Bayesian Hyperparameter Optimization: Overfitting, Ensembles and Conditional Spaces

Thèse

Julien-Charles Lévesque

Doctorat en génie électrique
Philosophiæ doctor (Ph.D.)

Québec, Canada

Bayesian Hyperparameter Optimization: Overfitting, Ensembles and Conditional Spaces

Thèse

Julien-Charles Lévesque

Sous la direction de:

Christian Gagné, directeur de recherche
Robert Sabourin, codirecteur de recherche

Résumé

Dans cette thèse, l'optimisation bayésienne sera analysée et étendue pour divers problèmes reliés à l'apprentissage supervisé. Les contributions de la thèse sont en lien avec 1) la surestimation de la performance de généralisation des hyperparamètres et des modèles résultants d'une optimisation bayésienne, 2) une application de l'optimisation bayésienne pour la génération d'ensembles de classifieurs, et 3) l'optimisation d'espaces avec une structure conditionnelle telle que trouvée dans les problèmes "d'apprentissage machine automatique" (AutoML).

Généralement, les algorithmes d'apprentissage automatique ont des paramètres libres, appelés hyperparamètres, permettant de réguler ou de modifier leur comportement à plus haut niveau. Auparavant, ces hyperparamètres étaient choisis manuellement ou par recherche exhaustive. Des travaux récents ont souligné la pertinence d'utiliser des méthodes plus intelligentes pour l'optimisation d'hyperparamètres, notamment l'optimisation bayésienne. Effectivement, l'optimisation bayésienne est un outil polyvalent pour l'optimisation de fonctions inconnues ou non dérivables, ancré fortement dans la modélisation probabiliste et l'estimation d'incertitude. C'est pourquoi nous adoptons cet outil pour le travail dans cette thèse.

La thèse débute avec une introduction de l'optimisation bayésienne avec des processus gaussiens (*Gaussian processes*, GP) et décrit son application à l'optimisation d'hyperparamètres. Ensuite, des contributions originales sont présentées sur les dangers du surapprentissage durant l'optimisation d'hyperparamètres, où l'on se trouve à mémoriser les plis de validation utilisés pour l'évaluation. Il est démontré que l'optimisation d'hyperparamètres peut en effet mener à une surestimation de la performance de validation, même avec des méthodologies de validation croisée. Des méthodes telles que le rebrassage des plis d'entraînement et de validation sont ensuite proposées pour réduire ce surapprentissage. Une autre méthode prometteuse est démontrée dans l'utilisation de la moyenne *a posteriori* d'un GP pour effectuer la sélection des hyperparamètres finaux, plutôt que sélectionner directement le modèle avec l'erreur minimale en validation croisée. Les deux approches suggérées ont montré une amélioration significative sur la performance en généralisation pour un banc de test de 118 jeux de données.

Les contributions suivantes proviennent d'une application de l'optimisation d'hyperparamètres pour des méthodes par ensembles. Les méthodes dites d'empilage (*stacking*) ont précédemment été employées pour combiner de multiples classifieurs à l'aide d'un métaclassifieur. Ces méthodes peuvent s'appliquer au résultat final d'une optimisation bayésienne d'hyperparamètres en conservant les meilleurs classifieurs identifiés lors de l'optimisation et en les combinant à la fin de l'optimisation. Notre méthode d'optimisation bayésienne d'ensembles consiste en une modification du pipeline d'optimisation d'hyperparamètres pour rechercher des hyperparamètres produisant de meilleurs modèles pour un ensemble, plutôt que d'optimiser pour la performance d'un modèle seul. L'approche suggérée a l'avantage de ne pas nécessiter plus d'entraînement de modèles qu'une méthode classique d'optimisation bayésienne d'hyperparamètres. Une évaluation empirique démontre l'intérêt de l'approche proposée.

Les dernières contributions sont liées à l'optimisation d'espaces d'hyperparamètres plus complexes, notamment des espaces contenant une structure conditionnelle. Ces conditions apparaissent dans l'optimisation d'hyperparamètres lorsqu'un modèle modulaire est défini – certains hyperparamètres sont alors seulement définis si leur composante parente est activée. Un exemple de tel espace de recherche est la sélection de modèles et l'optimisation d'hyperparamètres combinée, maintenant davantage connu sous l'appellation AutoML, où l'on veut à la fois choisir le modèle de base et optimiser ses hyperparamètres. Des techniques et de nouveaux noyaux pour processus gaussiens sont donc proposées afin de mieux gérer la structure de tels espaces d'une manière fondée sur des principes. Les contributions présentées sont appuyées par une autre étude empirique sur de nombreux jeux de données.

En résumé, cette thèse consiste en un rassemblement de travaux tous reliés directement à l'optimisation bayésienne d'hyperparamètres. La thèse présente de nouvelles méthodes pour l'optimisation bayésienne d'ensembles de classifieurs, ainsi que des procédures pour réduire le surapprentissage et pour optimiser des espaces d'hyperparamètres structurés.

Abstract

In this thesis, we consider the analysis and extension of Bayesian hyperparameter optimization methodology to various problems related to supervised machine learning. The contributions of the thesis are attached to 1) the overestimation of the generalization accuracy of hyperparameters and models resulting from Bayesian optimization, 2) an application of Bayesian optimization to ensemble learning, and 3) the optimization of spaces with a conditional structure such as found in automatic machine learning (AutoML) problems.

Generally, machine learning algorithms have some free parameters, called hyperparameters, allowing to regulate or modify these algorithms' behaviour. For the longest time, hyperparameters were tuned by hand or with exhaustive search algorithms. Recent work highlighted the conceptual advantages in optimizing hyperparameters with more rational methods, such as Bayesian optimization. Bayesian optimization is a very versatile framework for the optimization of unknown and non-derivable functions, grounded strongly in probabilistic modelling and uncertainty estimation, and we adopt it for the work in this thesis.

We first briefly introduce Bayesian optimization with Gaussian processes (GP) and describe its application to hyperparameter optimization. Next, original contributions are presented on the dangers of overfitting during hyperparameter optimization, where the optimization ends up learning the validation folds. We show that there is indeed overfitting during the optimization of hyperparameters, even with cross-validation strategies, and that it can be reduced by methods such as a reshuffling of the training and validation splits at every iteration of the optimization. Another promising method is demonstrated in the use of a GP's posterior mean for the selection of final hyperparameters, rather than directly returning the model with the minimal cross-validation error. Both suggested approaches are demonstrated to deliver significant improvements in the generalization accuracy of the final selected model on a benchmark of 118 datasets.

The next contributions are provided by an application of Bayesian hyperparameter optimization for ensemble learning. Stacking methods have been exploited for some time to combine multiple classifiers in a meta classifier system. Those can be applied to the end result of a Bayesian hyperparameter optimization pipeline by keeping the best classifiers and combining them at the end. Our Bayesian ensemble optimization method consists in a modification of the Bayesian optimization pipeline to search for the best hyperparameters to use for an ensemble, which is different from optimizing hyperparameters for the performance of a single model. The approach has the advantage of not requiring the training of more models than a regular Bayesian hyperparameter optimization. Experiments show the potential of the suggested approach on three different search spaces and many datasets.

The last contributions are related to the optimization of more complex hyperparameter spaces, namely spaces that contain a structure of conditionality. Conditions arise naturally in hyperparameter optimization when one defines a model with multiple components – certain hyperparameters then only need to be specified if their parent component is activated. One example of such a space is the combined algorithm selection and hyperparameter optimization, now better known as AutoML, where the objective is to choose the base model and optimize its hyperparameters. We thus highlight techniques and propose new kernels for GPs that handle structure in such spaces in a principled way. Contributions are also supported by experimental evaluation on many datasets.

Overall, the thesis regroups several works directly related to Bayesian hyperparameter optimization. The thesis showcases novel ways to apply Bayesian optimization for ensemble learning, as well as methodologies to reduce overfitting or optimize more complex spaces.

Contents

List of Tables

List of Figures

Of all the paths you take in life, make sure a few of them are dirt.

John Muir

Remerciements

J'aimerais d'abord remercier mon directeur Christian Gagné pour m'avoir offert de belles opportunités au cours de mon doctorat. Nos échanges étaient toujours fructueux, pertinents et enrichissants. Je remercie également mon codirecteur Robert Sabourin pour ses efforts soutenus dans ma supervision malgré nos rencontres sporadiques. Tu as su trouver le bon dosage entre la critique et l'encouragement pour me garder stimulé et motivé.

Merci aux nombreux organismes subventionnaires qui m'ont aidé lors de mon parcours, le FRQNT, le CRSNG et le programme Mitacs Accélération. Merci également aux groupes de Calcul Québec et Calcul Canada pour votre infrastructure et votre support. Merci à Thales Canada (Daniel Lafond, Jean-François Gagnon) et E Machine Learning (Éric Beaulieu) pour m'avoir permis de poursuivre ma recherche en milieu industriel.

Je me dois également de remercier mes collègues et amis du laboratoire LVSN (Max, Audrey, Marc-André, Yannick, Félix-Antoine, François-Michel et bien d'autres). Le laboratoire aura toujours été un endroit agréable où travailler et discuter – c'est grâce à vous. Merci pour toutes les conversations, pour tous les débats, pour le support technique et pour les gâteaux. Merci Max pour ta camaraderie et ton authenticité pendant les 16 dernières années, qui j'espère vont continuer.

Un merci particulier à mes amies Marie et Audrey, qui avec Gabrielle avez su créer de nombreux souvenirs à la fois désopilants et chérissables. Que ce soit pour le travail en groupe ou pour se décontracter, nos moments étaient toujours efficaces et appréciables.

De plus, j'aimerais remercier mes parents d'avoir su entretenir une curiosité intellectuelle chez moi, dès le plus jeune âge, sans pour autant m'étouffer. Vous avez été et continuez d'être une source d'inspiration.

Finalement, j'aimerais remercier Gabrielle, mon amour. Merci pour ton support, beau temps, mauvais temps. Je suis chanceux de t'avoir dans ma vie.

Chapter 1

Introduction

In the recent past, machine learning has been applied to solve regression, classification, clustering, and other problems in diverse fields of application. As time flows, the number of applications of machine learning grows exponentially. Some would say we are on the verge on a new industrial revolution where even cognitive tasks will be automated, a revolution which is propelled by the avalanche of data emerging from the increasingly interconnected world in which we live.

At their core, machine learning algorithms are optimization procedures, which given a training dataset learn a prediction function, producing class probabilities, continuous values, or other output depending on the problem at hand. Most machine learning algorithms, with a few rare exceptions, have some free parameters that define their behaviour, regulate their optimization, or in some cases define the structure of their models. We will call these free parameters *hyperparameters*, in contrast to the *parameters* of the models which are optimized by learning algorithms.

The problem with hyperparameters is that they are often left to the choice of the machine learning practitioner, often with no established procedure to select them. Traditionally, they were selected by building a combinatorial grid over possible values for each individual hyperparameter, and all the resulting tuples would be tested out independently – a procedure called a *grid search*. Hyperparameters are evaluated by posing a model with them (this usually requires the retraining of the model), and then they are evaluated on a dataset that does not overlap with the training set, the validation set. For example, given two parameters and their sets of possible values $a \in \{1, 2\}$ and $b \in \{3, 4\}$, the grid would become $\{(a = 1, b = 3), (a = 1, b = 4), (a = 2, b = 3), (a = 2, b = 4)\}$; it is the product of both individual sets. Grid search suffers from some very obvious drawbacks such as an algorithmic complexity that is exponential in the number of hyperparameters. With an equal number of observations, grid search is easily outperformed by a random search (Bergstra and Bengio, 2012). This is explained by the fact that random search covers the search space independently for each dimension. Additional uninformed search methods include sobol sequences (Sobol, 1967) and other forms of quasi-random low-discrepancy sequences.

Hyperparameter optimization is the process of applying a second layer of optimization on top of the base learning algorithm, in order to carefully tune its hyperparameters. The field of hyperparameter optimization benefits from a rich literature on Bayesian optimization (also known as sequential model-based optimization, SMBO), that is concerned with the optimization of unknown functions (Hennig and Schuler, 2012; Jones, 2001; Mockus et al., 1978; Wang et al., 2012). Bayesian optimization is perfectly suited for the problem of hyperparameter optimization. Given that the cost of evaluating the objective function is high (training a model), it is advisable to spend some quantity of time in choosing the next point to evaluate. This is seen in the swiftly growing literature on Bayesian hyperparameter optimization (Bartz-Beielstein et al., 2005; Feurer et al., 2015a; Hutter et al., 2011; Shahriari et al., 2016; Snoek et al., 2012; Snoek et al., 2015; Thornton et al., 2013).

This thesis concentrates on hyperparameter optimization and its applications. We will mostly be using Bayesian optimization with Gaussian processes (GPs; Rasmussen and Williams, 2006) given their desirable properties as a probabilistic regression model, such as uncertainty estimation. However, the contributions of the thesis are mostly agnostic to the choice of the hyperparameter optimization method, meaning that they should remain relevant no matter the underlying optimization mechanism – apart from Chapter 5, which is specifically aimed at GPs. There exist many other approaches that can be applied to optimize hyperparameters, and we will cover some of those through the chapters of this thesis.

Hyperparameter optimization is also gradually enabling the emergent field of *automatic machine learning* (AutoML). AutoML is the automation of the creation and configuration of machine learning pipelines, including the preprocessing and transformation of data, the selection of a machine learning algorithm and the tuning of all hyperparameters involved. This approach is justified by the fact that no single learning algorithm is optimal for all problems (known as the no free lunch theorem; Wolpert, 1996), and also by the fact that some algorithms rely heavily on hyperparameter tuning, such as kernel

SVMs. The intent is that when the field of AutoML will be mature and well-established, the application of a prediction algorithm on any defined problem will require nothing more than the press of a button, effectively removing the machine learning expert from the loop. The said machine expert will then be free to concentrate on more complex challenges, such as general artificial intelligence.

The thesis is focused on hyperparameter optimization, with contributions at different levels. The first contribution will be on the topic of overfitting in the selection of hyperparameters during their optimization. The second contribution is an application of hyperparameter optimization to ensemble methods. The last contribution of the thesis concerns the solving of AutoML problems – also known as combined algorithm selection and hyperparameter optimization problems (CASH problems; Thornton et al., 2013) – as well as discussing properties of the hyperparameter spaces generated by such problems. In the following, the objectives of the thesis and the resulting research questions will be detailed.

Objective 1: Assess the impact of overfitting in hyperparameter optimization and propose strategies to reduce it. It has been shown for evolutionary algorithms that evaluating many times in succession on a validation dataset and choosing a model with regards to the performance on the said validation set can result in overfitting (Igel, 2013). Unsurprisingly, this was also shown to happen for model selection (Cawley and Talbot, 2010). Thus, one of the problems targeted by this thesis will be the assessment of this overfitting in the context of Bayesian hyperparameter optimization. Given the presence of overfitting in the optimization of hyperparameters, we will then propose some strategies to obtain better estimates of the generalization performance of hyperparameters.

The work associated with this objective aims at answering the following research questions:

1. Does significant overfitting occur with regards to the cross-validation data during the Bayesian optimization of hyperparameters?
2. Can we alleviate this overfitting by dynamic resampling of training and validation splits?
3. Can the use of a selection split, only observed at the end of the optimization, further reduce the overfitting?

Objective 2: Propose an application of hyperparameter optimization to generate heterogeneous ensembles of classifiers. The second objective of the thesis will be to showcase an application of hyperparameter optimization, namely the generation of better ensembles of classifiers. The justification for the introduction of ensemble methods in this work is simple. Ensemble methods have been shown to outperform their single model counterparts on many occasions (Bell and Koren, 2007; Breiman, 2001; Kuncheva, 2004; Snoek et al., 2015, etc.), through a reduction of variance of predictions by combination. Hence, it seems promising to combine ensemble learning and hyperparameter optimization.

The objective could be split in two parts, with the first part being to show that it is possible to generate an ensemble for free as we optimize the hyperparameters of a learning algorithm. The second part is to propose a framework allowing Bayesian optimization of hyperparameters for ensembles, and thus optimize hyperparameters and models directly suited for combining in an ensemble. There are many approaches that might seem suitable for this problem. We will discuss them and show which ones are feasible without duplicating the training of classifiers needlessly.

The research questions associated with this objective are:

1. What mechanism of optimization allows for the efficient optimization of hyperparameters for ensembles?
2. Can we obtain better performing ensembles by directly optimizing hyperparameters for the performance of the ensemble in comparison with an optimization for the single-best classifier?

Objective 3: Propose methods for the optimization of hyperparameters with conditionality structures. Some hyperparameters have structure embedded in their definition. For instance, a deep neural network has more hyperparameter to define than a shallower network – depending on the number of layers, the type of layers, and so on. Such hyperparameters also appear when AutoML problems are considered, for instance, where one hyperparameter is the choice of the learning or preprocessing algorithm. The parameters of different learning algorithms usually have nothing in common with each other and should not be compared explicitly with a distance-based kernel.

The research questions this last work aims at answering are:

1. What is the impact or importance of inactive hyperparameter value imputation for spaces with a conditional structure?

2. Can the Bayesian optimization of hyperparameters in a space with conditions be improved by the use of a kernel forcing zero covariance between samples from different conditions?

The remainder of the thesis is structured as follows. Chapter 2 will cover the basics of Bayesian optimization and show how it can be applied to hyperparameter optimization. Chapter 3 will introduce the concept of overfitting and discuss some situations where it can happen, followed by an extensive study of overfitting in the context of hyperparameter optimization. Chapter 4 showcases an application of Bayesian optimization to generate ensembles of classifiers by optimizing directly for the best classifier to add to an ensemble with regards to hyperparameters. Chapter 5 proposes some new approaches for the optimization of conditional hyperparameter spaces, such as are often found in AutoML problems. Finally, Chapter 6 wraps up the thesis with some concluding remarks.

Chapter 2

Bayesian Hyperparameter Optimization

In this chapter, we will introduce the Bayesian optimization framework and show how it can be applied for hyperparameter optimization. Bayesian optimization will be the main tool for all the contributions contained in this thesis, so it deserves a thorough introduction.

Bayesian optimization has already been used for decades for the optimization of black-box functions (Mockus et al., 1978), a process also known as derivative-free optimization. The main characteristics of this setting are that functions are not analytically solvable and can be expensive to evaluate, thus creating a need to keep the number of evaluations low. Popular tools for derivative-free optimization include evolutionary algorithms such as the covariance matrix adaptation evolution strategy (CMA-ES; Hansen, 2005), and variants of Bayesian optimization, also known as Sequential Model-based Optimization (SMBO; Hutter et al., 2011; Jones et al., 1998). CMA-ES is a population-based approach, whose state is defined by its population and some internal parameters, such as the so-called covariance matrix. On the other hand, the state of a Bayesian optimization procedure is strictly defined by the set of observed inputs and corresponding values for the optimized function – the fact that Bayesian optimization has no internal state will be instrumental for the problem treated in Chapter 4.

Typically, we are looking for the minimum of an unknown function $f(\mathbf{x})$, with $f : \mathcal{X} \to \mathbb{R}$:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} f(\mathbf{x}), \tag{2.1}$$

where $\mathbf{x}$ is the parameter of interest in a bounded space $\mathbf{x} \in \mathcal{X}$. Bayesian optimization aims at solving the problem defined above iteratively, by reusing all information acquired at each iteration, or all the values that were observed for the unknown function $f(\mathbf{x})$. This can be done by posing a *surrogate* model of the function $f(\mathbf{x})$, which will then be used to determine the next point of f to evaluate $\mathbf{x}'$ balancing exploration and exploitation criteria to be defined. The key element of Bayesian optimization is to exploit all the available information to guide the optimization procedure at each iteration. For that purpose, a history of observed points is accumulated, $H = \{\mathbf{x}_i, f(\mathbf{x}_i)\}_{i=1}^{t}$. This history is used to guide the exploration and optimization of the function of interest. In this context, gradient descent can be described as a purely exploitative strategy which only uses local information in the form of the current position $\mathbf{x}_t$ and the local curvature, or the gradient.

There can be many choices of models or priors for the function $f(\cdot)$. In the following, we will briefly discuss two common choices for Bayesian optimization, along with their strengths and weaknesses. Gaussian Processes (GPs; Rasmussen and Williams, 2006; Snoek et al., 2012) are the most prevalent model choice for Bayesian optimization, having a lot of desirable properties. GPs can be described succinctly as a kernel-based Bayesian regression method. GPs make an assumption of Gaussian noise over the function of interest, which may not be suitable for all problems. GPs allow for the definition of a posterior probability distribution over functions, given some observations with Gaussian noise. A GP also induces a Gaussian distribution in every point of the space $\mathcal{X}$, which results in variance or uncertainty estimates over the covered space. Those variance estimates are key elements for Bayesian optimization; they essentially represent our knowledge about the input space. The other common model used for Bayesian optimization is random forests (RFs; Breiman, 2001; Hutter et al., 2011), an ensemble of decision trees trained on random subspaces of the input data. RFs are a strong classification and regression tool and can generally scale better than GPs. However, their variance estimates come from computing the variance of the outcomes for each individual tree in the forest, and it has been shown that these variance estimates are unrealistically small (Lakshminarayanan et al., 2016) – in other words the model is overconfident. For this reason, we will focus on GPs for the remainder of this chapter and most of the thesis.

2.1 Gaussian Process

A GP defines a probability distribution over functions and is entirely specified by its mean and covariance functions:

$$m(\mathbf{x}) = \mathbb{E}\left[f(\mathbf{x})\right], \tag{2.2}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}\left[(f(\mathbf{x}) - m(\mathbf{x}))\,(f(\mathbf{x}') - m(\mathbf{x}'))\right], \tag{2.3}$$

and the GP is defined as

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')). \tag{2.4}$$

The covariance function encodes the smoothness of the functions described by the GP with regards to the samples used to condition it. Given a set of n pairs of inputs and their observed function values, $\{\mathbf{x}_i, y_i\}_{i=1}^n$, a multivariate Gaussian distribution on $\mathbb{R}^n$ is induced. This multivariate distribution can now be used to query unobserved points $\mathbf{x}'$ in the input space $\mathcal{X}$, rendering mean and variance information. Some random Gaussian noise is assumed on observations:

$$y = f(\mathbf{x}) + \varepsilon, \tag{2.5}$$

with $\varepsilon \sim \mathcal{N}(0, \nu)$. The predictive mean and variance functions for a Gaussian process at a given test point $\mathbf{x}_*$ are then given by

$$\hat{\mu}(\mathbf{x}_*) = \mathbf{k}_*^T (K + \nu I)^{-1} \mathbf{y}, \tag{2.6}$$

$$\hat{\mathbb{V}}(\mathbf{x}_*) = \hat{\sigma}^2(\mathbf{x}_*) = k(\mathbf{x}_*, \mathbf{x}_*) + -\mathbf{k}_*^T (K + \nu I)^{-1} \mathbf{k}_*, \tag{2.7}$$

where $\mathbf{k}_*$ is the covariance of the query input point $\mathbf{x}_*$ with each of the observations in the history $X = \{\mathbf{x}_i\}_{i=1}^n$, and K is the covariance matrix of the inputs X. The *training* of the Gaussian process mostly consists of the inversion of the matrix $(K + \nu I)$, which has a complexity of $\mathcal{O}(n^3)$, where n is the number of observations. Ideally, the number of observations stays low (a few hundred points at most), and the time required to evaluate the underlying function will be many orders of magnitudes higher than that of training of the GP, hence justifying its application. The theory and practice of GPs is explained in greater detail in the book of Rasmussen and Williams (2006).

Figure 2.1 shows an example of a GP posterior, where a few noisy observations of a simulated function $f(\mathbf{x})$ have been gathered. The observations at the current moment $\mathcal{H} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ are used to compute a posterior distribution. The orange curve and its surrounding shaded area represent the predictive mean and variance functions (of Eq. 2.6 and 2.7) for the conditioned GP, with the shaded area representing the spread for one standard deviation. This shaded area represents our confidence about the value of the function, and it should be noted that it shrinks around observed points.

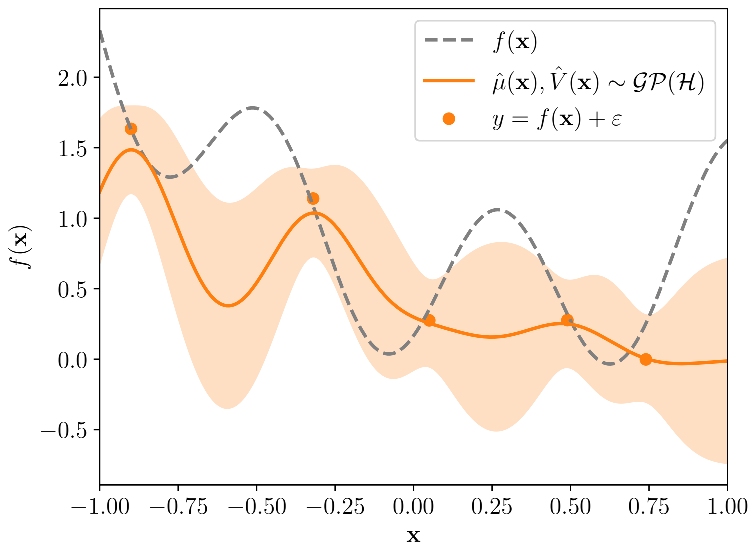


Figure 2.1: Example of Gaussian process. The dashed line shows the true function, the orange dots show observed samples of the function, and the orange line and envelope show the GP posterior mean and variance functions (for one standard deviation).

2.2 Acquisition Function

The next step is to exploit the posterior distribution induced by observations to choose the next point to evaluate in the input space. Let us introduce the *acquisition function* $a : \mathcal{X} \to \mathbb{R}$, a measure of the utility provided by the evaluation of a given point in the input space. The acquisition function will serve to determine the next point to evaluate through the proxy optimization

$$\mathbf{x}_{t+1} = \arg\max_{\mathbf{x}} a(\mathbf{x}). \tag{2.8}$$

Several acquisition functions have been suggested in the literature, most of which aim at balancing exploration and exploitation. One such function is the *expected improvement* (EI; Mockus et al., 1978):

$$a_{EI}(\mathbf{x}|\, \hat{\mu}(\mathbf{x}), \hat{\sigma}^2(\mathbf{x})) = \mathbb{E}\Big[\max\{0, f_{best} - \hat{\mu}(\mathbf{x})\}\Big], \tag{2.9}$$

where f_{best} is the lowest objective value observed so far during the optimization. This acquisition function has a closed form solution that can be computed with the mean and variance functions of the posterior (Jones, 2001):

$$a_{EI}(\mathbf{x}|\, \hat{\mu}(\mathbf{x}), \hat{\sigma}^2(\mathbf{x})) = \hat{\sigma}(\mathbf{x}) \left(z\, \Phi(z) + \phi(z)\right), \tag{2.10}$$

where

$$z = \frac{f_{best} - \hat{\mu}(\mathbf{x})}{\hat{\sigma}(\mathbf{x})}, \tag{2.11}$$

and $\Phi(\cdot)$ and $\phi(\cdot)$ are the normal cumulative distribution and density functions, respectively.

The *probability of improvement* (PI; Kushner, 1964) over the best value was previously used for Bayesian optimization, but it often results in greedier choices, with expected improvement showing better empirical performance. The *upper confidence bound* (UCB) has also been studied for Bayesian optimization with GPs (Srinivas et al., 2010), with some interesting theoretical guarantees, however it introduces a new free parameter which needs tuning. Finally, other acquisition functions of interest include the *entropy search* (ES; Hennig and Schuler, 2012), which aims at maximizing the information gain about the position of the global minimizer of the function. Similarly, the *predictive entropy search* (PES; Hernández-Lobato et al., 2014) acquisition function is a different approach for maximizing the information gain, which results in better properties, namely a completely Bayesian handling of the GP hyperparameters. Throughout this thesis, we will use the expected improvement acquisition function because it is straightforward to compute and has been shown to perform well. The design of acquisition functions in itself is an important avenue for research, but it is not the focus of this work.

The proxy optimization of the acquisition function (Eq. 2.8) can be accomplished through gradient descent in GPs, by propagating the gradient through the underlying model, and optimizing with L-BFGS (Snoek et al., 2012), something which is not possible with random forests. The non-convexity of the objective function is handled through multiple restarts with different starting points, and the best result is chosen at the end. Others have applied derivative free optimization methods, such as CMA-ES (Bergstra et al., 2011), and methods related to coordinate ascent in the SMAC toolbox[1], or simply random sampling. In most cases, we will optimize the acquisition function with L-BFGS and a gradient propagated from the underlying GP model. This matter will be discussed further in Chapter 5. This proxy optimization is justified by the fact that computing the acquisition function is extremely cheap compared to evaluating the unknown function $f(\mathbf{x})$, so it should be optimized properly.

Figure 2.2 shows the same example as above with the addition of an acquisition function, in this case the expected improvement. It can be seen that the acquisition function has local optima in areas where uncertainty is highest, and around areas close to the current minimum. The minimum of the acquisition function can be found through any

[1]The original SMAC publication (Hutter et al., 2011) makes no mention of this acquisition function optimization technique.

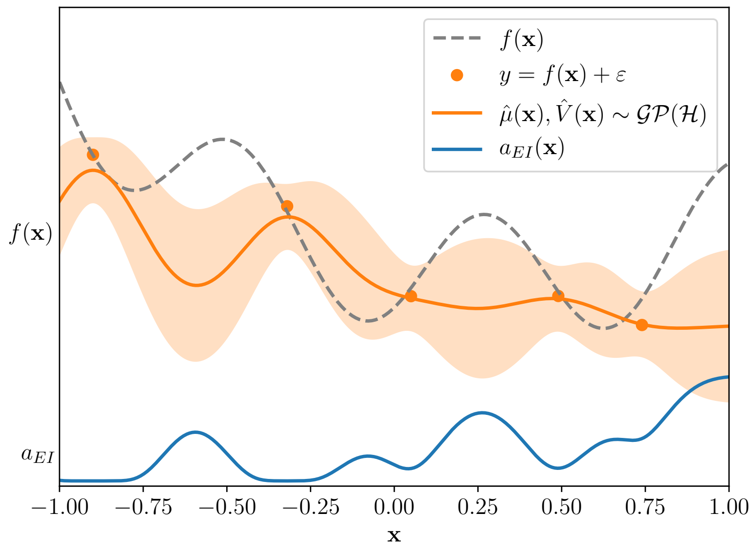


Figure 2.2: Example of GP with expected improvement acquisition function. Above, the objective function to minimize and its posterior model. Below, the acquisition function to maximize obtained from the posterior distribution.

Algorithm 2.1 Bayesian optimization of an unknown function with Gaussian processes.

```
Input: $B$ the maximal number of iterations
1: $\mathcal{H}_0 \leftarrow \varnothing$
2: for $t \in 1, \ldots, B$ do
3:     $\hat{\mu}(\mathbf{x}), \hat{\sigma}^2(\mathbf{x}) \leftarrow \mathcal{GP}(\mathcal{H}_{t-1})$   // Fit GP, get mean and variance functions
4:     $\mathbf{x}_t \leftarrow \arg\max_{\mathbf{x}} a(\mathbf{x} | \hat{\mu}(\mathbf{x}), \hat{\sigma}^2(\mathbf{x}))$   // Choose next point
5:     $y_t \leftarrow f(\mathbf{x}_t) + \varepsilon$   // Compute function value
6:     $\mathcal{H}_t \leftarrow \mathcal{H}_{t-1} \cup \{(\mathbf{x}_t, y_t)\}$   // Update observations
7: end for
8: return $\arg\min_{(\mathbf{x},y) \in \mathcal{H}_T} y$   // Return best found value
```

of the proxy optimization methods mentioned in the previous paragraph.

A step by step description of Bayesian optimization is shown in Algorithm 2.1. The procedure consists of iterating the steps of computing the posterior distribution, using it to find the maximum of the acquisition function, and evaluating the function at the new point. The process continues until either a maximum number of iterations or time limit is reached.

2.3 Hyperparameter Optimization

Choosing the hyperparameters of learning algorithms can be done through Bayesian optimization, and has gotten increasingly more common and accessible over the past years (Bergstra et al., 2011; Hutter et al., 2011; Snoek et al., 2012). Hyperparameters can be defined as any free parameter of a learning algorithm that is not optimized directly by the learning procedure. Examples include the width of an SVM kernel, the number of neurons or layers in a neural network, the learning rate for gradient descent, the type of preprocessing applied, and so on and so forth. In the present context, they are called *hyperparameters* in opposition to the learned model *parameters*, which are also known as model weights. This defines a bi-level optimization problem (Guyon et al., 2010), where on the first level the learning algorithm is run on a training dataset, and on the second, upper level, we cycle through the fitting of many models in order to find the best hyperparameters.

Taking the case of classification, let us assume pairs of labelled examples drawn from an unknown underlying distribution $(\mathbf{x}, y) \sim \mathcal{D}$, with $\mathbf{x} \in \mathbb{R}^d$ and $y \in \mathcal{Y}$. A finite number of such sample pairs are gathered and split in training and validation datasets $\mathcal{D}_T$ and $\mathcal{D}_V$, and a third set $\mathcal{D}_G$ is kept out of reach to estimate the generalization error at the very end, often called the testing set. Any given learning algorithm A is most likely going to have free hyperparameters $\gamma \in \Gamma$ defining its behaviour[2]. This learning algorithm is used to produce a classifier or hypothesis from the training data $\mathcal{D}_T$:

$$h_\gamma = A(\gamma, \mathcal{D}_T). \tag{2.12}$$

In the following, h_γ will be used as a short notation for a classifier trained with hyperparameters γ, with $h_\gamma : \mathbb{R}^d \to \mathcal{Y}$. Hyperparameter optimization attempts to find the best hyperparameters for the learning algorithm A:

$$\gamma^* = \underset{\gamma \in \Gamma}{\arg\min} \underset{(\mathbf{x},y)\sim\mathcal{D}}{\mathbb{E}} \ell(h_\gamma(\mathbf{x}), y), \tag{2.13}$$

where $\ell(\cdot, y)$ is a metric of error for the problem at hand (popular examples include the zero-one loss for classification, the mean squared error for regression models, and the cross entropy for probabilistic models). Since the distribution $\mathcal{D}$ is unknown, Equation 2.13 cannot be solved exactly, and it is instead minimized with regards to the validation dataset $\mathcal{D}_V$:

$$L(h_\gamma, \mathcal{D}_V | \, \ell) = \frac{1}{|\mathcal{D}_V|} \sum_{(\mathbf{x},y)\in\mathcal{D}_V} \ell(h_\gamma(\mathbf{x}), y). \tag{2.14}$$

[2]The usual boldface notation will be omitted for γ, as it will almost always refer to a vector of hyperparameters.

Algorithm 2.2 Bayesian optimization of a learning algorithm's hyperparameters.

```
Input: $\mathcal{D}_T$ and $\mathcal{D}_V$, some training and validation datasets, $A$ the learning algorithm, and $\ell$ the loss function
1: $\mathcal{H}_0 \leftarrow \varnothing$
2: for $t \in 1, \ldots, B$ do
3:     $\hat{\mu}(\gamma), \hat{\mathbb{V}}(\gamma) \leftarrow \mathcal{GP}(\mathcal{H}_{t-1})$   // Fit GP, get mean and variance functions
4:     $\gamma_t \leftarrow \arg\max_\gamma a(\gamma | \hat{\mu}(\gamma), \hat{\mathbb{V}}(\gamma))$   // Choose next hypers
5:     $h_{\gamma_t} \leftarrow A(\gamma_t, \mathcal{D}_T)$   // Train model
6:     $L_{\gamma_t} \leftarrow L(h_{\gamma_t}, \mathcal{D}_V | \ell)$   // Compute validation loss
7:     $\mathcal{H}_t \leftarrow \mathcal{H}_{t-1} \cup \{(\gamma_t, L_{\gamma_t})\}$   // Update observations
8: end for
9: return $\arg\min_{(h, L_h) \in \mathcal{H}_T} L_h$   // Return best model
```

Without knowledge about the underlying algorithm A, the minimization problem posed in Equation 2.13 is effectively a black-box optimization problem. In other words, if one wants to define a methodology to optimize the hyperparameters of *any* learning algorithm, the only solution is black-box optimization or Bayesian optimization. With knowledge about the underlying algorithm A, one can devise algorithms or methods to find the exact best hyperparameters for the algorithm, which has been done in the past (Cawley and Talbot, 2007; Jun et al., 2016), but as one would expect this is much more involved and it cannot work for all hyperparameters and learning algorithms. If we take a neural network as an example, one cannot infer the impact an extra layer will have on the prediction function, and thus finding an analytical solution or gradient is infeasible.

Algorithm 2.2 shows the step-by-step procedure for Bayesian optimization of hyperparameters. The only difference with Algorithm 2.1 is the evaluation of the function $f(\cdot)$ which now requires the training of a model. One can also replace steps 5-6 of the algorithm by a cross-validation loop, which trains multiple models and evaluates them on multiple validation splits, for example with a k-fold cross-validation methodology Alpaydin, 2014. This helps to limit overfitting in the selection of hyperparameters and will be the subject of Chapter 3.

2.4 Surrogate Model Hyperparameters

Another important piece of the Bayesian optimization pipeline is the treatment of the surrogate model's own hyperparameters, or one could say (hyper)^{2}parameters. For a Gaussian process, those are the noise scale ν, the amplitude of the covariance function

σ_a^2, the d length scales $\boldsymbol{\ell}$ for an automatic relevance determination (ARD) type kernel[3], and lastly the value of the mean function m, which is often assumed to be constant through the space. The first three hyperparameters are used in the kernel function

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_a^2 \exp\left(\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^T \mathrm{diag}(\boldsymbol{\ell})^{-2}(\mathbf{x}_i - \mathbf{x}_j)\right) + \nu\delta_{ij}, \tag{2.15}$$

where δ_{ij} is the Kronecker delta ($\delta_{ij} = 1$ iff $i = j$ and 0 otherwise). Here, the likelihood will be used to optimize the hyperparameters of the GP. Under the GP model, the likelihood of the observations $(X, \mathbf{y})$ is a Gaussian $\mathbf{y} \sim \mathcal{N}(m\mathbf{1}, K + \nu I)$, where $\mathbf{1}$ is a vector with of 1-valued entries (Rasmussen and Williams, 2006). The log-likelihood of the data is then given by

$$\log p(\mathbf{y}|X) = -\frac{1}{2}\mathbf{y}^T(K + \nu I)^{-1}\mathbf{y} - \frac{1}{2}\log|K + \nu I| - \frac{n}{2}\log 2\pi. \tag{2.16}$$

One way to determine the hyperparameters of the GP is to optimize this log-likelihood directly. In order to get a fully Bayesian treatment of the hyperparameters, we will use *slice sampling*, which samples from the posterior distribution over GP hyperparameters (Murray and Adams, 2010; Snoek et al., 2012). Integrating over these samples results in a Monte Carlo estimation of the GP model over the whole distribution of hyperparameters. These create an integrated acquisition function over the GP hyperparameters, defined as $\theta = (m, \sigma_a^2, \nu, \boldsymbol{\ell})$:

$$\hat{a}(\mathbf{x}) = \int a(\mathbf{x}|\theta)p(\theta|X, \mathbf{y})d\theta, \tag{2.17}$$

with $p(\theta|X, \mathbf{y}) \propto p(\mathbf{y}|X, \theta)p(\theta)$, and where $p(\theta)$ is the prior. When using GPs, the hyperparameters will be tuned using this slice sampling methodology, taken from Snoek et al., 2012.

2.5 Related Works

As far as hyperparameter optimization goes, many methods can be considered other than Bayesian optimization, or the now well-established approach with GPs described in this chapter. Bergstra and Bengio (2012) have shown that simple random search performs surprisingly well, effectively eliminating any reason to use brute force methods such as grid search. Black-box optimization techniques such as CMA-ES can also be applied, having been shown more efficient than Bayesian optimization

[3]The ARD kernel is often chosen because it allows for the scaling of each dimension of the input space independently, see Rasmussen and Williams (2006) for more details.

on some high-dimensional benchmarks (Hutter et al., 2013). Bayesian optimization has some advantages with regards to those methods, mainly revolving around the use of a probabilistic modelling of the objective function, which allows for a principled exploration of the search space.

Hutter et al. (2011) explored the use of random forests for Bayesian optimization, presenting the sequential model-based algorithm configuration (SMAC) library. SMAC has been used with some success to optimize complex search spaces such as AutoWEKA (Thornton et al., 2013) or AutoSklearn (Feurer et al., 2015a), both problems which aim at finding the best model and preprocessing method from most or all models available in given machine learning toolboxes.

Amongst more recent contributions to hyperparameter optimization, meta-learning has been successfully applied to kick-start hyperparameter optimizations (Feurer et al., 2015b). Solutions that performed well on similar problems are evaluated first, which allows the hyperparameter optimization procedure to converge faster and identify better solutions. Swersky et al. (2013) also extrapolated the performance of hyperparameter configurations across different datasets or varying training and validation split sizes through a multi-task formalism.

In order to speed up optimization, Domhan et al. (2015) and Swersky et al. (2014) consider the extrapolation of learning curves to prune underperforming models early on. Dataset subsampling has also been exploited to speed up hyperparameter optimization with great success. For example, Klein et al. (2016) model the function of validation error with regards to dataset size and let the Bayesian optimization procedure choose the subset size with an acquisition function that incurs a penalty for longer runtimes. Inspired by bandit algorithms, Li et al. (2017) iteratively increase the size of the training set and prune unpromising solutions with the successive halving strategy.

2.6 Hyperparameter Optimization Examples

To facilitate understanding, we will now present some examples of hyperparameter optimization applied to simple models. First, the hyperparameters to optimize will be defined, or the search space Γ, which usually takes the form of boundaries. Then the given search space will be optimized on a simple dataset. This will be done for two different models: an SVM with a radial basis function (RBF) kernel, and a random forest.

2.6.1 SVM RBF kernel width

As a first one-dimensional example, let us optimize τ the width of an RBF kernel[4] for a support vector machine (SVM):

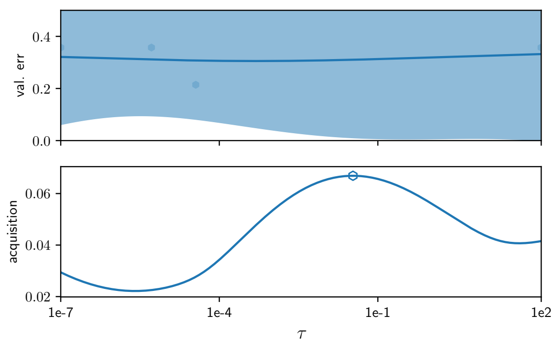
$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\tau \|\mathbf{x} - \mathbf{x}'\|^2\right). \tag{2.18}$$

In this particular case, the regularization parameter C of the SVM will be left constant, at $C = 1$, in order to be able to visualize the process. We will search for the optimal value of the τ parameter in the range $[10^{-5}, 10^2]$. Values of τ are explored in the logarithmic space, meaning that we are optimizing for u with $\tau = 10^u$, and the values of u are also rescaled between 0 and 1. In the case of a parameter such as the width of a kernel, this can have a significant impact on performance. In order to automatically determine the proper scaling for parameters, Snoek et al. (2014) defined some tunable warping functions for GP inputs which showed some desirable properties. We will not make use of such input warping in this thesis, although they can be of interest when one looks to optimize non-stationary functions.

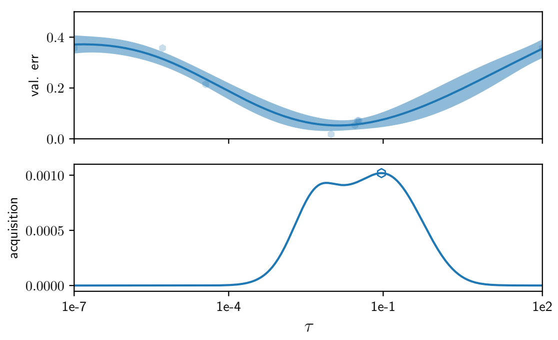
We then optimized the τ hyperparameter for an RBF SVM trained on the Ionosphere dataset (a small binary dataset with 351 samples and 33 features), taken from UCI (Frank and Asuncion, 2010). The objective function is the performance of the SVM on a hold-out validation dataset, such as defined in Equation 2.13. A test split is generated with one fifth of the dataset, and a validation split is generated from the remaining training data with one fifth of those training samples. Figure 2.3 shows snapshots of the optimization procedure at different number of iterations (or number of observations of the optimized function), one per subfigure. For each subfigure the top part represents the actual validation error optimized with regards to the τ parameter, whereas the bottom part presents the value of the acquisition function with regards to the hyperparameter optimized. The single point drawn on the acquisition curve represents the maximum of the acquisition function, which will next be evaluated by training a model with the given hyperparameters.

From Figure 2.3, we can infer multiple things. First, the optimal area for the parameter τ appears to be around 1e$-$2.5, and we can see that the acquisition function lead to many sampled models in the optimal zone, i.e. the area where hyperparameters lead to models with minimal empirical error rates. We can also see that the regression produced by the GP is rather smooth, which is appropriate for the problem at hand – this aspect

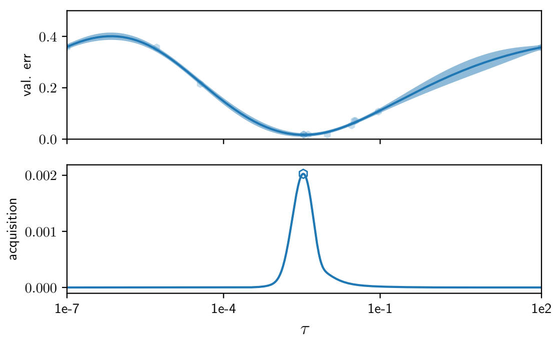
[4]Normally, the kernel width parameter is called γ, but to avoid confusion with the notation in Section 2.3 we use τ.



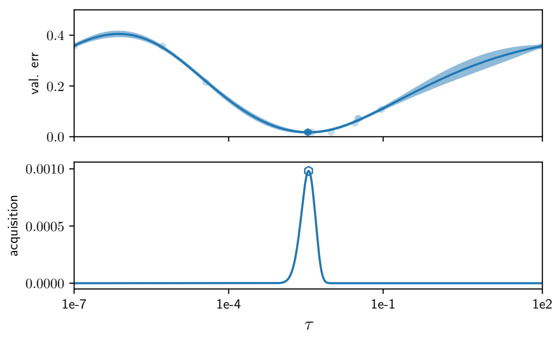
(a) 4 iterations



(b) 8 iterations



(c) 12 iterations



(d) 20 iterations

Figure 2.3: Example of hyperparameter optimization for an RBF SVM on the Ionosphere dataset. Subfigures represent snapshots of the optimization after given numbers of iterations.

is automatically handled by the slice sampling of GP hyperparameters, with the GP producing a higher log likelihood for hyperparameters inducing a smooth surface.

The above figure showed the process of hyperparameter optimization, however it does not tell us how well these hyperparameters and their corresponding models perform on a given testing set. Figure 2.4 shows the validation and testing error rates of the best found model as the optimization progresses. The plateaus in the figure represent areas where new models are trained, but which do not result in better validation performance. The model whose testing error is reported is the one with the minimum in validation error at each given iteration t:

$$\hat{h}_t = \underset{h \in \mathcal{H}_t}{\arg\min}\, L(h, \mathcal{D}_V). \tag{2.19}$$

Given the simplicity of the problem at hand, the performance rapidly attains a plateau and stays there until the end of the optimization, set at 50 iterations in this case. There also appears to be a slight amount of overfitting in the selection of hyperparameters,

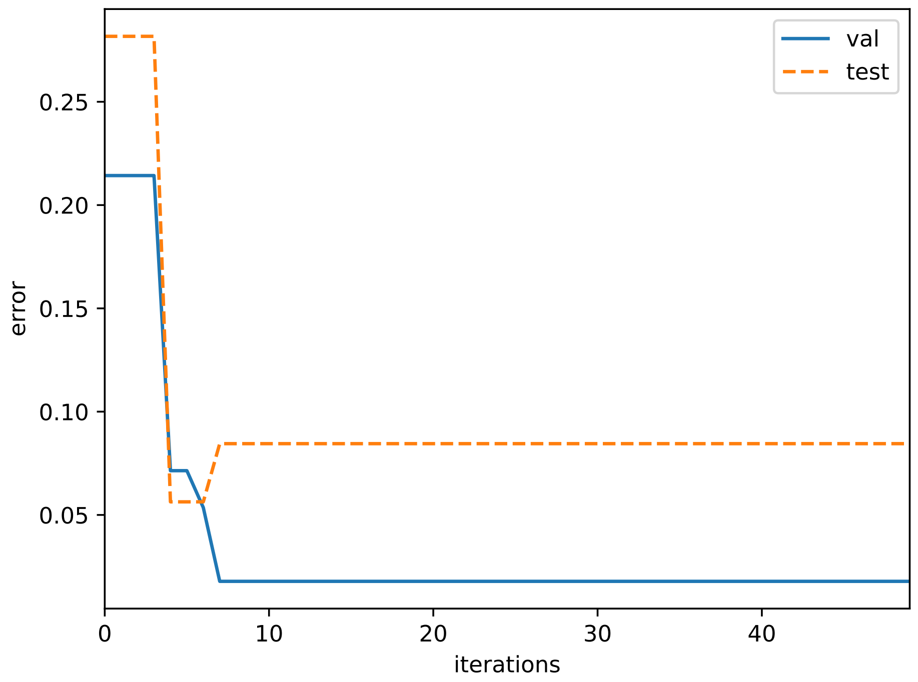


Figure 2.4: Validation and testing errors of the SVM with regards to the number of models trained (or iterations of the Bayesian optimization loop).

shown by the increase of testing error for the last iterations. The final hyperparameters selected would be $\tau = 0.06943$ for this particular dataset. The developers of the libsvm library recommend a default value of $\tau = 1/d$ as a starting point, which would equal 0.0303 for the dataset Ionosphere which has 33 dimensions. One interesting fact about Bayesian optimization is that, if one has knowledge about the function optimized, it can easily be injected in the procedure by, for instance, using a suggested configuration as a starting point. Feurer et al. (2015b) have done this through the use of meta-learning about dataset properties, using promising hyperparameter configurations on similar datasets as starting points. They showed great improvement in convergence speeds and were also able to find better final solutions.

2.6.2 Random forest hyperparameters

As an example of optimization for more than one hyperparameter, let us optimize the hyperparameters of a random forest, an ensemble of decision trees. In this case, the higher number of hyperparameters will prevent us from visualizing the posterior distribution over hyperparameters, but there are still insights to be drawn.

Table 2.1 shows the space which will be optimized for the random forest. The vector of values which represent the hyperparameters is the vector of normalized values for each hyperparameter, concatenated in $\gamma = \{$Num. estimators, Stopping criterion, Max. depth, Min. samples split, Min. samples leaf$\} \in \Gamma$. The maximum depth, minimum samples per leaf, and stopping criterion are the same for all trees of the forest. Categorical hyperparameters can be represented with integers or with a one-hot encoding, although in the case of a variable with only two possible choices, both have the same effect. For this search space, the evaluation of the validation error will be achieved with 5-fold cross-validation, effectively training 5 random forests to evaluate each hyperparameter tuple γ, in order to obtain more robust estimations of the generalization error.

Table 2.1: Random forest hyperparameters.

Hyperparameter	**Space**
1 - Num. estimators	Linearly in $\{2, \dots, 50\}$
2 - Stopping criterion	{gini, entropy}
3 - Max. depth	Linearly in $\{2, \dots, 150\}$
4 - Min. samples per split	Linearly in $\{2, \dots, 30\}$
5 - Min. samples per leaf	Linearly in $\{1, \dots, 30\}$

Figure 2.5 shows the result in validation and testing error rates for the optimization of a random forest's hyperparameters on the Adult dataset (taken from UCI). All datasets features were preprocessed and normalized to fall in the $[0, 1]$ range, feature-wise. In this case, the results produced are the average of five repetitions of the Bayesian optimization, with different train and validation splits. The shaded area in Figure 2.5 show the standard deviation over the five repetitions. We can see once again that the performance decreases rapidly and hits a plateau around iteration 40. There also seems to be some degree of overfitting in that the validation error rates keep decreasing after iteration 20, but the testing error rate either stays the same or increases. Overfitting will be discussed in greater depth in the next chapter.

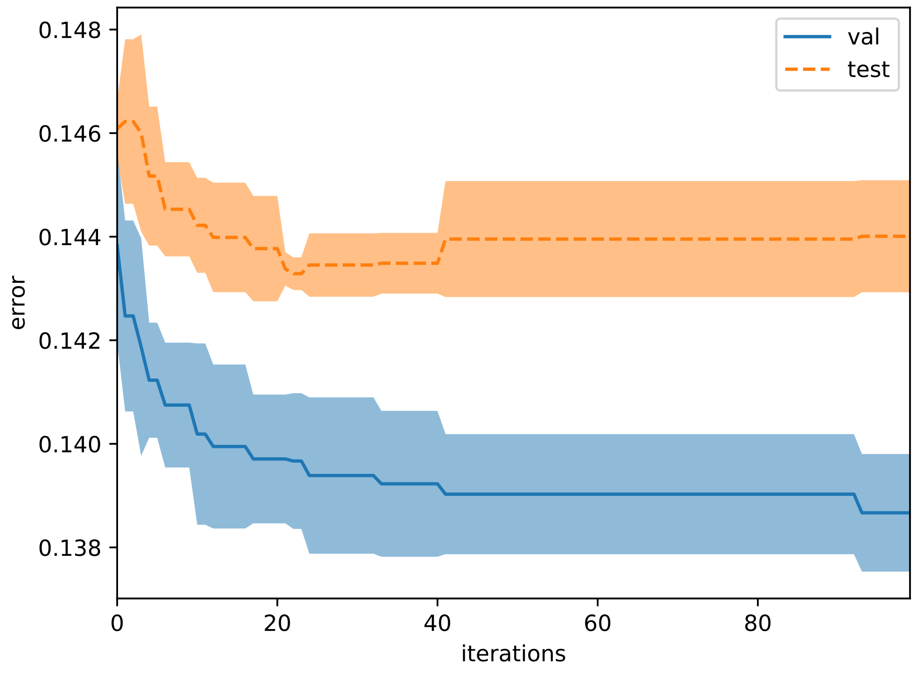


Figure 2.5: Validation and testing errors of the random forest with regards to the number of iterations of the Bayesian optimization procedure. The lines represent the average of five repetitions with different training and validation splits, and the shaded areas represent the standard deviation over those repetitions.

Chapter 3

Evaluation of Generalization Performance in Hyperparameter Optimization

One problem with hyperparameter optimization methods is that as the number of iterations spent optimizing hyperparameters increases, so does the knowledge of the validation data split(s). This results in an overestimation of the learner's generalization performance, generally called overfitting – also known as oversearching (Quinlan and Cameron-Jones, 1995). Another potential explanation for this overfitting is the possibility of finding a model performing strongly on the validation data purely by chance, which increases with the number of iterations. The phenomenon is not limited to hyperparameter optimization, and can apply anywhere an algorithm's performance is evaluated in rapid succession on the same validation data with regards to varying parameters (Dos Santos et al., 2009; Igel, 2013).

The default solution to the problem of overfitting in hyperparameter optimization is to use k-fold cross-validation. The repetitions provided by the k-folds of validation provide a reliable estimate of generalization performance, albeit at the cost of running the training algorithm k times. However, it is not clear whether this is the best choice in all cases, as it is merely considered a *good enough* choice. Wainer and Cawley (2017) have recently conducted an empirical study on the number of validation folds for cross-validation evaluation, and they concluded that two or three validation folds are sufficient for optimal hyperparameter selection. This is all related to the quantity of data available – if present in very large quantities, the problem of overfitting should be limited as the estimation of the generalization error will be more reliable.

Guyon et al. (2015) have created a competition for the automatic tuning and selection of machine learning algorithms (AutoML), and they highlight the overfitting of hyperparameters as one of the important problems that arise in the process. Computing the posterior probability of hyperparameters given the data and model space has been shown to be a good solution to reduce overfitting in the selection of hyperparameters (Cawley and Talbot, 2007; Rasmussen and Williams, 2006). However, this approach requires knowing the learning algorithm beforehand, effectively treating it as a white box, and cannot apply to all model families.

A similar problem has been studied and observed in the literature on evolutionary machine learning. Igel (2013) discusses the sources of overfitting in this context, namely: overfit on *training data*, overfit on *validation data*, and overfit on *final selection data*. He shows that overfit can occur on validation data even with a k-fold cross-validation procedure for a feature selection problem. Dos Santos et al. (2009) also used a separate selection data split for the final step of an ensemble construction technique, showing that it can result in better ensembles.

In this chapter, we consider the problem of overfitting the validation data in the context of hyperparameter optimization. We evaluate the empirical performance of various state of the art hyperparameter optimization methods and compare strategies to avoid overfitting. We propose to use two relatively unexplored ideas to improve hyperparameter optimization procedures, that is 1) to keep a second validation dataset used exclusively for the selection of the final hyperparameters and 2) to reshuffle the training and validation sets at each iteration of the hyperparameter optimization.

We evaluate some state of the art methods as well as the proposed approaches on a benchmark of 118 datasets (taken from Fernández-Delgado et al., 2014). Our experiments confirm that there is a degree of overfitting shown by Bayesian and evolutionary optimization methods. Generalization performance can be improved by reshuffling the training and validation splits over the course of a sequential optimization. However, using a second hold-out selection dataset does not seem to help reduce the overfitting in the choice of hyperparameters. Finally, we also consider a more Bayesian approach to the selection of the final model based on the posterior mean and variance of a surrogate model rather the minimum of the cross-validation error.

The chapter is organized as follows. We first present some concepts related to overfitting in Section 3.1, building on the framework of hyperparameter optimization introduced in Section 2.3. Then, we discuss in Section 3.2 about the problem of evaluating the performance of given hyperparameters and propose strategies to avoid overfitting. We present the experiments and corresponding results in Section 3.3 for what we call the arg min model selection. Finally, in Section 3.4.1, we introduce posterior mean selection and evaluate in on the same benchmark, showing improvements over generalization accuracy.

3.1 Overfitting

Hyperparameter optimization can be seen as a bi-level optimization problem, where the first optimization is the learning algorithm A, responsible of finding the model's parameters, and the second level is the optimization of the performance with regards to the hyperparameters $\gamma \in \Gamma$. Let us recall the two separate datasets used for training and for validation, $\mathcal{D}_T$ and $\mathcal{D}_V$, each assumed to be sampled i.i.d. from the underlying distribution $\mathcal{D}$. As was stated in Chapter 2, the ultimate goal of hyperparameter optimization is to find the hyperparameters which result in minimal loss on the true distribution $\mathcal{D}$:

$$\gamma_* = \arg\min_{\gamma} \left[\mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}} \ell(h_\gamma(\mathbf{x}), y) \right], \tag{3.1}$$

where $\ell(\cdot, y)$ is the loss function to minimize, and h_γ represents a model trained with hyperparameters γ: $h_\gamma = A(\mathcal{D}_T, \gamma)$. In this chapter and most of the thesis, we will consider the zero-one loss function:

$$\ell_{0-1}(\hat{y}, y) = \mathbb{I}(\hat{y} \neq y), \tag{3.2}$$

where $\mathbb{I}$ is the indicator function, which returns a 1 if the specified condition is true, and 0 otherwise.

Unfortunately the distribution $\mathcal{D}$ is unknown, therefore one must resort to an empirical evaluation of the generalization loss. The objective function can take the form of the empirical generalization error on $\mathcal{D}_V$:

$$L(h_\gamma, \mathcal{D}_V|\ell) = \frac{1}{|\mathcal{D}_V|} \sum_{(\mathbf{x},y)\in\mathcal{D}_V}^{|\mathcal{D}_V|} \ell(h_\gamma(\mathbf{x}), y). \tag{3.3}$$

The typical Bayesian optimization pipeline described in Section 2.3 can then be applied to find the best hyperparameters:

$$\hat{\gamma} = \arg\min_{\gamma} L(h_\gamma, \mathcal{D}_V|\ell). \tag{3.4}$$

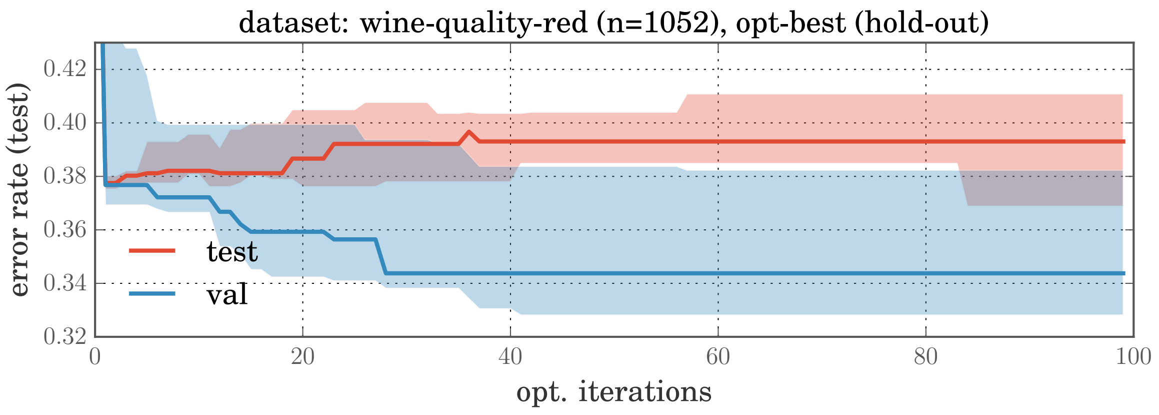


Figure 3.1: Example of overfitting on the hyperparameter level (error averaged over 10 repetitions). Gaussian Process used for hyperparameter optimization.

Similarly to the way a classifier can overfit training data, hyperparameter optimization can result in overfitting on the validation data, although hyperparameters most likely offer less flexibility or degrees of liberty for this overfitting. If the empirical estimation of the generalization error is not strong enough or if the hyperparameters offer a lot of flexibility, selected hyperparameters will result in models that learn overly complex decision boundaries achieving lower validation error (perhaps through sheer luck), but which poorly capture the true underlying distribution $\mathcal{D}$. Figure 3.1 shows an example of overfitting on hyperparameter selection for a RBF SVM learner with hold-out validation.

The overfitting of a classifier directly on the training set has largely been solved by the use of regularization. However, it introduces new hyperparameters in the form of regularization weights, which need to be selected through hyperparameter optimization. One could argue that the problem has just been pushed back to the next level, however regularization has many virtues apart from reducing overfitting (Zhang et al., 2017). Another classical approach to reduce overfitting of training data is early stopping, which makes use of the validation split to stop the optimization in an iterative procedure.

Cross-validation techniques are a simple way to obtain a better estimate of the generalization performance for some given hyperparameters. If we are able to better estimate the optimized value, the optimization overall should be improved. Cross-validation comes at the cost of repeating the training procedure multiple times, for instance, k times for a k-fold cross-validation and $|\mathcal{D}_T|$ times for a leave-one-out procedure. The k-fold method has been shown to be comparable to a fully Bayesian treatment for a Gaussian Process model (Rasmussen and Williams, 2006), and is often used by default for hyperparameter optimization.

The properties and bounds of hold-out or cross-validation estimations have been thoroughly covered in the literature. These can be broken down in bias and variance terms. The *bias* represents the quantity by which, in average, the estimator is off from the true generalization error of the model. The *variance* represents how the estimation will deviate on average from its mean and allows the practitioner to derive confidence bounds.

Both the hold-out estimation and the cross-validation estimation are computed on models trained with a subset of the whole data available for training and validation $\mathcal{D}_{TV}$. The fact that fewer data points are used for training can result in a lower accuracy for the model, depending on properties of the model and the size of the splits. This is one source of bias for the estimation of generalization accuracy, given that the final model is usually retrained on the whole training and validation data.

A source of variance is the number of samples used in computing the empirical estimation of the error. With hold-out validation, that number is the size of the validation split $n_V = |\mathcal{D}_V|$, with $\mathcal{D}_V = \mathcal{D}_{TV} \setminus \mathcal{D}_T$. The interest of k-fold cross-validation is that it allows to compute the generalization error on all samples available for training and validation through the cycling between folds. If the training algorithm produced the same model regardless of the training split used, the k-fold procedure would result in the best estimation of generalization error possible, although in practice this is rarely the case. One issue with the k-fold cross-validation is that there exists no unbiased estimator of its variance, meaning that one cannot obtain strong bounds for the generalization error computed with k-fold (Bengio and Grandvalet, 2004). The leave-one-out procedure provides more reliable estimates, however the cost of training $|\mathcal{D}_{TV}|$ models is prohibitively high.

A problem with Bayesian optimization is that no matter the statistical reliance or validity of these estimators, the fact that they are repeatedly called upon means that there will be some form of overfitting on the validation data. In practice, even reusing twice the same hold-out split would be enough to invalidate the statistical guarantees of its estimator, because the data are no longer independent and identically distributed. Dwork et al. (2015) attempt to solve this problem by using differential privacy inspired methods, answering only binary queries with regards to the validation error estimation, but this would not allow for the type of Bayesian optimization considered in this thesis. Tsamardinos et al. (2015) show that the Tibshirani-Tibshirani and nested cross-validation procedures show lower bias than the typical k-fold cross-validation.

A fully Bayesian treatment can limit overfitting on the selection of hyperparameters (Cawley and Talbot, 2007; Rasmussen and Williams, 2006). The marginal likelihood of hyperparameters is given by $p(\gamma|\mathbf{y}, X) \propto p(\mathbf{y}|X, \gamma)p(\gamma)$. This likelihood contains an integral over all possible models, which performs a type of regularization and helps to reduce overfitting. Hyperparameters can then be selected by maximizing this function, giving the maximum a posteriori estimate of hyperparameters. One problem with this approach is that it requires integrating over all possible models, which is not feasible for all families of models. It also must be carefully designed for specific models, which would render impossible the application of Bayesian optimization to any type of algorithm.

3.2 Strategies to Limit Overfitting

We will now propose some strategies to limit overfitting in the selection of hyperparameters. Note that we will also consider and compare two families of approaches for the optimization in itself: Bayesian optimization and evolutionary algorithms. More particularly, the evolutionary algorithm we will consider is the *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES; Hansen, 2005). CMA-ES is a strong tool for the solving of non-linear and non-convex black-box real-valued optimization problems. Hyperparameter optimization can be represented as such a problem (provided all optimized variables are real-valued), so we selected CMA-ES as a method for our experiments. It is said to be population-based, because at each iteration many points (in our case hyperparameters) are evaluated in parallel, which form a *population*. The method has been shown to be competitive with BO on a benchmark suite for real-valued black-box optimization (Hutter et al., 2013). CMA-ES is a second order approach estimating a covariance matrix that can be related to the inverse Hessian. This distribution is used to shift a population towards high accuracy zones in the hyperparameter space.

The population-based methods are distinct from the Bayesian optimization methods because they allow for different selection mechanisms of the best model, based on populations. These mechanisms will be introduced below.

3.2.1 Reshuffling

In order to reduce the amount of overfitting, we propose to resample the training and validation sets at each iteration. This is similar to a bootstrap estimate of the generalization error, renewed at each iteration. Given the available data for training and validation $\mathcal{D}_{TV}$, at each iteration t of the optimization procedure new training and validation sets will be sampled without replacement, and those datasets $\mathcal{D}_T^t$ and $\mathcal{D}_V^t$ will be used only for the current iteration. This will have the effect of slightly shifting the landscape of the objective function, not allowing an algorithm to advance too deep in a narrow valley, which would essentially be a case of overfitting.

3.2.2 Selection split

A second method to reduce the amount of overfitting is to let the hyperparameter optimization run its course with the regular hold-out or cross-validation evaluation, and to perform the final selection on a previously unobserved selection split $\mathcal{D}_S$. Generalization performance on the selection split is evaluated and cached as the optimization proceeds, but it is not observed by the algorithm before the end of the optimization. This way, selection of the final hyperparameter set is achieved through an unbiased estimation of the empirical error.

Note that the same amount of data remains available for the whole procedure, so some training and validation data must be sacrificed in order to build the selection dataset. Datasets are selected without overlap, giving $\mathcal{D}_{TV(S)} = \{\mathcal{D}_T \cup \mathcal{D}_V \cup \mathcal{D}_S\}$. Once the final hyperparameters are chosen, we retrain the model on the full data available for training, validation and selection (like we would for cross-validation).

For Bayesian optimization, the final model is chosen with a minimum on the generalization error of all models on the selection split. For population-based methods, several options are possible, described in the following.

Population-based selection split strategies

Since population-based methods have an internal state in the form of a population, more choices are available. The same three strategies used by Dos Santos et al. (2009) for ensemble generation will be exploited here for hyperparameter selection. Given the population at iteration t, $P_t = \{h_{\gamma_i}\}_{i=1}^{p}$, let us define two variables, the best solution in the current population according to the validation data split $\mathcal{D}_V$ and the selection data split $\mathcal{D}_S$:

$$b_V = b_V^t = \underset{h_\gamma \in P_t}{\arg\min}\, L(h_\gamma, \mathcal{D}_V) \tag{3.5}$$

$$b_S = b_S^t = \underset{h_\gamma \in P_t}{\arg\min}\, L(h_\gamma, \mathcal{D}_S). \tag{3.6}$$

Given these, the population-based selection strategies are the following:

- *Partial validation* selects the best solution from the current population with the selection dataset: b_S
- *Backwarding* only updates an archived solution a^{t-1} if the current best of population b_V is better according to the *selection split*:

$$a^t = \begin{cases} b_V & \text{if } L(b_V, \mathcal{D}_S) < L(a^{t-1}, \mathcal{D}_S) \\ a^{t-1} & \text{otherwise} \end{cases}. \tag{3.7}$$

- *Global validation* updates the archived solution a^{t-1}, but comparing only the best solution in the population according to the selection split:

$$a^t = \begin{cases} b_S & \text{if } L(b_S, \mathcal{D}_S) < L(a^{t-1}, \mathcal{D}_S) \\ a^{t-1} & \text{otherwise} \end{cases}. \tag{3.8}$$

3.3 Experiments

We compared the evaluation methods discussed in Section 3.2 on a benchmark of 118 datasets (taken from Fernández-Delgado et al., 2014), ranging from a size of 50 instances to 85,000 instances. The datasets come mainly from the UCI and statlog repositories (the full list of datasets is available in Appendix B). The median dataset size is 433 and the average 3102. The benchmark provides a good spread of dataset sizes, excluding very large scale problems, which would be better suited by a different learning algorithm and search space. Although there is a trend for larger datasets, the everyday practitioner is still likely to run into small and medium datasets.

Experiments are conducted for a single base learning algorithm, a support vector machine (libsvm implementation with scikit-learn wrappers) with a radial basis function kernel. The hyperparameters to optimize were the regularization parameter $C \in [10^{-5}, 10^5]$ and the kernel width $\tau \in [10^{-5}, 10^5]$, both optimized in the logarithmic space. Performance was evaluated with the following four different hyperparameter optimization methods, each with a budget of 100 function evaluations:

- Axis-aligned grid search (grid), a 10x10 grid in logarithmic space;
- Random search (rs), uniform sampling over logarithmic space;
- Bayesian optimization with Gaussian processes (gp), using the Spearmint package (Snoek et al., 2012) with the GPEIOptChooser and without parallelization (hence completely sequential);
- CMA-ES (cma) with a $\mu + \lambda$ evolution strategy ($\mu = 3$ and $\lambda = 6$, meaning 6 classifiers are trained per CMA-ES iteration), runs being made on the DEAP framework (Fortin et al., 2012).

The generalization error evaluation strategies and their abbreviated names are the following:

- *grid*, *rs*, *gp*, and *cma*: single validation, returning the best-of-run;
- *gp-r and cma-r*: reshuffling of the training and validation splits, also returning the best-of-run;
- *gp-sel*: perform the final model selection using the generalization error on a separate selection dataset $\mathcal{D}_S$.
- In the case of population-based methods, the use of a selection split takes the three forms discussed in Section 3.2:
  - *cma-pv*: partial validation;
  - *cma-bw*: backwarding;
  - *cma-gv*: global validation;
- *gp-sel-r*, *cma-pv-r*, *cma-bw-r*, and *cma-gv-r*: variants with both reshuffling and a separate selection set. These did not shuffle the selection set, since that would break its purpose of being unobserved until the end.

Note that the reshuffling and hold-out selection split variants were not evaluated for random search and grid search because they only look at the hyperparameter validation performance once for the final model selection, therefore they already have an unbiased estimate of the generalization error.

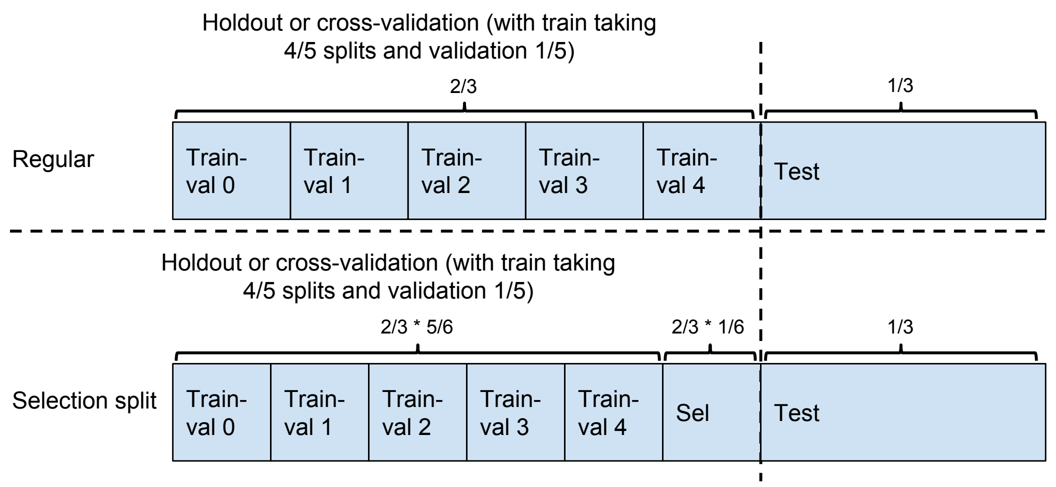


Figure 3.2: Data splits used for experiments. In a given repetition, the testing split is the same for all methods.

The way the data were split for regular optimization and optimization with a selection split is illustrated in Figure 3.2. Elements of importance to note are that the selection split is always the same, even for k-folds and reshuffling, in order to guarantee that it is never indirectly observed through folds. Generalization error estimates on the selection split are still computed by averaging over the 5 models trained on the 5 different training splits. When a testing split was specified in the dataset, this split was used instead of taking apart 1/3 of the dataset.

At the end of each optimization, the best hyperparameters are selected and a fresh model is retrained with those hyperparameters on the whole data available for training, validation, and selection (figures and tables present the testing error of this final model). For a deterministic model such as an SVM, there is no need to validate this final model – for a stochastic model, a validation split would be required. Training set sizes were capped at 5,000 in order to reduce the total optimization time - in those cases the remaining data is spread amongst the validation and selection split (if there is one).

All of the described experiments were repeated 20 times with a fresh resampling of all splits at the beginning, except for pre-specified test splits. These experiments result in a combination of 28 hyperparameter optimization methods and model selection strategies, across 118 datasets[1] for 20 repetitions each, giving a total of 66,080 hyperparameter optimization runs, and about 19.8 million classifiers trained. The total time required

[1]Out of the 122 datasets present in the study by Fernández-Delgado et al. (2014), three datasets were omitted because they were too small (balloons, lenses and trains), and molec-biol-protein-second was not used because of improper formatting.

for computing all experiments was roughly 9.8 days with 64 cores (on 8 computing nodes).

3.3.1 Results

Three metrics will be used to compare methods across datasets: generalization error, rank, and pairwise win rate. Generalization error is presented because it is an intuitive metric, but averaging generalization errors across datasets is tricky since different datasets have varying optimal error rates and noise levels. Rankings and pairwise win rates are generally considered more robust ways to compare classifiers or algorithms across multiple datasets (Demšar, 2006).

As in the work of Demšar (2006), statistical comparisons will be computed on the average generalization error across repetitions. Given a dataset d, a method i and its average generalization error on the testing split across q repetitions, e_i^d, the ranking is computed as follows:

$$rank_i^d = 1 + \sum_{j=0,\, j \neq i}^{m} comp(e_i^d, e_j^d), \tag{3.9}$$

where m is the number of methods, and *comp* is a comparison operator returning 0.5 if the compared values are equal (effectively assigning half a point to two tied methods):

$$comp(e_i, e_j) = \begin{cases} 0 & \text{if } e_i < e_j \\ 0.5 & \text{if } e_i == e_j \\ 1 & \text{otherwise} \end{cases}. \tag{3.10}$$

A global rank is then computed by averaging over datasets $Z = \{d^1, \ldots, d^k\}$. Unlike the generalization error, ranks (or win rates) can be averaged without any problem since all ranks are in the same range (1 up to the number of methods compared). A lower rank means the method performs better than the compared alternatives, with 1 being the best method, and rank m the worst method (for a ranking of m methods).

Pairwise win rate is the rate at which method i beat the other m methods, averaged over datasets, with draws also being assigned a 0.5 weight:

$$pw_i^d = \frac{1}{m} \sum_{j=1}^{m} comp(e_i^d, e_j^d). \tag{3.11}$$

Figure 3.3 presents the generalization error (upper) and rank (lower) of chosen methods with regards to the optimization iterations, averaged over all datasets. The left column

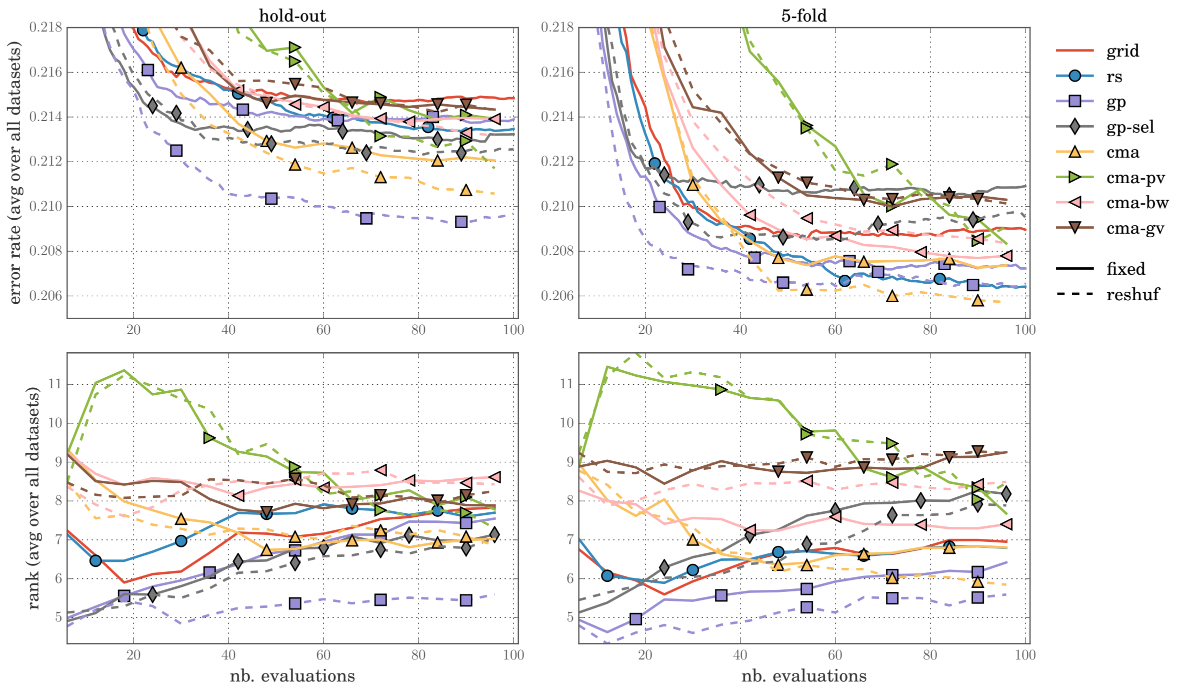


Figure 3.3: Performance for the proposed methods, left side is hold-out validation, right side is 5-fold validation. Top presents empirical error averaged on the test set, bottom presents average ranks (ranks computed by dataset). Dashed lines represent methods with reshuffled training and validation splits at each iteration, whereas full lines represent methods with fixed training and validation splits throughout a full optimization.

presents the evaluation of $f(\gamma)$ with a hold-out procedure, and the right column presents the k-fold cross-validation. One important point to notice is that methods were aligned with regards to the number of function evaluations. Since CMA-ES iterations contain 6 function evaluations, ranks with respect to BO methods are computed only for matching numbers of function evaluations (6, 12, ...). Table 3.1 shows the numerical values of median, interquartile ranges, pairwise win rates, and rank. Once again, hold-out is grouped on the left side of the table and cross-validation on the right side, since they are not equivalent in terms of computing resources used.

Tables 3.2 and 3.3 show pairwise counts of wins/losses computed per dataset and statistically significant differences according to Wilcoxon pairwise tests. For each dataset, the method at row i is compared with the method at column j. If the testing error of method i is lower than that of method j the count is incremented by 1. The count is decremented by 1 if the opposite condition is true, and it is unchanged on a draw. The value at the intersection of methods i and j is put in boldface and underlined

Table 3.1: Summary of performance metrics for all methods at the final iteration. Methods sorted by 5-fold rank.

	Hold-out			5-fold		
Method	Med (Q3-Q1)	Winrate	Rank	Med (Q3-Q1)	Winrate	Rank
cma-gv	17.22 (26.82)	47.09	7.91	16.73 (26.00)	37.62	9.23
cma-gv-r	16.96 (25.97)	44.67	8.25	16.35 (26.02)	37.86	9.20
cma-bw-r	16.72 (25.77)	43.34	8.43	16.07 (25.65)	42.95	8.49
cma-pv-r	16.68 (26.17)	51.36	7.31	15.53 (25.91)	43.07	8.47
gp-sel	17.05 (25.39)	52.63	7.13	16.40 (25.35)	45.04	8.19
gp-sel-r	17.22 (26.27)	53.27	7.04	15.95 (25.41)	47.12	7.90
cma-pv	17.05 (25.33)	47.97	7.78	15.56 (25.24)	48.64	7.69
cma-bw	17.25 (26.11)	42.04	8.61	15.57 (25.34)	50.64	7.41
grid	16.95 (26.47)	47.64	7.83	15.98 (25.22)	53.84	6.96
cma	16.90 (25.80)	53.54	7.00	16.10 (25.40)	54.90	6.81
rs	16.81 (25.97)	48.52	7.71	15.52 (24.73)	55.02	6.80
gp	17.43 (26.11)	49.85	7.52	15.73 (25.66)	57.87	6.40
cma-r	16.29 (26.53)	54.36	6.89	15.58 (24.77)	61.92	5.83
gp-r	15.82 (25.37)	63.71	5.58	15.53 (25.18)	63.50	5.61

if a Wilcoxon signed-rank using testing errors across all datasets was found significant with $\alpha = 0.05$. The Wilcoxon signed-rank test is a statistical test for comparing methods across multiple datasets, which is a non-parametric version of the Student's t-test that does not assume normal distributions and is less sensitive to outliers (Demšar, 2006). The Wilcoxon signed-rank test is very strong in this setting since the datasets where performance is equal will not influence the outcome of the test (they will have a difference of 0, and hence will be ranked similarly). This is desired because when a dataset or problem is easy, it does not say much about the methods we are trying to compare, therefore the statistical test should not infer anything from such a result. This is in opposition to a Student's t-test, which would be more conservative in this setting – not to mention the problem of commensurability across datasets which is avoided by ranking.

3.3.2 Analysis

Figure 3.3 and Table 3.1 indicate that the final generalization error averaged over all datasets has a relatively small spread across all methods. This is not very surprising

Table 3.2: Win/loss counts by dataset, with hold-out validation. Significant differences on a Wilcoxon signed-rank test between methods are boldfaced and underlined (row vs col, $p < 0.05$). Methods sorted by average rank over datasets.

Method	1	2	3	4	5	6	7	8	9	10	11	12	13	14
(1) cma-bw	0	-1	-3	-13	-5	-17	-20	-20	-**28**	-32	-**27**	-**24**	-**25**	-**48**
(2) cma-bw-r	1	0	-7	-16	-2	-14	-17	-17	-**20**	-24	-14	-**19**	-**28**	-**43**
(3) cma-gv-r	3	7	0	-10	-5	-5	-11	-21	-4	-23	-**20**	-**29**	-**18**	-**40**
(4) cma-gv	13	16	10	0	-4	4	-6	-4	-14	-13	-21	-**19**	-**18**	-**40**
(5) grid	5	2	5	4	0	-3	-6	2	-4	-5	-13	-16	-**19**	-**30**
(6) cma-pv	17	14	5	-4	3	0	-1	-5	-18	0	-14	-10	-7	-**47**
(7) rs	20	17	11	6	6	1	0	-3	-9	-15	-20	-20	-13	-**30**
(8) gp	20	17	21	4	-2	5	3	0	-7	-3	-11	-8	-8	-**36**
(9) cma-pv-r	**28**	**20**	4	14	4	18	9	7	0	-7	1	-8	-6	-**39**
(10) gp-sel	32	24	23	13	5	0	15	3	7	0	-4	-3	0	-**28**
(11) gp-sel-r	**27**	14	**20**	21	13	14	20	11	-1	4	0	0	-9	-**26**
(12) cma	**24**	**19**	**29**	**19**	16	10	20	8	8	3	0	0	-9	-**30**
(13) cma-r	**25**	**28**	**18**	**18**	**19**	7	13	8	6	0	9	9	0	-**16**
(14) gp-r	**48**	**43**	**40**	**40**	**30**	**47**	**30**	**36**	**39**	**28**	**26**	**30**	**16**	0

given that finding suboptimal yet working hyperparameters is relatively easy for the chosen learner (RBF SVM). However, one can see a clear benefit to using a reshuffling for most methods in both settings. The benefits of using a separate selection split are not as obvious - in fact most methods seem to be performing better without the selection split. This means that sacrificing data for the hold-out selection dataset hurts the optimization process, resulting in a suboptimal hyperparameter choice at the end.

The generalization error is brought close to what it is with a 5-fold cross-validation procedure using a fraction of the computing budget (compare gp and gp-r lines), which in itself is a very interesting result. The improvement provided by reshuffling is not as marked for 5-fold cross-validation, which is expected since k-fold provides a more reliable statistical estimate of the generalization error.

Best overall method

One clear winner from the use of reshuffling is Bayesian optimization with Gaussian processes (for instance, compare the results of *gp* with *gp-r*). The fact that there is a gain from continually reshuffling the training and validation splits shows that there is overfitting during the optimization procedure. Bayesian optimization with GPs and reshuffling (*gp-r*) performs consistently better than all other approaches, with average

Table 3.3: Win/loss counts by dataset, with 5-fold validation. Significant differences on a Wilcoxon signed-rank test between methods are boldfaced and underlined (row vs col, $p < 0.05$). Methods sorted by average rank over datasets.

Method	1	2	3	4	5	6	7	8	9	10	11	12	13	14
(1) cma-gv	0	6	-12	-14	-26	-**39**	-**18**	-**32**	-**38**	-**35**	-**46**	-**42**	-**56**	-**57**
(2) cma-gv-r	-6	0	-10	-9	-17	-**37**	-21	-**36**	-**31**	-**37**	-**38**	-**47**	-**49**	-**63**
(3) cma-bw-r	12	10	0	1	1	-9	-10	-20	-**18**	-**37**	-**28**	-**36**	-**55**	-**44**
(4) cma-pv-r	14	9	-1	0	-2	-6	-12	-20	-23	-**34**	-**28**	-**35**	-**44**	-**47**
(5) gp-sel	26	17	-1	2	0	5	-11	-15	-19	-**21**	-**18**	-**48**	-**38**	-**43**
(6) gp-sel-r	**39**	**37**	9	6	-5	0	-5	-8	-16	-10	-**25**	-**34**	-**36**	-**47**
(7) cma-pv	**18**	21	10	12	11	5	0	-11	-12	-14	-**15**	-**17**	-**27**	-**26**
(8) cma-bw	**32**	**36**	20	20	15	8	11	0	-8	-18	-**9**	-17	-**35**	-**34**
(9) grid	**38**	**31**	**18**	23	19	16	12	8	0	-1	-5	-10	-2	-**20**
(10) cma	**35**	**37**	**37**	**34**	**21**	10	14	18	1	0	3	-4	-**22**	-**22**
(11) rs	**46**	**38**	**28**	**28**	**18**	**25**	**15**	**9**	5	-3	0	-6	-19	-18
(12) gp	**42**	**47**	**36**	**35**	**48**	**34**	**17**	17	10	4	6	0	-13	-**23**
(13) cma-r	**56**	**49**	**55**	**44**	**38**	**36**	**27**	**35**	2	**22**	19	13	0	-2
(14) gp-r	**57**	**63**	**44**	**47**	**43**	**47**	**26**	**34**	**20**	**22**	18	**23**	2	0

ranking of 5.58 and 5.61 respectively for hold-out and 5-fold $f(\gamma)$ evaluation. According to the Wilcoxon signed-rank tests across datasets (see Tables 3.2 and 3.3), the approach is also significantly better than all other methods for hold-out, and significantly better than 11 methods out of 13 for 5-fold validation.

Population-based methods

It is interesting to note the performance of CMA-ES on this benchmark, since it has not been highlighted often in the literature on hyperparameter optimization. First, there was no significant difference between CMA-ES and GP for both hold-out and 5-fold $f(\gamma)$ evaluation methodologies. Hence, covariance matrix adaptation seems to be a good candidate solution for hyperparameter optimization, and perhaps other variants of CMA-ES should be considered in future works.

Second, CMA-ES also benefited from a reshuffling of training and validation splits during the optimization, with *cma-r* performing second best of all methods with ranks of 6.89 and 5.83 for hold-out and 5-fold cross-validation respectively. In all cases, the use of a separate selection split decreased the performance of CMA-ES methods in comparison with not using it. The effect seemed to be worse when using a 5-fold cross-validation strategy. This leads us to discourage the use of a separate selection split in the current setting, meaning small to medium scale datasets.

Reshuffling, overfitting and problem difficulty

In both the case of GP and the case of CMA-ES, reshuffling provided an increase in generalization performance, meaning there was overfitting on the validation split (or splits) in the hold-out and 5-fold settings. This overfitting took place on a relatively easy problem, namely choosing the parameters of an RBF SVM (Gaussian kernel width σ and regularization constant C). Based upon this, it could be argued that overfitting will be more pronounced on harder problems with more hyperparameters. The hypothesis space resulting from these hyperparameter configurations should be more complex, therefore allowing more overfitting on the validation set. However, more experimentation will be required to assert this hypothesis.

Comparison with regards to the dataset size

The benchmark used could be criticized as having a bias towards methods performing better on small datasets since it contains a lot of small-sized dataset. In order to clear up this question, the datasets were sorted into size categories (roughly 30 datasets per category) and new ranks were computed in each of them. Figure 3.4 shows the obtained ranks for categories with the dataset size boundaries $[1 - 133 - 455 - 1403 - 1e5]$ (computed analytically), the first category containing datasets of size $[1 - 133]$, the second $[133-455]$, and so on. The term dataset size refers to the number of data samples used for training and validation, without considering the number of data points in the testing set since they are not used during the optimization and should have no impact on a method's generalization performance. In order to reduce clutter, the methods with a selection split are not drawn on the figure - only grid, random search, CMA-ES and GPs along with their reshuffled variants are presented.

It can be seen from Figure 3.4 that GPs with reshuffling actually tend to improve as the dataset size increases (dashed lines), while the opposite is seen for CMA-ES. This provides reason to believe that overfitting is not limited to smaller datasets.

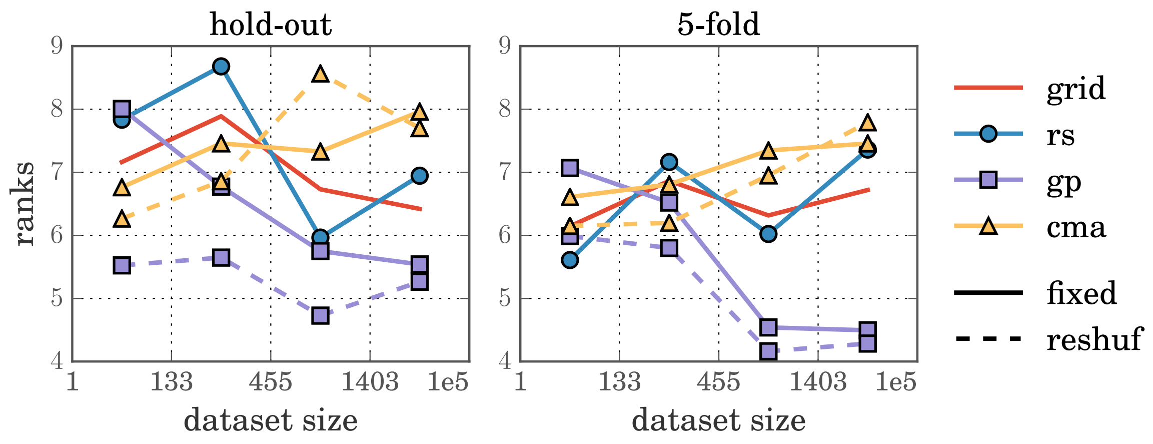


Figure 3.4: Ranks computed with regards to the dataset size (number of samples for training and validation). The same legend as Figure 3.3 is used.

3.4 Selection with posterior mean

3.4.1 Hyperparameter selection with the posterior mean

For Bayesian optimization methods, we could also select the final hyperparameters by minimizing the Gaussian process's posterior mean rather than simply using the arg min over all trained models. The Gaussian process and its hyperparameters include a model of the noise function in the covariance function:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_a^2 \exp\left(\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^T \mathrm{diag}(\boldsymbol{\ell})^{-2}(\mathbf{x}_i - \mathbf{x}_j)\right) + \nu\delta_{ij}, \tag{3.12}$$

where the noise amplitude is given by the parameter ν and is tunable through slice sampling or maximization of the likelihood (ideally with proper priors). In other words, the selection through an arg min operation is likely to pick a noisy sample, an artefact which is not going to happen if we instead use the posterior mean of the Gaussian process. We can visualize this idea with a simple one-dimensional example (and it stands to reason that the problem should only be exacerbated in a higher dimensional space). Figure 3.5 shows an example of a one-dimensional function with the noisy observations scattered in green. The three '$\times$' represent the three final selections that would be performed with either an arg min selection (in green), a posterior mean selection (in orange) or with knowledge of the true function (in grey). We can see that both selections are off the true function minimum by some amount, but also that the posterior mean is more conservative, smoothing out the noise of individual observations.

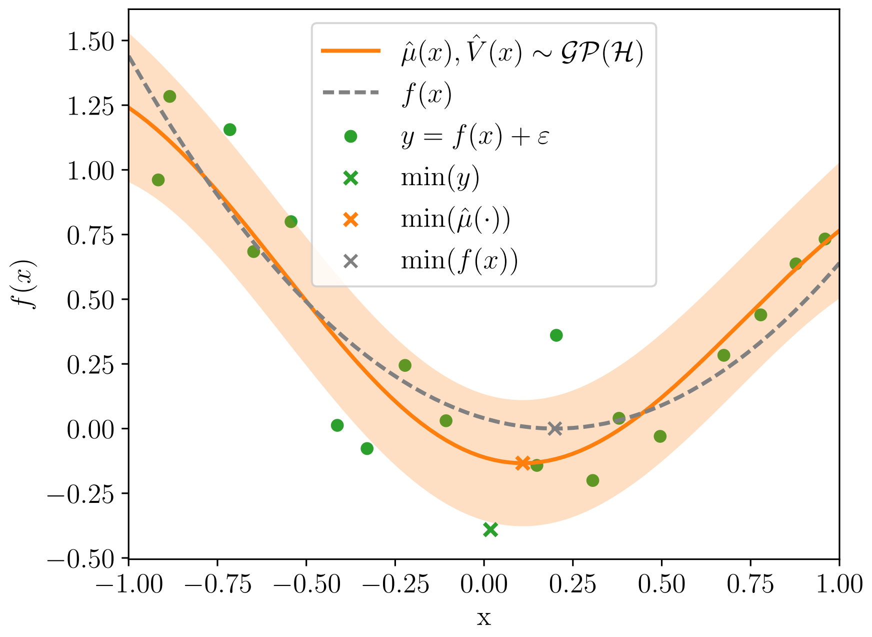


Figure 3.5: Comparison of optimum selection through posterior mean and arg min, with regards to the true function minimum.

Furthermore, it is our belief that posterior mean selection should synergize well with the reshuffling of the splits at each iteration, given that these produce naturally noisier observations, because some validation splits are easier or harder than others. The posterior mean of the GP should not only produce a better fit, closer to the true performance of hyperparameters than a GP trained on static splits, but it should also learn a better value for the noise parameter ν. In the case of GP without resampling of the training and validation splits, the sources of noise are the change in model resulting from different hyperparameters and the noise in retraining the same model with the same hyperparameters – for an SVM the training is deterministic and the resulting model does not vary from one repetition to the other. With reshuffling, there is another noise variable added, which is the noise of sampling the training and validation splits. This source of noise is most important for generalization accuracy estimation, because it helps us measure how much the accuracy of a model varies according to the validation split sample.

This method implies training a new model once those hyperparameters are identified, a cost which is reasonable if it leads to a better performing final model. In most hyperparameter optimization setups, this cost is already incurred when retraining the model on all data available for training and validation before deploying.

Table 3.4: Win/loss counts by dataset for selection with posterior mean (-pm). Significant differences on a Wilcoxon signed-rank test between methods are boldfaced and underlined (row vs col, $p < 0.05$).

(a) Hold-out

Method	1	2	3	4	5	6
(1) grid	0	-2	-4	-3	-**15**	-**54**
(2) rs	2	0	9	-8	-**17**	-**31**
(3) gp	4	-9	0	4	-**31**	-**43**
(4) gp-pm	3	8	-4	0	-**34**	-**54**
(5) gp-r	**15**	**17**	**31**	**34**	0	-20
(6) gp-pm-r	**54**	**31**	**43**	**54**	20	0

(b) 5-fold cross-validation

Method	1	2	3	4	5	6
(1) grid	0	12	-8	-11	-11	-**27**
(2) rs	-12	0	-4	1	-21	-**33**
(3) gp	8	4	0	-8	-**25**	-**28**
(4) gp-pm	11	-1	8	0	-26	-**25**
(5) gp-r	11	21	**25**	26	0	-**15**
(6) gp-pm-r	**27**	**33**	**28**	**25**	**15**	0

3.4.2 Results

The proposed posterior mean selection was executed on the exact same benchmark as in the previous section. In order to reduce the amount of clutter, we will not show population-based methods or methods making use of an external selection split. We present the axis-aligned grid (grid), random search (rs), Gaussian process (gp), along with reshuffling (gp-r) for comparison. We perform posterior mean selection on the two GP-based approaches, giving gp-pm and gp-r-pm for reshuffling. This only requires training a single new model, otherwise reusing the integral observations of the gp and gp-r methods.

The pairwise win counts are shown in Table 3.4, for both hold-out and 5-fold cross-validation on 10 repetitions of hyperparameter optimization for the same datasets and the same hyperparameter space (SVM RBF) as presented in the previous section.

We can see in Table 3.4 that using the posterior mean to choose the final hyperparameters appears to benefit the generalization accuracy in the presence of reshuffling. The normal GP without reshuffling does not appear to strongly benefit from the posterior mean selection – it performs worse on 4 datasets on the hold-out case, better on 8 datasets in the 5-fold case, with no significant difference according to the Wilcoxon test in both cases. However, gp-pm-r performs better than gp-r on 20 datasets in the hold-out case, and better than gp-r on 15 datasets in the 5-fold case. According to the Wilcoxon signed-rank test, gp-pm-r is significantly better than gp-r with 5-fold cross-validation, but not with hold-out validation.

Lastly, Table 3.5 shows a summary of the performance metrics for the same methods. We can see from this table that the average rank of gp-pm-r is the lowest amongst

Table 3.5: Generalization error, pairwise win rate and rank for all methods at the final iteration.

	Hold-out			5-fold		
Method	Med (Q3-Q1)	Winrate	Rank	Med (Q3-Q1)	Winrate	Rank
grid	16.95 (26.42)	44.49	3.83	16.39 (25.09)	46.82	3.69
rs	17.08 (26.45)	46.82	3.69	15.78 (25.42)	45.13	3.79
gp	17.16 (24.98)	44.70	3.82	15.76 (25.13)	46.54	3.71
gp-pm	17.74 (24.28)	44.28	3.84	15.86 (25.86)	47.67	3.64
gp-r	15.95 (25.97)	55.44	3.17	15.94 (25.96)	54.80	3.21
gp-pm-r	16.04 (25.24)	64.27	2.64	15.40 (25.82)	59.04	2.96

all methods for the hold-out validation. The same thing happens in the 5-fold cross-validation case, with an average rank of 2.96. Finally, one point of particular interest is the fact that the gp-r and gp-pm-r methods with single validation fold achieve median error rates of 15.95 and 16.04 respectively, which are pretty close to the median error rate of the classical gp approach using 5-fold cross-validation (15.76).

In fact, if we directly compare the average error rate of gp-pm-r-1f (for GP with reshuffling and posterior mean selection, using hold-out validation) with gp-5f (for GP with 5-fold cross-validation) using a Wilcoxon signed-rank test across the 118 datasets tested, no significant difference is found ($p = 0.573$). Put differently, our method with hold-out validation is *not significantly different from doing a 5-fold cross-validation procedure*, which is significantly more costly since 5 models are trained for each hyperparameter evaluation instead of one.

3.5 Conclusion

In this chapter, we showcased the performance of several approaches for the hyperparameter optimization of an SVM learner, over an extensive benchmark of datasets. The research questions targeted by this work were:

1. Does significant overfitting occur with regards to the cross-validation data during the Bayesian optimization of hyperparameters?

2. Can we alleviate this overfitting by dynamic resampling of training and validation

splits?

3. Can the use of a selection split, only observed at the end of the optimization, further reduce the overfitting?

This work shows that overfitting can and does occur during hyperparameter optimization, especially with a single hold-out validation set, answering (1) qualitatively. (2) and (3) are both answered through the benchmark evaluation. It can be seen that reshuffling the data between training and validation sets during a hyperparameter optimization procedure can be beneficial to the generalization performance, answering question (2) positively. On the other side, the selection split is not shown to provide a significant improvement over the selection of the final model, answering question (3) negatively.

A contribution of this work is providing evidence that the reshuffling procedure improves the performance of a sequential model-based optimization with GPs as well as that of a covariance matrix adaptation evolution strategy. We also show that CMA-ES is a good solution for continuous hyperparameter optimization, while often ignored in the literature. Finally, we show the interest of reusing the surrogate model and its posterior mean estimate to select the final model, over a simple arg min selection, which is overfitting in the noise of sampling data folds.

Chapter 4

Bayesian Hyperparameter Optimization for Ensemble Learning

The previous chapters and most of the literature on Bayesian optimization are focused on the optimization of the performance for a single model. It is generally accepted that ensembles can perform better than single classifiers, one of many striking examples being the winning entry of the Netflix challenge (Bell and Koren, 2007). In fact, machine learning competitions such as Kaggle competitions are often won by ensemble methods (Sun and Pfahringer, 2011). One of the reasons for the improved performance of ensembles with regards to single models is the reduction of the individual models' variance, which helps in generalization (Breiman, 2001).

Hyperparameter optimization generates a lot of trained models, and rather than directly discarding them it might be interesting to generate an ensemble from their predictions. This concept has been applied by Feurer et al. (2015a), who generate so-called *post-hoc* ensembles, meaning ensembles of classifiers that are formed without altering the hyperparameter optimization procedure. These classifier ensembles are constructed using forward greedy selection and combined with majority voting, a technique which has been shown to be robust to overfitting and better than more elaborate weighting or stacking methods (Caruana et al., 2004).

These previous lines of work make for a compelling argument to directly apply Bayesian optimization of hyperparameters for ensemble learning. We avoid optimizing a full vector of free hyperparameters for all models of the ensemble, a problem which is likely hard and which duplicates a lot of computation. Instead, we pose a performance model with regards to hyperparameters for the addition of a single new classifier to an already established ensemble. This is achieved by reusing models previously assessed during the optimization, evaluating performance changes induced by adding them one at a time to the ensemble. This allows us to compute observations of the true ensemble loss with regards to the hyperparameter values. These observations are used to condition a Bayesian optimization prior, creating mean and variance estimates over the hyperparameter space which will be used to optimize the configuration of a new classifier to add to the ensemble. Finally, we consider different possibilities to maintain and build the ensemble as the optimization progresses, and settle on a round-robin optimization of the classifiers in the ensemble. This ensemble optimization procedure comes at a small additional cost compared with a regular Bayesian optimization of hyperparameter yet yields better generalization accuracy for the same number of trained models.

We evaluate our proposed approach on a benchmark of medium datasets for two different hyperparameter spaces, one consisting solely of SVM algorithms with different kernel types, and one larger space with various families of learning algorithms. In both search spaces, our approach is shown to outperform regular Bayesian optimization as well as post-hoc ensemble generation from pools of classifiers obtained by classical Bayesian optimization of hyperparameters. We also evaluate our approach on a search space of convolutional neural networks trained on the CIFAR-10 dataset. The proposed approach is also able to provide better performance in this case.

The work presented in this chapter can be considered as the extension of (Lévesque et al., 2016), with some significant revisions, improvements and additional experiments.

In the next section we highlight some related work on hyperparameter optimization and ensemble generation. Section 4.2 presents the main contributions of this chapter, which can be summarized as a methodology for Bayesian optimization of ensembles through hyperparameter tuning. Section 4.3 presents some loss functions that can be used to evaluate the performance of each ensemble, other than the straight zero-one loss. Finally, Section 4.4 presents the experiments and an analysis of the results.

4.1 Hyperparameter Optimization and Ensembles

The idea of generating ensembles with hyperparameter optimization has already received some attention. Bergstra and Cox (2013) applied hyperparameter optimization in a multi-stage approach akin to boosting in order to generate better representations of images. Lacoste et al. (2014b) proposed the sequential model-based ensemble optimization (SMBEO) method to optimize ensembles by bootstrapping the validation datasets to simulate multiple independent hyperparameter optimization processes and combined the results with the agnostic Bayesian combination method.

The process of hyperparameter optimization generates many trained models, and is usually concluded by selecting a model according to the hold-out (or cross-validation) generalization error, $\hat{\gamma} = \arg\min_{\gamma} L(h_{\gamma}, \mathcal{D}_V)$. This single model selection at the end of the optimization is the equivalent of a point estimate, and it can result in overfitting. One strategy to limit this overfitting in the selection of a final model is to select multiple models instead of one, reducing the risk of overfitting and thus increasing the generalization performance.

A simple strategy to build an ensemble from a hyperparameter optimization is to keep the trained models as they are generated for evaluation instead of discarding them (Feurer et al., 2015a). This effectively generates a *pool* of classifiers to combine at the end of the optimization, a process which is called **post-hoc ensemble generation**. Forward greedy selection has been shown to perform well in the context of pruning a pool of classifiers (Caruana et al., 2004). At each iteration, given a pool of trained classifiers H to select from, a new classifier is added to the ensemble, selected according to the minimum ensemble generalization error. At the first iteration, the classifier added is simply the single best classifier. At step t, given the ensemble $E = \{h_{e_1}, h_{e_2}, \ldots, h_{e_{t-1}}\}$, the next classifier is chosen to minimize the empirical error on the validation dataset when added to E:

$$h_t = \underset{h \in H}{\arg\min}\, L(E \cup \{h\}, \mathcal{D}_V) \tag{4.1}$$

$$L(E \cup \{h\}, \mathcal{D}_V) = \frac{1}{|\mathcal{D}_V|} \sum_{(\mathbf{x},y) \in \mathcal{D}_V} \ell_{0-1}\Big(g(\mathbf{x}, E \cup \{h\}), y\Big), \tag{4.2}$$

where $g(\mathbf{x}_i, E)$ is a function combining the predictions of the classifiers in E on sample $\mathbf{x}_i$. In this case, the combination rule is majority voting, as it is less prone to overfitting (Caruana et al., 2004; Feurer et al., 2015a). Other possible combination rules include weighted voting, stacking (Kuncheva, 2004) and agnostic Bayesian combination (Lacoste et al., 2014a), to name only a few.

Figure 4.1 shows an example of a post-hoc ensemble generation from the optimization of the hyperparameters of an SVM with Gaussian kernel, namely the kernel width and regularization constant. Validation error is estimated with 5-fold cross-validation and 50 models are trained in total. As suggested by Caruana et al. (2004), we perform

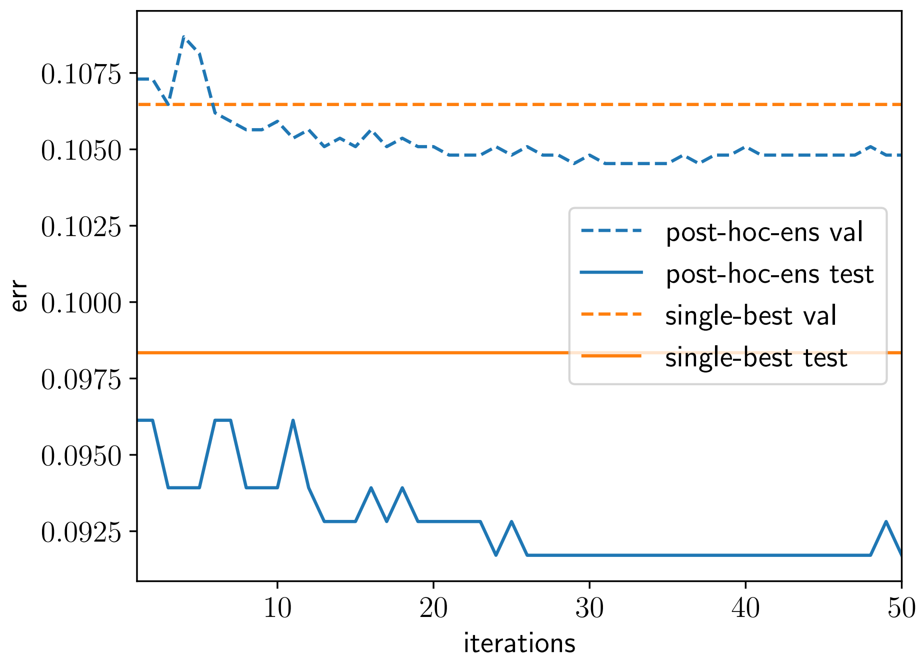


Figure 4.1: Example of post-hoc ensemble optimization. Dashed lines represent the validation error, and full lines represent testing error.

bagging over the complete pool of models and kick-start the ensemble selection with the 5 best models to reduce overfitting. The dataset used is the bank dataset, available from the UCI repository. We can see that in this example the ensemble is indeed able to improve on the performance of the single-best model. The fact that models are selected with replacement and are selected from a subsample of all models allow for the routine to be run for an indefinite number of iterations. Without bagging over models, this selection procedure is very likely to result in overfitting on the validation split, so we either limit the number of iterations or perform bagging over models.

It could be worthwhile to note that although we solve the ensemble selection problem with a forward greedy selection, it is not a submodular problem and hence we are not guaranteed to converge to an optimal solution (Krause and Golovin, 2014). This is tolerable because the optimal combination is most likely overfitting on the validation data.

4.2 Ensemble Optimization

In this chapter, we aim at directly optimizing an ensemble of classifiers through Bayesian hyperparameter optimization. The strategies discussed in the previous section mostly aimed at reusing the product of a completed hyperparameter optimization after the fact. Instead of only constructing an ensemble after optimizing hyperparameters for a single classifier, the hyperparameter space will be searched to maximize the performance of an ensemble. One goal of our approach is to make an online selection of hyperparameters that could be more interesting for an ensemble, but which do not necessarily optimize the objective function for classical hyperparameter optimization (Equation 2.13) on their own.

One way to go about this task would be to optimize a joint space of hyperparameters, e.g. with an ensemble of size m we would have $f(E) = f(h_{\gamma_1}, h_{\gamma_2}, \ldots, h_{\gamma_m})$ with each hyperparameter tuple $\gamma_i \in \Gamma$. Assuming d hyperparameters for the underlying classifier, this would result in a search space of dimensionality md, a very hard and inefficient optimization problem, which would needlessly duplicate the training of many models. Some information should be transferable between positions of classifiers in the ensemble, or, in other words, the performance of given hyperparameter tuples should not be dependent on their position in the ensemble.

In lieu of this costly optimization, we propose to keep a unified pool of trained classifiers with their hyperparameters, and evaluate the impact of adding classifiers from this pool to a fixed ensemble. Given a current ensemble of fixed size $E_t = \{h_{e_1}, h_{e_2}, \ldots, h_{e_m}\}$, the objective function will be posed as the loss of this ensemble $E = E_t$ if we augment it with a classifier trained with hyperparameters γ:

$$f(\gamma|E) = L\left(E \cup \{A(\gamma, \mathcal{D}_T)\}, \mathcal{D}_V\right) + \varepsilon, \tag{4.3}$$

where A is the actual learning algorithm returning a trained model using hyperparameters γ, $L(E, \mathcal{D}_V)$ is the empirical loss on a validation set (as defined in Equation 4.2), ε is some Gaussian noise on the observations, and the ensemble members are combined through a majority vote. The combination rule is one of the parameters of this model, and the choice of the majority vote will be explained below in greater detail. Given the pool of classifiers trained so far during the optimization $H_t = \{h_1, h_2, \ldots, h_t\}$ and a given ensemble E, observations of Equation 4.3 can be computed by adding each classifier in H to E. These observations can then be used to condition a probabilistic prior on the objective function, which will be used to perform Bayesian optimization. At this point, a lot of questions are still unanswered, but the main element of our method is established: an objective function that allows for the optimization of an ensemble in the hyperparameter space, combined with a method to generate observations for this objective function. The observations are obtained at a small extra cost compared with

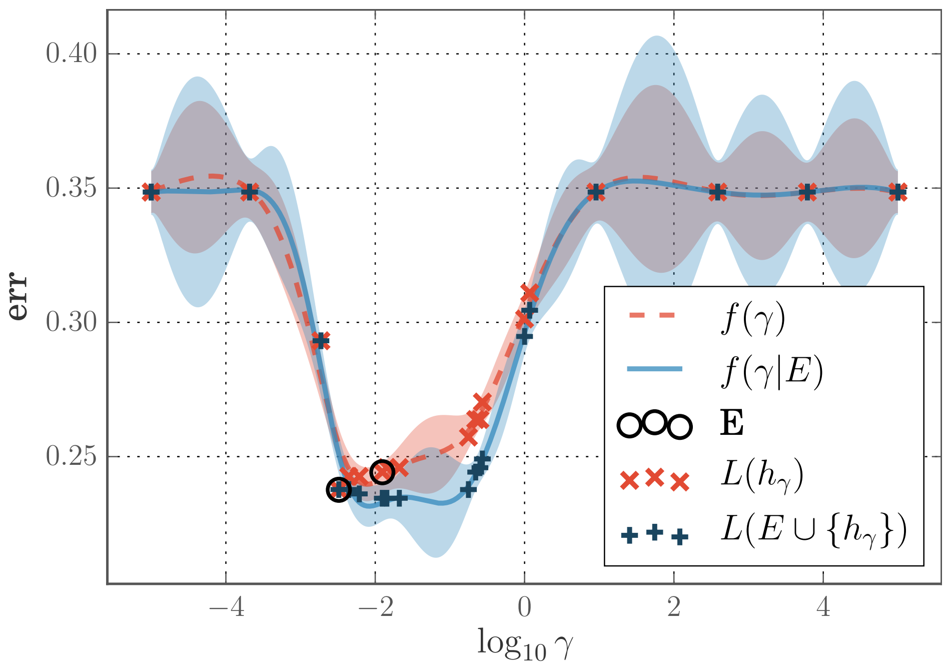


Figure 4.2: Example of an ensemble optimization. The red '$\times$' marks represent trained models and their standalone generalization error, and black circles represent two models selected for the current ensemble E. Blue '+' marks represent the performance of an ensemble when we add the trained model with corresponding hyperparameters γ.

a classical hyperparameter optimization (meaning it does not require the training of additional classifiers). More precisely, the extra cost is the computation of the $L(\cdot)$ term in Equation 4.3 for each classifier in the pool H_t.

Given a zero-one loss function (see Equation 3.2) and an empty ensemble $E = \varnothing$, Equation 4.3 falls back to classical hyperparameter optimization, and the objective function will be minimized by the best hyperparameters for a single model γ^*.

The power of the suggested framework is illustrated with an example shown in Figure 4.2. This figure presents one iteration of ensemble optimization given 20 trained SVMs in a one-dimensional hyperparameter space, where the single hyperparameter is the width of the RBF kernel ($\sigma \in [10^{-5}, 10^5]$). The dataset used is the Pima Indian Diabetes dataset available from UCI (Frank and Asuncion, 2010), with separate training and validation splits. The current ensemble E consists of two models selected by forward greedy selection shown by black circles. Ensemble evaluation and member selection strategies will be discussed further; for now let us assume a fixed ensemble. The red '$\times$' represent the generalization error of single models and the red curve represents a Gaussian process prior conditioned on those observations, in other words, a model of $f(\gamma)$. The blue '+' represent the generalization error of the ensemble E when the corresponding classifiers are added to it, and the blue curve is again a Gaussian process prior conditioned on the ensemble observations, or, more generally speaking, a model of $f(\gamma|E)$. For both Gaussian processes, the variance estimates are represented by shaded areas. Following a Bayesian optimization methodology, the next step would be to apply an acquisition function with the ensemble mean and variance estimates to select the next hyperparameters to evaluate.

Figure 4.2 shows that the objective function of an ensemble and a single classifier can be different. It can also be observed in this case that the generalization error of the ensemble is lower than that of a single model, hence the interest in optimizing ensembles directly.

4.2.1 Alternate formulations

In order to be able to generalize over the space of hyperparameters, it is crucial to have an ensemble which does not contain all the classifiers in H, because if it did there would be no information added in the computation of Equation 4.3. A different problem formulation could be derived which compares classifiers with the whole pool of trained models, which would take the form $f(\gamma|H) = q(h_\gamma|H, \mathcal{D}_V)$, where $q(\cdot)$ is a metric of performance for a classifier with regards to the pool. For example, a diversity inducing metric such as pairwise disagreement (Kuncheva, 2004) could be used, but this would lead to degenerate pools of classifiers, as diversity is easily increased by trivial and degenerate classifiers (voting all for one class or the other).

Multi-objective optimization approaches have been considered for the maximization of both diversity and accuracy, a problem typically solved with genetic algorithms (Tsymbal et al., 2005). However, this problem formulation does not guarantee a better performing ensemble – only a more diverse pool of classifiers – with the hope that it will lead to better generalization performance. Directly optimizing diversity in classifier ensembles has been met with mixed evidence so far (Didaci et al., 2013; Kuncheva, 2003).

Lastly, an inverse problem could be posed, measuring the difference in the generalization error by removing classifiers from the history one by one, and optimizing this difference. One problem with such a model is that it would be vulnerable to redundancy – very good hyperparameters present in multiple copies in the history would be falsely marked as having little impact on the generalization error.

Given these arguments, the correct approach for Bayesian optimization of ensembles appears to be the proposed one, with a fixed ensemble E, smaller than the set of all available classifiers H. This method requires that the trained models be kept in H during the optimization, and discarded only once the optimization is over, except the models in E.

4.2.2 Determining the ensemble E

The approach described above starts with a predetermined ensemble E and leads to a probabilistic model of $f(\gamma|E)$. In this section, we will propose a method to define and update this ensemble as the optimization progresses. Algorithm 4.1 presents the proposed method in detail. Each iteration will focus on optimizing one classifier of the ensemble E, keeping all other classifiers static. The first step of each iteration is thus to determine the classifier to update, with the procedure C. In this work, we use a round robin selection of all classifiers, given a fixed ensemble size m:

$$C_{RR}(i, m) = i \bmod m. \tag{4.4}$$

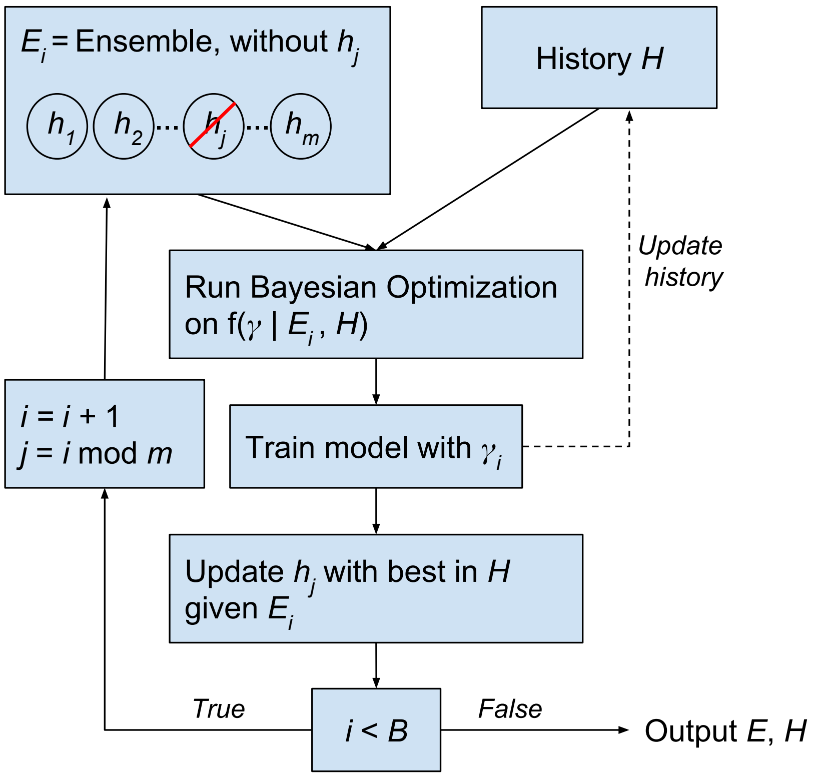
The round-robin procedure optimizes each *slot* in the ensemble iteratively. At every iteration i, the classifier position at position $C_{RR}(i, m)$ will be removed from the ensemble. The next step (lines 5-6) is to condition our surrogate model on observations of Equation 4.3 by adding each classifier in H to E (without replacement in this case) and computing the loss function l over the validation set $\mathcal{D}_V$. A set of hyperparameters is then chosen for the next model to train through the acquisition function (lines 7-8). Finally, the classifier at position j of the ensemble E is updated again greedily (lines 11-12). Figure 4.3 presents a graphical version of this procedure which contains the same operations.

This round-robin optimization allows for the classifiers at each position of the ensemble to be updated sequentially. The classifiers in the pool H should get better as the optimization progresses, and the round-robin selection will replace the early classifiers with better ones eventually, provided the ensemble size m is not too large in comparison with the total number of iterations B for the complete optimization.

The construction of the ensemble and the combination with majority voting are key components of this method. They have been shown empirically to outperform more complex combination methods, such as stacking with logistic regression and Bayesian model averaging, when applied on pools of classifiers (Caruana et al., 2004). One issue with stacking and other fancy combination methods is that they have more

Algorithm 4.1 Ensemble Optimization Procedure.

```
Input: $\mathcal{D}_T$ and $\mathcal{D}_V$, training and validation sets, $A$ the learning algorithm, $m$ the
    ensemble size, $C$ the ensemble member selection, $\ell$ the ensemble loss function, $B$
    the maximal number of iterations
Output: $E$, the final ensemble
 1: $H, G, E \leftarrow \varnothing$
 2: for $i \leftarrow 1, \dots, B$ do
 3:     $j \leftarrow C(i, m)$   // Get position to optimize
 4:     $E_i \leftarrow E \setminus \{h_j\}$   // Empty corresponding slot in ensemble
 5:     $\mathbf{L}_i \leftarrow \{L(E_i \cup \{h\}, \mathcal{D}_V|\ell)\}_{h \in H}$   // Loss on validation for available $h$
 6:     $\hat{\mu}(\gamma|E_i), \hat{\mathbb{V}}(\gamma|E_i) \leftarrow \mathcal{GP}(G, \mathbf{L}_i)$   // Fit model
 7:     $\gamma_i \leftarrow \arg\max_{\gamma \in \Gamma} a(\gamma|\, \hat{\mu}(\gamma|E_i), \hat{\mathbb{V}}(\gamma|E_i))$   // Next hypers
 8:     $h_i \leftarrow \mathrm{A}(\mathcal{D}_T, \gamma_i)$   // Train model
 9:     $G \leftarrow G \cup \{\gamma_i\}$
10:     $H \leftarrow H \cup \{h_i\}$
11:     $h_j \leftarrow \arg\min_{h \in H} L(E_i \cup \{h\}, \mathcal{D}_V|\ell_{0-1})$   // New model at $j$
12:     $E \leftarrow E_i \cup \{h_j\}$   // Update ensemble
13: end for
```



E_i = Ensemble, without h_j
h_1 h_2 ... h_j ... h_m
History H
Update history
Run Bayesian Optimization on f(γ | E_i, H)
$i = i + 1$
$j = i \bmod m$
Train model with γ_i
Update h_j with best in H given E_i
True
$i < B$
False
Output E, H

Figure 4.3: Outline of the Ensemble Optimization procedure with round-robin selection.

power to overfit in the combination of classifiers, hence resulting in poor generalization accuracy. Another important issue is the computational complexity of determining the combination weights, which is $O(1)$ for majority votes, compared with a complexity of up to $O(|\mathcal{D}_V|^2)$ for more sophisticated stacking models (e.g., a kernel SVM). Furthermore, if the combination stage becomes too complex, the cost of estimating the performance greedily for each classifier in the pool H could become comparable with the cost of training the classifiers in the first place, which may be an issue.

4.2.3 Visualizing each position of the ensemble

It is expected that some classifiers will specialize. For instance, when replacing an individually strong classifier, another strong classifier will most likely be required. Figure 4.4 shows an example of optimization on a one-dimensional hyperparameter space run for 50 iterations, where an ensemble of five classifiers was optimized. The five diamonds in the figure represent the five classifiers picked for the ensemble E, plotted at their individual classification error rate. The generalization error rate of the ensemble E is represented by the dotted and dashed line at the bottom of the figure, and the error rate of individual classifiers is represented by the dashed curve, or in other words the objective function for the minimization of the generalization error with regards to a single model $f(\gamma)$.

The coloured lines of Figure 4.4 are produced by successively removing each of the five members i, and by conditioning the Gaussian prior on the performance of the ensemble $E \setminus \{h_i\}$ given the models in the pool as per lines 5-6 of Algorithm 4.1. In this case, the pool of classifiers H contains 50 classifiers in total (i.e. we let the optimization run its course for 50 iterations). We can see from this figure that the hyperparameters which minimize the ensemble error are different for each *slot* in the ensemble, illustrating our concept. Some model seem uninteresting on their own, and probably act as biases to the ensemble prediction (yellow and purple lines and markers).

4.3 Loss Functions

The objective function defined in Equation 4.3 contains a loss function, which up until now referred to the empirical loss of the ensemble, or the zero-one loss. However, directly optimizing the zero-one loss of the ensemble is problematic because of the strong discontinuity present in the loss function. During the first iterations of the optimization procedure, most classifiers are likely to be bad, and a point could be

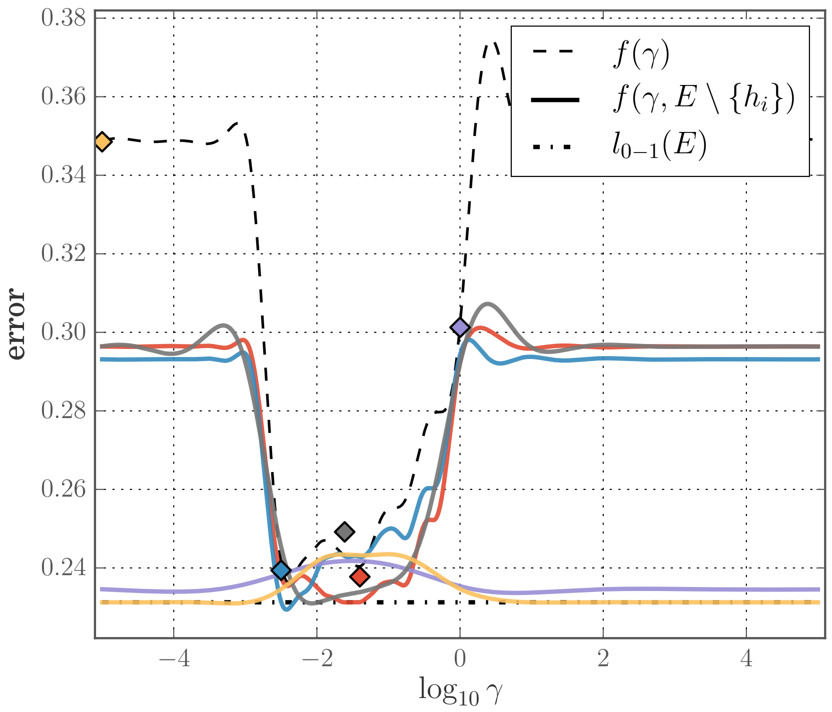


Figure 4.4: Example of objective function $f(\gamma|E \setminus \{h_i\})$ given a pool of 50 trained classifiers on a 1-D hyperparameter optimization problem. Each colour represents the different optimization problems seen by removing each of the classifiers in the ensemble, represented by dots of the same colour.

reached where swapping a bad classifier for a good one does not result in a change of the ensemble prediction. The optimization would effectively be stuck in a plateau caused by the multiple bad classifiers. For this reason, we need to consider different loss functions for the ensemble evaluation. In this section, we will present three different loss functions to be used for the optimization of ensembles: a margin-based loss function, a loss function based on PAC-Bayesian theory (Germain et al., 2015), and a smooth approximation of the zero-one loss function. In Section 4.3.4, we will also present some toy problems exhibiting the trade-offs of each loss function.

4.3.1 Margin-based loss functions

One interesting metric to analyze the performance of ensembles is the *margin* (Schapire and Freund, 2012). Given an ensemble of classifiers E outputting label predictions on a binary problem $y \in \mathcal{Y} = \{-1, 1\}$, the *normalized* margin for a sample $(\mathbf{x}, y)$ is defined as follows:

$$M(E,\mathbf{x},y) = \frac{1}{|E|}\sum_{h\in E} y \cdot h(\mathbf{x}). \tag{4.5}$$

The normalized margin $M \in [-1, 1]$ takes the value 1 when all classifiers of the ensemble correctly classify the sample $\mathbf{x}$, -1 when all the classifiers are wrong, and somewhere in between otherwise. In the case of multi-class problems, predictions of classifiers can be brought back to a binary domain by attributing 1 for a correct classification and -1 for a misclassification. The margin becomes

$$M_{mc}(E,\mathbf{x},y) = \frac{1}{|E|}\sum_{h\in E}[1 - 2\ell_{01}\left(h(\mathbf{x}), y\right)], \tag{4.6}$$

which would also have a value of -1 when all classifiers are wrong and 1 when all classifiers are right. Let us now derive some loss functions from the margin. The margin itself could be the objective, since it is desirable that the margin of the ensemble be high. Rescaled to have a maximum loss of one, this gives the following margin-based loss function:

$$\ell_M(E,\mathbf{x},y) = \frac{1 - M(E,\mathbf{x},y)}{2}. \tag{4.7}$$

This loss function should not be used to optimize an ensemble because it is maximized by the accuracy of *individual* classifiers. In other words, given a validation dataset $\mathcal{D}_V$ and a set of classifiers H to evaluate, the classifier minimizing Equation 4.7 is always the classifier with the lowest empirical error on its own, without regards to the ensemble performance. A proper loss function must take in account the behaviour of the ensemble, not only that of single classifiers. A loss function which achieves this is the squared margin-based loss function:

$$\ell_{M^2}(E,\mathbf{x},y) = \frac{(1 - M(E,\mathbf{x},y))^2}{4}. \tag{4.8}$$

This places a higher emphasis on samples misclassified by the ensemble, and decreases the importance of samples as the margin grows closer to 1. Note that the squared margin loss enforces some diversity between the classifiers of the ensemble, given that the returns are strictly diminishing as the margin increases.

4.3.2 PAC-Bayes and the $\mathcal{C}$-bound

There are some parallels to draw between the ℓ_{M^2} loss function and the PAC-Bayesian ensemble learning framework. Germain et al. (2015) have shown that there is a relation between the performance of an ensemble of classifiers with combination rule Q and the first and second moments of the margin. The combination rule Q takes the form of a probability distribution over classifiers, with better classifiers having higher density. They derive the $\mathcal{C}$-bound on the empirical risk of the ensemble Q. The empirical risk of the ensemble Q on a validation dataset $\mathcal{D}_V$ is defined as follows:

$$\mathcal{R}_Q^{\mathcal{D}_V} = \Pr_{\mathbf{x},y\sim\mathcal{D}_V} \left(M_Q(\mathbf{x},y) \leq 0\right) \tag{4.9}$$

$$M_Q(\mathbf{x},y) = \mathop{\mathbb{E}}_{h\sim Q} y \cdot h(\mathbf{x}), \tag{4.10}$$

where Pr is the probability computed along the given data sample – it is simply a different way to state the empirical zero-one loss of the ensemble. This definition is close to that of the previous section, except that the distribution Q in our case is simply an equal probability over all selected classifiers (and a null probability for non-selected classifiers). Given these definitions, the $\mathcal{C}$-bound takes the following form:

$$\mathcal{R}_Q^{\mathcal{D}_V} \leq \mathcal{C}_Q^{\mathcal{D}_V} = 1 - \frac{\left(\mu_1^{\mathcal{D}_V}(M_Q)\right)^2}{\mu_2^{\mathcal{D}_V}(M_Q)}, \tag{4.11}$$

where $\mu_1^{\mathcal{D}_V}$ and $\mu_2^{\mathcal{D}_V}$ represent the moments of the margin with regards to the validation dataset $\mathcal{D}_V$:

$$\mu_z^{\mathcal{D}_V} = \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}_V} \left[M_Q(\mathbf{x},y)^z\right]. \tag{4.12}$$

The $\mathcal{C}$-bound is an approximation on the risk of the combiner Q, which is then used to provide a bound on the generalization error of the ensemble (described in greater details in Section 5 of Germain et al. (2015)). By minimizing the $\mathcal{C}$-bound, one can strictly reduce an upper bound on the generalization error of the ensemble.

The goal of the $\mathcal{C}$-bound is to allow the optimization of combination rules Q, however in our case the combination rule Q is fixed, it is the majority voting rule. Nonetheless, we can still use the $\mathcal{C}$-bound to compare the contribution of different classifiers h_k when performing a forward greedy selection on $E \cup \{h_k\}$. In other words, all classifiers are equiprobable with majority voting, so the probability distribution Q will never change, but the underlying classifiers will. This will result in different values of the $\mathcal{C}$-bound, which can be directly optimized through hyperparameter optimization.

Taking another angle, the comparison of different ensembles with forward greedy selection can also be seen as the comparison of different combination rules Q in the space of all classifiers available in the pool. In other words, not selecting a classifier is the equivalent of giving it a weight or probability of 0, while selecting it is the equivalent of giving it a weight of $1/m$, where m is the number of classifiers in the majority voting ensemble. From both angles, the $\mathcal{C}$-bound can be exploited in the current framework just like the other losses defined in this section.

Links to the squared-margin loss: We can also tie the $\mathcal{C}$-bound to the squared margin loss defined in the previous section. If we expand the nominator in the squared margin loss function, it becomes:

$$\ell_{M^2}(E, \mathbf{x}, y) = \frac{1 - 2M(E, \mathbf{x}, y) + M(E, \mathbf{x}, y)^2}{4}, \tag{4.13}$$

and taking the expectation of this loss on the validation set, we have:

$$\begin{aligned}
\mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}_V} \ell_{M^2}(E, \mathbf{x}, y) &= \frac{1}{4} \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}_V} \left[1 - 2M(E, \mathbf{x}, y) + M(E, \mathbf{x}, y)^2\right] &(4.14)\\
&= \frac{1}{4}\left(1 - 2 \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}_V} [M(E, \mathbf{x}, y)] + \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}_V} \left[M(E, \mathbf{x}, y)^2\right]\right) &(4.15)\\
&= \frac{1 - 2\mu_1^{\mathcal{D}_V}(M_E) + \mu_2^{\mathcal{D}_V}(M_E)}{4}. &(4.16)
\end{aligned}$$

Hence, the squared margin loss can also be expressed in terms of moments of the margin, even though it results in a different trade-off. In the case of the $\mathcal{C}$-bound, the relation between the moments of the margin is multiplicative, whereas in the squared margin loss the relation is additive. Since both our squared margin loss function and the $\mathcal{C}$-bound are terms to be minimized, high first moment values are rewarded (average margin is high), while low second moment values are rewarded. The second moment $\mu_2(M)$ is directly related to the expected disagreement amongst classifiers in the ensemble, like so[1]:

$$\begin{aligned}
\mathrm{dis}_E^{\mathcal{D}_V} &= \frac{1}{2}\left(1 - \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{D}_\mathcal{V}} \left[\mathop{\mathbb{E}}_{h\sim E} h(\mathbf{x})\right]^2\right) &(4.17)\\
&= \frac{1}{2}\left(1 - \mu_2^{\mathcal{D}_V}(M_E)\right) &(4.18)\\
\mu_2^{\mathcal{D}_V}(M_E) &= 1 - 2\mathrm{dis}_E^{\mathcal{D}_V}. &(4.19)
\end{aligned}$$

From this, we can see that minimizing the squared margin loss consists in maximizing the average margin and disagreement:

$$\mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}_V} \ell_{M^2}(E, \mathbf{x}, y) = \frac{1 - \mu_1^{\mathcal{D}_V}(M_E) - \mathrm{dis}_E^{\mathcal{D}_V}}{2}. \tag{4.20}$$

The same happens for the $\mathcal{C}$-bound loss, although the disagreement can have a stronger impact since it is a denominator.

[1]Full developments in Section 4.1 of Germain et al. (2015).

Alteration for ensemble optimization: One small modification is required to apply the $\mathcal{C}$-bound in our setting, and that is to deal with classifiers which have $\mu_1(M)$ lower than 0, meaning classifiers that are on average more wrong than right. In the setting of Germain et al. (2015), the average error rate of classifiers is assumed to be below 0.5, an assumption which is reasonable and also required for boosting algorithms. However, in our case this is not guaranteed to be true. Early hyperparameter settings can be worse than random, or in the case of multiclass problems, better than random but still wrong more than 50% of the time. Thus, we split the $\mathcal{C}$-bound in two parts, giving the following adapted $\mathcal{C}$-bound:

$$\mathcal{C}_Q^{\mathcal{D}_V} = \frac{1}{2}\begin{cases} 1 - \frac{\left(\mu_1^{\mathcal{D}_V}(M_Q)\right)^2}{\mu_2^{\mathcal{D}_V}(M_Q)} & \text{if } \mu_1^{\mathcal{D}_V}(M_Q) \geq 0 \\ 1 + \frac{\left(\mu_1^{\mathcal{D}_V}(M_Q)\right)^2}{\mu_2^{\mathcal{D}_V}(M_Q)} & \text{otherwise} \end{cases}, \tag{4.21}$$

with an added normalization term to keep the bound between 0 and 1. The previous equation can also be expressed more concisely:

$$\mathcal{C}_Q^{\mathcal{D}_V} = \frac{1}{2}\left(1 - \operatorname{sign}(\mu_1^{\mathcal{D}_V}(M_Q))\frac{\left(\mu_1^{\mathcal{D}_V}(M_Q)\right)^2}{\mu_2^{\mathcal{D}_V}(M_Q)}\right). \tag{4.22}$$

This effectively forces average margin values below zero to be penalized. Figure 4.5 shows the possible values of the $\mathcal{C}$-bound and squared margin loss with regards to the first moment of the margin and the average disagreement. Due to the nature of the computation, some values are impossible and are thus left blank. We can see from this figure that the optimal values of the squared margin loss and $\mathcal{C}$-bound are on the right side of the spaces and consist of trade-offs between the first moment and the second moment. Most interesting are the lines with constant value showing how much weight is put on diversity, based on the underlying loss function. According to the squared margin loss, the ensembles with an average margin of 0.5 and no disagreement is roughly equivalent to an ensemble with a zero margin average but maximal disagreement. The same can be said for the $\mathcal{C}$-bound, although all ensembles can converge to a minimal bound value with maximal disagreement, due to the multiplicative nature of the bound.

4.3.3 Sigmoid as a smooth approximation of the 0-1 loss

Let us introduce one last loss function that will act as a smooth approximation to the real classification loss, or the zero-one loss. We will approximate the step function with a sigmoid function, which has the desirable property of having a value of 0.5 at

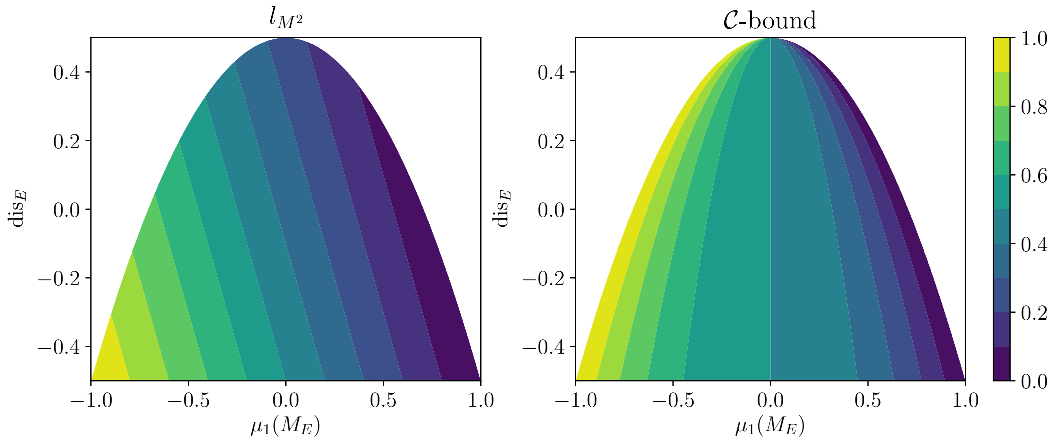


Figure 4.5: Possible values for the squared margin loss and $\mathcal{C}$-bound with regards to the first moment of the margin and the average disagreement.

the transition from wrong to correct classification. For a binary classification task, the zero-one loss function can be approximated as:

$$l_{sig}(E, \mathbf{x}, y) = 1 - s(M(E, \mathbf{x}, y), a), \tag{4.23}$$

where $s(x, a) = 1/(1 + e^{-ax})$ is the sigmoid function, with a scaling parameter a. There are two reasons for using this sigmoid loss function, the first is that it will eliminate the discontinuity at $M(E, \mathbf{x}, y) = 0$, and the second is that it will allow for the differentiation of ensembles in the limits of the margin. Given an ensemble with m classifiers, changing one classifier's vote results in a shift of the margin by $\pm\frac{2}{m}$. Figure 4.6 shows the zero-one loss function and the sigmoid approximation presented here, along with all possible values of the loss functions for an ensemble of size $m = 7$.

One of the objectives of the sigmoid loss function is to allow for the differentiation of classifiers in the stable zones $M \in [-1, 0)$ and $M \in (0, 1]$. The a parameter will thus be determined to have a *minimal difference* of ϵ between every possible pair of points on the curve. Intuitively, the minimal difference between two points is between by the left-most and right-most pairs of points in Figure 4.6, the zone of the sigmoid function where the slope has the lowest value. Concretely, this difference is the shift in loss function value when going from m wrong predictions to $m - 1$ wrong predictions (or

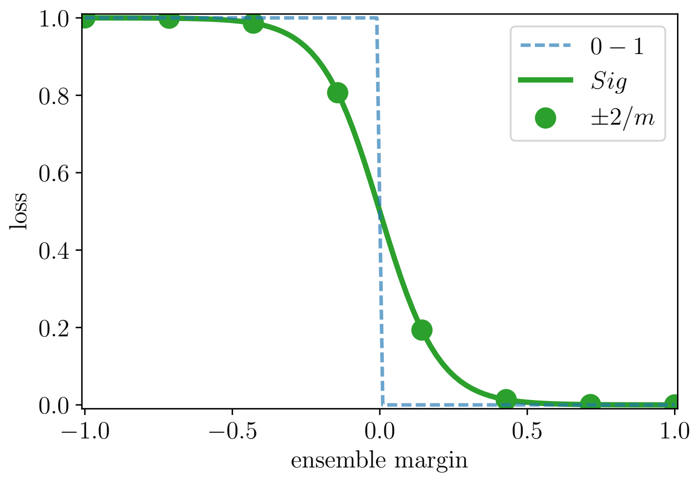


Figure 4.6: Zero-one loss and sigmoid approximation ($a = 10$). Scatter points show possible function values with an ensemble of size $m = 7$.

equivalently, from m to $m-1$ correct predictions):

$$\begin{aligned} d_{min}(a|m) &= 1 - s(-1,a) - (1 - s(-1+2/m, a)) \\ &= -s(-1,a) + s(-1+2/m) \\ &= -s(1-2/m, a) + s(1,a) \\ &\leq \epsilon. \end{aligned}$$

If we solve for $d_{min}(a|m) = \epsilon = 10^{-3}$, the equation has two solutions, one for $a < 1$ and one for $a > 1$. The solution with $a < 1$ gives a sigmoid which behaves almost linearly in $[-1, 1]$, whereas the solution with $a > 1$ has the sharp transition around $M = 0$ that is desired for approximating the zero-one loss, so we pick this one. Note that for every ensemble size a different scaling parameter a should be determined in order to maintain the desired minimum difference.

Multiclass approximation: The above zero-one loss smooth approximation works only for binary classification problems. For the multiclass case, the transition is not necessarily centered at $M = 0$, but can vary depending on the number of votes for each class. Drawing from the multiclass SVM hinge loss (Crammer and Singer, 2001), we can propose an adaptation to the sigmoid loss that will work for multiclass problems.

The prediction of a majority voting ensemble in the multiclass case is defined as the class with the most votes:

$$H(\mathbf{x}) = \arg\max_{y_p} \left[\sum_{h \in E} \mathbb{I}(h(\mathbf{x}) = y_p) \right], \tag{4.24}$$

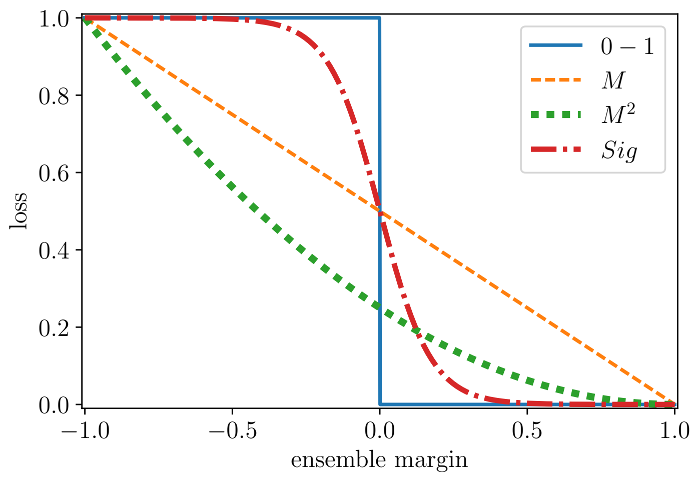


Figure 4.7: Loss functions with respect to the margin of the ensemble.

where $\mathbb{I}$ is the indicator function, returning one if the given condition is true and zero otherwise. With $c > 2$ classes, it is possible for a class to be chosen with a minimum of m/c votes, and the ensemble is guaranteed to predict class c if it obtains more than $m/2$ votes. The transition zone varies with regards to the votes of the other classifiers. Let us define the support of the ensemble for the *correct* class:

$$v_*(E, \mathbf{x}, y) = \sum_{h \in E} \mathbb{I}[h(\mathbf{x}) = y]. \tag{4.25}$$

The other needed quantity is based on the class label y' with the most votes that is *not* the correct label y:

$$v_{max}(E, \mathbf{x}, y) = \max_{y' \neq y} \sum_{h \in E} \mathbb{I}[h(\mathbf{x}) = y']. \tag{4.26}$$

If the ensemble correctly predicts the label of the sample $H(\mathbf{x}) = y$, then $v_* > v_{max}$. If the ensemble wrongly predicts the sample, then $v_{max} > v_*$. In the case of a tie, the winning label is randomly chosen. In all cases, the local loss of the ensemble depends on those two vote counts, v_* and v_{max}. More specifically, the transition area from one ensemble prediction to another is centered around v_{max}, because when v_* becomes greater than v_{max}, the ensemble starts to correctly predict the sample. Knowing this, by posing a sigmoid function centered at 0, with $v_* - v_{max}$ as its input, it is possible to have a smooth approximation of the empirical error once again.

The loss functions presented in this section are plotted with regards to the margin in Figure 4.7, with the exception of the $\mathcal{C}$-bound, which is not directly relatable to the margin of a single instance. The loss function can be seen as a criterion for

Table 4.1: Toy problem with an ensemble $E = \{h_0, h_1, h_2\}$ and potential classifiers to add h'_i. Left are the zero-one losses of single classifiers, right are the ensembles and the loss functions discussed in this section. More details in surrounding text.

	$l_{01}^{\mathbf{x}_0}$	$l_{01}^{\mathbf{x}_1}$	$l_{01}^{\mathbf{x}_2}$	$\bar{l}_{01}(h)$
h_0	0	1	1	0.67
h_1	1	0	1	0.67
h_2	1	1	0	0.67
h'_0	1	0	0	0.33
h'_1	1	1	1	1.00

	$\bar{l}_{01}$	$\bar{l}_M$	$\bar{l}_{M^2}$	$\bar{l}_{sig}$	$\bar{l}_{\mathcal{C}}$
E	1.00	0.67	0.44	0.98	1.00
$E + h'_0$	**1.00**	**0.58**	**0.35**	**0.67**	**0.67**
$E + h'_1$	**1.00**	0.75	0.56	1.00	1.00

the construction of an ensemble, balancing accuracy and diversity, and in this regard there is a hefty literature on ensemble construction which could be applied to this problem (Kuncheva, 2004; Martínez-Muñoz et al., 2009). It is relevant to point out that maximizing diversity on its own leads to ensembles of degenerate classifiers – the difficulty lies in finding how much diversity is required, and this is arguably still an open question (Brown et al., 2005; Didaci et al., 2013; Kuncheva, 2003).

4.3.4 Toy problems

We will now present some examples of loss functions in order to gain a better understanding of the various trade-offs available. Table 4.1 shows an example where the existing ensemble E is filled with bad classifiers. The classifiers h_i represent the classifiers in the ensemble, the classifiers h'_i represent the classifiers available for evaluation. Columns to the left represent metrics for individual classifiers (loss on single examples, average loss for the classifier), and columns to the right of the vertical space represent ensemble metrics. The top parts represent the current state, or the current ensemble $E = \{h_0, h_1, h_2\}$ and its loss values, while the bottom part shows new candidate classifiers h'_i and the resulting ensemble losses if those classifiers were added to E. Ensemble metrics in boldface represent the best value among the available classifiers (in this case the choice is between h'_0 and h'_1). A good loss function should be able to identify that classifier h'_0 would bring the ensemble closer to having correct predictions, and we can see that the classical zero-one loss is unable to do so. All other loss functions are able to identify that classifier h'_0 would be a better choice.

Table 4.2 shows another example in which the available classifiers present a less obvious trade-off. All individual classifiers have an empirical error of 0.4, and at first the ensemble has the same error rate. This example shows that the squared margin and

Table 4.2: Toy problem with an ensemble $E = \{h_0, h_1, h_2, h_3\}$ and potential classifiers to add h'_i. Left are the zero-one losses of single classifiers, right are the ensembles and the loss functions discussed in this section. More details in surrounding text.

	$l_{01}^{\mathbf{x}_0}$	$l_{01}^{\mathbf{x}_1}$	$l_{01}^{\mathbf{x}_2}$	$l_{01}^{\mathbf{x}_3}$	$l_{01}^{\mathbf{x}_4}$	$\bar{l}_{01}(h)$
h_0	1	1	0	0	0	0.40
h_1	0	0	0	1	1	0.40
h_2	0	0	1	1	0	0.40
h_3	1	0	0	1	0	0.40
h'_0	0	0	1	0	1	0.40
h'_1	1	1	0	0	0	0.40
h'_2	0	0	0	1	1	0.40

	$\bar{l}_{01}$	$\bar{l}_M$	$\bar{l}_{M^2}$	$\bar{l}_{sig}$	$\bar{l}_{\mathcal{C}}$
E	0.40	0.40	0.20	0.30	0.40
$E + h'_0$	**0.20**	**0.40**	**0.18**	**0.24**	**0.31**
$E + h'_1$	0.40	**0.40**	0.19	0.38	0.38
$E + h'_2$	**0.20**	**0.40**	0.21	**0.24**	0.41

$\mathcal{C}$-bound losses do not directly optimize for classification loss, whereas the sigmoid more closely follows the zero-one loss. This can be shown by the fact that classifier h'_1 has lower loss value than the classifier h'_2 with the squared margin and $\mathcal{C}$-bound losses, while it results in a higher ensemble zero-one error. We can also see that the simple margin loss only depends on the single classifier loss and would not be able to differentiate between the three candidates in this case.

These two examples demonstrate that the straight ensemble zero-one loss and the simple margin loss are inadequate for ensemble optimization. These toy problems also highlight differences between the squared margin, $\mathcal{C}$-bound, and the sigmoid approximation of the zero-one loss, for instance the sigmoid loss directly maximizes ensemble performance, whereas the other two losses encourage some diversity.

4.4 Experiments

We showcase the performance of the proposed ensemble optimization approach on a few different problems. The datasets for the first two problems are taken from UCI. For every repetition, a different hold-out testing partition was sampled with 33% of the total dataset size (unless the dataset had a pre-specified testing split, in which case it was used for all repetitions). The remaining instances of the dataset were used to complete a 5-fold cross-validation procedure. Final models are retrained on all the data available for training and validation.

Unless otherwise noted, the prior on the objective function is a Gaussian process with

Matérn-52 kernel using automatic relevance determination[2]. The noise, amplitude, and length-scale parameters are obtained through slice sampling (Snoek et al., 2012). The slice sampling of GP hyperparameters for ensemble optimization must be reinitialized at every iteration given that different ensembles E can change the properties of the optimized function drastically. The acquisition function is the Expected Improvement over the best solution found so far. The methods and their abbreviated names are the following:

- Classical Bayesian optimization (BO-best). It returns a single model selected with an argmin on validation performance (Snoek et al., 2012).
- Post-hoc ensemble constructed from the pool of classifiers with Bayesian optimization (BO-post). The post-hoc ensemble is initiated by picking the three best classifiers from the pool before proceeding with forward greedy selection – this form of warm starting is recommended by Caruana et al. (2004) to reduce overfitting.
- Random search, single best model found and post-hoc ensemble. The same selection methods described above for Bayesian optimization are applied to hyperparameters sampled uniformly from the defined search spaces, giving RS-best and RS-post.
- Our ensemble optimization approach, where either the ensemble E found during the optimization is returned (EO), or a post-hoc ensemble is built from scratch using forward greedy optimization, using the trained models as a pool of classifiers (EO-post).

4.4.1 SVM search space

The models used in this benchmark are SVM models, and the parameterization includes the choice of the kernel along with the various hyperparameters needed per kernel. The hyperparameter space optimized Γ can be described as follows:

- One hyperparameter for the kernel choice: linear, RBF, polynomial, or sigmoid;
- Regularization constant $C \in [10^{-5}, 10^{5}]$ (for all kernels);
- RBF and sigmoid kernels both have a kernel width parameter $\tau_{RBF} \in [10^{-5}, 10^{5}]$;

[2]Code from `http://github.com/JasperSnoek/spearmint`.

- Polynomial kernel has a degree parameter $d \in [1, 10]$;
- Sigmoid and polynomial kernels both have an intercept parameter, $c \in [10^{-2}, 10^{2}]$.

The hyperparameter space contains a categorical parameter, the kernel choice. It is represented using a continuous parameter which is later discretized. This does not deal with the fact that hyperparameters for different kernels are disjoint and should not be modelled jointly. Chapter 5 describes tools to better optimize spaces with structure such as this one, but they are not the concern in this application. Since all the compared methods are using the same technique, the comparison is fair.

The ensemble optimization method uses the squared margin loss function, first introduced in (Lévesque et al., 2016). The next section introduces more extensive experiments on all loss functions. The EO-post method is also included, where a post-hoc ensemble is constructed from the final pool of models trained during the optimization, in order to see if it helps to restart the ensemble generation procedure at the end. The procedure applied is exactly the same as for BO-post, only the pool of classifiers change.

All compared approaches optimized the same search space. Each method is given a budget of $B = 200$ iterations, or 200 hyperparameter tuples tested, to optimize hyperparameters with a 5-fold cross-validation procedure. The ensemble selection stage exploits this cross-validation procedure, considering the next classifier which reduces the most the generalization error over all the cross-validation folds. Selected hyperparameters are retrained on the whole training and validation data, and combined directly on the testing split to generate the generalization error values presented in this section. The ensemble optimization method is run with an ensemble size $m = 12$, left fixed for all problems. The optimal zone for ensemble size appears to be between 10 and 30, according to experiments not shown in the thesis. The ensemble size is left fixed for all problems in this benchmark to show that the method can perform strongly without tuning the ensemble size. Future work could investigate strategies to dynamically size the ensemble as the optimization progresses, with no fixed limit.

Results

The results are compiled into rank-based comparisons, followed by a table containing the raw error rates over all datasets. The ranks for the rank-based tests are computed by taking the average generalization error of the test set over all repetitions, and then

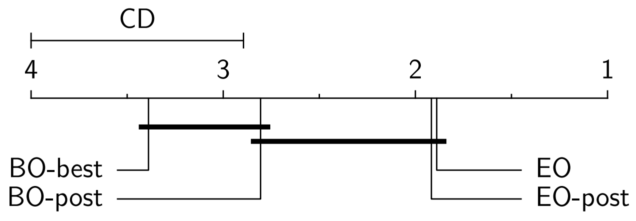


Figure 4.8: Nemenyi post-hoc test for SVM search space. Methods joined by a bold line are not significantly different ($p > 0.05$). The methods are sorted by average rank, with 4 being the worst rank (on the left) and 1 being the best (on the right).

Table 4.3: Wilcoxon pairwise test p-values for the SVM hyperparameter space. Bold entries highlight significant differences ($p \leq 0.05$) and parentheses are added when method at row i is worse than the method at column j according to ranks.

	1	2	3	4
1 - BO-best	–	(0.33)	(**0.00**)	(**0.00**)
2 - BO-post	0.33	–	(**0.00**)	(**0.01**)
3 - EO-post	**0.00**	**0.00**	–	(0.21)
4 - EO	**0.00**	**0.01**	0.21	–

sorting methods per dataset, as described in the Section 3.3.1. Given these ranks, the Friedman test is then the first test to run, which will tell us if there is any difference between the group of all methods (Demšar, 2006). This test finds a significant difference with p-value of 5.5e$-$4. We then follow the Friedman test with a post-hoc Nemenyi test, shown in Figure 4.8, also defined in greater details in Demšar (2006). Methods linked by bold lines are not significantly different according to the Nemenyi test with a significance level of $\alpha = 0.05$. The Nemenyi post-hoc test gives a more visual insight as to what is going on, but it is sensitive to the pool of tested methods – the outcome of the test can change if new methods are inserted in the experiments. According to this test, EO and EO-post are both significantly different from BO-best, meaning that ensemble optimization is significantly better than the single best classifier returned by Bayesian optimization. The same is not true for the post-hoc ensemble built from the output of the Bayesian optimization, BO-post, which is not found significantly different from all three other methods.

For a finer look at the results, we also present pairwise Wilcoxon signed-rank tests for all pairs of methods in the experiment (Demšar, 2006). The results of this procedure

Table 4.4: Generalization error on SVM hyperparameter space, averaged over 10 repetitions, 5-fold cross-validation. Last column shows the rank of methods averaged over all datasets.

	adlt	bnk	car	ches	ltr	mgic	msk	p-blk	pim	sem	spam	s-gc	s-im	s-sh	s-pl	thy	tita	wine	ranks
BO-best	<u>15.52</u>	10.67	1.27	<u>16.86</u>	2.45	<u>12.49</u>	0.29	3.06	25.52	4.43	6.47	23.20	<u>3.57</u>	<u>0.10</u>	<u>23.91</u>	3.09	20.59	<u>35.28</u>	3.39
BO-post	15.38	10.71	<u>1.56</u>	<u>16.72</u>	2.50	12.21	**0.28**	3.01	25.65	**4.37**	6.47	23.45	<u>2.94</u>	**0.08**	**22.58**	<u>3.17</u>	20.59	<u>35.09</u>	2.81
EO-post	**15.27**	10.60	0.95	15.08	2.36	12.21	0.28	**2.97**	24.03	4.40	**6.36**	23.40	**2.55**	0.09	22.63	2.69	20.57	33.70	1.92
EO	<u>15.39</u>	**10.44**	**0.81**	**15.06**	**2.34**	**12.18**	0.30	3.14	**23.70**	4.58	6.45	**23.05**	2.73	0.09	22.61	**2.51**	**20.27**	**33.29**	1.89

are shown in Table 4.3, with significant differences at $\alpha = 0.05$ highlighted in bold. We can see from the Wilcoxon tests that our ensemble optimization methods (both EO and EO-post) significantly outperform the single model and post-hoc ensemble built from regular Bayesian optimization. There is no significant difference between BO-best and BO-post according to the Wilcoxon test, however BO-post does have a lower average rank over all the datasets (2.81 against 3.39).

Finally, there is no significant difference between EO and EO-post, suggesting that there is no need for a post-hoc ensemble construction – in other words, there is no need to rebuild a fresh ensemble after the optimization is finished. Caruana et al., 2004 presented some strategies to reduce overfitting in the forward greedy procedure – such as bagging from the pool of models – which could be considered in order to achieve more with the same pool, although this is left for future work.

At the lowest level of analysis, Table 4.4 presents the generalization error on the test split for the selected methods on each dataset, averaged over the 10 repetitions. The last column shows the ranks of each method averaged over all datasets, where the best rank is 1 and the worst rank is 4. For each dataset, the lowest error score is highlighted in bold, and methods that are significantly worse than this best method (according to a Wilcoxon pairwise test on the repetitions) are underlined. Summarily, we can see that the EO method achieves the lowest average error rate (across 10 repetitions) on 10 datasets out of 18, and is significantly outperformed only once by EO-post. In contrast, BO-best achieves the lowest error rate on none of the datasets, and is outperformed on 7 datasets.

4.4.2 Loss functions and scikit-learn search space

Next, we evaluate the same methods as in the previous section on another search space with the addition of different loss functions. The search space consists of multiple base learning algorithms, all available from `scikit-learn`[3]. The search space includes multiple different base learners, and thus an extra hyperparameter is added for the choice of the classifier. The models and their hyperparameters are as follows:

- *K nearest neighbours* with the number of neighbours `n_neighbours` in $[1, 30]$;
- *RBF SVM* with the penalty `C` logarithmically in $[10^{-5}, 10^{5}]$ and the width of the RBF kernel τ_{RBF} logarithmically in $[10^{-5}, 10^{5}]$;
- *linear SVM* with the penalty `C` logarithmically scaled in $[10^{-5}, 10^{5}]$;
- *decision tree* with the maximal depth `max_depth` in $[1, 10]$, the minimum number of examples in a node to split `min_samples_split` in $[2, 100]$, and the minimum number of training examples in a leaf `min_samples_leaf` in $[2, 100]$;
- *random forest* with the number of trees `n_estimators` in $[1, 30]$, the maximal depth `max_depth` in $[1, 10]$, the minimum number of examples in a node to split `min_samples_split` in $[2, 100]$, and the minimum number of training examples in a leaf `min_samples_leaf` in $[2, 100]$;
- *AdaBoost* with the number of weak learners `n_estimators` in $[1, 30]$;
- *Gaussian Naive Bayes (GNB)* and *Linear Discriminant Analysis (LDA)* both without any hyperparameters;
- *Quadratic Discriminant Analysis (QDA)* with the regularization `reg_param` logarithmically in $[10^{-3}, 10^{3}]$.

There is a difference with the previous section's experiment in that the inactive hyperparameters are imputed before being fed to the Gaussian process surrogate model. This means that only the hyperparameters for the currently chosen learning algorithm take a value in the dense hyperparameter vector, with other hyperparameters having a default value, hence not influencing the covariance value. More details on this matter can be found in Chapter 5.

We also added a new baseline optimization method, the covariance matrix adaption evolution strategy (CMA-ES). It is not expected that this method will perform very well since the space has integer-valued parameters and some structureDatasets identified

[3]Available at `http://scikit-learn.org/`.

by numbers were taken from OpenML, the numbers represents their ID in the OpenML database.

Since the search space contains base models that are ensemble methods themselves, our ensemble optimization procedure can end up building ensembles of ensembles. At this point, one could wonder what is the additional benefit of this stacking in comparison with a single ensemble, which is itself optimized jointly on a greater number of samples (the training data). The answer comes in part from the "no free lunch theorem" (Wolpert, 1996), meaning that a single model cannot be optimal for all problems. This is shown in the study of Feurer et al. (2015a), where they compared the optimization over a joint space with disjointed optimizations for isolated models on 13 datasets. They show that optimizing jointly with a post-hoc ensembling performed the best or not significantly different from the best on 12 out of 13 datasets, while Gradient Boosted Trees achieved the best performance on 10 out of 13, AdaBoost on 8 out 13, and Ranfom Forests on 7 out of 13. Thus, the conclusion is that it is justifiable to consider the construction of ensemble of ensembles, and to let the optimization procedure decide what to pick. Furthermore, if any single ensemble method is performing much better than other methods, it should be reflected by the performance of the BO-best method, which is the single best model identified by Bayesian optimization.

All three loss functions presented in Section 4.3 are evaluated, namely the squared margin loss (EO-sqm), the C-bound loss (EO-C) and the smooth zero-one loss (EO-sig). The maximum number of iterations B is again set to 200, and results present the average over 10 repetitions.

Results

The difference between the methods' ranks was assessed with a Friedman's rank test, with positive result (p = 1.43e$-$10). This allows us to follow up with a Nemenyi post-hoc test, which compares all methods against themselves. The results of the Nemenyi test are presented in Figure 4.9. We can see that most Bayesian methods (BO/EO) are on the same level, with no significant statistical difference between them according to the Nemenyi test. Note that the BO-best and BO-post are not significantly different from a post-hoc ensemble built from random hyperparameter optimization (RS-post). Our ensemble optimization methods (with all loss functions) are the only ones significantly better than a post-hoc ensemble built with random hyperparameters. Additionally, according to this Nemenyi test the single best model found by regular Bayesian optimization is not significantly different from the single best model found by

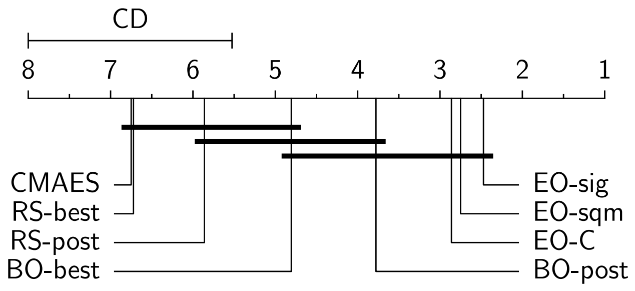


Figure 4.9: Nemenyi post-hoc test based on ranks. Methods joined by a bold line are not statistically different according to the Nemenyi test with level $\alpha = 0.05$. The methods are sorted by average rank, with 8 being the worst rank (far left) and 1 being the best (far right).

random search.

Pairwise Wilcoxon signed-rank tests for all pairs of methods are presented in Table 4.5. We can see from those tests that EO-sqm and EO-sig both outperform all baselines. On the other hand, the EO-C method is not able to significantly outperform the BO-post baseline, even though the C-bound loss gives guarantees on the generalization error. It is possible that the optimization was somehow impeded by the use of the C-bound as an objective, which strongly benefits from diversity in the ensemble. The same can be said about the squared margin loss, albeit to a lesser extent. The squared margin still directly maximizes the margin, only with diminishing returns.

Furthermore, we can see that CMA-ES performs roughly as good as a random search in this search space, which is understandable given the structured nature of the space (only one classifier is active for any given hyperparameter vector).

These Wilcoxon tests also show that there is no significant difference between the proposed loss functions, although the average ranks suggest that the sigmoid loss function is the stronger one. Interestingly, we also remark that random search, both for the single-best classifier and post-hoc ensemble, is outperformed by all Bayesian optimization methods.

Table 4.6 shows the generalization errors on the test sets, average over repetitions for the described methods. Once again, for each dataset, the lowest error score is highlighted in bold, and methods that are significantly worse than this best method (according to a Wilcoxon pairwise test on the repetitions) are underlined. We will highlight some

Table 4.5: Pairwise Wilcoxon signed-rank tests on each pair of methods in the experiment. Each entry represents the outcome of a row-vs-column test, results in bold indicate significant differences, and results in parenthesis indicate that the method in the row was worse than the method in the column (according to average rank).

	1	2	3	4	5	6	7	8
1 - CMA	–	(0.81)	(0.16)	(**0.00**)	(**0.00**)	(**0.00**)	(**0.00**)	(**0.00**)
2 - RS-best	0.81	–	(0.43)	(**0.00**)	(**0.00**)	(**0.00**)	(**0.00**)	(**0.00**)
3 - RS-post	0.16	0.43	–	(0.08)	(**0.01**)	(**0.00**)	(**0.00**)	(**0.00**)
4 - BO-best	**0.00**	**0.00**	0.08	–	(**0.01**)	(**0.01**)	(**0.00**)	(**0.00**)
5 - BO-post	**0.00**	**0.00**	**0.01**	**0.01**	–	(0.06)	(**0.03**)	(**0.01**)
6 - EO-C	**0.00**	**0.00**	**0.00**	**0.01**	0.06	–	(0.81)	(0.72)
7 - EO-sqm	**0.00**	**0.00**	**0.00**	**0.00**	**0.03**	0.81	–	(0.21)
8 - EO-sig	**0.00**	**0.00**	**0.00**	**0.00**	**0.01**	0.72	0.21	–

Table 4.6: Error rates on the test set for the baseline methods, and ensemble optimization with three different loss functions. Methods sorted by rank.

	adlt	bnk	car	ches	ltr	mgic	msk	p-blk	pim	sem	spam	s-gc	s-im	s-sh	s-pl	thy	tita	wine	ranks
CMA	14.48	10.64	2.63	19.08	3.04	12.66	1.85	3.09	23.96	6.10	5.69	23.75	3.66	0.07	25.06	1.09	23.74	35.56	6.75
RS-best	14.44	10.70	1.45	20.73	3.14	12.76	0.64	3.34	23.83	6.07	5.67	23.25	3.77	0.09	24.09	1.10	24.06	37.30	6.72
RS-post	14.31	10.61	2.31	24.38	3.29	12.92	1.78	2.79	23.83	6.10	5.02	**22.65**	3.23	0.07	22.28	1.13	21.95	36.02	5.86
BO-best	14.39	10.35	0.84	16.18	3.48	12.49	0.26	2.60	24.61	5.29	5.01	23.10	2.71	0.03	23.17	1.09	21.52	33.17	4.81
BO-post	14.29	10.32	0.95	14.52	3.42	12.14	0.28	2.58	**23.70**	5.33	4.91	23.15	2.53	0.03	22.74	1.10	21.25	33.01	3.78
EO-C	14.41	10.27	1.01	14.30	2.27	12.02	0.47	**2.50**	23.83	4.55	4.68	23.30	**2.42**	0.03	**21.41**	1.09	20.61	**31.40**	2.86
EO-sqm	14.49	**10.21**	**0.66**	14.18	2.27	12.28	0.29	2.57	24.22	**4.40**	**4.58**	22.80	2.55	0.03	21.51	1.09	**20.48**	32.11	2.75
EO-sig	**14.24**	10.34	0.95	**13.94**	**2.23**	**11.95**	**0.24**	2.50	24.35	4.55	4.80	22.90	2.62	**0.02**	22.10	**1.07**	20.77	32.45	2.47

similar metrics as in the previous section: the method with the lowest average rank EO-sig has the lowest error rate on 7 out of 18 datasets, and is outperformed only on one dataset (wine). In comparison the BO-best method is significantly worse than the best on 7 datasets out of 18, while being the best on none of the datasets. EO-sqm is significantly worse than the best method on three datasets, and EO-C on two datasets. Overall, these tests are only presented to give a greater level of detail to the interested reader, the tests to which we attribute the most importance are the Nemenyi and Wilcoxon tests performed across all datasets.

4.4.3 Convolutional neural networks

Lastly, we evaluated the performance of our approach when fine-tuning the parameters of a convolutional neural network for the CIFAR-10 dataset. In order to have a reproducible baseline, the `cuda-convnet` implementation was used with the reference model files given which achieves 18% generalization error on the testing dataset, without

Table 4.7: Average generalization error and standard deviation in validation and testing after post-hoc ensembling, computed across repetitions.

Method	Validation	Testing
BO-post	16.61 ± 0.24	15.50 ± 0.42
EO-post	15.36 ± 0.23	14.06 ± 0.44

data augmentation[4]. One batch of the training data was set aside for validation (batches 1-4 used for training, 5 for validation, and 6 for testing). Performance of the baseline configuration on the given training batches was around $22.4\% \pm 0.9$ for 250 epochs of training. The parameters optimized were the same as in (Snoek et al., 2012), namely the learning rates and weight decays for the convolution and softmax layers, and the parameters of the local response normalization layer (size, power and scale). The number of training epochs was kept fixed at 250.

We computed 10 repetitions of a standard Bayesian optimization and our proposed ensemble optimization with ensemble size $m = 7$, both with a budget of $B = 100$ hyperparameter tuples to evaluate. Figure 4.10 shows the performance of ensembles generated from both pools of classifiers with a post-hoc ensemble generation. In order to limit overfitting, the first three models of each ensemble were selected directly based on accuracy, as suggested in (Caruana et al., 2004). In both cases, the ensemble size benefits the generalization accuracy, although the classifiers generated by the ensemble optimization procedure do perform slightly better. The difference in generalization error between BO-post and EO-post at the last iteration is found significant by a Wilcoxon test with a p-value of 0.005. Table 4.7 shows the raw generalization error figures after 100 iterations of post-hoc ensemble selection. Further work should investigate strategies to use the remaining validation data once the models are chosen, to further improve generalization accuracy.

4.5 Conclusion

In this chapter, we presented a methodology for the optimization of ensembles through Bayesian hyperparameter optimization. We tackled the various challenges posed by ensemble optimization in this context, and the result is an optimization strategy that is able to exploit trained models efficiently and generate better ensembles of classifiers

[4]Code available at `https://code.google.com/archive/p/cuda-convnet/` and network configuration file used is `layers-18pct.cfg`.

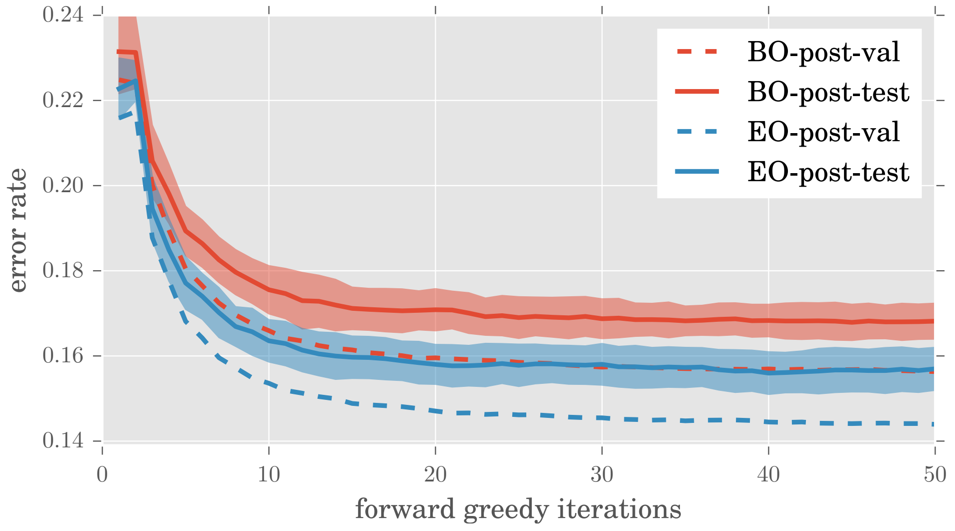


Figure 4.10: Generalization errors of a post-hoc ensemble with classical Bayesian optimization (BO-post) and a post-hoc ensemble generated from our ensemble optimization approach (EO-post) on the CIFAR-10 dataset with regards to the number of classifiers in the final ensemble. Results averaged over 10 repetitions.

at a computational cost comparable to a regular hyperparameter optimization. The research questions targeted by this work were:

1. What mechanism of optimization allows for the efficient optimization of hyperparameters for ensembles?
2. Can we obtain better performing ensembles by directly optimizing hyperparameters for the performance of the ensemble in comparison with an optimization for the single-best classifier?

The answer to question (1) is presented in Section 4.2 in the form of the suggested ensemble optimization methodology. Finally, the answer to question (2) is a definite yes, as shown by the empirical evaluation in Section 4.4.

We showcase the performance of our approach on three different problem suites, and in all cases observe a significant difference in generalization accuracy between our approach and post-hoc ensembles built on top of a classical hyperparameter optimization, according to Wilcoxon signed-rank tests. This is a strong validation of our method, especially considering that it involves little extra computation in the form of a forward greedy pass at each iteration, and does not require to train more models than a regular hyperparameter optimization.

Chapter 5

Kernels for Conditional Hyperparameter Spaces

As was seen in Chapter 4, the scope of hyperparameter optimization can be extended to include the choice of a learning algorithm. Rather than only concern ourselves with the hyperparameters of a single learning algorithm chosen beforehand, we can optimize jointly both the choice of the learning algorithm and the hyperparameters of all available models. If a model family is performing poorly, less time should be spent to tune its hyperparameters, something that can not be achieved if hyperparameters are tuned separately for each model.

This optimization problem has been called a Combined Algorithm Selection and Hyperparameter optimization (CASH) problem, examples of which are solved in the Auto-WEKA (Thornton et al., 2013) and Auto-Sklearn frameworks (Feurer et al., 2015a). Given a classification or regression dataset, the Auto-WEKA platform attempts to identify the best algorithm and the best configuration amongst a wide selection of algorithms available in the WEKA library. The Auto-Sklearn framework takes the same approach with the scikit-learn library, but also adds the choice of the preprocessing and data transformation methods.

The aforementioned works can be regrouped under the emerging field of automatic machine learning (AutoML). The objective of AutoML is to completely automate the machine learning pipeline, including preprocessing method and model choice. The ultimate goal of AutoML is to take the machine learning expert out of the loop for the generation of carefully tuned predictive models. Another example of AutoML framework is the automatic statistician, a probabilistic regression tool (Duvenaud et al., 2013; Lloyd et al., 2014), where a composition of many different kernels is used with GPs to model a wide variety of function types.

One challenge with the optimization problems posed by AutoML is the *conditionality* of some hyperparameters. Conditional hyperparameters need only be specified if another hyperparameter 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 hyperparameters specific to this classifier are relevant. Regular learning algorithm hyperparameters can also induce conditionality, for instance a neural network may have a varying number of hyperparameters to tune based on its depth. The Sequential Model-based Algorithm Configuration (SMAC; Hutter et al., 2011) and the Tree Parzen Estimators (TPE; Bergstra et al., 2011), both handling conditional or hierarchical hyperparameters, were compared on the Auto-WEKA problem, with SMAC outperforming TPEs. Bayesian optimization with Gaussian processes and conditional hyperparameters have been studied by Bergstra et al. (2011) and Hoffman et al. (2014), but appear to be inferior to both TPE and SMAC with random forests as a surrogate model on the Auto-WEKA benchmark (Eggensperger et al., 2013).

In this chapter, we show that Bayesian optimization with GPs also 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 favourably 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 include some form of conditionality.

The work in this chapter is largely based on Lévesque et al. (2017), with some minor revisions and additions.

In the next section, we briefly describe some previous work on hyperparameter optimization. In Section 5.2, 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 5.4, our proposed kernels for the optimization of conditional spaces are presented, which are the conditional kernel and the Laplace kernel. Section 5.5 presents experiments on a CASH problem with models from scikit-learn, and following discussions.

5.1 Related Works

Bayesian optimization and related techniques were introduced in Chapter 2. Amongst the methods of particular interest for the problem at hand are TPEs, which define a custom hierarchy for Bayesian optimization with tree-structured Parzen density estimators (Bergstra et al., 2011). TPEs supports conditions in the hyperparameter space through the branching of the tree. SMAC is another toolbox for the Bayesian optimization of hyperparameters for learning algorithms and SAT solvers, with support for conditional hyperparameters (Hutter et al., 2011). 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 (Thornton et al., 2013; Eggensperger et al., 2013). 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 (Feurer et al., 2015a).

The work presented in this chapter is compatible with most of the recent improvements to the regular Bayesian optimization pipeline, such as meta-learning or dynamic data set size configuration. To a larger extent the methods mentioned in the literature review in Section 2.5 are still relevant here.

5.2 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 $\mathcal{D}_V$:

$$L(h_\gamma, \mathcal{D}_V) = \frac{1}{|\mathcal{D}_V|} \sum_{(\mathbf{x},y)\in\mathcal{D}_V} \ell(h_\gamma(\mathbf{x}), y), \tag{5.1}$$

$$f(\gamma) = L(h_\gamma, \mathcal{D}_V) + \varepsilon, \tag{5.2}$$

where h_γ is the prediction rule obtained by running learning algorithm A with hyperparameters γ on a given training dataset $\mathcal{D}_T$, and $\ell(\cdot, y)$ 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 $\mathrm{GP}(\mu, k)$ 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 $\mathrm{GP}(\mu, k)$ on observed outputs $\mathbf{y} = \{L(h_{\gamma_i}, \mathcal{D}_V)\}_{i=1}^{t}$ associated with inputs $G = \{\gamma_1, \dots, \gamma_t\}$, 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

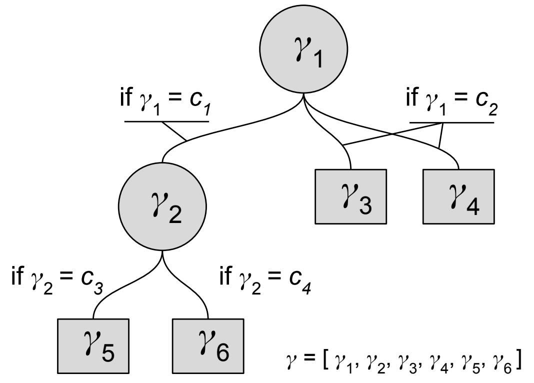
$$\hat{\mu}(\gamma_*) = \mathbf{k}_*^T (K + \sigma_n^2 I)^{-1} \mathbf{y}, \tag{5.3}$$

$$\hat{\mathbb{V}}(\gamma_*) = k(\gamma_*, \gamma_*) - \mathbf{k}_*^T (K + \sigma_n^2 I)^{-1} \mathbf{k}_*, \tag{5.4}$$

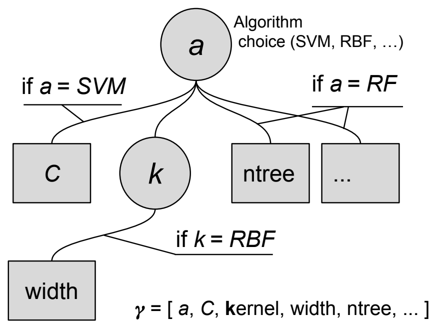
where $\mathbf{k}_* = [k(\gamma_1, \gamma_*), \dots, k(\gamma_N, \gamma_*)]^T$ and K is the positive semi-definite kernel matrix $[k(\gamma, \gamma')]_{\forall (\gamma, \gamma') \in (G \times G)}$. On each trial t, the GP is conditioned on the full history of observations $\mathcal{H} = \{\gamma_i, L(h_{\gamma_i},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

5.3 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 $\mathcal{A} = \{A_1, A_2, \dots, A_m\}$ and their respective hyperparameter spaces $\Gamma_1, \Gamma_2, \dots, \Gamma_m$, the resulting search space becomes $\mathcal{K} = \mathcal{A} \cup \Gamma_1 \cup \Gamma_2, \dots, \Gamma_m$, 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 $1 + \sum_{i=1}^{m} |\Gamma_i|$. 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 $\boldsymbol{\kappa}_i = \{a_i, \gamma_{1,i}, \dots, \gamma_{m,i}\}$ and L_i is the empirical loss of algorithm $A_i = A_{a_i}$ using hyperparameters $\gamma_{a_i,i}$. This formulation is the equivalent of assuming that there is information shared between the hyperparameters of different learning algorithms, through the GP kernel. A GP with a regular distance-based kernel would not be able to distinguish between active and inactive parameters in a dense input vector $\boldsymbol{\kappa_i}$, generating a confusion concerning the responsibility of hyperparameters for model performance, or, in other words, making an invalid credit assignment.



(a) Generic space


a
Algorithm choice (SVM, RBF, ...)
if a = SVM
if a = RF
C
k
ntree
...
if k = RBF
width
γ = [a, C, kernel, width, ntree, ...]

(b) Combined algorithm selection and hyperparameter optimization

Figure 5.1: Hyperparameter spaces with conditions.

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 chapter. 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 5.1a shows a generic configuration space with hierarchical structure, where various equality conditions are defined. Given hyperparameters γ_i (with subscript now representing index in the hyperparameter vector), the conditionality is presented 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_3, and γ_1 takes the value c_1 (otherwise parameter γ_2 would not 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. Figure 5.1b also shows a concrete example of hyperparameter space for a CASH problem, where the root hyperparameter represents the choice of the learning algorithm. The figure only presents two base classifier for visualization purposes.

5.3.1 Imputation

In order to train a GP or RF surrogate model of $f(\gamma)$, some values need to be given even for inactive parameters, since the models expect dense input values. 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. (2015a).

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 observed through the covariance function, which can be drawn in 2D. This type of similarity analysis is not possible with non-kernel-based models such as the random forest.

Let us define a dummy search space with four parameters, one being a categorical classifier choice parameter (with possible values a, b, and c) 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:

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(\frac{-\|\mathbf{x} - \mathbf{x}'\|_2^2}{\ell}\right), \tag{5.5}$$

where ℓ is a length-scale parameter arbitrarily set to 0.5 in this example. Multiple vectors of hyperparameters were densely sampled in the unit cube $[0, 1]^d$, giving $X \in \mathbb{R}^{n\times d}$, with $n = 100$ and $d = 4$ in this case. Categorical parameters are converted to the nearest valid value. The covariance matrix is then expressed as $K \in \mathbb{R}^{n\times n}$, where $K_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$. Figure 5.2 shows the resulting covariance matrix for two cases, the case where the evaluated hyperparameters were the unaltered samples X, 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, 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 $b - c$. Most importantly, we can see that imputation helps reducing the amount of noise when comparing pairs of samples from the same conditional category – this translates into a higher covariance between samples belonging to the same condition. No imputation can result in samples being considered further apart than they really are.

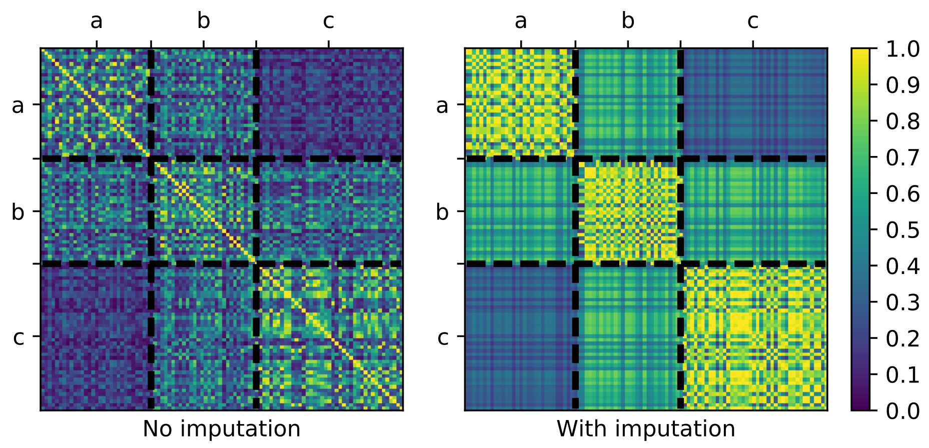


Figure 5.2: Two covariance matrices for the same sample. On the left is the covariance for densely sampled hyperparameters, with inactive values left set to random value, and on the right side the inactive values have been replaced with a default value. The three blocks correspond to three different classifiers (samples are sorted along the value of the classifier choice parameter to highlight differences).

In this example, and throughout this chapter, the categorical parameters are encoded as integers, which has the disadvantage of assuming a continuity in parameter choices or values. A categorical parameter might be better suited by a one-hot encoding, although it is still fundamentally incorrect when used in conjunction with an automatic relevance determination kernel. Furthermore, we will see that there are other issues involved which limit the impact of the choice of representation for these parameters.

5.3.2 Surrogate model optimization

At each iteration t during a model-based optimization procedure, a so-called *acquisition function* is used to choose the next hyperparameter values to evaluate. This acquisition function uses the 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, reach some local optima. The next hyperparameters to evaluate are chosen through a proxy optimization $\gamma_t = \arg\max_\gamma a(\gamma|f(\cdot))$, which should be comparatively much cheaper than evaluating the underlying objective function (i.e., training a classifier on the given dataset). Popular choices of acquisition functions include the Expected Improvement:

$$a_{EI}(\gamma|\mu,\sigma) = \mathbb{E}\left[\max\left(0, f(\gamma_{best} - f(\gamma))\right)\right], \tag{5.6}$$

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 (Hernández-Lobato et al., 2014).

One way to optimize the acquisition function is through random sampling, trying many uniformly sampled hyperparameters then selecting those which result in the maximal value for $a(\cdot)$. 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. (2012) use LBFGS to find the minimum of the acquisition function, while (Bergstra et al., 2011) use the Covariance Matrix Adaptation Evolution Strategy (CMA-ES). These methods are suitable for dense spaces, but the conditionality of the search spaces considered in this chapter 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 (Feurer et al., 2015a) uses a technique called local search, which is similar to coordinate ascent, performing steps on separate coordinates one at a time. The value of the acquisition function for a hyperparameter configuration γ is evaluated and compared with that of its neighbouring configurations, by alternatively taking steps in each direction for each *active parameter*. Given a starting point γ_i, the local search updates the candidate hyperparameter tuple to evaluate for a number of iterations with the following rule:

$$\gamma_{i+1} = \underset{\gamma \in S(\gamma_i)}{\arg\max}\, a(\gamma | f(\cdot)), \tag{5.7}$$

$$S(\gamma_i) = \{\gamma_i\} \cup \{N_j(\gamma_i)\}_{j=1}^{|\Gamma|}, \tag{5.8}$$

where the function N_j returns the neighbours of a configuration according to hyperparameter or coordinate j. The neighbours 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 neighbours are every other value possible, and for integer values they are a step with plus or minus one. The procedure is iterated until a maximum number of iterations is spent, until a maximum has been reached (i.e., the best acquisition value is the left-hand term in Equation 5.8), or until the gain in acquisition per step falls below a threshold value. This procedure is restarted a number of times with different starting points, sampled from valid configurations with inactive hyperparameter imputation.

5.4 Kernels for Conditional Spaces

The main contribution of this chapter 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 is the *conditional kernel*, which 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 (Lakshminarayanan et al., 2014; Balog et al., 2016). 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, bring Bayesian optimization back to state of the art levels on problems with conditions such as combined algorithm and hyperparameter selection.

5.4.1 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 enforced 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:

$$k_C(\gamma, \gamma') = \begin{cases} k(\gamma, \gamma') & \text{if } \gamma_c = \gamma'_c \quad \forall c \in C \\ 0 & \text{otherwise} \end{cases} . \tag{5.9}$$

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 5.1a, where the graph to parse contains two conditions, γ_1 which would always be active, and γ_2 which would only be active when $\gamma_1 = a$. This graph is descended and the descent stops once hyperparameters have no children, and all other conditions not encountered during the parsing of the tree are deemed inactive. 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 using an ARD kernel (Equation 2.15) and 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. For a similar effect with categorical conditional variables encoded with a one-hot encoding, all length scales corresponding to the different categories should be set to a low value. However, just using length scales would not allow the definition of more complex conditions, such as conditions depending on more than one variable, and range conditions.

This kernel has the attractive property of completely isolating otherwise unrelated hyperparameters, 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 straightforward, 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 5.9 is true, then the covariance matrix will be filled with small dense squares of non-zero values with one-valued diagonals (since $k(\mathbf{x}, \mathbf{x}') = 1$). Figure 5.3 shows the values of the covariance matrix evaluated for the same example proposed for Figure 5.2.

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:

$$K_C = \begin{bmatrix} K_{cond_1} & 0 & 0 \\ 0 & K_{cond_2} & 0 \\ 0 & 0 & K_{cond_3} \end{bmatrix}, \tag{5.10}$$

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):

$$K_C^{-1} = \begin{bmatrix} K_{cond_1}^{-1} & 0 & 0 \\ 0 & K_{cond_2}^{-1} & 0 \\ 0 & 0 & K_{cond_3}^{-1} \end{bmatrix}. \tag{5.11}$$

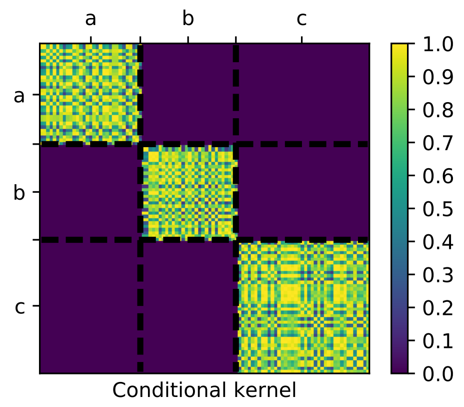
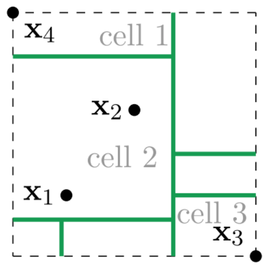


Figure 5.3: Covariance matrix for the same sample and space used defined in Section 5.3. The conditional kernel enforces a null covariance if samples have different choices for activated hyperparameters. The three blocks correspond to three different conditions (samples are sorted along the value of the conditional parameter to highlight differences).

This 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. Furthermore, at any given iteration, given the results of the previous iteration, only one submatrix has had a new point added to it, the other submatrices staying the same. Thus, only the submatrix with the new point needs to be inverted with all other inverts remaining equal. This way, the new complexity can become $\mathcal{O}(m_i^3)$, where i is the index of the subspace with an added point. This process is the same as training a different GP for each conditional subspace and sharing the same noise and amplitude hyperparameters across all distinct GPs, although with the conditional kernel one does not need to maintain this hierarchy of independent GPs. It is also relevant to note that GP hyperparameter optimization procedures such as slice sampling add an additional loop to the algorithmic complexity of the GP.

Finally, an implementation of conditional kernels is available in the hyperparameter optimization toolbox Chocolate[1]. It does not contain the discussed speedup based on piecewise integration of the covariance matrix, but should provide a nice starting point for a practitioner wanting to experiment with conditional spaces.

[1]`https://github.com/novasyst/chocolate`



$\mathbf{x}$	$\phi(\mathbf{x})^T$
$\mathbf{x}_1$	[0 1 0]
$\mathbf{x}_2$	[0 1 0]
$\mathbf{x}_3$	[0 0 1]
$\mathbf{x}_4$	[1 0 0]

Figure 5.4: Example of features generated by a Mondrian process in a two-dimensional space. Left are the splits generated by process, and right is the resulting feature vector.

5.4.2 Laplace kernel

Lastly, we propose the use of the Laplace kernel, which has been shown to have ties with random forests. Balog et al. (2016) have recently proposed the Mondrian kernel, a kernel which proceeds by drawing multiple samples of Mondrian processes (Roy and Teh, 2009), and encoding an explicit feature mapping from them. Balog et al. (2016) describe Mondrian processes as a process with d competing exponential clocks with life time λ, one per dimension. When a clock rings, a random split is generated in the corresponding space (also with random split point). Figure 5.4 shows an example of Mondrian process sample over a 2D space. The Mondrian kernel is then expressed by the co-occurence of points within a same split. Given a sample of Mondrian process i, the corresponding kernel would be:

$$k_{M_i}(\mathbf{x}, \mathbf{x}') = \phi_i(\mathbf{x})^T \phi_i(\mathbf{x}') \tag{5.12}$$

$$= \begin{cases} 1 & \text{if } \mathbf{x}, \mathbf{x}' \text{ in same partition cell} \\ 0 & \text{otherwise,} \end{cases} \tag{5.13}$$

where ϕ_i is the feature map resulting from the Mondrian process sample. Typically, there are M different Mondrian process samples, and the final Mondrian kernel is the average of the dot product of individual mappings:

$$k_M(\mathbf{x}, \mathbf{x}') = \frac{1}{M} \sum_{m=1}^{M} \phi_m(\mathbf{x})^T \phi_m(\mathbf{x}'). \tag{5.14}$$

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:

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|_1}{\ell}\right). \tag{5.15}$$

Mondrian processes and Mondrian forests (Lakshminarayanan et al., 2014) have strong ties to random forests, where Mondrian forests draw partitions of the data as random Mondrian process samples and random forests obtain the partitions 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 (Thornton et al., 2013). 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 (Hutter et al., 2011), and it was observed empirically by Lakshminarayanan et al., 2016 that this leads to overconfidence in the outputs (too low variance). 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 by Balog et al. (2016), 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 – both worthwhile endeavours but not required to evaluate the potency of these approaches for the optimization of hyperparameter spaces with conditions.

Figure 5.5 shows the resulting covariance matrix for the Mondrian[2] 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.

We began with the rather noisy and blurry covariance estimates of the squared exponential kernels with or without imputation of Figure 5.2, and we have now defined

[2]Computed with source code from: https://github.com/matejbalog/mondrian-kernel

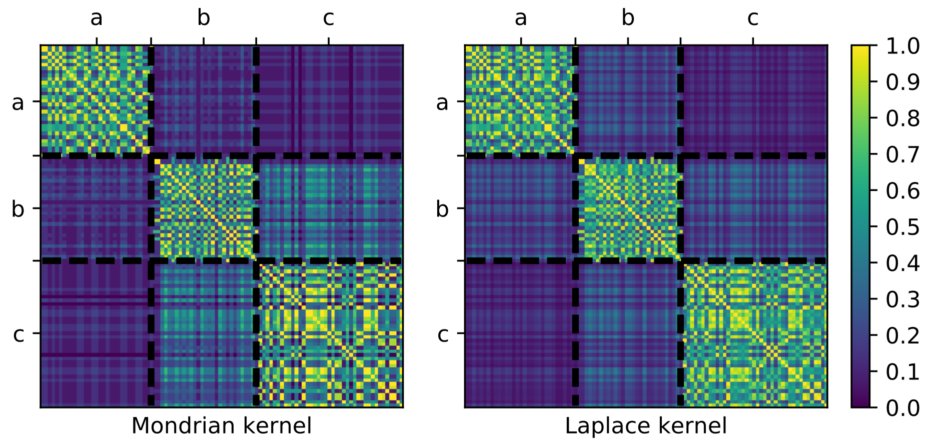


Figure 5.5: Covariance matrix for the same sample and space used defined in Section 5.3. The conditional kernel enforces a null covariance if samples have different choices for activated hyperparameters. The three blocks correspond to three different conditions (samples are sorted along the value of the conditional parameter to highlight differences).

much better suited kernels for conditional spaces. In the next section, these new kernels will be evaluated on a benchmark of problems.

5.5 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[3]. The models and their hyperparameters are listed in Table 5.1.

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 it still does not resolve the fact that information will be shared when none should be. We used the ConfigSpace library provided by the maintainers of SMAC to define the search space along with the relationships between the hyperparameters[4].

[3]Source: `https://www.scikit-learn.org`, also available directly from PyPI.
[4]Source code: `https://github.com/automl/ConfigSpace`

Table 5.1: Algorithms of interest and hyperparameters.

Learning algorithm	**Hyperparameter**	**Space**
k nearest neighbours (knn)	Number of neighbours	Linearly in $\{1, \ldots, 30\}$
Gaussian kernel SVM (svm)	Regularization C	Logarithmically in $[10^{-5}, 10^{5}]$
	Kernel width τ	Logarithmically in $[10^{-5}, 10^{5}]$
Linear SVM (linsvm)	Regularization C	Logarithmically in $[10^{-5}, 10^{5}]$
Decision tree (dt)	Maximal depth	Linearly in $\{1, \ldots, 10\}$
	Min. number of samples per split	Linearly in $\{2, \ldots, 100\}$
	Min. number of samples per leaf	Linearly in $\{2, \ldots, 100\}$
Random forest (rf)	Number of trees	Linearly in $\{1, \ldots, 30\}$
	Maximal depth	Linearly in $\{1, \ldots, 10\}$
	Min. number of samples per split	Linearly in $\{2, \ldots, 100\}$
	Min. number of samples per leaf	Linearly in $\{2, \ldots, 100\}$
AdaBoost (adab)	Number of weak learners	Linearly in $\{1, \ldots, 30\}$
Gaussian naive Bayes (gnb)	No parameter.	
Linear discriminant analysis (lda)	No parameter.	
Quadratic discriminant analysis (qda)	Regularization constant	Logarithmically in $[10^{-3}, 10^{3}]$

The acquisition function used for methods requiring it (i.e., all methods 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 by Snoek et al. (2012), 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[5]. 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 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 by Hansen (2005). The DEAP toolbox is used[6]. No imputation of inactive parameters is performed.

Bayesian optimization with a GP and without imputation (GP-matern-noimpute) : This is similar to Spearmint, but the acquisition function is optimized through random sampling instead of LBFGS. 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 5.9, 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;

[5]Source code: `https://github.com/automl/SMAC3`
[6]DEAP: `https://github.com/DEAP`

- 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 (Murray and Adams, 2010; Snoek et al., 2012). 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[7]. 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 new data splits are sampled for each repetition, including a test with 20% of the data if it was not provided in the original dataset. The same splits are used across all methods for the same repetition.

5.5.1 Results and discussion

Table 5.2 shows the empirical error rate of each method on the test set, averaged over the 10 repetitions. The best performance per dataset is highlighted in bold, and for

[7]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.

Table 5.2: 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	$\underline{18.39}$	14.20	$\underline{46.61}$	46.17	11.85	14.44	10.70	1.45	$\underline{20.73}$	3.14	$\underline{12.76}$	$\underline{0.64}$	$\underline{3.34}$	23.83	23.25	$\underline{3.77}$	24.09	$\underline{0.09}$	$\underline{6.07}$	$\underline{5.67}$	$\underline{1.10}$	$\underline{24.06}$	$\underline{37.30}$	7.08
Spearmint	0.22	$\underline{33.02}$	20.32	44.63	46.37	12.05	$\underline{15.53}$	$\underline{10.83}$	$\underline{6.16}$	$\underline{31.77}$	$\underline{8.81}$	13.61	$\underline{2.66}$	$\underline{3.17}$	23.44	23.10	$\underline{4.52}$	$\underline{27.42}$	$\underline{0.86}$	$\underline{9.47}$	$\underline{6.86}$	2.09	21.50	$\underline{35.62}$	9.15
SMAC	0.09	13.62	14.22	44.63	46.27	$\underline{12.60}$	14.42	10.59	0.92	$\underline{17.40}$	2.83	$\underline{12.66}$	**0.26**	$\underline{3.48}$	**23.38**	23.45	$\underline{3.31}$	23.96	$\underline{0.08}$	**4.40**	$\underline{5.84}$	$\underline{1.12}$	$\underline{22.70}$	$\underline{36.61}$	5.52
CMA	0.09	$\underline{18.02}$	14.26	44.66	46.67	11.88	$\underline{14.48}$	10.64	$\underline{2.63}$	19.08	3.04	$\underline{12.66}$	$\underline{1.85}$	$\underline{3.09}$	23.96	23.75	$\underline{3.66}$	$\underline{25.06}$	$\underline{0.07}$	$\underline{6.10}$	$\underline{5.69}$	1.09	$\underline{23.74}$	$\underline{35.56}$	6.94
GP-matern-noimpute	0.11	$\underline{25.11}$	17.52	44.63	46.82	11.61	$\underline{14.50}$	10.61	$\underline{4.65}$	$\underline{26.45}$	$\underline{3.87}$	12.40	$\underline{2.38}$	$\underline{3.05}$	23.51	23.50	$\underline{3.42}$	25.91	$\underline{0.08}$	$\underline{8.98}$	$\underline{5.87}$	$\underline{1.61}$	$\underline{23.79}$	33.23	7.75
GP-matern	0.09	12.68	**14.10**	45.32	48.36	**11.20**	14.36	$\underline{10.72}$	0.75	18.29	3.16	$\underline{12.53}$	0.27	2.55	23.70	23.35	3.51	23.53	$\underline{0.05}$	4.98	$\underline{5.27}$	$\underline{1.11}$	21.36	33.54	4.83
GP-cond	0.11	18.72	15.48	44.91	47.21	$\underline{15.92}$	**14.33**	10.51	**0.72**	18.75	3.47	$\underline{12.44}$	0.28	2.50	24.29	23.75	2.84	$\underline{24.04}$	0.04	4.89	$\underline{5.28}$	1.10	$\underline{21.77}$	32.76	5.48
GP-laplace	0.13	$\underline{21.60}$	17.08	44.89	**45.97**	12.36	14.39	10.51	$\underline{2.34}$	23.25	$\underline{8.18}$	$\underline{13.03}$	$\underline{2.11}$	$\underline{2.72}$	24.68	23.45	$\underline{3.27}$	24.14	$\underline{0.09}$	$\underline{7.43}$	$\underline{5.28}$	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	$\underline{12.49}$	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	$\underline{17.87}$	14.68	44.89	47.06	11.82	14.38	**10.20**	1.39	$\underline{19.56}$	4.19	$\underline{12.72}$	$\underline{1.73}$	**2.49**	23.51	23.30	3.23	24.27	0.04	$\underline{7.59}$	$\underline{5.17}$	$\underline{1.35}$	**20.45**	32.98	5.48

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 5.2, 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 ($p = 9\text{e}{-12}$). Figure 5.6 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 5.3 shows the result of pairwise Wilcoxon signed-rank tests. Methods highlighted

Table 5.3: 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**)	–

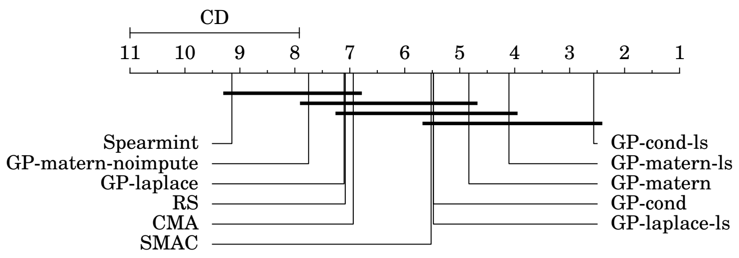


Figure 5.6: Nemenyi post-hoc test.

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.

Figure 5.7 shows the rank for some of the methods evaluated, averaged over datasets, with regards to the number of iterations of the optimization procedure. The left subfigure shows the validation performance and the left figure shows the testing performance. Ordering of the methods is roughly preserved between validation and testing, although gp-matern-ls seemed closer to the best method gp-cond-ls in validation. SMAC, using random forests, seemed to get worse as iterations progressed, perhaps it got stuck in some local optima of the hyperparameter space. Interestingly random search did not show signs of improvement as the iteration budget increased.

5.5.2 Visualization of classifier choices

Another angle providing insight into the behaviour of combined algorithm selection and hyperparameter optimization is to look at the final classifier choice, once the

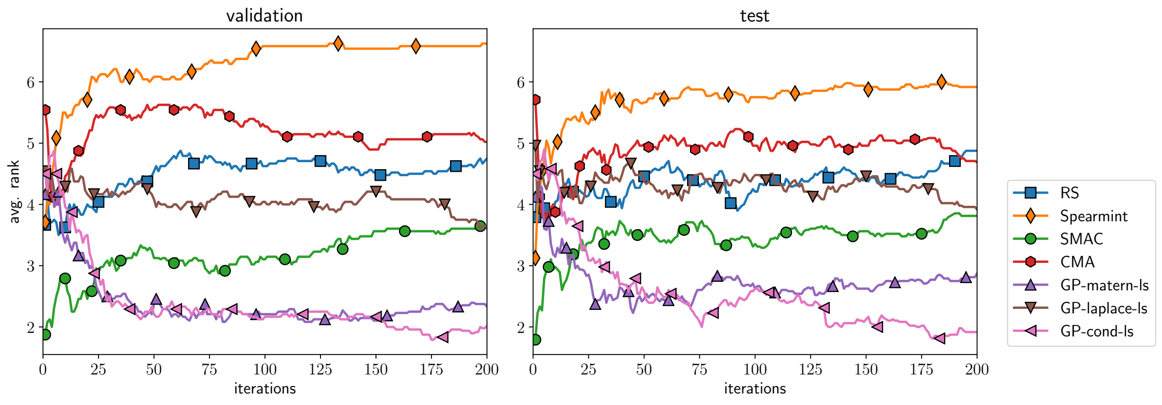


Figure 5.7: Ranks per iteration of the process, averaged over all datasets. The validation performance is shown on the left and the testing performance is shown on the right.

optimization procedure is over. Here we look at the results for a single optimization method for simplicity, GP-cond-ls. Figure 5.8 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 5.9 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 titanic (tita) 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.

5.6 Conclusion

In this chapter we proposed two kernels for the optimization of hyperparameter spaces with conditional variables, such as combined algorithm selection and hyperparameter optimization. Research questions of this work were:

1. What is the impact or importance of inactive hyperparameter value imputation for spaces with a conditional structure?
2. Can the Bayesian optimization of hyperparameters in a space with conditions be improved by the use of a kernel forcing zero covariance between samples from different conditions?

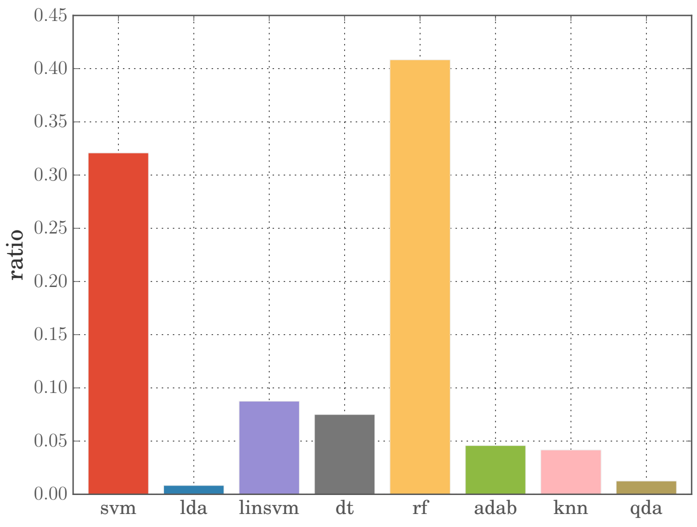


Figure 5.8: Resulting classifier choices for an optimization with GP-cond-ls over all 24 datasets and 10 repetitions.

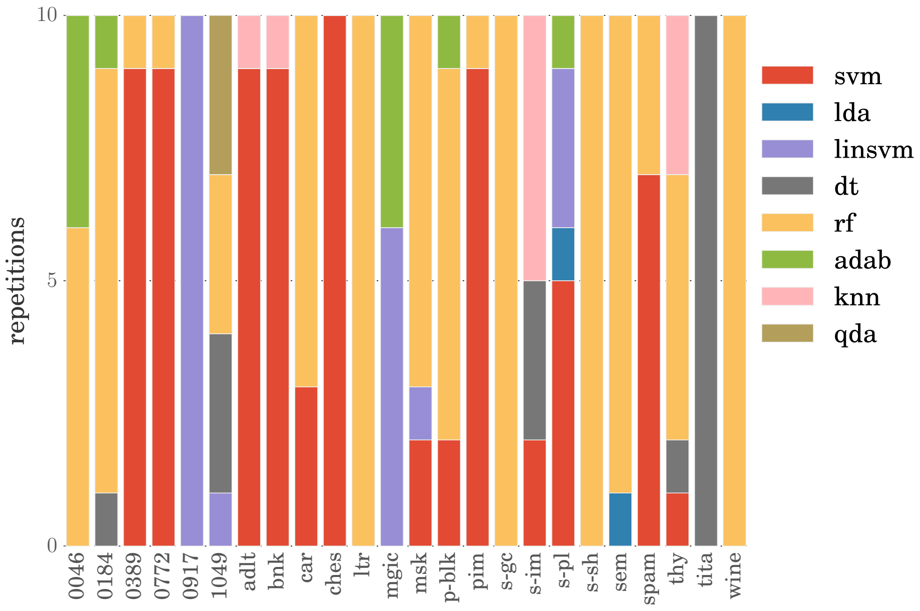


Figure 5.9: Resulting classifier choices for an optimization with GP-cond-ls. Each column represents the distribution of classifier choices across the 10 repetitions on a single dataset. Figure best viewed in colour.

Question (1) pertained to the importance of proper imputation of values for inactive hyperparameters in order to obtain valid covariance estimates, rather than noise. The answer to question (1) is that imputation has a rather high impact on hyperparameter optimization as shown in Section 5.5 by the comparison methods with and without imputation. Question (2) is answered positively, since the conditional kernel performs better than other kernels for the optimization of a space with conditional structure. Overall, 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.

Chapter 6

Conclusion

In this thesis, we proposed new Bayesian optimization applications and provided insight on difficult problems in Bayesian optimization such as the optimization of conditional spaces and overfitting. The contributions of the thesis support the rapidly evolving field of Bayesian optimization for hyperparameter tuning. Even though the contributions are all directly related to hyperparameter optimization, they have potential for impact in different lines of work, with contributions in ensemble methods and Automatic Machine Learning (AutoML). In this chapter, we present a brief overview of each objective of the thesis (as presented in Chapter 1), state how it was met, present contributions resulting from each objective along with a short discussion of each work's limitations. The objectives are somewhat aligned with the main chapters of the thesis (Chapters $3 - 5$).

The first objective mentioned in the introduction is to 1) **assess the impact of overfitting in hyperparameter optimization (also known in this setting as oversearching) and propose strategies to reduce it**, an objective that is met by the work presented in Chapter 3. The impact of overfitting in hyperparameter optimization is shown by the decrease of generalization error when adding extra validation folds. The proposed methodology for *resampling of the training and validation splits* shows a promising direction in which to go in order to reduce this overfitting, one of the contributions of this thesis. Another contribution of the chapter is the demonstration that *the posterior mean of the GP should be favoured as a way to select the final model instead of an* $\arg\min$ *selection.* This contribution is logical, since the argmin selection ignores the noise in sampling the validation splits and training the model, and is the same as picking the observation with the lowest error and highest noise value. Furthermore, the combination of these two contributions also result in an even stronger reduction of overfitting, which we believe is due to the better estimation of the noise component for the probabilistic regression enabled by resampling. These contributions have a high potential for impact in the machine learning community, given the simplicity of their application and the conceivable benefits. Improvements can be expected for all sizes of datasets present in the studied benchmark, including the largest ones. Although they are still small in comparison to some popular large scale datasets, these dataset sizes are encountered in many real world problems, and thus can benefit from the proposed approaches. One downside of the research conducted for this objective is the restriction in the experiments to a single search space for SVM hyperparameters. This model choice helps to reduce the number of variables in the study, since SVMs are not sensitive to initialization. We expect the conclusions of the work to extend straightforwardly to models with similar properties. The proposed method should also be applicable for non-deterministic models and models sensitive to initialization by adding an extra model selection layer once the final hyperparameters are selected. Finally, another way to apply our contributions to models sensitive to initialization would be to perform random seed optimization.

The second objective of this thesis is to 2) **propose an application of hyperparameter optimization to generate heterogeneous ensembles of classifiers**. This objective is met with the work presented in Chapter 4. We propose a modification of the Bayesian hyperparameter optimization loop to *directly optimize for the performance of an ensemble instead of a single model*. The approach is shown to be performing better than a single model for many different types of model, constituting the main contribution of the chapter. The contribution is an important one because of the little overhead at which this ensemble construction method comes for, which is a single iteration of forward greedy selection for an ensemble combined with majority votes. The method was shown to perform better on many different datasets, and for different base classifiers, with three search spaces: one for SVMs with different kernels, one for base scikit-learn models, and one for convolutional neural networks (cuda-convnets). This illustrates the versatility of the proposed approach. Interestingly, the issue of overfitting is very much present in this work, perhaps more so than for a single model as studied in Chapter 3. Because of this, one should exert caution when applying this method – a cross-validation strategy is recommended. There is also room for improvement upon the suggested method, one of many ways being the mechanism for constructing and updating the ensemble. In this regard, the source code of this work is available online, hopefully this will help other researchers build up upon this work and improve it. The field is evolving fast, and recent work has already shown potential benefits by building two-layer Bayesian optimization techniques, where the first layer identifies promising base classifiers, and the second layer optimizes the hyperparameters of a stacking model (Wistuba et al., 2017).

Lastly, the third objective of this thesis is to 3) **propose methods for the optimization of hyperparameters with conditionality structures**, with specific applications in the AutoML field. This objective is met by the work presented in Chapter 5, which studied the use of methods such as imputation and proposed new kernels for GP regression. The *conditional kernel* is shown to have a significant impact on the performance of a Bayesian optimization in a space with conditions, which constitutes the main contribution of the chapter. Another minor contribution of this work is to highlight the importance of proper handling of the conditional structure, something which was not made explicit in the literature. The work is also open sourced in the modular Bayesian optimization toolbox Chocolate, although the implementation of the conditional kernel is straightforward in itself – the key being to model the structure of the search space. The contributions presented in Chapter 5 have a high potential of impact for AutoML applications, where the number of conditions in a typical search space is usually high. For search spaces with both a very large number of dimensions and a highly conditional structure, the covariance matrix typically becomes extremely sparse, which can give rise to other issues. The use of GPs in such search spaces remains a hard problem, perhaps bandit-based approaches are better suited (e.g., Hyperband; Li et al., 2017). Moreover, note that one should not expect to achieve good results in optimizing a search space with dimensionality d with n observations when $n << d$ or even $n \sim d$.

Finally, although the work presented in this thesis is mainly based on Bayesian optimization with GPs, apart from the contributions of Chapter 5, the contributions are easily extensible to other surrogate models provided they allow variance or uncertainty estimation over the modelled function. Acquisition functions are also subject to the same treatment. Our work is not applicable to methods which do not preserve the state of their optimization, meaning the function observations (cross-validation error rates) and their corresponding hyperparameters. Those are key elements of all contributions of this thesis, except for the demonstration in Chapter 3 that CMA-ES can benefit from a reshuffling of the training and validation splits, without ties to Bayesian optimization.

Before concluding, let us briefly touch on avenues for future work based on the contributions of this thesis. On the topic of overfitting introduced in Chapter 3, directions for future work include the application of our methodology to stochastic or non-deterministic learning algorithms. Another interesting direction for future work would be derivation of theoretic bounds for the variance of validation split resampling. Concerning the work introduced in Chapter 4, promising directions for future work include dynamically determining the ensemble size m, as well as a procedure for automatically choosing the classifier to optimize at each iteration of the Bayesian optimization loop. The choice of the combination rule was also maintained fixed through the work, with a simple majority vote, and there could be better alternatives, although recent works have already begun touching on this matter (Wistuba et al., 2017). Finally, in regards to the work presented in Chapter 5, many directions offer opportunities for future contributions. First, although the conditional kernel is performing well on the studied benchmarks, high dimensional spaces still remain a challenge, possibly due to the very little amount of information shared between observations in a space with a very sparse covariance matrix. Other variants could be considered, with partial sharing of information rather than hard cutting of the space. Some recent work on this matter considers the learning of subpartitions (Wang et al., 2018), rather than injecting prior knowledge, although both could be combined.

In closing, this thesis presents multiple contributions centered around Bayesian optimization of hyperparameters. It is expected that these contributions will help push the field further towards truly automatic model selection and hyperparameter optimization, or simply AutoML. Most of the contributions in the thesis are published in proceedings (this is true for parts of Chapter 4 and Chapter 5), with source code available online in some form, hopefully helping fellow researchers building upon this work. The contributions achieved by the thesis are varied and complementary, touching on overfitting, ensemble methods and AutoML. On the pertinence of each area of focus in the thesis, we can end with the following notes:

- Dealing with *overfitting* is vital to ensure that the models and hyperparameters selected properly generalize in operational settings, provided i.i.d. assumptions are met.
- The application of *ensemble methods* is helpful for settings in which it is important to drive the performance of models to their maximum, in order to squeeze a few extra percentage points of accuracy. Arguably, this should always be the case, especially when the cost is so low.

- Finally, the conditional kernel can help perform better on AutoML problems, which should only become more ubiquitous as computing resources available to practitioners keep on growing – the idea being that it is better to let a computer do the work of selecting the model than to require the time of a highly qualified worker.

The ultimate endeavour of this work, and others in the same field, is to *provide efficient tools that can automatically select the best model and the right hyperparameters for any given problem represented as a dataset.* These tools are suitable for a humbling quantity of applications, where the challenge of the practitioner becomes the definition of a satisfactory objective function – if all else fails, human preference can become the objective function (Christiano et al., 2017; Durand and Gagné, 2017). With each published paper or thesis, we are inching closer to the concerted objective of a complete framework for the Bayesian optimization of hyperparameters and models.

Appendix A

List of publications

The thesis lead to the following peer reviewed publications.

A.1 Conferences

- J.-C. Lévesque, A. Durand, C. Gagné, and R. Sabourin (2017). "Bayesian Optimization for Conditional Hyperparameter Spaces". In: *Proceedings of the 2017 International Joint Conference on Neural Networks*, pp. 286–293. DOI: 10.1109/IJCNN.2017.7965867.
- J.-C. Lévesque, C. Gagné, and R. Sabourin (2016). "Bayesian Hyperparameter Optimization for Ensemble Learning". In: *Proceedings of the 32nd Conference on Uncertainty in Artificial Intelligence*, pp. 437–446.
- J.-C. Lévesque, A. Durand, C. Gagné, and R. Sabourin (2012). "Multi-Objective Evolutionary Optimization for Generating Ensembles of Classifiers in the ROC Space". In: *Proceedings of the 14th Conference on Genetic and Evolutionary Computation*, pp. 879–886.

A.2 Workshops

- J.-C. Lévesque, C. Gagné, and R. Sabourin (2013). "Ensembles of Budgeted Kernel Support Vector Machines for Parallel Large Scale Learning". In: *NIPS 2013 Workshop on Big Learning: Advances in Algorithms and Data Management*, p. 5.

A.3 Journal papers in preparation

- J.C. Lévesque, C. Gagné, and R. Sabourin (2017). “Strategies for Evaluating Generalization Performance in Hyperparameter Optimization”.

Appendix B

Datasets and Results of Chapter 2

Table B.1: Generalization error for all datasets, averaged over repetitions (standard deviation in parenthesis). Hold-out validation.

Dataset/Method	grid	rs	gp	gp-sel	gp-r	gp-sel-r	cma	cma-pv	cma-bw	cma-gv	cma-r	cma-pv-r	cma-bw-r	cma-gv-r
abalone	34.0(0.7)	34.0(0.9)	34.3(1.1)	34.0(0.9)	34.1(1.2)	34.2(1.4)	34.2(1.0)	34.3(1.2)	34.3(1.0)	34.2(0.9)	34.1(0.8)	34.1(0.9)	34.3(1.1)	34.1(0.9)
acute-inflammation	0.6(2.6)	0.2(1.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.2(1.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)
acute-nephritis	0.4(1.2)	0.2(1.1)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.2(1.1)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)
adult	15.4(0.2)	15.4(0.2)	15.4(0.2)	15.4(0.1)	15.4(0.1)	15.5(0.2)	15.4(0.2)	15.5(0.1)	15.4(0.1)	15.5(0.2)	15.5(0.2)	15.4(0.1)	15.5(0.1)	15.4(0.1)
annealing	24.1(0.5)	24.2(0.7)	25.0(3.1)	24.1(0.2)	24.6(1.4)	24.1(0.2)	25.6(4.7)	25.5(4.6)	25.3(1.8)	24.2(0.9)	24.4(1.0)	24.1(0.2)	25.5(1.7)	24.4(0.9)
arrhythmia	33.0(2.3)	33.0(3.3)	32.5(3.0)	32.9(2.8)	33.4(2.2)	33.7(2.1)	32.6(3.0)	32.3(2.5)	33.0(2.6)	33.2(3.4)	33.4(2.6)	33.8(2.9)	33.4(3.1)	33.2(2.6)
audiology-std	34.8(2.9)	31.8(3.9)	33.8(8.3)	32.4(3.8)	35.2(2.4)	33.8(3.2)	32.2(3.9)	34.0(3.7)	34.4(3.2)	32.0(4.0)	35.8(0.9)	33.8(3.2)	35.6(1.2)	34.2(3.2)
balance-scale	2.0(1.7)	1.7(1.6)	1.9(2.0)	1.8(1.9)	0.8(0.9)	0.9(1.0)	2.3(2.2)	3.1(4.5)	2.4(3.9)	2.6(3.1)	1.8(2.5)	2.0(2.4)	1.6(1.6)	1.3(1.5)
bank	10.6(0.5)	10.6(0.5)	11.0(0.5)	11.0(0.7)	10.9(0.6)	10.8(0.6)	10.8(0.5)	11.0(0.6)	10.8(0.6)	10.5(0.6)	10.7(0.4)	11.3(0.9)	11.2(0.8)	10.9(0.8)
blood	22.4(1.0)	22.2(1.5)	22.6(1.3)	21.9(1.6)	22.5(2.1)	22.1(2.8)	23.2(2.2)	23.9(1.6)	23.7(1.4)	22.9(2.0)	22.7(1.8)	23.6(0.7)	23.5(1.4)	23.0(2.1)
breast-cancer	30.2(3.3)	30.5(3.9)	30.7(2.3)	29.3(4.7)	28.8(3.2)	30.2(4.3)	30.3(3.1)	28.8(2.0)	29.5(2.8)	30.7(4.7)	30.1(3.8)	29.2(1.8)	29.4(2.4)	30.7(4.1)
breast-cancer-wisc	4.1(1.3)	4.2(1.0)	3.7(1.1)	3.9(1.2)	3.6(1.1)	4.2(1.0)	4.0(1.0)	3.7(1.0)	3.9(1.5)	4.1(1.0)	3.5(1.2)	3.8(1.3)	3.9(0.9)	4.1(1.2)
breast-cancer-wisc-diag	3.9(2.4)	4.2(2.3)	3.6(1.1)	4.0(2.1)	3.5(0.9)	3.3(0.7)	3.6(1.2)	3.2(0.9)	3.3(0.9)	3.5(1.3)	3.4(1.1)	3.4(1.1)	3.6(1.0)	3.2(1.0)
breast-cancer-wisc-prog	22.7(4.0)	21.3(3.5)	22.3(3.2)	22.3(1.8)	22.1(3.3)	23.5(2.8)	22.1(4.1)	22.6(3.0)	22.2(3.3)	22.3(3.3)	21.8(3.9)	23.4(0.8)	22.9(2.2)	23.6(3.7)
breast-tissue	32.2(6.2)	32.5(6.5)	34.5(6.8)	33.9(7.4)	31.5(5.5)	32.8(6.7)	31.5(6.2)	34.1(6.1)	36.6(11.0)	35.0(6.2)	32.9(4.3)	33.8(4.5)	34.6(4.6)	32.8(5.4)
car	1.7(0.7)	1.6(0.7)	1.7(0.8)	1.7(0.8)	1.6(0.9)	1.5(0.7)	1.5(0.7)	1.7(0.8)	1.8(0.9)	1.7(0.9)	1.7(0.9)	1.6(0.9)	1.6(0.9)	1.8(1.0)
cardiotocography-10clas..	17.6(1.6)	17.0(1.1)	17.8(2.2)	17.0(1.9)	17.2(1.3)	17.3(1.9)	17.7(1.8)	17.7(1.3)	17.5(1.4)	17.5(1.4)	17.6(1.5)	17.7(1.5)	17.5(1.8)	17.8(1.8)
cardiotocography-3class..	8.3(1.2)	8.3(1.2)	8.5(1.6)	8.2(1.1)	8.3(0.6)	8.2(0.5)	8.4(0.9)	8.9(1.0)	8.8(1.1)	8.8(0.8)	8.7(0.9)	8.9(1.0)	9.0(1.2)	8.7(0.9)
chess-krvk	40.5(0.7)	40.2(0.8)	40.0(0.7)	39.7(0.6)	39.9(0.6)	39.9(0.7)	40.3(0.6)	40.2(0.7)	39.9(0.7)	39.8(0.6)	40.3(0.7)	40.3(0.6)	40.2(0.6)	40.3(0.5)
chess-krvkp	1.0(0.3)	0.9(0.3)	1.0(0.4)	1.0(0.4)	0.9(0.3)	0.9(0.3)	1.0(0.3)	1.0(0.4)	1.0(0.4)	1.1(0.4)	1.0(0.4)	1.0(0.4)	1.0(0.3)	1.0(0.4)
congressional-voting	39.6(2.0)	39.1(1.3)	39.4(1.4)	39.8(1.8)	39.8(2.8)	39.0(2.2)	39.5(1.4)	40.2(2.1)	39.3(1.6)	39.6(2.1)	39.1(1.7)	39.9(2.6)	39.4(1.6)	39.4(2.1)
conn-bench-sonar-mines-..	15.4(5.6)	17.7(6.4)	19.1(7.3)	18.7(6.1)	13.9(4.2)	17.4(6.3)	18.0(6.1)	17.8(6.3)	21.0(7.1)	19.0(5.8)	15.2(5.4)	17.4(6.2)	19.6(6.9)	18.3(6.6)
conn-bench-vowel-deterd..	0.0(0.1)	0.0(0.0)	0.0(0.1)	0.1(0.1)	0.0(0.1)	0.0(0.1)	0.0(0.0)	0.0(0.2)	0.1(0.3)	0.1(0.2)	0.0(0.1)	0.0(0.2)	0.1(0.2)	0.1(0.2)
connect-4	18.4(0.4)	18.7(0.6)	18.7(1.4)	18.3(0.3)	18.4(0.3)	19.2(2.2)	18.5(0.4)	19.8(2.5)	19.2(1.9)	19.2(1.9)	19.0(1.4)	18.5(0.6)	18.3(0.3)	18.3(0.3)
contrac	46.7(3.0)	46.4(2.6)	45.7(3.1)	46.1(2.4)	45.3(2.2)	45.9(2.2)	45.3(1.8)	46.3(2.6)	46.4(1.8)	46.0(2.4)	46.6(2.2)	45.8(2.1)	46.5(2.0)	45.9(3.0)
credit-approval	15.3(1.7)	15.0(2.0)	14.0(2.0)	13.6(1.9)	13.7(1.8)	14.3(1.7)	14.5(2.1)	13.6(2.1)	13.7(1.9)	13.8(2.1)	13.7(1.9)	13.8(1.8)	13.8(1.8)	13.7(1.8)
cylinder-bands	24.6(2.6)	24.8(1.9)	24.2(2.3)	24.6(2.9)	25.6(2.5)	24.9(3.4)	24.6(1.9)	26.3(3.4)	26.2(2.9)	25.5(2.6)	25.3(2.4)	25.3(2.7)	25.5(3.3)	24.2(2.5)
dermatology	3.4(1.6)	3.3(1.2)	3.0(1.4)	3.0(1.3)	3.0(1.2)	3.1(1.4)	3.5(1.8)	3.1(1.2)	3.0(1.1)	3.2(1.3)	3.0(1.5)	3.0(1.5)	3.1(1.2)	3.2(1.6)
echocardiogram	19.1(7.8)	21.3(8.1)	18.7(6.5)	18.2(7.5)	17.7(6.1)	20.6(7.8)	19.7(6.6)	17.9(4.9)	17.1(4.4)	17.7(5.5)	18.8(6.5)	15.4(2.5)	17.2(5.3)	19.8(7.9)
ecoli	15.1(2.7)	15.4(3.0)	15.9(3.6)	15.2(2.6)	14.6(3.2)	15.0(2.6)	15.7(3.1)	14.7(2.8)	14.7(2.9)	15.9(3.1)	15.0(2.8)	14.6(2.9)	14.7(2.6)	16.0(4.2)
energy-y1	5.9(1.4)	5.6(1.7)	5.4(1.8)	5.3(1.5)	5.0(1.7)	5.6(1.6)	5.6(1.8)	7.2(10.6)	6.7(4.3)	6.4(2.5)	5.5(1.7)	5.3(2.0)	5.6(1.8)	5.2(1.9)
energy-y2	8.9(1.2)	8.1(1.6)	8.6(1.5)	8.5(1.4)	8.7(1.1)	8.3(1.3)	8.3(1.3)	9.0(1.7)	9.2(1.5)	8.4(1.2)	8.7(1.4)	8.7(1.6)	9.0(2.1)	8.4(1.5)
fertility	14.6(0.9)	16.1(3.5)	14.1(0.6)	14.3(0.0)	15.0(2.0)	14.3(0.0)	14.9(1.9)	14.6(0.9)	14.6(1.2)	14.6(1.2)	14.6(0.9)	14.7(1.4)	15.4(3.1)	15.4(3.2)
flags	53.7(4.5)	53.1(5.8)	54.2(6.5)	53.9(5.4)	55.6(6.1)	54.6(5.7)	53.7(4.0)	56.8(7.0)	57.3(5.4)	55.9(5.2)	56.1(4.1)	54.7(5.6)	55.4(5.9)	54.4(5.0)
glass	36.1(5.8)	35.0(5.7)	33.6(4.8)	34.4(5.5)	32.4(4.9)	34.6(5.1)	35.4(6.0)	32.6(6.0)	35.1(5.4)	33.8(6.1)	34.9(6.1)	33.2(5.1)	37.1(5.7)	35.5(6.2)
haberman-survival	28.0(3.0)	27.8(2.4)	27.2(1.6)	28.0(3.0)	27.5(2.1)	28.7(3.2)	27.5(1.9)	27.6(2.4)	27.3(1.7)	28.2(2.3)	28.2(3.4)	27.6(1.9)	27.2(2.0)	28.1(2.4)
hayes-roth	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)
heart-cleveland	43.7(2.9)	44.3(3.1)	43.0(3.5)	44.4(3.0)	44.5(2.8)	44.4(3.4)	43.4(2.8)	42.7(3.3)	43.2(2.9)	43.6(2.5)	43.2(2.4)	44.4(2.6)	43.9(2.2)	43.6(2.7)
heart-hungarian	19.5(2.9)	19.3(2.8)	19.3(2.9)	20.0(3.9)	17.8(2.6)	19.8(3.0)	19.3(2.9)	18.1(2.6)	19.3(2.8)	18.8(3.2)	18.8(2.7)	18.0(2.7)	19.5(3.9)	20.0(3.5)
heart-switzerland	62.6(5.6)	60.1(5.7)	63.6(4.7)	63.2(4.5)	61.2(3.9)	62.2(5.9)	63.9(5.8)	61.6(4.2)	61.0(3.9)	63.4(5.3)	63.8(6.9)	61.8(4.2)	61.6(4.2)	62.0(5.2)
heart-va	68.7(4.5)	69.8(4.2)	70.3(2.9)	68.7(3.8)	69.8(4.6)	69.0(4.2)	69.5(5.0)	70.8(2.9)	69.6(4.0)	68.6(4.3)	68.6(5.4)	69.9(3.8)	68.9(4.2)	69.6(4.6)
hepatitis	18.6(3.1)	19.0(2.8)	19.9(2.5)	19.2(3.8)	19.4(2.9)	19.8(2.9)	18.7(2.9)	19.5(3.5)	19.2(3.3)	18.2(3.0)	19.2(3.0)	19.0(3.3)	19.7(2.6)	18.8(3.6)
hill-valley	48.9(1.7)	49.5(1.3)	48.7(1.8)	49.0(1.5)	49.8(1.1)	49.5(1.4)	49.4(1.4)	48.9(1.9)	48.7(1.8)	48.7(1.6)	49.6(1.4)	49.3(1.6)	50.1(0.7)	49.8(1.2)
horse-colic	37.8(2.7)	38.5(2.1)	38.0(2.6)	38.5(2.4)	36.9(3.5)	37.6(2.8)	37.5(3.2)	37.2(2.7)	36.2(2.8)	37.4(3.1)	36.2(3.1)	36.7(2.8)	35.3(2.4)	35.6(3.5)
ilpd-indian-liver	30.7(2.5)	31.1(2.7)	29.9(2.1)	30.0(2.2)	30.7(3.3)	29.7(2.3)	30.0(2.0)	29.2(2.0)	28.5(0.6)	29.8(1.9)	29.8(2.8)	28.6(0.6)	29.4(1.9)	30.5(2.7)
image-segmentation	84.1(4.4)	84.5(4.2)	77.9(9.5)	77.2(9.9)	78.7(8.7)	82.2(6.5)	79.4(9.7)	76.8(9.9)	70.4(8.9)	81.1(8.0)	78.1(9.4)	79.3(8.9)	72.3(9.7)	77.8(10.6)
ionosphere	7.4(2.2)	9.0(3.3)	7.4(2.4)	8.3(2.7)	7.2(3.0)	8.6(3.3)	7.5(2.8)	7.8(1.9)	8.6(3.2)	8.0(3.0)	8.1(2.3)	7.9(2.5)	8.7(3.5)	8.5(3.0)
iris	6.8(6.2)	5.8(6.3)	7.0(7.1)	7.3(7.2)	6.3(6.2)	7.6(7.0)	5.4(6.2)	4.2(3.3)	4.4(3.0)	5.6(6.3)	4.5(4.0)	4.7(4.9)	7.2(7.1)	8.7(7.4)

Dataset/Method	grid	rs	gp	gp-sel	gp-r	gp-sel-r	cma	cma-pv	cma-bw	cma-gv	cma-r	cma-pv-r	cma-bw-r	cma-gv-r
led-display	28.4(2.2)	28.4(2.5)	28.7(2.3)	29.3(2.4)	28.4(2.2)	28.3(2.1)	28.3(2.1)	29.1(2.1)	29.0(2.4)	28.9(2.5)	28.3(1.9)	28.5(2.3)	28.5(2.3)	27.9(2.2)
letter	6.2(0.3)	5.9(0.3)	5.8(0.4)	5.8(0.4)	5.8(0.4)	5.8(0.4)	5.8(0.4)	5.8(0.4)	5.8(0.4)	5.8(0.4)	5.8(0.4)	5.8(0.4)	5.8(0.4)	5.7(0.4)
libras	17.1(3.7)	17.3(4.3)	17.9(3.6)	17.3(4.1)	16.0(2.3)	18.0(3.6)	17.2(3.4)	17.0(3.8)	18.7(4.4)	17.5(4.2)	17.7(4.3)	16.9(3.5)	17.8(4.9)	16.8(4.2)
low-res-spect	10.3(2.2)	11.5(2.1)	10.3(2.1)	10.4(1.8)	10.9(1.9)	10.9(1.9)	10.9(1.8)	10.6(2.2)	11.1(2.1)	10.4(2.0)	10.7(2.1)	10.8(1.9)	10.6(1.9)	10.6(1.9)
lung-cancer	56.5(8.9)	57.7(6.7)	58.1(5.1)	60.8(4.1)	56.5(10.1)	59.2(5.5)	58.1(5.7)	60.8(2.3)	58.8(8.9)	60.8(7.7)	54.2(10.5)	58.1(7.5)	58.5(6.2)	60.0(5.2)
lymphography	19.0(4.9)	17.9(4.7)	18.5(6.1)	17.0(5.9)	17.2(4.8)	17.6(5.7)	17.7(6.0)	17.4(4.2)	18.3(6.0)	18.0(5.9)	17.4(4.9)	18.7(6.1)	18.3(4.7)	19.4(5.9)
magic	13.3(0.4)	13.3(0.3)	13.3(0.3)	13.3(0.3)	13.3(0.3)	13.3(0.3)	13.3(0.3)	13.4(0.3)	13.4(0.3)	13.4(0.3)	13.2(0.4)	13.5(0.4)	13.4(0.3)	13.4(0.4)
mammographic	17.6(2.2)	17.4(1.9)	17.1(1.8)	17.4(1.9)	18.0(2.0)	17.3(1.9)	17.4(2.1)	17.3(2.2)	17.8(2.4)	17.0(2.0)	17.9(2.3)	17.3(1.9)	17.7(2.0)	17.1(2.2)
miniboone	8.2(0.2)	8.3(0.2)	8.0(0.2)	8.0(0.2)	7.9(0.2)	8.0(0.2)	8.0(0.2)	8.0(0.2)	8.0(0.2)	8.0(0.2)	7.9(0.2)	8.0(0.2)	8.0(0.2)	8.0(0.2)
molec-biol-promoter	20.4(5.5)	22.1(9.3)	22.6(7.2)	21.7(6.5)	22.1(6.0)	22.8(6.9)	20.4(5.4)	22.5(6.8)	22.4(7.2)	23.0(7.6)	23.7(7.4)	20.5(5.0)	21.2(5.7)	23.7(5.6)
molec-biol-splice	14.2(1.0)	14.3(0.8)	14.6(1.2)	14.5(0.9)	14.3(1.0)	14.8(1.2)	14.6(0.9)	14.4(0.9)	14.3(0.8)	14.5(0.8)	14.4(1.2)	14.6(1.1)	14.6(1.1)	14.7(1.0)
monks-1	38.1(10.3)	41.9(7.3)	38.2(8.8)	39.7(7.6)	39.1(9.7)	36.4(8.6)	38.8(9.8)	38.9(7.2)	40.6(7.5)	38.7(8.3)	38.4(7.8)	39.6(8.1)	39.6(7.7)	43.4(6.3)
monks-2	34.1(0.6)	34.2(0.6)	33.9(0.5)	34.0(0.6)	34.3(0.3)	34.1(0.5)	34.0(0.7)	34.2(0.7)	34.0(0.5)	33.9(0.6)	34.1(0.5)	33.6(0.6)	34.1(0.5)	34.2(0.6)
monks-3	45.2(11.5)	45.3(8.4)	48.3(6.5)	44.2(10.2)	45.7(7.5)	46.9(8.9)	39.4(9.8)	48.9(5.7)	47.7(7.2)	47.3(8.7)	42.2(10.1)	42.2(11.0)	43.4(10.1)	43.7(10.7)
mushroom	0.0(0.0)	0.0(0.0)	0.0(0.1)	0.0(0.1)	0.0(0.0)	0.0(0.1)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)
musk-1	8.1(3.1)	7.8(2.4)	8.7(3.1)	8.0(3.1)	6.7(2.2)	7.3(2.4)	7.7(2.8)	8.6(3.6)	7.3(2.1)	8.2(2.9)	7.6(2.5)	8.3(4.5)	8.2(2.8)	7.9(2.6)
musk-2	0.6(0.2)	0.5(0.1)	0.5(0.2)	0.6(0.3)	0.5(0.2)	0.6(0.2)	0.5(0.2)	0.6(0.2)	0.6(0.2)	0.5(0.2)	0.5(0.2)	0.5(0.2)	0.5(0.2)	0.6(0.2)
nursery	0.9(0.2)	1.0(0.2)	0.9(0.3)	0.9(0.2)	0.9(0.2)	0.9(0.1)	0.9(0.2)	0.9(0.2)	0.9(0.2)	0.9(0.2)	0.9(0.2)	4.2(14.3)	1.0(0.3)	1.0(0.3)
oocytes-merluccius-nucl..	15.3(1.7)	16.1(2.0)	15.5(1.9)	15.4(2.5)	15.3(1.8)	16.1(2.3)	15.5(2.2)	15.8(2.1)	16.1(2.5)	16.1(2.3)	16.0(2.3)	15.6(2.4)	16.2(2.7)	15.5(2.4)
oocytes-merluccius-stat..	9.0(1.4)	8.4(1.4)	8.4(1.3)	8.6(1.3)	8.1(1.2)	8.3(1.2)	8.6(1.2)	8.9(1.4)	8.7(1.2)	8.6(1.3)	8.7(1.2)	9.7(5.2)	8.9(1.5)	8.7(1.4)
oocytes-trisopterus-nuc..	15.7(2.0)	15.6(2.1)	16.0(2.2)	16.5(2.2)	15.1(1.6)	15.5(1.9)	15.9(1.7)	16.0(2.1)	15.9(2.7)	15.9(1.6)	15.4(1.5)	15.9(1.6)	16.2(2.3)	16.0(2.3)
oocytes-trisopterus-sta..	7.1(1.7)	6.8(1.6)	7.1(1.3)	6.8(1.4)	6.7(1.3)	6.6(1.1)	6.7(1.3)	6.9(1.4)	7.0(1.5)	6.7(1.2)	6.9(1.4)	6.9(1.4)	7.1(1.2)	7.0(1.5)
optical	2.7(0.4)	2.7(0.3)	2.8(0.3)	2.8(0.3)	2.7(0.3)	2.9(0.4)	2.8(0.3)	2.8(0.3)	2.8(0.3)	2.8(0.3)	2.8(0.4)	2.9(0.3)	3.0(0.5)	2.9(0.3)
ozone	3.3(0.5)	3.1(0.4)	3.1(0.5)	3.0(0.3)	3.0(0.2)	2.9(0.1)	3.1(0.3)	2.9(0.2)	2.9(0.1)	3.2(0.4)	3.0(0.3)	2.9(0.0)	2.9(0.1)	3.0(0.3)
page-blocks	3.4(0.3)	3.5(0.4)	3.4(0.4)	3.4(0.3)	3.4(0.3)	3.6(0.5)	3.5(0.3)	3.4(0.3)	3.4(0.4)	3.4(0.3)	3.4(0.4)	3.5(0.4)	3.4(0.4)	3.4(0.3)
parkinsons	10.7(4.4)	8.4(4.3)	11.5(5.0)	11.5(4.7)	10.0(4.6)	8.3(4.5)	10.3(4.9)	13.1(6.1)	13.9(5.8)	11.1(5.8)	7.6(2.5)	11.0(4.9)	11.3(5.2)	10.3(5.3)
pendigits	2.8(0.2)	2.7(0.2)	2.7(0.2)	2.7(0.1)	2.7(0.2)	2.7(0.2)	2.8(0.3)	2.7(0.2)	2.8(0.3)	2.7(0.2)	2.7(0.2)	2.8(0.3)	2.7(0.2)	2.8(0.2)
pima	24.8(3.0)	23.8(2.6)	25.2(3.6)	23.9(3.0)	23.3(2.3)	23.9(3.2)	24.5(2.9)	23.4(2.2)	23.2(2.3)	23.3(2.0)	23.5(2.0)	24.3(4.1)	22.9(1.9)	23.4(2.3)
pittsburg-bridges-MATER..	20.4(4.2)	20.9(5.0)	20.7(5.4)	19.7(4.4)	20.1(3.9)	20.8(4.6)	21.5(4.6)	22.0(4.5)	22.2(3.6)	20.3(2.9)	20.4(3.1)	20.1(5.0)	20.3(4.9)	20.4(3.7)
pittsburg-bridges-REL-L	35.7(7.4)	34.6(7.0)	35.5(7.2)	35.7(6.7)	36.6(7.3)	37.7(8.9)	37.4(6.4)	38.6(7.2)	37.7(6.5)	33.9(6.2)	37.3(5.8)	36.4(6.3)	37.2(8.4)	38.6(8.3)
pittsburg-bridges-SPAN	35.3(8.3)	34.8(7.3)	37.3(9.4)	37.6(8.9)	32.3(6.3)	35.0(8.2)	32.9(7.6)	35.6(6.8)	34.7(7.1)	34.5(5.6)	33.8(6.4)	33.5(5.9)	30.9(5.5)	33.0(6.8)
pittsburg-bridges-T-OR-..	16.1(4.6)	16.0(5.5)	15.3(4.6)	15.6(4.0)	15.4(4.0)	13.7(2.9)	15.3(4.8)	15.3(2.1)	14.4(2.1)	15.4(3.5)	14.9(3.1)	16.4(4.8)	15.1(3.5)	15.4(4.4)
pittsburg-bridges-TYPE	45.0(7.1)	44.5(7.0)	46.1(6.9)	48.8(4.7)	47.6(5.7)	47.8(7.0)	46.1(8.1)	50.9(7.2)	49.6(5.4)	47.9(5.5)	45.5(8.5)	47.2(7.3)	52.1(4.5)	47.8(8.1)
planning	32.1(6.8)	30.5(5.4)	29.9(2.7)	30.1(4.1)	28.7(3.0)	29.2(2.9)	29.0(2.4)	28.7(4.5)	28.9(3.7)	31.1(5.7)	29.2(5.2)	27.9(0.9)	28.8(2.6)	31.4(7.5)
plant-margin	17.2(1.0)	17.5(1.2)	17.6(1.1)	17.7(1.2)	17.7(1.4)	17.8(1.2)	17.5(1.2)	17.6(1.2)	17.8(1.1)	17.6(1.5)	17.6(1.1)	18.0(1.6)	17.8(1.3)	17.5(1.1)
plant-shape	31.2(1.8)	31.4(1.8)	30.8(1.7)	31.1(1.7)	31.3(2.2)	31.2(1.8)	31.1(1.9)	30.6(1.8)	31.1(2.0)	30.8(1.9)	31.3(1.8)	31.3(1.4)	31.2(1.8)	31.5(1.5)
plant-texture	16.2(1.2)	16.0(1.4)	16.0(1.3)	15.6(1.5)	15.7(1.6)	15.9(1.4)	15.9(1.2)	16.0(1.7)	15.8(1.4)	15.7(1.3)	15.9(1.3)	15.9(1.5)	16.2(1.3)	15.9(1.6)
post-operative	33.8(3.6)	34.2(5.1)	33.6(5.3)	32.0(1.9)	31.6(1.4)	31.7(2.0)	32.8(4.3)	32.0(1.7)	32.3(2.0)	34.7(5.3)	33.1(3.3)	31.7(1.1)	31.9(1.2)	33.6(4.3)
primary-tumor	58.0(3.3)	58.3(3.4)	58.3(3.6)	58.7(3.4)	58.4(2.6)	58.6(4.3)	57.6(2.7)	58.5(3.5)	59.5(4.8)	58.1(4.1)	58.3(3.7)	59.3(2.5)	59.0(4.0)	58.7(3.1)
ringnorm	1.7(0.3)	1.6(0.3)	1.5(0.3)	1.6(0.2)	1.6(0.2)	1.6(0.3)	1.6(0.3)	1.6(0.3)	1.6(0.3)	1.6(0.2)	1.8(0.3)	1.7(0.4)	1.6(0.2)	1.6(0.3)
seeds	9.2(7.0)	7.6(3.1)	7.9(3.5)	8.3(3.6)	8.0(3.5)	7.8(3.5)	6.8(3.6)	7.8(3.0)	8.9(3.4)	8.3(3.7)	8.0(3.4)	7.1(2.6)	7.6(3.2)	10.5(13.0)
semeion	5.1(0.9)	5.1(1.0)	5.1(1.3)	5.7(1.2)	5.1(1.1)	4.9(1.1)	4.7(1.0)	5.5(1.3)	5.2(1.1)	5.3(1.1)	4.9(0.9)	5.0(0.8)	5.3(1.1)	5.3(1.0)
soybean	58.5(8.5)	58.7(13.1)	59.6(13.2)	55.5(8.3)	54.7(8.6)	53.7(7.2)	58.9(13.1)	58.6(13.4)	54.6(6.6)	57.1(11.3)	53.6(8.4)	55.6(9.5)	54.7(9.7)	51.7(5.0)
spambase	6.8(0.5)	6.8(0.5)	6.8(0.5)	6.7(0.5)	6.8(0.5)	6.7(0.6)	6.7(0.5)	6.7(0.5)	6.8(0.5)	6.8(0.6)	6.7(0.5)	6.7(0.5)	6.8(0.5)	6.6(0.6)
spect	41.6(4.4)	40.4(4.0)	43.6(3.1)	43.0(2.5)	41.9(2.9)	41.4(4.3)	41.2(3.8)	41.5(4.3)	42.8(3.1)	40.2(3.9)	41.0(3.7)	43.5(2.6)	42.8(3.2)	41.0(3.8)
spectf	29.0(36.4)	12.2(18.3)	8.0(0.0)	12.2(18.3)	8.0(0.0)	12.2(18.3)	16.4(25.2)	8.0(0.0)	8.0(0.0)	16.4(25.2)	8.0(0.0)	8.0(0.0)	12.2(18.3)	16.4(25.2)
statlog-australian-cred..	33.4(2.8)	34.1(2.7)	32.2(0.2)	32.9(1.8)	33.0(1.7)	33.2(1.8)	32.9(2.1)	32.3(0.5)	32.6(0.9)	34.1(3.0)	33.7(2.5)	32.2(0.1)	33.2(2.0)	34.3(3.6)
statlog-german-credit	24.8(2.0)	24.8(1.8)	24.3(1.9)	24.5(2.0)	24.4(1.8)	24.2(1.7)	24.7(1.5)	25.2(2.3)	24.7(2.6)	24.8(2.1)	24.3(2.1)	25.4(2.4)	24.3(1.6)	25.0(1.6)
statlog-heart	16.8(3.6)	16.6(3.7)	17.3(4.1)	17.1(4.1)	16.0(3.2)	17.2(4.2)	16.6(3.7)	17.1(3.0)	17.4(3.0)	17.9(3.9)	16.5(2.9)	18.3(9.2)	16.1(4.8)	19.0(5.2)
statlog-image	3.7(0.6)	3.9(0.7)	3.8(0.8)	3.8(0.6)	4.0(0.8)	3.6(0.7)	4.0(0.7)	3.9(0.8)	4.0(0.8)	3.8(0.7)	3.8(0.8)	3.8(0.8)	3.8(0.7)	3.6(0.7)
statlog-landsat	8.4(0.7)	8.7(0.6)	8.3(0.5)	8.4(0.5)	8.5(0.4)	8.4(0.4)	8.6(0.6)	9.4(0.8)	9.0(0.8)	8.8(0.6)	9.2(0.7)	8.9(0.6)	9.2(1.0)	8.9(0.6)

Dataset/Method	grid	rs	gp	gp-sel	gp-r	gp-sel-r	cma	cma-pv	cma-bw	cma-gv	cma-r	cma-pv-r	cma-bw-r	cma-gv-r
statlog-shuttle	0.3(0.1)	0.3(0.1)	0.3(0.1)	0.3(0.1)	0.3(0.0)	0.3(0.0)	0.2(0.0)	0.3(0.1)	0.3(0.1)	0.3(0.1)	0.2(0.1)	0.2(0.0)	0.3(0.0)	0.3(0.1)
statlog-vehicle	17.7(2.5)	17.8(2.3)	17.7(2.1)	18.1(2.9)	17.3(2.1)	17.3(2.8)	17.7(2.4)	17.7(2.3)	18.6(2.5)	17.7(2.5)	16.6(2.0)	17.4(2.8)	18.6(3.2)	18.3(3.4)
steel-plates	25.9(1.7)	25.6(1.6)	25.3(1.6)	25.1(1.5)	24.6(1.4)	25.8(2.0)	25.9(1.5)	25.9(1.6)	26.3(2.2)	26.1(2.0)	26.2(1.8)	25.9(1.5)	25.8(2.0)	25.6(1.9)
synthetic-control	1.1(0.8)	1.5(0.9)	1.4(0.9)	2.7(7.0)	1.2(0.8)	1.2(0.8)	1.5(1.0)	1.5(0.9)	1.5(0.8)	1.4(0.7)	1.5(1.2)	1.3(1.0)	1.4(0.8)	1.3(0.8)
teaching	45.1(4.3)	46.6(5.6)	46.4(5.7)	47.3(5.0)	46.2(5.8)	46.3(5.8)	46.3(5.8)	47.4(4.6)	46.3(4.5)	47.2(7.7)	47.0(6.5)	47.8(4.4)	48.4(4.0)	46.1(5.8)
thyroid	3.0(0.6)	2.8(0.5)	2.8(0.6)	3.1(0.5)	3.1(0.4)	2.9(0.5)	2.8(0.6)	2.7(0.5)	3.0(0.4)	3.0(0.4)	2.9(0.5)	3.1(0.3)	2.9(0.5)	3.0(0.5)
tic-tac-toe	0.4(0.7)	0.4(0.7)	0.6(0.8)	0.4(0.7)	0.7(0.8)	0.4(0.7)	0.4(0.8)	0.4(0.6)	0.7(0.8)	0.5(0.8)	0.8(0.9)	0.4(0.6)	0.9(0.9)	0.5(0.8)
titanic	21.8(1.1)	21.5(1.0)	21.4(1.0)	21.4(1.0)	21.4(1.2)	21.4(1.0)	21.6(1.0)	21.6(1.0)	21.7(1.1)	21.7(1.1)	21.6(1.1)	21.5(1.0)	21.4(1.1)	21.5(1.0)
twonorm	2.3(0.3)	2.2(0.2)	2.3(0.2)	2.2(0.2)	2.2(0.3)	2.3(0.3)	2.2(0.3)	2.2(0.3)	2.2(0.3)	2.2(0.2)	2.3(0.3)	2.3(0.2)	2.2(0.3)	2.3(0.3)
vertebral-column-2class..	15.2(3.3)	16.0(4.3)	16.3(3.2)	15.8(2.6)	14.7(2.5)	15.7(3.2)	15.8(4.4)	16.0(4.9)	16.6(4.4)	16.8(5.6)	14.7(2.9)	14.3(3.1)	14.4(3.3)	14.4(3.1)
vertebral-column-3class..	16.4(3.7)	15.9(3.1)	18.0(3.9)	17.1(4.1)	14.9(3.0)	17.2(5.2)	16.3(3.1)	15.0(3.7)	15.7(3.6)	16.2(4.4)	15.4(3.5)	15.5(3.3)	14.9(3.0)	15.2(3.1)
wall-following	8.0(0.7)	8.5(0.8)	8.3(0.8)	8.4(0.7)	8.1(0.6)	8.3(0.7)	8.6(0.9)	8.3(0.8)	8.6(0.8)	8.2(0.6)	8.6(0.8)	8.3(0.9)	8.8(1.2)	8.5(1.0)
waveform	13.3(0.7)	13.2(0.7)	13.1(0.6)	13.1(0.8)	13.2(0.6)	13.3(0.8)	13.2(0.6)	13.2(0.7)	13.2(0.7)	13.1(0.7)	13.3(0.7)	13.2(0.6)	13.1(0.8)	13.2(0.7)
waveform-noise	14.0(0.5)	14.1(0.7)	14.1(0.5)	14.1(0.6)	14.1(0.7)	14.0(0.6)	14.1(0.6)	14.1(0.7)	14.2(0.6)	14.2(0.6)	14.0(0.6)	14.0(0.6)	14.0(0.5)	14.0(0.5)
wine	2.4(1.6)	3.3(2.3)	4.3(7.6)	3.8(4.6)	4.0(3.7)	2.9(1.8)	3.2(2.3)	2.8(1.5)	3.3(1.8)	2.9(2.1)	3.3(2.4)	2.8(1.4)	3.4(1.7)	3.3(1.9)
wine-quality-red	39.1(1.5)	38.6(2.2)	38.5(1.4)	39.1(2.0)	38.5(1.9)	38.2(1.9)	39.0(2.4)	41.1(1.7)	41.2(2.7)	39.8(2.0)	39.7(2.1)	40.6(2.5)	41.3(1.9)	40.0(2.3)
wine-quality-white	38.9(1.1)	37.1(1.6)	36.8(0.8)	36.7(1.6)	36.5(0.9)	36.6(0.8)	36.7(1.0)	37.1(2.0)	37.3(2.1)	37.2(2.3)	36.5(1.1)	36.9(1.2)	37.2(1.3)	36.8(1.6)
yeast	41.1(1.8)	41.2(1.5)	41.7(1.7)	41.2(1.9)	41.1(1.6)	41.7(2.2)	41.2(1.4)	40.9(1.6)	41.5(1.6)	41.6(2.2)	41.3(1.5)	41.3(1.0)	41.3(1.4)	41.7(1.2)
zoo	6.4(4.6)	7.3(4.5)	9.9(7.1)	7.8(7.7)	5.7(3.6)	5.8(3.6)	6.8(4.0)	6.1(3.4)	6.4(3.2)	6.9(3.3)	5.7(3.6)	5.8(3.4)	6.1(3.4)	6.4(3.4)

Table B.2: Generalization error for all datasets, averaged over repetitions (standard deviation in parenthesis). 5-fold cross-validation.

Dataset/Method	grid	rs	gp	gp-sel	gp-r	gp-sel-r	cma	cma-pv	cma-bw	cma-gv	cma-r	cma-pv-r	cma-bw-r	cma-gv-r
abalone	33.8(0.7)	33.9(1.0)	33.9(0.8)	33.9(1.1)	33.8(0.9)	34.0(1.0)	33.9(0.9)	33.9(1.0)	34.0(1.1)	34.1(0.8)	33.9(0.9)	34.0(0.8)	34.1(0.9)	34.2(1.1)
acute-inflammation	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.4(1.6)	0.4(1.6)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)
acute-nephritis	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)
adult	15.4(0.1)	15.4(0.1)	15.4(0.1)	15.5(0.2)	15.4(0.1)	15.4(0.1)	15.4(0.1)	15.4(0.0)	15.4(0.1)	15.5(0.2)	15.4(0.1)	15.4(0.1)	15.4(0.1)	15.4(0.1)
annealing	24.0(0.0)	24.0(0.0)	24.0(0.0)	24.0(0.0)	24.0(0.0)	24.1(0.2)	24.0(0.0)	24.4(1.0)	25.6(4.6)	23.9(0.3)	24.0(0.0)	24.0(0.0)	24.2(0.9)	24.1(0.9)
arrhythmia	31.3(2.4)	31.5(2.6)	31.5(2.6)	31.5(2.1)	31.8(2.2)	31.8(2.8)	32.2(4.3)	33.0(5.4)	33.7(4.9)	33.2(4.9)	31.3(2.5)	32.5(4.7)	31.9(2.3)	32.0(2.8)
audiology-std	36.0(0.0)	31.2(3.7)	32.2(4.5)	34.4(7.7)	33.4(2.9)	32.4(3.6)	31.8(3.5)	31.6(3.3)	34.0(3.2)	32.6(3.6)	33.4(3.4)	33.0(3.5)	33.2(3.4)	32.4(3.8)
balance-scale	1.4(1.3)	1.6(1.7)	0.9(1.1)	1.5(1.7)	0.6(0.8)	1.2(1.9)	1.4(1.8)	2.2(3.1)	1.6(1.8)	2.2(2.8)	1.2(1.3)	1.9(2.8)	1.8(2.2)	1.6(2.2)
bank	10.5(0.3)	10.5(0.3)	10.9(0.5)	10.9(0.5)	10.7(0.5)	10.6(0.6)	10.6(0.5)	10.9(0.7)	10.7(0.4)	10.6(0.5)	10.7(0.5)	11.0(0.5)	10.8(0.5)	10.7(0.5)
blood	22.2(0.9)	21.8(1.1)	22.3(1.2)	22.0(1.4)	22.1(1.0)	22.2(1.3)	22.2(1.2)	23.4(2.5)	23.1(1.7)	22.5(1.4)	22.4(1.8)	22.7(1.3)	22.9(1.6)	22.9(1.2)
breast-cancer	30.0(2.4)	29.3(2.8)	29.8(2.3)	28.5(2.1)	29.4(1.6)	29.0(2.4)	29.8(3.1)	28.7(1.6)	29.0(2.2)	29.4(2.7)	29.1(1.7)	29.8(2.7)	30.2(1.3)	29.6(2.8)
breast-cancer-wisc	3.5(1.1)	3.6(1.0)	3.6(1.1)	4.2(1.1)	3.8(1.1)	4.2(0.8)	3.4(0.7)	3.5(1.1)	3.7(1.1)	3.9(0.7)	3.6(1.0)	3.5(0.7)	3.7(1.2)	4.0(1.3)
breast-cancer-wisc-diag	3.3(0.9)	3.1(1.2)	3.1(0.8)	3.6(1.1)	3.2(1.1)	3.6(2.0)	3.2(0.8)	3.2(0.9)	3.2(0.9)	3.8(1.2)	3.0(0.9)	3.2(1.1)	3.4(0.9)	3.2(0.9)
breast-cancer-wisc-prog	20.9(3.7)	21.1(2.9)	22.7(2.7)	22.9(2.1)	22.4(2.7)	22.6(2.6)	21.8(3.1)	23.1(3.6)	22.2(2.9)	22.7(3.2)	21.7(3.9)	22.5(2.0)	22.2(3.4)	22.9(3.6)
breast-tissue	31.5(4.4)	33.4(4.6)	34.0(5.4)	33.5(5.9)	32.1(5.7)	33.9(5.8)	33.8(5.6)	33.5(6.1)	34.1(6.8)	34.2(6.2)	33.4(5.5)	33.9(4.8)	34.4(6.3)	33.2(7.4)
car	1.3(0.5)	1.3(0.5)	1.4(0.6)	1.5(0.6)	1.3(0.6)	1.4(0.7)	1.3(0.6)	1.5(0.5)	1.5(0.8)	1.3(0.6)	1.3(0.6)	1.4(0.7)	1.3(0.6)	1.3(0.6)
cardiotocography-10clas..	16.7(1.2)	17.1(0.8)	16.5(1.2)	17.0(1.7)	16.6(1.6)	17.2(1.9)	16.8(1.5)	17.0(1.7)	17.0(1.5)	17.5(2.2)	16.8(1.2)	17.5(1.4)	17.5(1.7)	17.2(1.6)
cardiotocography-3class..	8.0(1.2)	8.2(0.7)	7.9(1.0)	8.1(1.0)	7.7(0.9)	8.1(0.9)	8.3(0.8)	8.6(1.0)	8.6(1.1)	8.8(0.9)	8.2(0.9)	8.6(0.8)	8.6(0.8)	8.6(1.2)

Dataset/Method	grid	rs	gp	gp-sel	gp-r	gp-sel-r	cma	cma-pv	cma-bw	cma-gv	cma-r	cma-pv-r	cma-bw-r	cma-gv-r
chess-krvk	40.3(0.5)	40.1(0.5)	39.6(0.4)	39.6(0.5)	39.6(0.4)	39.6(0.5)	39.8(0.5)	40.2(0.6)	40.0(0.6)	40.0(0.6)	40.0(0.4)	40.3(0.5)	40.1(0.5)	40.1(0.4)
chess-krvkp	1.0(0.3)	1.0(0.4)	0.9(0.3)	1.0(0.3)	0.9(0.3)	0.9(0.3)	0.9(0.3)	0.9(0.2)	0.9(0.2)	0.9(0.3)	0.9(0.3)	1.0(0.5)	0.9(0.3)	1.0(0.4)
congressional-voting	39.0(1.5)	39.2(1.4)	39.3(1.1)	40.0(1.8)	39.1(1.7)	39.5(1.8)	39.2(1.6)	39.3(0.9)	39.0(0.8)	39.5(2.1)	38.7(1.0)	39.9(2.5)	39.7(2.4)	39.5(2.0)
conn-bench-sonar-mines-..	13.4(4.6)	14.3(4.6)	14.2(4.2)	15.2(5.7)	13.3(4.1)	15.6(4.9)	13.5(4.4)	14.6(5.2)	15.6(5.8)	16.0(6.0)	14.0(4.4)	14.6(5.5)	15.1(5.1)	15.1(5.2)
conn-bench-vowel-deterd..	0.0(0.1)	0.0(0.1)	0.0(0.1)	0.0(0.1)	0.0(0.1)	0.0(0.1)	0.0(0.2)	0.0(0.0)	0.0(0.1)	0.0(0.1)	0.0(0.0)	0.0(0.2)	0.1(0.2)	0.0(0.2)
connect-4	18.2(0.3)	18.5(0.4)	18.8(2.0)	18.2(0.3)	18.5(1.4)	18.5(1.4)	18.4(0.4)	19.7(2.3)	18.8(1.3)	18.8(1.3)	18.5(0.5)	19.2(1.9)	18.7(0.8)	18.6(0.8)
contrac	45.2(1.9)	45.3(2.0)	45.1(2.0)	45.5(1.7)	45.3(2.2)	45.6(1.6)	45.2(1.7)	45.7(1.7)	45.3(1.8)	45.6(1.8)	45.2(2.0)	45.4(2.0)	45.6(1.9)	45.8(1.9)
credit-approval	14.2(2.0)	13.8(2.0)	13.9(2.0)	13.5(1.9)	14.0(2.1)	13.5(2.0)	13.7(2.0)	13.7(2.0)	13.6(2.0)	13.6(2.0)	13.9(1.9)	15.1(7.0)	13.8(1.9)	13.7(1.6)
cylinder-bands	24.5(2.4)	23.9(3.2)	24.0(2.4)	22.6(2.7)	23.9(2.8)	23.8(3.4)	23.6(3.4)	25.3(2.8)	25.1(2.7)	24.0(3.0)	23.9(2.5)	24.7(2.1)	25.7(3.4)	24.3(3.1)
dermatology	3.2(1.7)	3.1(1.2)	3.1(1.3)	3.3(1.3)	3.1(1.4)	3.3(1.5)	3.1(1.2)	3.2(1.3)	3.3(1.5)	3.2(1.5)	3.0(1.0)	3.3(1.3)	3.2(1.4)	3.4(1.7)
echocardiogram	16.2(3.9)	15.2(2.7)	16.7(4.3)	17.8(6.0)	15.2(2.8)	16.0(4.3)	16.7(5.0)	16.3(5.1)	15.6(2.6)	17.6(5.9)	15.3(2.5)	14.8(2.4)	16.3(3.3)	17.0(5.4)
ecoli	14.8(3.1)	14.7(3.0)	14.7(2.8)	14.6(2.4)	14.4(3.2)	14.6(2.2)	14.7(3.0)	14.9(2.5)	14.2(2.4)	14.9(2.4)	14.9(2.9)	14.9(3.0)	15.0(2.8)	14.4(2.9)
energy-y1	5.2(1.5)	5.0(1.1)	4.5(1.2)	4.7(1.4)	4.1(1.1)	4.6(1.5)	4.4(1.1)	4.8(1.7)	4.5(1.1)	4.8(1.3)	4.5(1.2)	4.7(1.5)	4.6(1.1)	4.7(1.4)
energy-y2	8.9(1.6)	8.7(1.6)	8.3(1.5)	8.7(1.3)	8.4(1.3)	8.7(1.1)	8.8(1.3)	8.5(1.3)	8.1(1.2)	8.5(1.4)	9.0(1.2)	8.6(1.1)	8.5(1.0)	8.3(1.1)
fertility	15.1(2.0)	15.3(2.6)	14.3(0.0)	14.4(0.6)	14.4(0.6)	14.3(0.0)	14.9(1.9)	15.3(3.2)	15.0(2.5)	14.4(1.4)	14.7(1.0)	15.3(2.6)	14.3(0.0)	14.7(1.9)
flags	53.6(5.2)	53.9(5.2)	53.5(4.5)	53.7(4.1)	54.8(4.6)	53.6(5.2)	54.8(3.6)	54.4(4.2)	53.2(3.2)	53.9(4.5)	54.2(5.3)	55.1(6.1)	55.0(5.6)	53.9(5.9)
glass	33.9(4.8)	33.2(4.8)	33.4(4.9)	33.8(4.5)	33.3(3.3)	33.8(4.7)	34.0(4.2)	32.1(5.4)	35.9(6.0)	34.0(6.6)	32.2(3.1)	33.7(5.8)	35.3(5.6)	35.0(6.2)
haberman-survival	27.8(2.0)	28.3(2.5)	28.0(2.0)	28.2(2.7)	28.0(2.2)	28.1(2.5)	27.6(2.3)	27.7(2.0)	28.1(2.6)	28.5(3.7)	27.9(2.4)	27.7(2.0)	27.7(2.3)	28.3(4.0)
hayes-roth	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)	96.4(0.0)
heart-cleveland	41.7(2.6)	42.0(2.7)	42.6(2.6)	42.8(2.9)	43.0(2.7)	43.7(2.6)	42.7(2.8)	42.3(2.2)	41.7(2.2)	43.4(3.2)	41.6(2.5)	42.8(2.4)	42.6(3.0)	43.0(3.1)
heart-hungarian	17.9(2.1)	18.1(2.2)	18.5(2.5)	18.9(2.1)	18.4(3.1)	19.1(3.5)	18.1(2.1)	19.3(3.2)	18.6(2.6)	19.6(2.8)	18.4(2.9)	18.9(2.3)	19.4(2.5)	19.4(2.3)
heart-switzerland	62.6(6.1)	62.4(4.3)	61.2(4.0)	62.3(5.3)	60.5(4.5)	61.4(6.1)	61.6(4.4)	61.6(5.2)	62.8(3.8)	63.8(5.6)	62.8(4.4)	61.5(2.5)	61.4(3.2)	63.9(6.2)
heart-va	69.6(5.1)	68.9(4.4)	70.8(3.8)	69.5(4.1)	69.7(4.4)	69.0(4.6)	70.1(4.1)	71.3(2.8)	70.6(5.2)	69.9(3.9)	69.1(5.3)	70.2(3.7)	70.6(3.0)	69.2(4.6)
hepatitis	17.9(3.3)	17.8(2.6)	20.4(1.3)	19.4(3.2)	19.5(2.3)	19.5(2.7)	18.3(2.8)	19.2(2.4)	18.4(3.3)	18.6(2.6)	18.5(3.1)	20.0(2.0)	19.3(2.5)	19.2(2.6)
hill-valley	49.6(1.2)	48.7(1.3)	48.9(1.4)	48.6(1.4)	48.8(1.3)	48.4(1.3)	49.3(1.4)	49.1(1.4)	49.2(1.4)	49.2(1.7)	49.6(1.2)	49.2(1.2)	49.2(1.2)	48.9(1.6)
horse-colic	39.1(1.8)	39.3(1.4)	39.7(0.0)	36.6(3.3)	39.1(1.8)	38.6(2.1)	37.5(3.1)	36.5(3.1)	35.0(3.4)	35.0(4.3)	39.0(1.8)	37.8(2.4)	37.0(2.6)	37.9(2.9)
ilpd-indian-liver	29.5(1.7)	29.8(1.5)	29.0(1.5)	29.5(2.4)	28.6(0.9)	29.1(1.2)	29.0(1.4)	28.3(0.6)	28.5(0.5)	30.2(1.9)	28.4(0.8)	28.5(0.7)	28.6(1.0)	30.1(1.9)
image-segmentation	85.5(0.7)	83.3(7.4)	78.9(8.7)	81.4(8.5)	82.6(6.4)	81.4(8.1)	80.4(8.5)	75.8(9.1)	71.5(9.1)	77.7(9.7)	77.1(9.5)	80.7(8.1)	70.6(8.7)	81.8(8.3)
ionosphere	6.2(2.1)	6.9(2.2)	6.8(1.5)	7.3(2.1)	7.1(2.0)	7.1(2.2)	7.9(2.0)	7.5(3.5)	7.8(3.5)	6.8(2.4)	7.0(2.2)	6.7(2.3)	6.7(2.2)	7.8(2.7)
iris	4.2(2.6)	3.9(2.2)	3.9(2.6)	7.1(6.6)	3.8(2.6)	7.5(7.0)	3.8(2.7)	4.3(3.6)	3.6(2.4)	6.3(7.0)	3.2(2.5)	3.5(2.1)	4.5(4.6)	6.3(6.5)
led-display	28.0(2.0)	27.6(2.2)	28.0(2.1)	28.3(2.3)	28.0(2.3)	28.2(1.9)	27.7(2.0)	28.3(1.9)	28.6(2.0)	28.4(2.4)	27.8(2.0)	28.1(1.9)	28.6(2.3)	28.3(2.1)
letter	6.2(0.3)	5.9(0.4)	5.8(0.4)	5.7(0.4)	5.8(0.4)	5.7(0.4)	5.9(0.4)	5.8(0.5)	5.7(0.4)	5.7(0.4)	5.9(0.3)	5.7(0.4)	5.7(0.4)	5.7(0.4)
libras	16.5(2.9)	16.1(3.0)	16.4(3.5)	17.0(3.9)	16.3(3.4)	16.4(3.8)	16.4(2.6)	17.7(5.5)	17.7(3.3)	17.6(3.8)	16.1(3.4)	17.1(3.6)	16.4(3.4)	16.7(3.4)
low-res-spect	10.5(2.1)	10.8(1.8)	10.3(1.6)	10.4(1.7)	10.6(1.6)	11.1(2.1)	11.3(2.0)	11.1(2.3)	10.3(1.7)	11.1(2.3)	10.3(1.9)	10.5(1.8)	10.5(2.0)	10.8(2.0)
lung-cancer	55.4(8.3)	55.8(6.8)	57.3(7.1)	60.0(5.8)	58.5(7.1)	57.7(10.2)	54.2(7.1)	59.6(8.4)	57.7(9.9)	56.5(8.5)	54.6(9.1)	58.5(7.5)	58.5(9.2)	58.8(7.8)
lymphography	18.7(4.0)	18.3(4.8)	18.8(5.3)	18.4(6.2)	17.7(3.7)	18.8(4.9)	19.3(5.6)	15.6(3.7)	17.8(4.6)	17.7(5.3)	18.2(6.0)	20.3(8.5)	18.3(5.2)	18.1(5.2)
magic	13.4(0.3)	13.3(0.3)	13.3(0.3)	13.3(0.3)	13.3(0.2)	13.2(0.4)	13.3(0.3)	13.2(0.4)	13.3(0.3)	13.3(0.3)	13.3(0.3)	13.3(0.4)	13.2(0.4)	13.3(0.4)
mammographic	17.5(2.2)	17.6(2.0)	17.6(1.9)	17.1(1.8)	17.7(1.9)	17.4(1.9)	17.6(2.0)	17.3(1.7)	16.8(2.1)	17.2(2.2)	17.5(1.8)	17.4(2.3)	17.3(1.9)	17.3(2.2)
miniboone	8.1(0.2)	8.2(0.3)	7.9(0.2)	7.9(0.2)	7.9(0.2)	7.9(0.2)	7.9(0.2)	7.9(0.2)	7.9(0.2)	7.9(0.2)	7.9(0.2)	7.9(0.2)	7.9(0.2)	7.9(0.2)
molec-biol-promoter	19.5(5.4)	17.9(5.3)	21.3(4.6)	22.9(5.7)	19.5(5.9)	23.0(8.0)	20.1(6.1)	23.3(6.7)	21.1(7.0)	19.5(5.8)	21.8(6.2)	23.0(6.4)	23.2(6.0)	20.8(6.8)
molec-biol-splice	13.9(0.8)	14.2(0.9)	14.0(0.8)	14.1(0.9)	14.2(1.2)	14.2(1.0)	14.2(1.0)	14.4(1.3)	14.3(0.8)	14.3(0.9)	14.1(0.9)	14.6(1.2)	14.4(1.1)	14.4(0.8)
monks-1	40.0(10.8)	38.6(9.1)	36.4(9.8)	41.3(9.0)	28.6(1.9)	35.7(9.0)	40.3(10.0)	39.3(9.2)	38.9(9.2)	42.2(8.4)	36.2(9.8)	38.4(9.5)	36.9(8.7)	36.5(8.3)
monks-2	34.4(0.2)	34.3(0.4)	34.2(0.3)	33.9(0.5)	34.4(0.2)	34.0(0.6)	34.3(0.4)	34.3(0.5)	34.0(0.5)	34.0(0.5)	34.5(0.2)	34.0(0.5)	34.1(0.6)	34.0(0.6)
monks-3	49.6(8.1)	40.0(10.4)	44.6(9.0)	46.1(8.7)	46.8(6.2)	48.4(6.3)	43.3(8.9)	37.5(8.9)	40.6(9.8)	41.4(11.8)	41.8(8.2)	37.3(10.1)	40.9(10.3)	39.5(11.8)
mushroom	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)	0.0(0.0)
musk-1	7.0(2.2)	6.9(2.5)	6.4(1.9)	7.4(2.5)	6.6(2.1)	7.7(2.5)	6.4(2.0)	7.1(2.2)	6.8(2.5)	7.6(2.3)	6.8(2.4)	7.0(2.3)	6.9(2.2)	7.6(2.2)
musk-2	0.5(0.2)	0.5(0.2)	0.5(0.2)	0.5(0.2)	0.5(0.2)	0.5(0.2)	0.5(0.2)	0.5(0.1)	0.6(0.2)	0.5(0.2)	0.5(0.2)	0.5(0.2)	0.5(0.2)	0.5(0.2)
nursery	0.9(0.1)	1.0(0.2)	1.0(0.2)	0.9(0.2)	1.0(0.2)	0.9(0.2)	1.0(0.2)	1.0(0.2)	0.9(0.2)	0.9(0.2)	1.0(0.2)	0.9(0.1)	0.9(0.1)	0.9(0.2)
oocytes-merluccius-nucl..	15.3(1.7)	15.1(1.5)	15.0(1.7)	15.4(2.2)	14.7(2.0)	15.2(2.4)	15.0(1.6)	14.9(2.0)	15.2(2.1)	15.3(2.2)	15.0(1.9)	14.7(1.9)	15.1(2.6)	14.9(2.4)
oocytes-merluccius-stat..	8.5(1.3)	9.0(1.2)	8.7(1.1)	8.3(1.4)	8.5(1.3)	8.6(1.3)	8.9(1.2)	8.5(1.4)	8.6(1.3)	8.8(1.2)	8.7(1.0)	8.6(1.2)	8.9(1.3)	8.6(1.2)
oocytes-trisopterus-nuc..	15.7(1.8)	15.1(1.3)	15.5(1.5)	15.6(1.8)	15.1(1.7)	15.3(1.4)	15.1(1.5)	15.3(1.7)	15.1(1.5)	15.2(1.7)	14.9(1.8)	15.1(1.7)	15.7(1.9)	15.6(1.6)

Dataset/Method	grid	rs	gp	gp-sel	gp-r	gp-sel-r	cma	cma-pv	cma-bw	cma-gv	cma-r	cma-pv-r	cma-bw-r	cma-gv-r
oocytes-trisopterus-sta..	6.3(1.4)	6.5(1.6)	6.4(1.3)	6.4(1.3)	6.3(1.2)	6.8(1.4)	6.2(1.1)	6.4(1.3)	6.5(1.2)	7.0(1.4)	6.2(1.1)	6.6(1.4)	6.4(1.1)	7.2(1.6)
optical	2.5(0.0)	2.7(0.2)	2.7(0.2)	2.7(0.2)	2.6(0.1)	2.7(0.2)	2.6(0.2)	2.7(0.1)	2.7(0.3)	2.6(0.2)	2.7(0.1)	2.7(0.2)	2.7(0.2)	2.7(0.2)
ozone	2.9(0.1)	3.0(0.2)	2.9(0.1)	2.9(0.1)	2.9(0.1)	2.9(0.1)	3.0(0.2)	2.9(0.1)	2.9(0.1)	3.1(0.2)	3.0(0.1)	2.9(0.0)	2.9(0.0)	3.0(0.2)
page-blocks	3.4(0.4)	3.3(0.3)	3.3(0.4)	3.3(0.3)	3.3(0.4)	3.4(0.4)	3.3(0.4)	3.4(0.4)	3.3(0.3)	3.3(0.3)	3.3(0.3)	3.4(0.3)	3.3(0.4)	3.3(0.3)
parkinsons	9.6(4.1)	8.8(4.0)	7.7(3.8)	9.0(3.8)	7.2(3.7)	7.9(4.7)	8.4(4.1)	9.9(3.4)	8.7(4.4)	8.1(4.0)	7.6(3.7)	9.7(3.7)	8.7(4.4)	8.4(4.9)
pendigits	2.7(0.2)	2.7(0.1)	2.6(0.1)	2.7(0.2)	2.6(0.1)	2.7(0.2)	2.6(0.1)	2.7(0.2)	2.6(0.1)	2.7(0.2)	2.6(0.2)	2.7(0.3)	2.6(0.1)	2.7(0.2)
pima	23.7(2.0)	23.5(2.5)	23.5(2.0)	23.1(2.0)	23.3(2.1)	23.0(2.0)	23.3(2.0)	23.1(2.0)	22.9(1.9)	23.8(2.3)	23.6(2.3)	23.1(1.8)	23.2(1.9)	23.3(2.3)
pittsburg-bridges-MATER..	17.3(4.0)	16.5(2.4)	18.8(4.2)	21.6(4.8)	18.6(4.2)	20.0(5.2)	17.2(4.0)	20.1(4.5)	18.8(4.1)	20.5(4.6)	18.2(5.0)	18.5(3.6)	18.5(5.1)	21.2(4.1)
pittsburg-bridges-REL-L	35.0(6.0)	34.3(6.0)	35.0(7.2)	37.7(8.7)	34.3(5.7)	34.9(5.8)	35.7(6.0)	33.5(5.2)	35.1(6.8)	34.1(6.2)	35.9(6.8)	35.5(5.9)	35.9(6.0)	34.6(7.5)
pittsburg-bridges-SPAN	29.8(4.1)	31.2(6.2)	33.2(7.9)	33.6(7.5)	30.3(5.8)	33.0(8.2)	29.7(5.8)	34.1(7.2)	31.2(6.2)	32.7(7.7)	30.2(5.5)	34.8(7.5)	32.7(7.0)	33.3(7.8)
pittsburg-bridges-T-OR-..	13.9(2.8)	14.3(2.6)	14.9(2.9)	15.4(4.3)	15.0(2.5)	14.9(4.3)	14.4(3.1)	14.9(3.2)	15.0(2.4)	16.3(5.6)	14.1(3.1)	15.0(2.5)	14.9(2.1)	15.4(4.6)
pittsburg-bridges-TYPE	45.5(6.5)	46.4(7.6)	47.0(6.4)	46.8(6.4)	48.3(7.2)	47.6(5.0)	49.1(7.1)	47.0(7.2)	49.1(5.5)	47.6(5.4)	47.2(7.8)	49.2(5.9)	49.2(7.3)	46.8(6.1)
planning	27.8(0.9)	27.7(1.5)	27.9(0.8)	29.3(2.6)	27.9(0.8)	28.6(1.3)	28.0(1.6)	27.9(0.8)	28.1(0.9)	28.6(2.7)	28.2(2.5)	27.9(0.8)	28.0(0.8)	28.7(2.0)
plant-margin	17.2(1.0)	17.4(1.2)	17.4(1.0)	17.3(1.3)	17.5(1.3)	17.4(1.0)	17.5(1.3)	17.4(1.1)	17.4(1.1)	17.5(1.1)	17.4(1.3)	17.6(1.4)	17.8(1.3)	17.7(1.0)
plant-shape	30.9(1.8)	30.6(1.4)	30.9(1.6)	31.1(1.9)	30.9(1.7)	31.0(1.9)	30.9(1.6)	31.1(1.8)	31.5(1.8)	31.1(1.9)	30.8(1.6)	30.9(1.5)	31.6(1.8)	31.1(1.8)
plant-texture	16.2(1.4)	15.8(1.6)	15.8(1.5)	15.9(1.2)	15.7(1.3)	15.5(1.3)	15.8(1.3)	15.8(1.5)	15.6(1.4)	15.8(1.4)	15.8(1.5)	15.7(1.4)	15.8(1.4)	16.0(1.3)
post-operative	32.3(2.7)	32.0(1.7)	31.6(1.4)	31.6(1.4)	31.6(1.4)	31.7(2.0)	31.9(1.6)	31.2(0.0)	31.7(1.1)	33.0(2.7)	31.2(0.0)	31.4(0.7)	31.4(0.7)	33.4(5.1)
primary-tumor	56.8(2.4)	56.5(2.2)	56.8(2.9)	57.0(2.8)	56.7(2.5)	56.7(2.9)	56.6(2.8)	57.5(3.4)	56.7(3.1)	57.1(2.8)	56.8(2.3)	58.3(4.6)	57.1(3.8)	57.7(4.5)
ringnorm	1.6(0.3)	1.5(0.2)	1.5(0.2)	1.6(0.2)	1.5(0.3)	1.6(0.2)	1.6(0.2)	1.7(0.2)	1.6(0.2)	1.6(0.3)	1.6(0.2)	1.6(0.2)	1.6(0.3)	1.7(0.3)
seeds	7.0(2.8)	7.7(3.6)	7.3(3.8)	7.4(3.3)	6.2(3.1)	8.8(6.7)	7.2(3.0)	7.2(3.1)	6.7(3.1)	7.2(3.4)	6.5(2.3)	6.5(2.7)	6.8(2.9)	7.5(3.1)
semeion	5.0(0.8)	4.8(0.9)	4.8(0.7)	4.9(1.1)	4.6(0.7)	4.7(0.7)	4.7(0.7)	4.7(0.8)	4.7(0.7)	4.8(0.9)	4.6(0.7)	4.7(0.7)	4.6(0.7)	4.8(0.8)
soybean	58.8(8.1)	55.7(8.6)	54.6(5.8)	60.7(11.5)	58.3(9.0)	57.7(8.8)	53.8(9.3)	53.0(3.4)	54.1(7.6)	59.1(11.1)	52.7(8.7)	53.2(6.9)	53.1(6.2)	58.5(11.2)
spambase	6.6(0.5)	6.6(0.5)	6.6(0.6)	6.6(0.5)	6.6(0.5)	6.7(0.5)	6.6(0.5)	6.6(0.5)	6.7(0.5)	6.7(0.5)	6.6(0.5)	6.7(0.5)	6.7(0.5)	6.7(0.5)
spect	44.6(0.0)	42.8(1.8)	45.2(0.0)	43.8(2.3)	45.1(0.2)	43.9(2.4)	43.8(1.9)	43.9(2.4)	43.8(1.9)	42.6(2.8)	42.0(3.3)	43.7(2.9)	43.4(2.7)	42.2(3.3)
spectf	12.2(18.3)	8.0(0.0)	8.0(0.0)	8.0(0.0)	8.0(0.0)	8.0(0.0)	8.0(0.0)	8.0(0.0)	8.0(0.0)	8.0(0.0)	8.0(0.0)	8.0(0.0)	8.0(0.0)	8.0(0.0)
statlog-australian-cred..	32.4(0.6)	33.2(1.6)	32.2(0.1)	32.5(0.8)	32.2(0.1)	32.7(1.0)	32.9(1.8)	32.2(0.0)	32.2(0.0)	32.8(1.7)	32.5(1.0)	32.4(0.7)	32.2(0.2)	33.0(1.7)
statlog-german-credit	24.1(1.7)	24.4(1.5)	24.1(1.8)	24.3(1.8)	24.3(1.6)	24.4(1.6)	24.0(1.7)	24.2(1.6)	24.1(1.6)	24.6(1.5)	24.2(1.6)	24.6(2.2)	24.5(2.0)	24.4(2.8)
statlog-heart	15.1(3.2)	15.0(3.2)	14.9(3.2)	16.9(3.9)	15.6(3.3)	16.6(4.1)	16.5(5.7)	15.5(3.8)	16.4(2.7)	17.2(3.3)	15.8(2.8)	15.7(3.5)	16.4(3.2)	17.0(3.3)
statlog-image	3.8(0.6)	3.7(0.6)	3.6(0.6)	3.7(0.6)	3.6(0.6)	3.7(0.7)	3.8(0.6)	3.7(0.5)	3.7(0.7)	3.6(0.6)	3.8(0.5)	3.7(0.6)	3.6(0.7)	3.7(0.6)
statlog-landsat	8.1(0.1)	8.7(0.6)	8.2(0.2)	8.3(0.2)	8.1(0.2)	8.2(0.3)	8.7(0.4)	8.8(0.4)	8.6(0.4)	8.7(0.4)	8.8(0.3)	8.6(0.3)	8.5(0.4)	8.5(0.4)
statlog-shuttle	0.2(0.0)	0.2(0.0)	0.2(0.0)	0.3(0.1)	0.2(0.0)	0.3(0.0)	0.2(0.0)	0.2(0.0)	0.2(0.0)	0.3(0.0)	0.2(0.0)	0.2(0.0)	0.2(0.0)	0.2(0.0)
statlog-vehicle	16.6(1.7)	16.8(2.0)	16.4(2.0)	16.9(2.5)	16.9(1.8)	17.1(2.5)	16.9(2.7)	16.5(1.9)	17.0(2.4)	17.5(2.7)	16.9(2.3)	17.1(2.4)	17.2(2.4)	17.9(2.3)
steel-plates	24.8(1.5)	24.7(1.4)	24.2(1.6)	25.0(1.9)	24.1(1.5)	24.9(1.6)	25.0(1.2)	25.4(1.9)	25.5(1.4)	25.5(1.7)	24.8(1.7)	25.2(2.0)	25.1(1.4)	25.5(1.3)
synthetic-control	0.9(0.6)	0.9(0.6)	1.1(0.9)	1.1(0.8)	0.9(0.9)	1.1(0.8)	0.9(0.8)	0.9(0.6)	1.0(0.8)	0.9(0.6)	1.0(0.8)	1.1(0.7)	1.1(0.9)	1.0(0.7)
teaching	47.2(6.0)	46.2(6.3)	46.8(5.2)	47.8(5.6)	47.2(5.5)	47.9(5.4)	47.4(4.2)	48.2(3.7)	47.3(4.8)	48.4(5.1)	47.3(5.1)	46.9(5.3)	48.2(5.5)	46.8(6.2)
thyroid	2.9(0.3)	2.8(0.6)	3.0(0.4)	2.8(0.5)	2.9(0.3)	3.0(0.4)	3.0(0.4)	3.0(0.3)	3.0(0.3)	2.9(0.5)	2.8(0.4)	3.2(0.3)	3.1(0.3)	2.9(0.4)
tic-tac-toe	0.2(0.3)	0.2(0.3)	0.3(0.5)	0.3(0.7)	0.1(0.2)	0.3(0.6)	0.1(0.2)	0.1(0.3)	0.4(0.7)	0.3(0.6)	0.1(0.2)	0.2(0.3)	0.4(0.8)	0.5(0.8)
titanic	21.6(1.0)	21.5(1.1)	21.4(1.2)	21.5(1.0)	21.3(1.1)	21.5(1.0)	21.5(1.0)	21.2(1.1)	21.5(1.1)	21.6(1.0)	21.3(1.1)	21.4(1.1)	21.3(1.0)	21.5(1.0)
twonorm	2.2(0.2)	2.2(0.2)	2.2(0.2)	2.2(0.3)	2.2(0.2)	2.2(0.2)	2.2(0.2)	2.2(0.2)	2.2(0.2)	2.3(0.2)	2.2(0.2)	2.2(0.2)	2.2(0.2)	2.2(0.2)
vertebral-column-2class..	15.1(3.6)	15.4(4.0)	15.3(3.7)	15.0(3.5)	15.5(3.9)	15.7(3.5)	14.6(3.0)	15.0(3.0)	14.6(3.3)	14.9(3.3)	15.2(3.0)	15.0(3.4)	14.6(3.2)	15.0(3.2)
vertebral-column-3class..	15.6(3.2)	15.6(3.1)	15.7(3.2)	15.9(3.2)	15.0(3.0)	15.9(3.4)	15.2(3.3)	15.0(3.2)	15.1(3.2)	15.1(3.5)	15.1(3.2)	15.2(3.1)	15.3(3.3)	15.3(3.7)
wall-following	8.0(0.7)	8.2(0.7)	8.1(0.6)	8.1(0.7)	8.0(0.5)	8.1(0.7)	8.1(0.6)	8.3(0.7)	8.2(0.8)	8.3(0.9)	8.1(0.7)	8.3(0.7)	8.2(0.8)	8.2(0.8)
waveform	13.1(0.7)	13.2(0.7)	13.2(0.7)	13.1(0.7)	13.1(0.7)	13.1(0.7)	13.2(0.7)	13.1(0.7)	13.1(0.8)	13.2(0.8)	13.2(0.7)	13.1(0.7)	13.1(0.7)	13.1(0.7)
waveform-noise	14.0(0.6)	14.0(0.6)	14.0(0.5)	14.0(0.6)	14.0(0.6)	14.0(0.6)	14.0(0.5)	14.0(0.6)	14.0(0.6)	14.0(0.5)	14.1(0.6)	14.0(0.5)	14.0(0.5)	14.0(0.6)
wine	2.1(1.4)	2.1(1.7)	2.1(1.6)	2.8(2.2)	2.1(2.0)	2.9(1.7)	2.1(1.4)	2.8(1.7)	2.5(1.5)	2.2(2.0)	2.4(2.0)	2.8(2.0)	3.5(1.6)	3.1(1.8)
wine-quality-red	38.8(1.7)	37.5(1.7)	37.1(1.7)	38.4(1.6)	37.4(1.3)	38.5(1.1)	38.8(2.3)	41.1(2.4)	40.3(2.6)	40.0(2.3)	38.4(2.1)	39.6(2.0)	39.3(1.9)	39.3(2.1)
wine-quality-white	39.2(1.0)	36.6(0.9)	36.6(0.9)	36.6(1.0)	36.5(0.7)	36.7(1.2)	36.4(0.8)	36.9(1.8)	36.5(0.9)	36.8(1.4)	36.5(1.0)	37.0(1.2)	36.6(1.0)	37.2(2.1)
yeast	40.8(1.3)	40.7(1.4)	40.7(1.2)	41.0(2.1)	40.8(1.1)	41.4(1.9)	41.1(1.4)	41.2(1.5)	41.3(1.6)	41.1(1.7)	41.2(1.8)	41.1(1.3)	41.2(1.3)	41.6(1.8)
zoo	5.4(3.4)	7.4(4.6)	6.2(3.5)	6.9(6.8)	6.1(3.2)	5.5(3.6)	6.9(3.3)	7.6(7.5)	7.6(7.8)	8.0(7.5)	6.6(3.5)	5.8(2.7)	5.9(3.4)	6.4(3.6)

Bibliography

Alpaydin, E. (2014). *Introduction to Machine Learning*. Third edit. MIT Press, p. 640.

Balog, M., B. Lakshminarayanan, Z. Ghahramani, D. M. Roy, and Y. W. Teh (2016). "The Mondrian Kernel". In: *Proceedings of the 32nd conference on Uncertainty in Artificial Intelligence*, pp. 32–41. arXiv: arXiv:1606.05241v1.

Bartz-Beielstein, T., C. W. G. Lasarczyk, and M. Preuss (2005). "Sequential Parameter Optimization". In: *IEEE Congress on Evolutionary Computation*. Vol. 1, pp. 773–780.

Bell, R. M. and Y. Koren (2007). "Lessons from the Netflix prize challenge". In: *ACM SIGKDD Explorations Newsletter* 9.2, pp. 75–79.

Bengio, Y. and Y. Grandvalet (2004). "No Unbiased Estimator of the Variance of K-Fold Cross-Validation". In: *Journal of Machine Learning Research* 5, pp. 1089–1105.

Bergstra, J., R. Bardenet, Y. Bengio, and B. Kégl (2011). "Algorithms for Hyper-Parameter Optimization". In: *Advances in Neural Information Processing Systems*, pp. 2546–2554.

Bergstra, J. and Y. Bengio (2012). "Random Search for Hyper-Parameter Optimization". In: *Journal of Machine Learning Research* 13, pp. 281–305.

Bergstra, J. and D. D. Cox (2013). "Hyperparameter Optimization and Boosting for Classifying Facial Expressions: How good can a " Null " Model be?" In: *Proceedings of the ICML Workshop on Representation and Learning.*

Breiman, L. (2001). "Random Forests". In: *Machine Learning* 45, pp. 5–32.

Brown, G., J. L. Wyatt, and P. Tino (2005). "Managing Diversity in Regression Ensembles". In: *Journal of Machine Learning Research* 6, pp. 1621–1650.

Caruana, R., A. Niculescu-Mizil, G. Crew, and A. Ksikes (2004). "Ensemble Selection from Libraries of Models". In: *Proceedings of the 21st International Conference on Machine Learning*, pp. 137–144.

Cawley, G. C. and N. L. C. Talbot (2007). "Preventing Over-Fitting during Model Selection via Bayesian Regularisation of the Hyper-Parameters". In: *Journal of Machine Learning Research* 8, pp. 841–861.

Cawley, G. C. and N. L. C. Talbot (2010). “On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation”. In: *Journal of Machine Learning Research* 11, pp. 2079–2107.

Christiano, P., J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei (2017). “Deep reinforcement learning from human preferences”. In: *arXiv stat.ML*, p. 17. arXiv: 1706.03741.

Crammer, K. and Y. Singer (2001). “On The Algorithmic Implementation of Multiclass Kernel-based Vector Machines”. In: *Journal of Machine Learning Research (JMLR)* 2, pp. 265–292. DOI: 10.1162/15324430260185628.

Demšar, J. (2006). “Statistical Comparisons of Classifiers over Multiple Data Sets”. In: *Journal of Machine Learning Research* 7, pp. 1–30. DOI: 10.1016/j.jecp.2010.03.005.

Didaci, L., G. Fumera, and F. Roli (2013). “Diversity in Classifier Ensembles : Fertile Concept or Dead End ?” In: *Multiple Classifier Systems*, pp. 37–48.

Domhan, T., J. T. Springenberg, and F. Hutter (2015). “Speeding up Automatic Hyperparameter Optimization of Deep Neural Networks by Extrapolation of Learning Curves”. In: *Proceedings of the 24th International Conference on Artificial Intelligence*, pp. 3460–3468.

Dos Santos, E. M., R. Sabourin, and P. Maupin (2009). “Overfitting cautious selection of classifier ensembles with genetic algorithms”. In: *Information Fusion* 10.2, pp. 150–162. DOI: 10.1016/j.inffus.2008.11.003.

Durand, A. and C. Gagné (2017). “Estimating Quality in User-Guided Multi-Objective Bandits Optimization”. In: *arxiv cs.LG*, p. 18. arXiv: 1701.01095.

Duvenaud, D., J. R. Lloyd, R. Grosse, J. B. Tenenbaum, and Z. Ghahramani (2013). “Structure Discovery in Nonparametric Regression through Compositional Kernel Search”. In: *Proceedings of the 30th International Conference on Machine Learning*. Vol. 28, pp. 1166–1174. arXiv: arXiv:1302.4922v4.

Dwork, C., V. Feldman, O. Reingold, M. Hardt, A. Roth, and T. Pitassi (2015). “Generalization in Adaptive Data Analysis and Holdout Reuse”. In: *Advances in Neural Information Processing Systems*, pp. 2350–2358. DOI: 10.1126/science.aaa9375.

Eggensperger, K., M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. H. Hoos, and K. Leyton-Brown (2013). “Towards an Empirical Foundation for Assessing Bayesian Optimization of Hyperparameters”. In: *NIPS 2013 Workshop on Bayesian Optimization in Theory and Practice*.

Fernández-Delgado, M., E. Cernadas, S. Barro, and D. Amorim (2014). “Do we Need Hundreds of Classifiers to Solve Real World Classification Problems ?” In: *Journal of Machine Learning Research* 15, pp. 3133–3181.

Feurer, M., A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter (2015a). “Efficient and Robust Automaated Machine Learning”. In: *Advances in Neural Information Processing Systems*, pp. 2755–2763.

Feurer, M., J. T. Springenberg, and F. Hutter (2015b). “Initializing Bayesian Hyperparameter Optimization via Meta-Learning”. In: *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pp. 1128–1135.

Fortin, F.-A., F.-M. D. Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné (2012). “DEAP: Evolutionary Algorithms Made Easy”. In: *Journal of Machine Learning Research* 13, pp. 2171–2175.

Frank, A. and A. Asuncion (2010). *UCI Machine Learning Repository*. URL: `https://archive.ics.uci.edu/ml/datasets.html`.

Germain, P., A. Lacasse, F. Laviolette, M. Marchand, and J.-F. Roy (2015). “Risk Bounds for the Majority Vote : From a PAC-Bayesian Analysis to a Learning Algorithm”. In: *Journal of Machine Learning Research* 16, pp. 787–860. arXiv: `arXiv:1503.08329v1`.

Guyon, I., K. Bennett, G. Cawley, H. J. Escalante, S. Escalera, T. K. Ho, N. Macià, B. Ray, M. Saeed, A. Statnikov, and E. Viegas (2015). “Design of the 2015 ChaLearn AutoML Challenge”. In: *Proceedings of the 2015 International Joint Conference on Neural Networks*. ChaLearn, pp. 1–8.

Guyon, I., a. Saffari, G. Dror, and G. Cawley (2010). “Model Selection : Beyond the Bayesian / Frequentist Divide”. In: *Journal of Machine Learning Research* 11, pp. 61–87.

Hansen, N. (2005). *The CMA evolution strategy: A tutorial*. Tech. rep. Inria.

Hennig, P. and C. J. Schuler (2012). “Entropy Search for Information-Efficient Global Optimization”. In: *Journal of Machine Learning Research* 13, pp. 1809–1837.

Hernández-Lobato, J. M., M. W. Hoffman, and Z. Ghahramani (2014). “Predictive Entropy Search for Efficient Global Optimization of Black-box Functions”. In: *Advances in Neural Information Processing Systems*, pp. 918–926.

Hoffman, M. W., B. Shahriari, and N. de Freitas (2014). “On correlation and budget constraints in model-based bandit optimization with application to automatic machine learning”. In: *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics* 33, pp. 365–374.

Hutter, F., H. H. Hoos, and K. Leyton-Brown (2011). “Sequential Model-Based Optimization for General Algorithm Configuration”. In: *Learning and Intelligent Optimization* 5, pp. 507–523.

Hutter, F., H. Hoos, and K. Leyton-Brown (2013). “An evaluation of sequential model-based optimization for expensive blackbox functions”. In: *Proceeding of the GECCO Workshop on Blackbox Optimization Benchmarking*. ACM Press, pp. 1209–1216. DOI: 10.1145/2464576.2501592.

Igel, C. (2013). “A note on generalization loss when evolving adaptive pattern recognition systems”. In: *IEEE Transactions on Evolutionary Computation* 17.3, pp. 345–352. DOI: 10.1109/TEVC.2012.2197214.

Jones, D. R. (2001). “A Taxonomy of Global Optimization Methods Based on Response Surfaces”. In: *Journal of Global Optimization* 21, pp. 345–383. DOI: 10.1023/A:1012771025575.

Jones, D. R., M. Schonlau, and W. J. Welch (1998). “Efficient Global Optimization of Expensive Black-Box Functions”. In: *Journal of Global Optimization* 13, pp. 455–492. DOI: 10.1023/a:1008306431147.

Jun, K.-S., F. Orabona, R. Willett, and S. Wright (2016). “Improved Strongly Adaptive Online Learning using Coin Betting”. In: *arXiv stat.ML*, p. 13. arXiv: 1610.04578.

Klein, A., S. Falkner, S. Bartels, P. Hennig, and F. Hutter (2016). “Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets”. In: *arXiv cs.LG*, p. 17. arXiv: 1605.07079.

Krause, A. and D. Golovin (2014). “Submodular Function Maximization”. In: *Tractability*. Ed. by L. Bordeaux, Y. Hamadi, P. Kohli, and R. Mateescu. Vol. 3. Cambridge: Cambridge University Press, pp. 71–104. DOI: 10.1017/CBO9781139177801.004.

Kuncheva, L. I. (2003). “That Elusive Diversity in Classifier Ensembles”. In: *Pattern Recognition and Image Analysis*, pp. 1126–1138.

Kuncheva, L. I. (2004). *Combining Pattern Classifiers: Methods and Algorithms*. Wiler-Interscience, p. 384.

Kushner, H. (1964). “A new method of locating the maximum of an arbitrary multipeak curve in the presence of noise”. In: *Journal of Basic Engineering* 86, pp. 97–106.

Lacoste, A., H. Larochelle, M. Marchand, and F. Laviolette (2014a). “Agnostic Bayesian Learning of Ensembles”. In: *Proceedings of the 31st International Conference on Machine Learning*, pp. 611–619.

Lacoste, A., H. Larochelle, M. Marchand, and F. Laviolette (2014b). “Sequential Model-Based Ensemble Optimization”. In: *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence*, pp. 440–448.

Lakshminarayanan, B., D. M. Roy, and Y. W. Teh (2014). “Mondrian Forests: Efficient Online Random Forests”. In: *Advances in Neural Information Processing Systems*. Vol. 27, pp. 3140–3148. arXiv: arXiv:1406.2673v1.

Lakshminarayanan, B., D. M. Roy, and Y. W. Teh (2016). “Mondrian Forests for Large-Scale Regression when Uncertainty Matters”. In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. Vol. 51, pp. 1478–1487.

Lévesque, J.-C., A. Durand, C. Gagné, and R. Sabourin (2012). “Multi-Objective Evolutionary Optimization for Generating Ensembles of Classifiers in the ROC Space”. In: *Proceedings of the 14th Conference on Genetic and Evolutionary Computation*, pp. 879–886.

Lévesque, J.-C., A. Durand, C. Gagné, and R. Sabourin (2017). “Bayesian Optimization for Conditional Hyperparameter Spaces”. In: *Proceedings of the 2017 International Joint Conference on Neural Networks*, pp. 286–293. DOI: 10.1109/IJCNN.2017.7965867.

Lévesque, J.-C., C. Gagné, and R. Sabourin (2013). “Ensembles of Budgeted Kernel Support Vector Machines for Parallel Large Scale Learning”. In: *NIPS 2013 Workshop on Big Learning: Advances in Algorithms and Data Management*, p. 5.

Lévesque, J.-C., C. Gagné, and R. Sabourin (2016). “Bayesian Hyperparameter Optimization for Ensemble Learning”. In: *Proceedings of the 32nd Conference on Uncertainty in Artificial Intelligence*, pp. 437–446.

Li, L., K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar (2017). “Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization”. In: *Proceedings of the 5th International Conference on Learning Representations*. arXiv: 1603.06560.

Lloyd, J. R., D. Duvenaud, R. Grosse, J. B. Tenenbaum, and Z. Ghahramani (2014). “Automatic Construction and Natural-Language Description of Nonparametric Regression Models”. In: *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pp. 1242–1250. arXiv: arXiv:1402.4304v2.

Martínez-Muñoz, G., D. Hernández-Lobato, and A. Suárez (2009). “An analysis of ensemble pruning techniques based on ordered aggregation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.2, pp. 245–59. DOI: 10.1109/TPAMI.2008.78.

Mockus, J., V. Tiesis, and A. Zilinskas (1978). “The application of Bayesian methods for seeking the extremum”. In: *Towards Global Optimization* 2, pp. 117–129.

Murray, I. and R. P. Adams (2010). “Slice Sampling Covariance Hyperparameters of Latent Gaussian Models”. In: *Advances in Neural Information Processing Systems*, pp. 1732–1740. arXiv: arXiv:1006.0868v2.

Quinlan, J. R. and R. M. Cameron-Jones (1995). “Oversearching and Layered Search in Empirical Learning”. In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pp. 1019–1024.

Rasmussen, C. E. and C. K. I. Williams (2006). *Gaussian Processes for Machine Learning*. MIT Press, p. 266.

Roy, D. and Y. W. Teh (2009). “The Mondrian Process”. In: *Advances in Neural Information Processing Systems*, pp. 1377–1384. URL: http://eprints.pascal-network.org/archive/00004694/.

Schapire, R. E. and Y. Freund (2012). *Boosting: Foundations and Algorithms*. MIT Press, p. 544.

Shahriari, B., K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas (2016). “Taking the human out of the loop: A review of Bayesian optimization”. In: *Proceedings of the IEEE* 104.1, pp. 148–175. DOI: 10.1109/JPROC.2015.2494218. arXiv: arXiv:1011.1669v3.

Snoek, J., H. Larochelle, and R. P. Adams (2012). “Practical Bayesian Optimization of Machine Learning Algorithms”. In: *Advances in Neural Information Processing Systems*, pp. 2951–2959.

Snoek, J., O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. M. A. Patwary, Prabhat, and R. P. Adams (2015). “Scalable Bayesian Optimization Using Deep Neural Networks”. In: *Proceedings of the 32nd International Conference on Machine Learning*, pp. 2171–2180.

Snoek, J., K. Swersky, R. Zemel, and R. P. Adams (2014). “Input Warping for Bayesian Optimization of Non-Stationary Functions”. In: *Proceedings of the 31st International Conference on Machine Learning*, pp. 1674–1682. arXiv: arXiv:1402.0929v3.

Sobol, I. M. (1967). “On the Distribution of Points in a Cube and the Approximate Evaluation of Integrals”. In: *Zhurnal Vychislitel’noi Matematiki i Matematicheskoi Fiziki* 7.4, pp. 784–802.

Srinivas, N., A. Krause, and M. Seeger (2010). “Gaussian Process Optimization in the Bandit Setting : No Regret and Experimental Design”. In: *Proceedings of the 27th International Conference on Machine Learning*, pp. 1015–1022. arXiv: arXiv:0912.3995v4.

Sun, Q. and B. Pfahringer (2011). “Bagging Ensemble Selection”. In: *AI 2011: Advances in Artificial Intelligence* 7106, pp. 251–260.

Swersky, K., J. Snoek, and R. P. Adams (2013). “Multi-Task Bayesian Optimization”. In: *Advances in Neural Information Processing Systems*, pp. 2004–2012.

Swersky, K., J. Snoek, and R. P. Adams (2014). “Freeze-Thaw Bayesian Optimization”. In: *arXiv stat.ML*, p. 12. arXiv: arXiv:1406.3896v1.

Thornton, C., F. Hutter, Holger H. Hoos, and K. Leyton-brown (2013). “Auto-WEKA : Combined Selection and Hyperparameter Optimization of Classification Algorithms”. In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 847–855. arXiv: arXiv:1208.3719v2.

Tsamardinos, I., A. Rakhshani, and V. Lagani (2015). “Performance-Estimation Properties of Cross-Validation-Based Protocols with Simultaneous Hyper-Parameter Optimization”. In: *International Journal on Artificial Intelligence Tools* 24.5, pp. 1–14. DOI: 10.1142/S0218213015400230.

Tsymbal, A., M. Pechenizkiy, and P. Cunningham (2005). “Diversity in Search Strategies for Ensemble Feature Selection”. In: *Information Fusion* 6.1, pp. 83–98.

Wainer, J. and G. C. Cawley (2017). “Empirical Evaluation of Resampling Procedures for Optimising SVM Hyperparameters”. In: *Journal of Machine Learning Research* 18.15, pp. 475–509.

Wang, Z., C. Gehring, P. Kohli, and S. Jegelka (2018). “Batched Large-scale Bayesian Optimization in High-dimensional Spaces”. In: *International Conference on Artificial Intelligence and Statistics*. arXiv: 1706.01445.

Wang, Z., M. Zoghi, D. Matheson, F. Hutter, and N. de Freitas (2012). “Bayesian Optimization in a Billion Dimensions via Random Embeddings”. In: *Journal of Machine Learning Research* 55.1, pp. 361–387. arXiv: arXiv:1301.1942v1.

Wistuba, M., N. Schilling, and L. Schmidt-thieme (2017). “Automatic Frankensteining : Creating Complex Ensembles Autonomously”. In: *Proceedings of the 2017 SIAM International Conference on Data Mining*, pp. 741–749.

Wolpert, D. H. (1996). “The Lack of A Priori Distinctions Between Learning Algorithms”. In: *Neural Computation* 8.7, pp. 1341–1390. DOI: 10.1162/neco.1996.8.7.1341.

Zhang, C., S. Bengio, M. Hardt, B. Recht, and O. Vinyals (2017). “Understanding Deep Learning Requires Re-thinking Generalization”. In: *Proceedings of the 5th International Conference on Learning Representations*. arXiv: arXiv:1611.03530v1.