randomized feature map to the Laplace kernel, converging to the Laplace kernel in the number of Mondrian process samples. The Laplace kernel is a l_1 distance-based kernel, which is rather similar in form to the squared exponential kernel:

$$k(x, x') = \exp\left(-\frac{\|x - x'\|_1}{\ell}\right). \quad (11)$$

Mondrian processes and Mondrian forests [12] have strong ties to random forests, where Mondrian forests draw partitionings as random Mondrian process samples and random forests obtain partitionings of the space through decision trees trained on random subspaces.

Random forests have been used in the past for Bayesian optimization due to their strong predictive accuracy, and potentially because they are better suited for higher dimensional spaces and discrete spaces [2]. However, one downside of random forests is that their predictive variance is usually computed by taking the variance of the output of the individual trees in the random forest [4]. What was observed empirically in [14] is that random forests are overconfident in their outputs. Therefore the Mondrian kernel and the Laplace kernel can result in models conceptually close to random forests, while maintaining the advantages of Gaussian Processes and allowing a completely Bayesian reasoning with respect to uncertainty.

As stated above, since the Mondrian kernel serves as a random feature approximation to the Laplace kernel and since the link between these and random forests has been clearly established in [13], we will be using the Laplace kernel for this work. The interest of using a Mondrian kernel in this case would be to have a noisier split of the data, or save time both in the tuning of the kernel λ and the training of the Gaussian Process, given that Mondrian kernels provide an explicit feature mapping. Figure 4 shows the resulting covariance matrix for the Mondrian[1] and Laplace kernels, with $m = 30$ Mondrian process samples for the Mondrian kernel and the same length-scale parameter as in the previous examples ($\ell = 0.75$, with the lifetime parameter of the Mondrian processes being the inverse of the length-scale $\lambda = 1/\ell$). We can see that the two kernels are indeed closely related, in fact at a quick glance they appear identical.

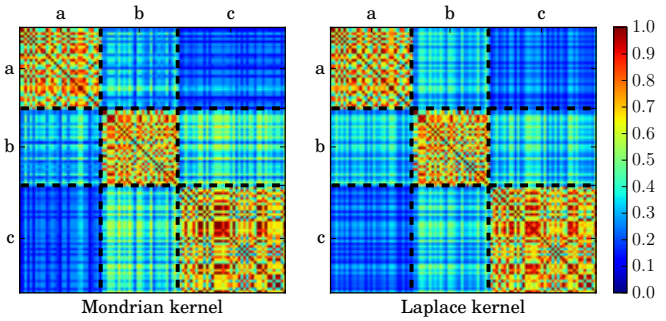

Fig. 4. Covariance matrix for the same sample and space used defined in Section III-A. The conditional kernel enforces a null covariance if samples have different choices for activated hyperparameters. The three blocks correspond to three different conditions (samples are sorted along the value of the conditional parameter to highlight differences).

We began with the rather noisy and blurry covariance estimates of the squared exponential kernels with or without imputation of Figure 2, and we have now defined much better suited kernels for conditional spaces. In the next section, these new kernels will be evaluated on a benchmark of problems.

V. Experiments

In order to assess the impact of using the suggested kernels for Bayesian optimization in comparison with state of the art methods, we compare them on a search space of medium scale with 14 hyperparameters and one level of conditionality. The search space consists of learning algorithms implemented in the scikit-learn library[2]. The models and their hyperparameters include the following:

- *K nearest neighbors (knn)* with the number of neighbors `n_neighbors` in $\{1, 2, \ldots, 30\}$;
- *RBF SVM (svm)* with the penalty `C` logarithmically in $[10^{-5}, 10^{5}]$ and the width of the RBF kernel γ_{RBF} logarithmically in $[10^{-5}, 10^{5}]$;
- *linear SVM (linsvm)* with the penalty `C` logarithmically scaled in $[10^{-5}, 10^{5}]$;
- *decision tree (dt)* with the maximal depth `max_depth` in $\{1, 2, \ldots, 10\}$, the minimum number of examples in a node to split `min_samples_split` in $\{2, 3, \ldots, 100\}$,

[1]Computed with source code from: github.com/matejbalog/mondrian-kernel
[2]Source: scikit-learn.org, also available directly from PyPI.

and the minimum number of training examples in a leaf `min_samples_leaf` in $\{2, 3, \ldots, 100\}$;
- *random forest (rf)* with the number of trees `n_estimators` in $\{1, 2, \ldots, 30\}$, the maximal depth `max_depth` in $\{1, 2, \ldots, 10\}$, the minimum number of examples in a node to split `min_samples_split` in $\{2, 3, \ldots, 100\}$, and the minimum number of training examples in a leaf `min_samples_leaf` in $\{2, 3, \ldots, 100\}$;
- *AdaBoost* (adab) with the number of weak learners `n_estimators` in $\{1, 2, \ldots, 30\}$;
- *Gaussian Naive Bayes (gnb)* and *Linear Discriminant Analysis (lda)* both without any hyperparameters;
- *Quadratic Discriminant Analysis (qda)* with the regularization `reg_param` logarithmically in $[10^{-3}, 10^{3}]$.

Finally, there is a root hyperparameter which is the choice of the learning algorithm – it determines which underlying hyperparameters should be active or inactive. Hyperparameters are rescaled in $[0, 1]$ for the surrogate model, and integer hyperparameters are rounded to the closest value. The classifier choice is represented as an integer. After the fact, it was remarked that a categorical variable would be more principled for the classifier choice, but some preliminary experiments showed no major difference. We used the ConfigSpace library provided by the maintainers of SMAC to define the search space along with the relationships between the hyperparameters[3].

The acquisition function used for all methods requiring one (i.e., all except random search and CMA) is the Expected Improvement. The hyperparameter optimization methods compared were all allocated a total of 200 observations (or classifiers trained). The methods are the following:

Random search (RS): An often overlooked baseline of comparison, random search still performs very well especially in complex search spaces. Hyperparameters are sampled uniformly in the search space defined above (in logarithmic or linear space, depending on the hyperparameter).

Bayesian optimization with GPs (Spearmint): This is the equivalent of the method proposed in [7], which is Bayesian optimization with slice sampling for GP hyperparameters and surrogate optimization to locate minima of the acquisition function. The kernel used is the Matern52 kernel as in the original article. No imputation of hyperparameters is done.

Sequential model-based algorithm configuration (SMAC): The implementation used is the Python version, or SMAC3[4]. No meta-features are used in order to be fair with other compared methods. Each configuration is allowed only one instance, and its performance is evaluated with the same 5-fold cross-validation procedure as other methods. Inactive hyperparameters are imputed to a default value (which is roughly the middle value of each space, so 0.5 for a parameter in $[0, 1]$).

Covariance matrix adaptation evolution strategy (CMA): The CMA-ES algorithm is also included to have a baseline of what a purely numerical optimization tool can achieve. Note that CMA-ES most likely deals very badly with conditional hyperparameters. The population size used is $\lambda = 4 + 3 \log d$, which is suggested as a rule of thumb in [15]. The DEAP toolbox is used[5]. No imputation of inactive parameters is performed.

Bayesian optimization with a GP and without imputation (GP-matern-noimpute): : This is similar to Spearmint, minus the optimization of the acquisition function. This extra comparison method is added to assess the importance of the imputation of hyperparameters.

Bayesian optimization with a conditional kernel (GP-cond): One of the proposed approaches, with the kernel defined in Equation 8, wrapped around a Matern52 kernel with automatic relevance determination. Length-scales and GP hyperparameters are determined through slice sampling. Note that there are two variants of this approach:

- one without local optimization (GP-cond), where the acquisition function is maximized over 1000 random samples from the search space;
- one with a local search optimization (GP-cond-ls), such as the one used in SMAC.

Bayesian optimization with a Laplace kernel: The other proposed kernel, with length-scales applied for each dimension (similar to the automatic relevance determination for the Matern52 kernel). Length-scales and GP hyperparameters are determined through slice sampling. There are also two variants of this approach, one without local optimization (GP-laplace) and one with a local search optimization (GP-laplace-ls). Inactive hyperparameters are imputed to a default value.

Bayesian optimization with a Matern52 kernel (GP-matern): In order to assess the impact of imputation, we compare against another variant of GP, where hyperparameter values used to condition the GP prior have inactive values imputed the same as the other methods in this section. There is also a variant of this method which had local optimization applied (GP-matern-ls).

All GP-based methods include the use of priors on GP hyperparameters, which are then tuned with slice sampling [7, 8]. A log normal prior with zero mean and unit standard deviation is placed on the length-scales $\boldsymbol{\ell}$ and the kernel amplitude σ_f, and a horseshoe prior with scale 1 is placed on the noise σ_n.

We evaluate the methods on a series of 24 datasets ranging from about 500 to 60K instances taken from UCI and OpenML[6]. All the datasets were preprocessed to have zero mean and unit standard deviation. At each iteration, the performance of chosen hyperparameters is evaluated with 5-fold cross-validation. Each experiment is repeated 10 times where

[3]Source code: github.com/automl/ConfigSpace

[4]Source code: github.com/automl/SMAC3

[5]DEAP: github.com/DEAP

[6]From UCI: Adult (adlt), Bank (bnk), Car (car), Chess-krvk (ches), Letter (ltr), Magic (mgic), Musk-2, Page-blocks (p-blk), Pima (pim), Semeion (sem), Spambase (spam), Stat-german-credit (s-gc), Stat-image (s-im), Stat-shuttle (s-sh), Steel-plates (s-pl), Titanic (tita), Thyroid (thy), Wine-quality-red (wine). Datasets identified by numbers were taken from OpenML, the numbers represents their ID in the OpenML database.

new data splits are sampled for each repetition, including a test with 20% of the data if it wasn't provided in the original dataset. The same splits are used across all methods for the same repetition.

A. Results and Discussion

Table I shows the empirical error rate of each method on the test set, averaged over repetitions. The best performance per dataset is highlighted in bold, and for each dataset the methods that are significantly worse than the best according to a Wilcoxon signed-rank test with α value 0.05 are underlined.

From Table I, we can see that no method significantly outperforms GP-cond-ls. In other words, it is not always the best method found (in bold), but it is never significantly different from the best. In comparison, GP-matern-ls is significantly worse than the best on only one dataset, while SMAC is worse on 10 datasets, and Spearmint is worse on 14 datasets. With respect to average ranks, we can see that the best method is again GP-cond-ls (2.56), followed by GP-matern-ls (4.10), and GP-matern (4.83).

When compared all together, the methods are significantly different according to a Friedman's test ($p = 9 \times 10^{-12}$). Figure 5 shows the result of a post-hoc Nemenyi test with α value of 0.05. All methods linked with a bold line are on the same level according to the Nemenyi test. Methods that have a difference in average rank over datasets greater than CD are deemed significantly different. We can see that optimization of the acquisition function is important to obtain a better performing solution, that is, all GP-X-ls methods perform better than their non-optimized counterparts. Finally, we can see that according to this test, the only method able to outperform a random search is GP-cond-ls, although it should be pointed that the Nemenyi test is rather conservative in its estimates. It might be interesting to note that GP-laplace-ls and SMAC perform on the same level and use roughly similar mechanisms, although one striking difference would be the fact that the length-scales of the Laplace kernel are optimized with slice sampling while the random forest hyperparameters are kept fixed for SMAC (default values used). This is surprising as we thought that a GP with a Laplace kernel would be able to outperform a random forest surrogate model by providing more reliable mean and variance estimates.

Table II shows the result of pairwise Wilcoxon signed-rank tests. Methods highlighted in bold show a significant difference between row and column, and parentheses highlight when the method on the corresponding row performs worse than the method in the corresponding column. From these tests, it is interesting to note that GP-cond-ls significantly outperforms all approaches apart from GP-matern-ls, and that the naive Spearmint application is significantly outperformed by all other methods, including the CMA evolution strategy.

B. Visualization of Classifier Choices

Another angle providing insight into the behavior of combined algorithm selection and hyperparameter optimization is to look at the final classifier choice, once the optimization procedure is over. Here we look at the results for a single optimization method for simplicity, GP-cond-ls. Figure 6 shows the distribution of classifier choices over all datasets and repetitions. We can see that the best performing classifiers are, by a wide margin, random forests and SVMs with an RBF kernel. Other classifiers do see limited use, although only around 5 or 10 percent of the time. Figure 7 provides further details, showing the classifier choices for each dataset in the benchmark. In this case, each column represents the 10 final classifier choices made for a single dataset. In this figure, we can see that some datasets benefit from different families of learning algorithms – for instance, the problem tita (titanic) always converged on a decision tree classifier, and dataset 0917 always resulted in a linear SVM classifier choice. Further analyses would definitely be interesting in this space of problems and learning algorithms, although this is left for future work.

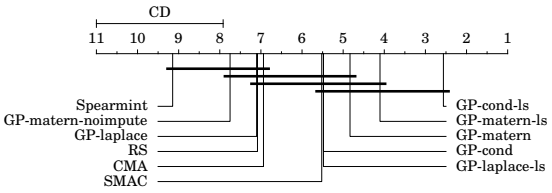

Fig. 5. Nemenyi post-hoc test. Methods joined by a bold horizontal line are not statistically different ($p > 0.05$).

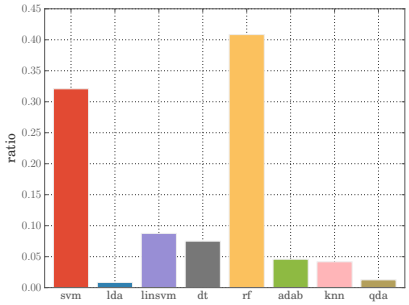

Fig. 6. Resulting classifier choices for an optimization with GP-cond-ls over all 24 datasets and 10 repetitions.

VI. Conclusion

In this paper we proposed two kernels for the optimization of hyperparameter spaces with conditional variables, such as combined algorithm selection and hyperparameter optimization. We also discussed the importance of proper imputation of values for inactive hyperparameters in order to obtain valid covariance estimates, rather than noise. Our comparison of suggested methods with the state of the art showed that one can indeed apply hyperparameter optimization with GPs and we show that the conditional kernel outperforms other suggested approaches. Future work should investigate different search spaces, and focus on large scale optimization problems.

TABLE I

AVERAGE TESTING ERROR OF EACH METHOD. BOLDFACE HIGHLIGHTS THE BEST PERFORMING METHOD PER DATASET, AND UNDERLINED METHODS HIGHLIGHT METHODS THAT WERE SIGNIFICANTLY DIFFERENT FROM THE BEST METHOD ACCORDING TO A WILCOXON SIGNED-RANK TEST ($p < 0.05$).

	46	184	389	772	917	1049	adlt	bnk	car	ches	ltr	mgic	msk	p-blk	pim	s-gc	s-im	s-pl	s-sh	sem	spam	thy	tita	wine	ranks
RS	0.09	18.39	14.20	46.61	46.17	11.85	14.44	10.70	1.45	20.73	3.14	12.76	0.64	3.34	23.83	23.25	3.77	24.09	0.09	6.07	5.67	1.10	24.06	37.30	7.08
Spearmint	0.22	33.02	20.32	44.63	46.37	12.05	15.53	10.83	6.16	31.77	8.81	13.61	2.66	3.17	23.44	23.10	4.52	27.42	0.86	9.47	6.86	2.09	21.50	35.62	9.15
SMAC	0.09	13.62	14.22	44.63	46.27	12.60	14.42	10.59	0.92	17.40	2.83	12.66	**0.26**	3.48	**23.38**	23.45	3.31	23.96	0.08	**4.40**	5.84	1.12	22.70	36.61	5.52
CMA	0.09	18.02	14.26	44.66	46.67	11.88	14.48	10.64	2.63	19.08	3.04	12.66	1.85	3.09	23.96	23.75	3.66	25.06	0.07	6.10	5.69	1.09	23.74	35.56	6.94
GP-matern-noimpute	0.11	25.11	17.52	44.63	46.82	11.61	14.50	10.61	4.65	26.45	3.87	12.40	2.38	3.05	23.51	23.50	3.42	25.91	0.08	8.98	5.87	1.61	23.79	33.23	7.75
GP-matern	0.09	12.68	**14.10**	45.32	48.36	**11.20**	14.36	10.72	0.75	18.29	3.16	12.53	0.27	2.55	23.70	23.35	3.51	23.53	0.05	4.98	5.27	1.11	21.36	33.54	4.83
GP-cond	0.11	18.72	15.48	44.91	47.21	15.92	**14.33**	10.51	**0.72**	18.75	3.47	12.44	0.28	2.50	24.29	23.75	2.84	24.04	0.04	4.89	5.28	1.10	21.77	32.76	5.48
GP-laplace	0.13	21.60	17.08	44.89	**45.97**	12.36	14.39	10.51	2.34	23.25	8.18	13.03	2.11	2.72	24.68	23.45	3.27	24.14	0.09	7.43	5.28	1.10	20.93	34.50	7.10
GP-matern-ls	**0.06**	**11.78**	20.10	45.60	46.27	11.88	14.39	10.35	0.84	**16.18**	3.48	12.49	0.26	2.60	24.61	23.10	2.71	**23.17**	**0.03**	5.29	5.01	**1.09**	21.52	33.17	4.10
GP-cond-ls	0.06	12.95	14.12	**44.56**	46.02	11.51	14.34	10.45	0.81	16.43	**2.62**	**12.22**	0.26	2.54	23.70	**23.05**	**2.64**	23.48	0.57	4.74	**4.86**	1.09	21.72	**32.20**	2.56
GP-laplace-ls	0.13	17.87	14.68	44.89	47.06	11.82	14.38	**10.20**	1.39	19.56	4.19	12.72	1.73	**2.49**	23.51	23.30	3.23	24.27	0.04	7.59	5.17	1.35	**20.45**	32.98	5.48

TABLE II

PAIRWISE WILCOXON SIGNED-RANK TESTS

	1	2	3	4	5	6	7	8	9	10	11
1 - RS	–	**0.01**	(**0.02**)	(0.76)	0.16	(**0.00**)	(0.19)	0.29	(**0.01**)	(**0.00**)	(0.24)
2 - Spearmint	(**0.01**)	–	(**0.00**)	(**0.00**)	(**0.00**)	(**0.00**)	(**0.00**)	(**0.00**)	(**0.00**)	(**0.00**)	(**0.00**)
3 - SMAC	**0.02**	**0.00**	–	**0.02**	**0.01**	(0.45)	(0.41)	0.17	(0.13)	(**0.00**)	(0.36)
4 - CMA	0.76	**0.00**	(**0.02**)	–	0.12	(**0.00**)	(0.20)	0.61	(**0.01**)	(**0.00**)	(0.16)
5 - GP-matern-noimp.	(0.16)	**0.00**	(**0.01**)	(0.12)	–	(**0.01**)	(**0.01**)	(0.11)	(**0.01**)	(**0.00**)	(**0.01**)
6 - GP-matern	**0.00**	**0.00**	0.45	**0.00**	**0.01**	–	0.61	**0.01**	(0.57)	(**0.03**)	0.24
7 - GP-cond	0.19	**0.00**	0.41	0.20	**0.01**	(0.61)	–	**0.02**	(0.36)	(**0.00**)	0.89
8 - GP-laplace	(0.29)	**0.00**	(0.17)	(0.61)	0.11	(**0.01**)	(**0.02**)	–	(**0.01**)	(**0.00**)	(**0.00**)
9 - GP-matern-ls	**0.01**	**0.00**	0.13	**0.01**	**0.01**	0.57	0.36	**0.01**	–	(0.10)	0.14
10 - GP-cond-ls	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.03**	**0.00**	**0.00**	0.10	–	**0.00**
11 - GP-laplace-ls	0.24	**0.00**	0.36	0.16	**0.01**	(0.24)	0.89	**0.00**	(0.14)	(**0.00**)	–

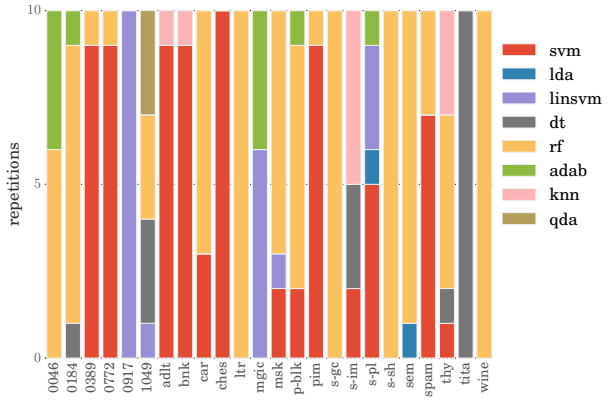

Fig. 7. Resulting classifier choices for an optimization with GP-cond-ls. Each column represents the distribution of classifier choices across the 10 repetitions on a single dataset. Figure best viewed in color.

ACKNOWLEDGEMENTS

This work was supported financially by the Mitacs program and E Machine Learning Inc. It also benefited from computing resources provided by Calcul Québec and Compute Canada.

REFERENCES

[1] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for Hyper-Parameter Optimization," in *Proceedings of NIPS*, 2011, pp. 2546–2554.

[2] C. Thornton, F. Hutter, Holger H. Hoos, and K. Leyton-brown, "Auto-WEKA : Combined Selection and Hyperparameter Optimization of Classification Algorithms," in *Proceedings of the 19th SIGKDD*, 2013, pp. 847–855.

[3] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter, "Efficient and Robust Automated Machine Learning," in *Proceedings of NIPS*, 2015.

[4] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential Model-Based Optimization for General Algorithm Configuration," *Learning and Intelligent Optimization*, pp. 507–523, 2011.

[5] M. W. Hoffman, B. Shahriari, and N. de Freitas, "On correlation and budget constraints in model-based bandit optimization with application to automatic machine learning," *Proceedings of the 17th AISTATS*, 2014.

[6] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. H. Hoos, and K. Leyton-Brown, "Towards an Empirical Foundation for Assessing Bayesian Optimization of Hyperparameters," in *NIPS Workshop on Bayesian Optimization in Theory and Practice*, 2013.

[7] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms," in *Proceedings of NIPS*, 2012.

[8] I. Murray and R. P. Adams, "Slice Sampling Covariance Hyperparameters of Latent Gaussian Models," in *Proceedings of NIPS*, 2010.

[9] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization," *ArXiv*, 2016.

[10] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets," *ArXiv*, 2016.

[11] J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani, "Predictive Entropy Search for Efficient Global Optimization of Black-box Functions," in *Proceedings of NIPS*, 2014.

[12] B. Lakshminarayanan, D. M. Roy, and Y. W. Teh, "Mondrian Forests: Efficient Online Random Forests," in *Proceedings of NIPS*, 2014.

[13] M. Balog, B. Lakshminarayanan, Z. Ghahramani, D. M. Roy, and Y. W. Teh, "The Mondrian Kernel," in *Proceedings of the 32nd UAI conference*, 2016.

[14] B. Lakshminarayanan, D. M. Roy, and Y. W. Teh, "Mondrian Forests for Large-Scale Regression when Uncertainty Matters," in *Proceedings of the 19th AISTATS*, 2016.

[15] N. Hansen, "The CMA evolution strategy: A tutorial," Inria, Tech. Rep., 2005.