The Mondrian Kernel

Matej Balog[*]
Department of Engineering
University of Cambridge

Balaji Lakshminarayanan
Gatsby Unit
University College London

Zoubin Ghahramani
Department of Engineering
University of Cambridge

Daniel M. Roy
Department of Statistical Sciences
University of Toronto

Yee Whye Teh
Department of Statistics
University of Oxford

Abstract

We introduce the Mondrian kernel, a fast *random feature* approximation to the Laplace kernel. It is suitable for both batch and online learning, and admits a fast kernel-width-selection procedure as the random features can be re-used efficiently for all kernel widths. The features are constructed by sampling trees via a Mondrian process [Roy and Teh, 2009], and we highlight the connection to Mondrian forests [Lakshminarayanan et al., 2014], where trees are also sampled via a Mondrian process, but fit independently. This link provides a new insight into the relationship between kernel methods and random forests.

1 INTRODUCTION

Kernel methods such as support vector machines and Gaussian processes are very popular in machine learning. While early work relied on dual optimization, recent large-scale kernel methods focus on the primal optimization problem where the input data are mapped to a finite-dimensional feature space and the weights are learned using fast linear optimization techniques, e.g., stochastic gradient descent. Rahimi and Recht [2007] proposed to approximate shift-invariant kernels by mapping the inputs to so-called *random features*, constructed so that the inner product of two mapped data points approximates the kernel evaluated at those two points (which is the inner product in the feature space corresponding to the kernel). Rahimi and Recht [2007] proposed two random feature construction schemes: *random Fourier features*, where data points are projected onto random vectors drawn from the Fourier transform of the kernel and then passed through suitable non-linearities; and *random binning*, where the input space is partitioned by a random regular grid into bins and data points are mapped to indicator vectors identifying which bins they end up in. Both of these approaches require specifying the kernel hyperparameters in advance, so that the appropriate distribution is used for sampling the random vectors or random grids, respectively. However, a suitable kernel width (length-scale) is often not known a priori and is found by cross-validation, or, where available, marginal likelihood optimization. In practice, this entails constructing a new feature space and training a linear learner from scratch for each kernel width, which is computationally expensive. Using a suitable kernel width is often more important than the choice of kernel type [Schölkopf and Smola, 2001], so a fast kernel width selection method is desirable.

We describe a connection between the Laplace kernel and the Mondrian process [Roy and Teh, 2009], and leverage it to develop a random feature approximation to the Laplace kernel that addresses the kernel width selection problem. This approximation, which we call the *Mondrian kernel*, involves random partitioning of data points using a Mondrian process, which can be efficiently reused for all kernel widths. The method preserves the nonparametric nature of kernel learning and is also suitable for online learning.

The Mondrian kernel reveals an interesting link between kernel methods and decision forests [Breiman, 2001, Criminisi et al., 2012], another popular class of nonparametric methods for black-box prediction tasks. The Mondrian kernel resembles *Mondrian forests*, a decision-forest variant introduced by Lakshminarayanan et al. [2014], where a Mondrian process is used as the randomization mechanism. The efficiently trainable Mondrian forests excel in the online setting, where their distribution is identical to the corresponding batch Mondrian forest, and have been successfully applied to both classification and regression [Lakshminarayanan et al., 2014, 2016]. Mondrian forests and the Mondrian kernel both lead to randomized, non-linear learning algorithms whose randomness stems from a Mondrian process. The former fits parameters corresponding to different Mondrian trees independently, while the latter fits them jointly. We compare these methods theoretically and thus establish a novel connection between the Laplace kernel and Mondrian forests via the Mondrian kernel.

[*]Also affiliated with Max-Planck Institute for Intelligent Systems, Tübingen, Germany.

The contributions of this paper are:

- a review of the Mondrian process using the simple notion of competing exponential clocks (Section 2);
- a novel connection between the Mondrian process and the Laplace kernel (Section 3), yielding a fast approximation to learning with the Laplace kernel;
- an efficient procedure for learning the kernel width from data (Section 4); and
- a comparison between Mondrian kernel and Mondrian forest that provides another connection between kernel learning and random forests (Section 6).

2 MONDRIAN PROCESS

For completeness, we review the Mondrian process [Roy and Teh, 2009, Roy, 2011, Chapter 5]. Although simple and perhaps well known to experts, our exposition through competing exponential clocks has not explicitly appeared in this form in the literature. Readers familiar with the Mondrian process may skip this section on first reading.

2.1 TERMINOLOGY

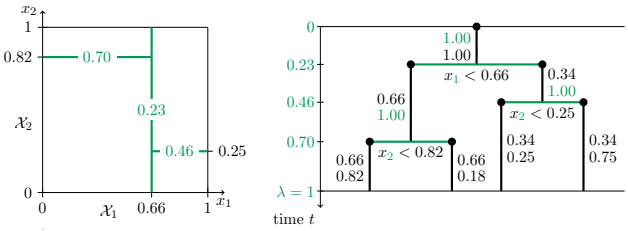
An *axis-aligned box* $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_D \subseteq \mathbb{R}^D$ is a Cartesian product of D bounded intervals $\mathcal{X}_d \subseteq \mathbb{R}$. Their total length $|\mathcal{X}_1| + \cdots + |\mathcal{X}_D|$ is the *linear dimension of* $\mathcal{X}$. A *guillotine partition* of $\mathcal{X}$ is a hierarchical partitioning of $\mathcal{X}$ using axis-aligned cuts. Such a partition can be naturally represented using a strictly binary tree.

An *exponential clock with rate* r takes a random time $T \sim \mathrm{Exp}(r)$ to ring after being started, where $\mathrm{Exp}(r)$ is the exponential distribution with rate (inverse mean) r. The notion of *competing exponential clocks* refers to D independent exponential clocks with rates $r_1, \ldots, r_D$, started at the same time. It can be shown that (1) the time until some clock rings has $\mathrm{Exp}(\sum r_d)$ distribution, (2) it is the d-th clock with probability proportional to r_d, and (3) once a clock rings, the remaining $D - 1$ clocks continue to run independently with their original distributions.

2.2 GENERATIVE PROCESS

The Mondrian process on an axis-aligned box $\mathcal{X} \subseteq \mathbb{R}^D$ is a time-indexed stochastic process taking values in guillotine-partitions of $\mathcal{X}$. It starts at time 0 with the trivial partition of $\mathcal{X}$ (no cuts) and as time progresses, new axis-aligned cuts randomly appear, hierarchically splitting $\mathcal{X}$ into more and more refined partitions. The process can be stopped at a lifetime $\lambda \in [0, \infty)$, which amounts to ignoring any cuts that would appear after time λ.

To describe the distribution of times and locations of new cuts as time progresses, we associate an independent exponential clock with rate $|\mathcal{X}_d|$ to each dimension d of $\mathcal{X}$. Let T be the first time when a clock rings and let d be the dimension of that clock. If $T > \lambda$ then this process terminates. Otherwise, a point a is chosen uniformly at random from $\mathcal{X}_d$ and $\mathcal{X}$ is split into $\mathcal{X}^< = \{\mathbf{x} \in \mathcal{X} \mid x_d < a\}$ and $\mathcal{X}^> = \{\mathbf{x} \in \mathcal{X} \mid x_d > a\}$ by a hyperplane in dimension d that is perpendicular to $\mathcal{X}_d$ at point a. After making this first cut, the remaining $D - 1$ clocks are discarded and the generative process restarts recursively and *independently* on $\mathcal{X}^<$ and $\mathcal{X}^>$. However, those processes start at time T rather than 0 and thus have less time left until the lifetime λ is reached.



(a) (b)

Figure 1: (a) Sample of a Mondrian process on the axis-aligned box $\mathcal{X} = [0, 1] \times [0, 1] \subseteq \mathbb{R}^2$ with lifetime $\lambda = 1.0$. Numbers on the cuts (shown in green) indicate the times when they appeared. The first cut appeared at time $T = 0.23$, in dimension $d = 1$, at location $a = 0.66 \in \mathcal{X}_1$. (b) Representing the Mondrian sample as a strictly binary tree, with new nodes (shown as circles) appearing as time (y-axis) progresses. The two numbers below each node show the rates of the two exponential clocks competing to split that node, with the winning clock's rate shown in green.

The specification of the generative process on $\mathcal{X}$ is now complete. Due to the properties of competing exponential clocks, the time until the first cut appears in $\mathcal{X}$ has exponential distribution with rate equal to the linear dimension of $\mathcal{X}$ and the dimension d in which the cut is made is chosen proportional to $|\mathcal{X}_d|$. This confirms equivalence of our generative process to the one proposed by Roy and Teh [2009]. Finally, we note that a.s. the Mondrian process does not explode, i.e., for every lifetime $\lambda \in [0, \infty)$, the process generates finitely many cuts with probability 1 [Roy, 2011].

2.3 PROJECTIVITY

If a Mondrian process runs on $\mathcal{X}$, what distribution of random partitions does it induce on an axis-aligned subbox $\mathcal{A} \subseteq \mathcal{X}$? (See Figure 2a for an illustration in $D = 2$ dimensions.) The Mondrian process was constructed so that the answer is the Mondrian process itself [Roy, 2011]. Here we explain this projectivity property using the notion of competing exponential clocks. To argue that the resulting process on $\mathcal{A}$ is indeed a Mondrian process, we show that the process running on $\mathcal{X}$ generates cuts in $\mathcal{A}$ in the same way as a Mondrian process running directly on $\mathcal{A}$ would.

Recall that each dimension d of $\mathcal{X}$ is associated with an exponential clock with rate $|\mathcal{X}_d|$ and if it rings first, the cut location is chosen uniformly at random from $\mathcal{X}_d$. This procedure can be equivalently represented using two competing clocks for each dimension (rather than just one):

- Clock $\mathcal{C}^d_{\mathcal{A}}$ with rate $|\mathcal{A}_d|$. If this clock rings first, the cut location is chosen uniformly at random from $\mathcal{A}_d$.
- Clock $\mathcal{C}^d_{\neg\mathcal{A}}$ with rate $|\mathcal{X}_d| - |\mathcal{A}_d|$. If it rings first, the cut location is sampled uniformly from $\mathcal{X}_d \setminus \mathcal{A}_d$.

(See Figure 2b.) Note that the clocks $\mathcal{C}^1_{\mathcal{A}}, \ldots, \mathcal{C}^D_{\mathcal{A}}$ represent the same cut distribution as a Mondrian process running on $\mathcal{A}$ would. If a clock $\mathcal{C}^d_{\neg\mathcal{A}}$ rings first, a cut is made outside of $\mathcal{A}$ and all of $\mathcal{A}$ remains on one side of this cut. None of the clocks $\mathcal{C}^d_{\mathcal{A}}$ have rung in that case and would usually be discarded and replaced with fresh clocks of identical rates, but by property (3) of competing exponential clocks, we can equivalently reuse these clocks (let them run) on the side of the cut containing $\mathcal{A}$. (Figure 2c shows a cut in dimension $d = 1$ that misses $\mathcal{A}$ and the reused clocks $\mathcal{C}^d_{\mathcal{A}}$). Hence, cuts outside $\mathcal{A}$ do not affect the distribution of the first cut crossing $\mathcal{A}$, and this distribution is the same as if a Mondrian process were running just on $\mathcal{A}$. When a cut is made within $\mathcal{A}$ (see Figure 2d), the process continues on both sides recursively and our argument proceeds inductively, confirming that the Mondrian process on $\mathcal{X}$ generates cuts in $\mathcal{A}$ in the same way as a Mondrian process on $\mathcal{A}$ would.

2.4 MONDRIAN PROCESS ON $\mathbb{R}^D$

The Mondrian process on $\mathbb{R}^D$ is defined implicitly as a time-indexed stochastic process such that its restriction to any axis-aligned box $\mathcal{X} \subseteq \mathbb{R}^D$ is a Mondrian process as defined in section 2.2. Fortunately, this infinite-dimensional object can be compactly represented by instantiating the Mondrian process only in regions where we have observed data. As we observe new data points, the Mondrian sample can be extended using the *conditional Mondrian* algorithm [Roy and Teh, 2009], a simple and fast sampling procedure for extending a Mondrian sample in an axis-aligned box $\mathcal{A}$ to a larger axis-aligned box $\mathcal{X} \supseteq \mathcal{A}$. The conditional Mondrian is useful for online learning and prediction, as it can be used to extend Mondrian samples to (yet) unobserved parts of the input space [Lakshminarayanan et al., 2014].

3 MONDRIAN KERNEL

For concreteness, our running example will be regression: the problem of learning a function $f : \mathbb{R}^D \to \mathbb{R}$ from a set of N training examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$. However, the Mondrian kernel applies equally well to classification, or any other learning task.

Learning with kernels involves choosing a kernel function $k : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ to act as a similarity measure between input data points. Evaluating $k(\cdot, \cdot)$ on all pairs of N data

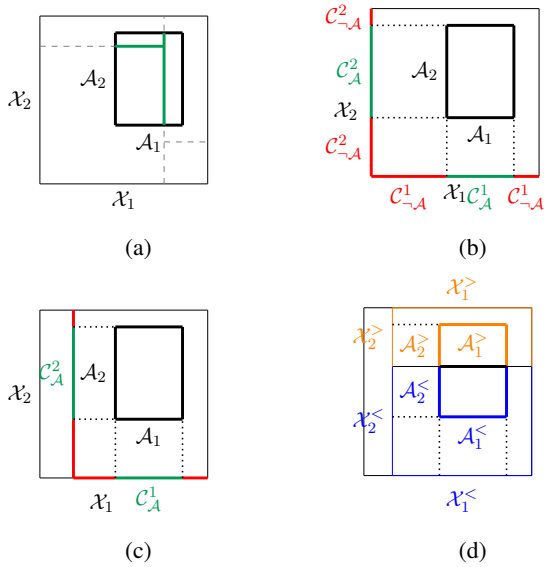


Figure 2: (a) A Mondrian process running on $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2$ generates cuts (dashed lines), some of which intersect $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$ (green lines) and thus induces a random partition of $\mathcal{A}$. (b) Representing the first cut distribution using $2D = 4$ competing exponential clocks: in each dimension d, clock $\mathcal{C}^d_{\mathcal{A}}$ corresponds to the region where making a cut splits $\mathcal{A}$ (shown in green) and clock $\mathcal{C}^d_{\neg\mathcal{A}}$ to the (disconnected) region where making a cut misses $\mathcal{A}$ (shown in red). (c) Cut outside $\mathcal{A}$: reusing the clocks $\mathcal{C}^1_{\mathcal{A}}$, $\mathcal{C}^2_{\mathcal{A}}$ on the side of the cut containing $\mathcal{A}$. (d) Cut inside $\mathcal{A}$ (shown in black): the argument proceeds by induction on both sides.

points takes $\Omega(N^2)$ operations, with some models also requiring a $\Theta(N^3)$ operation on an $N \times N$ kernel matrix. This generally makes exact kernel methods unsuitable for large-scale learning. Rahimi and Recht [2007] proposed a fast approximation through a randomized construction of a low-dimensional feature map $\phi : \mathbb{R}^D \to \mathbb{R}^C$ such that

$$\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^D \qquad k(\mathbf{x}, \mathbf{x}') \approx \phi(\mathbf{x})^T \phi(\mathbf{x}')$$

and then using a linear learning method in the feature space $\mathbb{R}^C$ implied by ϕ. For example, linear regression $\mathbf{y} \approx \mathbf{\Phi w}$, where $\mathbf{\Phi} \in \mathbb{R}^{N \times C}$ is the feature matrix with n-th row $\phi(\mathbf{x}_n)^T$, is solvable exactly in time linear in N. In general, the primal problem also lends itself naturally to stochastic gradient descent approaches for learning $\mathbf{w}$.

We use the Mondrian process to construct a randomized feature map for the (isotropic) *Laplace kernel*:

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\lambda \|\mathbf{x} - \mathbf{x}'\|_1) = \exp(-\lambda \sum_{d=1}^{D} |x_d - x'_d|).$$

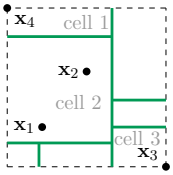
Here $\lambda \geq 0$ is the inverse kernel width (inverse length-scale), which we call the *lifetime* parameter of the kernel. We use a non-standard parametrization as this lifetime parameter will be linked to the Mondrian process lifetime.

3.1 MONDRIAN KERNEL

Consider the following randomized construction of a feature map $\phi : \mathbb{R}^D \to \mathbb{R}^C$:

1. Sample a partition of $\mathbb{R}^D$ via a Mondrian process on $\mathbb{R}^D$ with lifetime λ. Label the cells of the generated partition by $1, 2, \ldots$ in arbitrary order.
2. To encode a data point $\mathbf{x} \in \mathbb{R}^D$, look up the label c of the partion cell $\mathbf{x}$ falls into and set $\phi(\mathbf{x})$ to be the (column) indicator vector that has a single non-zero entry at position c, equal to 1.

The Mondrian process on $\mathbb{R}^D$ generates infinitely many partition cells and cannot be stored in memory, but projectivity comes to the rescue. As we only ever need to evaluate ϕ on finitely many data points, it suffices to run the Mondrian on the smallest axis-aligned box containing all these points. Also, we only label partition cells containing at least one data point, in effect removing features that would be 0 for all our data points. Then, the dimensionality C of ϕ equals the number of non-empty partiton cells and each data point has a single non-zero feature, equal to 1.



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

Figure 3: Feature expansions of 4 data points in $\mathbb{R}^2$.

However, note that the set of points on which the feature map ϕ is evaluated need not be known in advance and can even grow in an online fashion. Indeed, the conditional Mondrian algorithm discussed in section 2.4 allows us to extend Mondrian samples to larger boxes as necessary, and we can increase the dimensionality of ϕ whenever a data point is added to a previously empty partition cell.

This feature map ϕ induces a kernel

$$k_1(\mathbf{x}, \mathbf{x}') := \phi(\mathbf{x})^T \phi(\mathbf{x}') = \begin{cases} 1 & \text{if } \mathbf{x}, \mathbf{x}' \text{ in same partition cell} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

which we call a *Mondrian kernel* of order 1.

Instead of using a single Mondrian sample (partition), we can use M independent samples and construct a feature map ϕ by concatenating and normalizing the feature maps $\phi^{(1)}, \ldots, \phi^{(M)}$ obtained from each individual sample as above:

$$\phi(\mathbf{x}) := \frac{1}{\sqrt{M}} \left[\phi^{(1)}(\mathbf{x})^T \quad \cdots \quad \phi^{(M)}(\mathbf{x})^T \right]^T. \quad (2)$$

This feature expansion is *sparse*: every data point has exactly M non-zero features. The corresponding kernel, which we call a *Mondrian kernel of order* M, is

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

This is the empirical frequency with which points $\mathbf{x}$ and $\mathbf{x}'$ end up in the same partition cell of a Mondrian sample.

Algorithm 1 Mondrian kernel

1: **for** $m = 1$ **to** M **do**
2: construct feature map $\phi^{(m)}$ ▷ *section 3.1*
3: join and rescale $\phi^{(1)}, \ldots, \phi^{(M)}$ into ϕ ▷ *equation (2)*
4: map data $\mathbf{X}$ to feature representations $\mathbf{\Phi}$ using ϕ
5: use linear learning method on $\mathbf{\Phi}$

3.2 MONDRIAN–LAPLACE LINK

By independence of the M Mondrian samples, a.s.

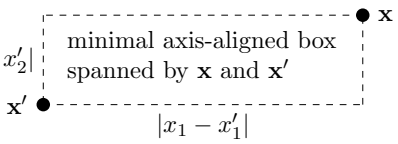
$$\lim_{M \to \infty} k_M(\mathbf{x}, \mathbf{x}') = \mathbb{E}\left[\phi^{(1)}(\mathbf{x})^T \phi^{(1)}(\mathbf{x}') \right] = \mathbb{E}\left[k_1(\mathbf{x}, \mathbf{x}') \right]$$

with convergence at the standard rate $\mathcal{O}_p(M^{-1/2})$. We thus define the *Mondrian kernel of order* ∞ as

$$k_\infty(\mathbf{x}, \mathbf{x}') := \mathbb{E}[k_1(\mathbf{x}, \mathbf{x}')].$$

Proposition 1 (Mondrian-Laplace link)**.** *The Mondrian kernel of order* ∞ *coincides with the Laplace kernel.*

Proof. As $k_1(\mathbf{x}, \mathbf{x}')$ (defined in (1)) is a binary random variable, $k_\infty(\mathbf{x}, \mathbf{x}')$ equals the probability that $\mathbf{x}$ and $\mathbf{x}'$ fall into the same partition cell of a Mondrian sample, which is equivalent to the sample having no cut in the minimal axis-aligned box spanned by $\mathbf{x}$ and $\mathbf{x}'$.



By projectivity, this probability is the same as the probability of not observing any cuts in a Mondrian process with lifetime λ running on just this minimal box. Noting that the linear dimension of this box is $\|\mathbf{x} - \mathbf{x}'\|_1$, we obtain

$$\begin{aligned} k_\infty(\mathbf{x}, \mathbf{x}') &= \mathbb{P}(\text{no cut between } \mathbf{x}, \mathbf{x}' \text{ until time } \lambda) \\ &= \mathbb{P}\left(T > \lambda\right) \text{ where } T \sim \text{Exp}\left(\|\mathbf{x} - \mathbf{x}'\|_1\right) \\ &= e^{-\lambda \|\mathbf{x} - \mathbf{x}'\|_1}. \end{aligned}$$

□

Note that the lifetime (inverse width) λ of the Laplace kernel corresponds to the lifetime of the Mondrian process used in the construction of the Mondrian kernel.

This link allows us to approximate the Laplace kernel with a Mondrian kernel k_M, which, unlike the Laplace kernel, admits a finite-dimensional feature expansion. The finite order M trades off kernel approximation error and computational costs (indirectly through the complexity of ϕ).

The following result confirms that the convergence of the Mondrian kernel approximation is exponentially fast in M *uniformly* on any fixed bounded input domain $\mathcal{X}$.

Proposition 2. *For any bounded input domain $\mathcal{X} \subseteq \mathbb{R}^D$ and $\delta > 0$, as $M \to \infty$,*

$$\mathbb{P}\left[\sup_{\mathbf{x},\mathbf{x}' \in \mathcal{X}} |k_M(\mathbf{x},\mathbf{x}') - k_\infty(\mathbf{x},\mathbf{x}')| > \delta\right] = \mathcal{O}\left(M^{2/3} e^{-M\delta^2/(12D+2)}\right).$$

Proof. Given in Supplement A. □

4 FAST KERNEL WIDTH LEARNING

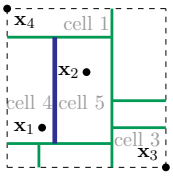
This section discusses the main advantage of our Mondrian approximation to the Laplace kernel: the efficient learning of kernel width from data. In particular, the approximation allows for efficient evaluation of all kernel lifetimes (inverse widths) $\lambda \in [0, \Lambda]$, where the terminal lifetime $\Lambda > 0$ need not be fixed a priori.

4.1 FEATURE SPACE REUSAL

We make the following recollections from earlier sections:

- the Mondrian process runs through time, starting at time 0 and only refining the generated partition as time progresses (cuts are never removed)
- the Mondrian process with lifetime λ is obtained by ignoring any cuts that would occur after time λ
- the lifetime λ of the Mondrian process used in constructing an explicit feature map ϕ for a Mondrian kernel corresponds to the lifetime (inverse width) of the Laplace kernel that it approximates

Running the Mondrian process from time 0 to some terminal lifetime Λ thus sweeps through feature spaces approximating all Laplace kernels with lifetimes $\lambda \in [0, \Lambda]$. More concretely, we start with $\lambda = 0$ and ϕ the feature map corresponding to M trivial partitions, i.e., for any data point $\mathbf{x}$, the vector $\phi(\mathbf{x})$ has length M and all entries set to the normalizer $M^{-1/2}$. As we increase λ, at discrete time points new cuts appear in the M Mondrian samples used in constructing ϕ. Suppose that at some time λ, the partition cell corresponding to the c-th feature in ϕ is split into two by a new cut that first appeared at this time λ. We update the feature map ϕ by removing the c-th feature and appending two new features, one for each partition cell created by the split. See Figure 4 for an example with $M = 1$.



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

Figure 4: A new cut (shown in thick blue) appeared, splitting cell $c = 2$ (cf. Figure 3) into two new cells $c = 4$ and $c = 5$. The table shows the update to ϕ, with the removed feature in gray italics and the two new features in bold blue.

This procedure allows us to approximate all Laplace kernels with lifetimes $\lambda \in [0, \Lambda]$ without having to resample new feature spaces for each lifetime. The total computational cost is the same (up to a multiplicative constant) as of constructing a single feature space just for the terminal lifetime Λ. This is because a strictly binary tree with $C^{(m)}$ leaves (partition cells in the m-th Mondrian sample at time Λ) contains at most $C^{(m)} - 1$ internal nodes (features that had to be removed at some time point $\lambda < \Lambda$).

4.2 LINEAR LEARNER RETRAINING

Evaluating suitability of a lifetime (inverse kernel width) λ requires training and evaluating a linear model in the feature space implied by ϕ. This can also be done more efficiently than retraining a new model from scratch every time a new cut is added and ϕ updated. We discuss the example of ridge regression with exact solutions, and a general case of models trainable using gradient descent methods.

4.2.1 Ridge regression

The MAP weights of the primal ridge regression problem are $\hat{\mathbf{w}} = \mathbf{A}^{-1}\mathbf{\Phi}^T\mathbf{y}$, where $\mathbf{A} := (\mathbf{\Phi}^T\mathbf{\Phi} + \delta^2\mathbf{I}_C)$ is the regularized feature covariance matrix and δ^2 is the regularization hyperparameter. Instead of inverting $\mathbf{A}$, it is numerically more stable to work with its Cholesky factor $\mathrm{chol}(\mathbf{A})$ [Seeger, 2003]. Phrasing the problem as Bayesian linear regression with, say, observation noise variance $\sigma_y^2 = \delta^2$ and prior weights variance $\sigma_w^2 = 1$, we can also obtain the log marginal likelihood $\mathcal{L}(\lambda)$ of the form

$$\mathcal{L}(\lambda) = -\frac{\|\mathbf{y} - \mathbf{\Phi}\hat{\mathbf{w}}\|_2^2}{2\delta^2} - \frac{\|\hat{\mathbf{w}}\|_2^2}{2} - \frac{1}{2}\ln\det\mathbf{A} + \text{const},$$

where the dependence on λ is implicit through ϕ.

When a new cut appears in one of the M Mondrian samples and ϕ is updated by deleting the c-th feature and appending two new ones, the corresponding update to the regularized feature covariance matrix $\mathbf{A}$ is to delete its c-th row and c-th column, and append two new rows and columns. Then both $\mathbf{A}^{-1}$ and $\mathrm{chol}(\mathbf{A})$ can be appropriately updated in $\mathcal{O}(C^2)$ time, faster than $\mathcal{O}(C^3)$ recomputation from scratch. Updating the Cholesky factor when the c-th row and column

are removed is slightly involved but can be achieved by first permuting the rows and columns so that the ones to be removed are the last ones [Seeger, 2004], after which the Cholesky factor is updated by deleting its last row and column. If C is the number of features at the terminal lifetime Λ, this $\mathcal{O}(C^2)$ update is performed $\mathcal{O}(C)$ times, for a total computational cost $\mathcal{O}(C^3)$. Note that performing the inversion or Cholesky factorization at just the terminal lifetime Λ would have the same time complexity.

After updating $\mathbf{A}^{-1}$ or $\mathrm{chol}(\mathbf{A})$, the optimal weights $\hat{\mathbf{w}}$ can be updated in $\mathcal{O}(C^2 + N)$ time and the determinant of $\mathbf{A}$ required for the marginal likelihood $\mathcal{L}(\lambda)$ can be obtained from $\mathrm{chol}(\mathbf{A})$ as the squared product of its diagonal elements in $\mathcal{O}(C)$ time. Exploiting sparsity of ϕ, evaluating the model on N_{test} data points takes $\mathcal{O}(N_{\text{test}}M)$ time.

Finally, we note that computing the marginal likelihood $\mathcal{L}(\lambda)$ for all $\lambda \in [0, \Lambda]$ and combining it with a prior $p(\lambda)$ supported on $[0, \Lambda]$ allows Bayesian inference over the kernel width λ^{-1}. We refer to Supplement B for more details.

4.2.2 Models trainable using gradient descent

Consider a linear model trained using a gradient descent method. If (an approximation to) the optimal weight vector $\mathbf{w}$ is available and then ϕ is updated by removing the c-th feature and appending two new features, a natural way of reinitializing the weights for subsequent gradient descent iterations is to remove the c-th entry of $\mathbf{w}$ and append two new entries, both set to the removed value (as points in the split cell are partitioned into the two new cells, this preserves all model predictions). Note that we have the freedom of choosing the number of gradient descent iterations after each cut is added, and we can opt to only evaluate the model (on a validation set, say) at several λ values on the first pass through $[0, \Lambda]$. One iteration of stochastic gradient descent takes $O(M)$ time thanks to sparsity of ϕ.

This efficient kernel width selection procedure can be especially useful with models where hyperparameters cannot be tweaked by marginal likelihood optimization (e.g., SVM).

5 ONLINE LEARNING

In this section, we describe how the Mondrian kernel can be used for online learning. When a new data point $\mathbf{x}_{N+1} \in \mathbb{R}^D$ arrives, incorporating it into M existing Mondrian samples (using the conditional Mondrian algorithm discussed in section 2.4) can create $0 \leq k \leq M$ new non-empty partition cells, increasing the dimensionality of the feature map ϕ. We set the new features to 0 for all previous data points $\mathbf{x}_1, \ldots, \mathbf{x}_N$.

In our running example of ridge regression, exact primal updates can again be carried out efficiently. The inverse $\mathbf{A}^{-1}$ or Cholesky factor $\mathrm{chol}(\mathbf{A})$ of the regularized feature covariance matrix $\mathbf{A}$ can be updated in two steps:

1. extend $\mathbf{A}^{-1}$ or $\mathrm{chol}(\mathbf{A})$ to incorporate the k new features (set to 0 for all existing data points) in $\mathcal{O}(C^2)$
2. incorporate the new data point $\mathbf{x}_{N+1}$, which is now a simple rank-1 update on $\mathbf{A}$, so $\mathbf{A}^{-1}$ or $\mathrm{chol}(\mathbf{A})$ can again be updated efficiently in $\mathcal{O}(C^2)$ time

We refer to Supplement C for more details.

With gradient descent trainable models, we maintain (an approximation to) the optimal weights $\mathbf{w}$ directly. When a new data point arrives, we expand the dimensionality of ϕ as described above. The previously optimal weights can be padded with 0's in any newly added dimensions, and then passed to the gradient descent method as initialization.

6 LINK TO MONDRIAN FOREST

We contrast Mondrian kernel with Mondrian forest [Lakshminarayanan et al., 2014, 2016], another non-linear learning method based on the Mondrian process. They both start by sampling M independent Mondrians on $\mathbb{R}^D$ to provide M independent partitions of the data. However, these partitions are then used differently in the two models:

- In a Mondrian forest, parameters of predictive distributions in each tree are fitted independently of all other trees. The prediction of the forest is the average prediction among the M trees.
- With Mondrian kernel, the weights of all random features are fitted jointly by a linear learning method.

Let $C^{(m)}$ count the leaves (non-empty partition cells) in the m-th Mondrian sample and let $C = \sum_{m=1}^{M} C^{(m)}$ be the total number of leaves. Let $\boldsymbol{\phi}_n^{(m)} := \phi^{(m)}(\mathbf{x}_n) \in \mathbb{R}^{C^{(m)}}$ be the indicator of the partition cell in the m-th sample into which the n-th data point falls (as in section 3.1). Also, as in equation (2), let $\boldsymbol{\phi}_n := \phi(\mathbf{x}_n) \in \mathbb{R}^C$ be the normalized concatenated feature encoding of the n-th data point. Recall that each vector $\boldsymbol{\phi}_n \in \mathbb{R}^C$ contains exactly M nonzero entries, all of which equal the normalizer $M^{-1/2}$.

For simplicity, we restrict our attention to ridge regression in this section and compare the learning objective functions of Mondrian kernel and Mondrian forest.

6.1 MONDRIAN KERNEL OBJECTIVE

The primal ridge regression problem in the feature space implied by ϕ is

$$\min_{\mathbf{w} \in \mathbb{R}^C} \sum_{n=1}^{N} (y_n - \mathbf{w}^T \boldsymbol{\phi}_n)^2 + \delta^2 \|\mathbf{w}\|_2^2.$$

Decomposing $\mathbf{w} = M^{-1/2}[\mathbf{w}^{(1)T} \cdots \mathbf{w}^{(M)T}]^T$, so that each (rescaled) subvector $\mathbf{w}^{(m)}$ corresponds to features from the m-th Mondrian, denoting by $\hat{y}_n^{(m)} := \mathbf{w}^{(m)T} \boldsymbol{\phi}_n^{(m)}$

the "contribution" of the m-th Mondrian to the prediction at the n-th data point, and writing $\text{loss}(y, \hat{y}) := (y - \hat{y})^2$, the Mondrian kernel objective function can be restated as

$$\min_{\mathbf{w}\in\mathbb{R}^C} \sum_{n=1}^{N} \text{loss}\left(y_n, \frac{1}{M}\sum_{m=1}^{M} \hat{y}_n^{(m)}\right) + \delta^2\|\mathbf{w}\|_2^2. \quad (3)$$

6.2 MONDRIAN FOREST OBJECTIVE

Assuming a factorizing Gaussian prior over the leaves in each Mondrian tree (i.e., without the hierarchical smoothing used by Lakshminarayanan et al. [2016]), the predictive mean parameters $\mathbf{w}^{(m)}$ in the leaves of the m-th Mondrian tree are fitted by minimizing

$$\min_{\mathbf{w}^{(m)}\in\mathbb{R}^{C^{(m)}}} \sum_{n=1}^{N} (y_n - \mathbf{w}^{(m)T}\boldsymbol{\phi}_n^{(m)})^2 + \gamma^2\|\mathbf{w}^{(m)}\|_2^2$$

where γ^2 is the ratio of noise and prior variance in the predictive model. The parameters $\mathbf{w}^{(m)}$ are disjoint for different trees, so these M independent optimization problems are equivalent to minimizing the average of the M individual objectives. Writing $\hat{y}_n^{(m)} := \mathbf{w}^{(m)T}\boldsymbol{\phi}_n^{(m)}$ for the m-th tree's prediction at the n-th data point and concatenating the parameters $\mathbf{w} := M^{-1/2}[\mathbf{w}^{(1)T} \cdots \mathbf{w}^{(M)T}]^T$, the Mondrian forest objective can be stated as

$$\min_{\mathbf{w}\in\mathbb{R}^C} \sum_{n=1}^{N} \frac{1}{M}\sum_{m=1}^{M} \text{loss}(y_n, \hat{y}_n^{(m)}) + \gamma^2\|\mathbf{w}\|_2^2. \quad (4)$$

6.3 DISCUSSION

Comparing (3) and (4), we see that subject to regularization parameters (priors) chosen compatibly, the two objectives only differ in the contribution of an individual data point n to the total loss:

$$\text{Mondrian kernel:} \quad \text{loss}\left(y_n, \frac{1}{M}\sum_{m=1}^{M} \hat{y}_n^{(m)}\right)$$

$$\text{Mondrian forest:} \quad \frac{1}{M}\sum_{m=1}^{M} \text{loss}(y_n, \hat{y}_n^{(m)})$$

Specifically, the difference is in the order in which the averaging $\frac{1}{M}\sum_{m=1}^{M}$ over Mondrian samples/trees and the nonlinear loss function are applied. In both models predictions are given by $\hat{y} = \frac{1}{M}\sum_{m=1}^{M} \hat{y}^{(m)}$, so the Mondrian kernel objective is consistent with the aim of minimizing empirical loss on the training data, while the forest objective minimizes average loss across trees, not the loss of the actual prediction (when $M > 1$) [Ren et al., 2015].

Ren et al. [2015] address this inconsistency between learning and prediction by proposing to extend random forests with a *global refinement* step that optimizes all tree parameters jointly, minimizing the empirical training loss. Our approximation of the Laplace kernel via the Mondrian kernel can be interpreted as implementing this joint parameter fitting step on top of Mondrian forest, revealing a new connection between random forests and kernel methods.

7 RELATED WORK

The idea of Rahimi and Recht [2007] to approximate shift-invariant kernels by constructing random features has been further developed by Le et al. [2013] and Yang et al. [2015], providing a faster method of constructing the random features when the input dimension D is high. The fast method of Dai et al. [2014] can adapt the number of random features, making it better-suited for streaming data. To the best of our knowledge, these methods require random features to be reconstructed from scratch for each new kernel width value; however, our solution allows us to efficiently learn this hyperparameter for the Laplace kernel.

Decision forests are popular for black-box classification and regression thanks to their competitive accuracy and computational efficiency. The most popular variants are Breiman's Random Forest [Breiman, 2001] and Extremely Randomized Trees [Geurts et al., 2006]. Breiman [2000] established a link between the Laplace kernel and random forests with an infinite number of trees, but unlike our work, made two additional strong assumptions, namely infinite data and a uniform distribution of features. From a computational perspective, Shen et al. [2006] approximated evaluation of an isotropic kernel using kd-trees, reducing computational complexity as well as memory requirements. Davies and Ghahramani [2014] constructed 'supervised' kernels using random forests and demonstrated that this can lead to linear-time inference. We refer to [Scornet, 2015] for a recent discussion on the connection between decision forests and kernel methods.

A key difference between decision forests and kernel methods is whether parameters are fit independently or jointly. In decision forests, the leaf node parameters for each tree are fit independently, whereas the weights of random features are fit jointly. Scornet [2015] shows that random forests can be interpreted as adaptive kernel estimates and discusses the theoretical properties of fitting parameters jointly. Ren et al. [2015] propose to extend random forests with a *global refinement* step, optimizing all tree parameters jointly to minimize empirical training loss.

The proposed Mondrian kernel establishes a link between Mondrian trees and Laplace kernel for finite data, without any assumptions on the distribution of the features. Unlike prior work, we exploit this connection to construct an adaptive random feature approximation and efficiently learn the kernel width.

8 EXPERIMENTS

We conducted three sets of experiments, with these goals:

1. verify that Mondrian kernel approximates the Laplace kernel, and compare to other random feature generation schemes (Section 8.1);
2. demonstrate usefulness of our efficient kernel width selection procedure, showing that it can quickly learn a suitable kernel width from data (Section 8.2); and
3. empirically compare the Mondrian kernel and Mondrian forests, supporting the insight into their relationship from Section 6 (Section 8.3).

With the exception of two experiments on synthetic data, we carried out our evaluation on the CPU dataset from [Rahimi and Recht, 2007], containing $N = 6554$ training and $N_{\text{test}} = 819$ test points with $D = 21$ attributes. Note that the CPU dataset is an adversarial choice here, as Rahimi and Recht [2007] report that random Fourier features perform better than binning schemes on this task. In all experiments, the ridge regularization constant was set to $\delta^2 = 10^{-4}$, the value used by Rahimi and Recht [2007], and the primal optimization problems were solved using stochastic gradient descent.

8.1 LAPLACE KERNEL APPROXIMATION

First we examined the absolute kernel approximation error $|k_\infty(\cdot,\cdot) - k_M(\cdot,\cdot)|$ directly. To this end, we sampled $N = 100$ data points uniformly at random in the unit square $[0,1]^2$ and computed the maximum absolute error over all N^2 pairs of points. The Laplace kernel k_∞ and Mondrian kernels k_M had a common lifetime (inverse width) $\lambda = 10$, so that several widths fit into the input domain $[0,1]^2$. We repeated the experiment 5 times for each value of M, showing the results in Figure 5. We plot the maximum error against the number M of non-zero features per data point, which is relevant for solvers such as Pegasos SVM [Shalev-Shwartz et al., 2007], whose running time scales with the number of non-zero features per data point. Under this metric, the Mondrian kernel and Random binning converged to the Laplace kernel faster than random Fourier features, showing that in some cases they can be a useful option. (The error of Random Fourier features would decrease faster when measured against the *total* number of features, as Mondrian kernel and Random binning generate sparse feature expansions.)

Second, we examined the approximation error indirectly via test set error on the CPU dataset. We repeated the experiment 5 times for each value of M and show the results in Figure 6. Even though Fourier features are better suited to this task, for a fast approximation with few ($M < 15$) non-zero features per data point, random binning and Mondrian kernel are still able to outperform the Fourier features.

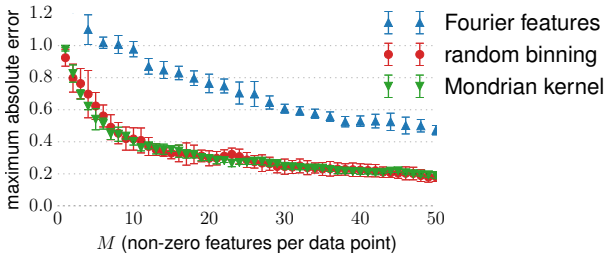


Figure 5: Maximum absolute kernel approximation error on all pairs of $N = 100$ data points in $[0,1]^2$.

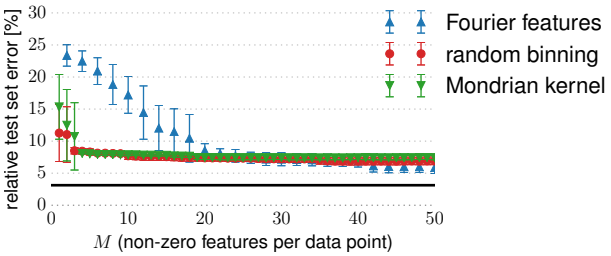


Figure 6: Test set error on the CPU dataset. The horizontal line at 3.1% indicates the error achieved with an exact, but expensive computation using the Laplace kernel.

8.2 FAST KERNEL WIDTH LEARNING

First, using a synthetic regression dataset generated from a Laplace kernel with known ground truth lifetime $\lambda_0 = 10$, we verified that the lifetime could be recovered using our kernel width selection procedure from Section 4. To this end, we let the procedure run until a terminal lifetime $\Lambda = 100$ and plotted the error on a held-out validation set as a function of the lifetime λ. The result in Figure 7 shows that the ground truth kernel lifetime $\lambda_0 = 10$ was recovered within an order of magnitude by selecting the lifetime $\hat{\lambda}$ minimizing validation set error. Moreover, this value of $\hat{\lambda}$ led to excellent performance on an independent test set.

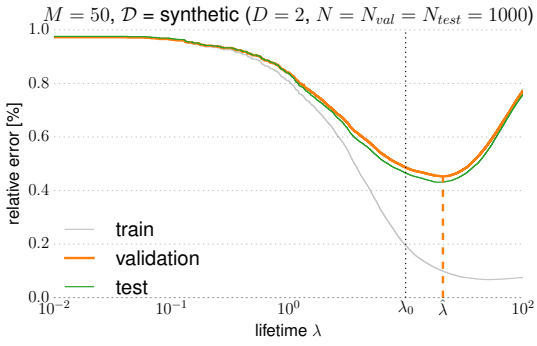


Figure 7: Recovering the ground truth lifetime $\lambda_0 = 10$ by selecting the value $\hat{\lambda} \approx 19$ minimizing validation set error.

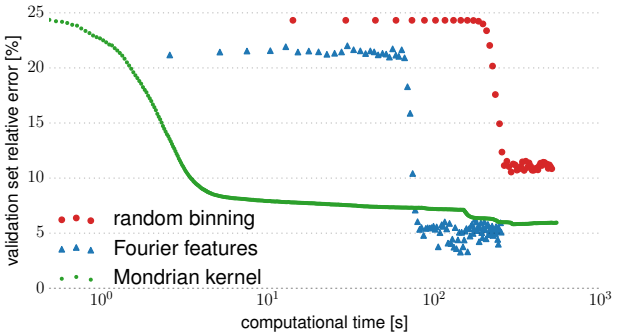


Figure 8: Validation set error as a function of computation time. Even though Fourier features are better suited to the CPU dataset [Rahimi and Recht, 2007] and eventually outperform the Mondrian kernel, the latter discovers suitable kernel widths at least an order of magnitude faster.

Second, we evaluated our kernel width selection procedure on the CPU dataset in order to demonstrate its practical usefulness. While the Mondrian kernel allows to efficiently sweep through lifetimes λ, Fourier features and random binning need to be reconstructed and retrained for each attempted lifetime value. We started the Fourier features and random binning at $\lambda = 1$, and in each step, we either doubled the maximum lifetime or halved the minimum lifetime considered so far, based on which direction seemed more promising. Once a good performing lifetime was found, we further optimized using a binary search procedure. All schemes were set to generate $M = 350$ non-zero features per datapoint. Figure 8 shows the performance of each scheme on a held-out validation set as a function of computation time. The result suggests that our kernel width learning procedure can be used to discover suitable lifetimes (inverse kernel widths) at least an order of magnitude faster than random Fourier features or random binning.

8.3 MONDRIAN KERNEL VS FOREST

We compared the performance of Mondrian kernel and "Mondrian forest" (quotes due to omission of hierarchical smoothing) based on the same $M = 50$ Mondrian samples, using the CPU dataset and varying the lifetime λ. Recall that higher values of λ lead to more refined Mondrian partitions, allowing more structure in the data to be modeled, but also increasing the risk of overfitting. Figure 9 shows that Mondrian kernel exploits the joint fitting of parameters corresponding to different trees and achieves a lower test error at lower lifetime values, thus producing a more compact solution based on simpler partitions. Figure 10 shows the parameter values learned by Mondrian kernel and Mondrian forest at the lifetime $\lambda = 2 \times 10^{-6}$. The distribution of weights learned by Mondrian kernel is more peaked around 0, as the joint fiting allows achieving more extreme predictions by adding together several smaller weights.

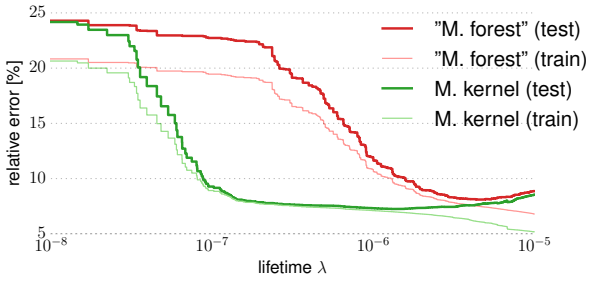


Figure 9: Comparison of Mondrian kernel and Mondrian forest models based on the same set of Mondrian samples.

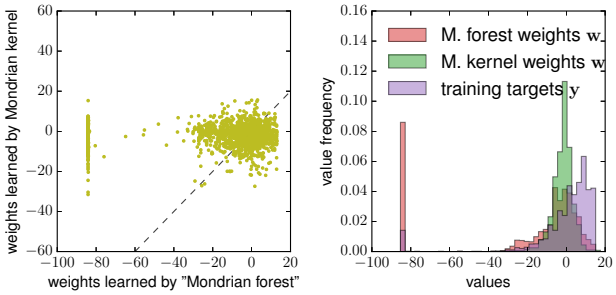


Figure 10: Weights learned by Mondrian forest and Mondrian kernel at the lifetime $\lambda = 2 \times 10^{-6}$ in Figure 9.

9 CONCLUSION

We presented the Mondrian kernel, a fast approximation to the Laplace kernel that admits efficient kernel width selection. When a different kernel or a different approximation is used, our procedure can provide a fast and simple way of initializing the kernel width for further optimization. While a Gaussian kernel is often considered a default choice, in many situations it imposes an inappropriately strong smoothness assumption on the modelled function and the Laplace kernel may in fact be a preferable option.

Our approach revealed a novel link between the Mondrian process and the Laplace kernel. We leave the discovery of similar links involving other kernels for future work.

Acknowledgements

We would like to thank Nilesh Tripuraneni for useful discussions. Part of this research was carried out while MB was at the University of Oxford. BL gratefully acknowledges generous funding from the Gatsby Charitable Foundation. ZG acknowledges funding from the Alan Turing Institute, Google, Microsoft Research and EPSRC Grant EP/N014162/1. DMR is supported by an NSERC Discovery Grant. YWT's research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) ERC grant agreement no. 617071.

References

L. Breiman. Some infinity theory for predictor ensembles. Technical report, University of California at Berkeley, 2000.

L. Breiman. Random forests. *Mach. Learn.*, 45:5–32, 2001.

A. Criminisi, J. Shotton, and E. Konukoglu. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Found. Trends Comput. Graphics and Vision*, 2012.

B. Dai, B. Xie, N. He, Y. Liang, A. Raj, M.-F. F. Balcan, and L. Song. Scalable kernel methods via doubly stochastic gradients. In *Adv. Neural Information Proc. Systems (NIPS)*, 2014.

A. Davies and Z. Ghahramani. The random forest kernel and other kernels for big data from random partitions. *arXiv preprint arXiv:1402.4293v1*, 2014.

P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Mach. Learn.*, 63(1):3–42, 2006.

B. Lakshminarayanan, D. M. Roy, and Y. W. Teh. Mondrian forests: Efficient online random forests. In *Adv. Neural Information Proc. Systems (NIPS)*, 2014.

B. Lakshminarayanan, D. M. Roy, and Y. W. Teh. Mondrian forests for large scale regression when uncertainty matters. In *Int. Conf. Artificial Intelligence Stat. (AISTATS)*, 2016.

Q. Le, T. Sarlós, and A. Smola. Fastfood-approximating kernel expansions in loglinear time. In *Proc. Int. Conf. Mach. Learn. (ICML)*, 2013.

A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Adv. Neural Information Proc. Systems (NIPS)*, 2007.

S. Ren, X. Cao, Y. Wei, and J. Sun. Global refinement of random forest. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 723–730, 2015.

D. M. Roy. *Computability, inference and modeling in probabilistic programming*. PhD thesis, Massachusetts Institute of Technology, 2011.

D. M. Roy and Y. W. Teh. The Mondrian process. In *Adv. Neural Information Proc. Systems (NIPS)*, 2009.

B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001. ISBN 978-0-262-19475-4.

E. Scornet. Random forests and kernel methods. *arXiv preprint arXiv:1502.03836v2*, 2015.

M. Seeger. *Bayesian Gaussian Process Models: PAC-Bayesian Generalisation Error Bounds and Sparse Approximations*. PhD thesis, University of Edinburgh, 2003.

M. Seeger. Low rank updates for the Cholesky decomposition. Technical report, University of California at Berkeley, 2004.

S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In *Proc. Int. Conf. Mach. Learn. (ICML)*, 2007.

Y. Shen, A. Ng, and M. Seeger. Fast Gaussian process regression using KD-trees. In *Adv. Neural Information Proc. Systems (NIPS)*, 2006.

Z. Yang, A. J. Smola, L. Song, and A. G. Wilson. A la Carte - Learning Fast Kernels. In *Int. Conf. Artificial Intelligence Stat. (AISTATS)*, 2015.