Machine Learning for Quantum Mechanics in a Nutshell

Matthias Rupp*

January 30, 2015

Abstract

Models that combine quantum mechanics (QM) with machine learning (ML) promise to deliver the accuracy of QM at the speed of ML. This hands-on tutorial introduces the reader to QM/ML models based on kernel learning, an elegant, systematically non-linear form of ML. Pseudo-code and a reference implementation are provided, enabling the reader to reproduce results from recent publications where atomization energies of small organic molecules are predicted using kernel ridge regression.

*Institute of Physical Chemistry and National Center for Computational Design and Discovery of Novel Materials (MARVEL), Department of Chemistry, University of Basel, Klingelbergstrasse 80, 4056 Basel, Switzerland

Contents

1 Introduction **3**
1.1 Why QM/ML models? . . . 3
1.2 The key idea . . . 5
1.3 Related work . . . 6

2 Machine learning **7**
2.1 Learning with kernels . . . 7
2.2 Kernel functions . . . 8
2.3 Specific kernels . . . 10
2.4 Linear regression . . . 11
2.5 Kernel ridge regression . . . 13
2.6 Implementation . . . 14
2.7 What about other methods? . . . 16
2.8 Model selection and performance estimation . . . 16

3 Predicting atomization energies **20**
3.1 Dataset . . . 20
3.2 Representation . . . 22
3.3 Model building . . . 22

4 What next? **24**

A Dataset **25**

> "The underlying physical laws necessary for the mathematical theory of a large part of physics and the whole of chemistry are thus completely known, and the difficulty is only that the exact application of these laws leads to equations much too complicated to be soluble. It therefore becomes desirable that approximate practical methods of applying quantum mechanics should be developed."
>
> Paul A.M. Dirac[1]

1 Introduction

This tutorial is meant to provide a short, practical introduction to interpolation of numerical quantum chemistry results using kernel-based machine learning methods. It has three parts: The introduction, explaining key ideas and providing context, a theory part, where kernel-based machine learning is introduced using the example of kernel ridge regression, and, a practical part that provides a worked example of how to predict atomization energies of small organic molecules. Exercises, marked by $\triangleright$, invite the reader to immediate practical experience with the subject. Algorithms are given in pseudo code for easy implementation. The supplementary material also provides a basic implementation of the used routines in *Mathematica*[2], a notebook with solutions to exercises, as well as a dataset to work with. Table 1 provides a glossary of used acronyms and notation.

1.1 Why QM/ML models?

Quantum mechanics (QM) provides a theory of matter at the atomic scale, and numerical solutions to Schrödinger's equation allow for calculation of virtually any property of a system. So why aren't more problems in materials science, organic chemistry, or biochemistry solved computationally? A major problem is the computational effort required, which increases so rapidly with system size that for solutions in agreement with experiment one is limited to small systems. As pointed out early last century by Paul Dirac[1], this situation necessitates approximations, trading in accuracy or generality for computational efficiency. Many such approximations were made, both on a conceptual level, such as the Born-Oppenheimer approximation, and on a numerical level, leading to a variety of approaches to solve Schrödinger's equation approximately (Table 2). To drive home the importance of differences in asymptotic runtime, consider doubling a system's size: For a coupled cluster method with runtime $O(n^7)$, runtime increases by a factor of $2^7 = 128$, whereas for a density functional theory method with runtime $O(n^3)$ it increases only by a factor of 8. Even so, after a few doublings one is bound to run out of computing resources.

So what to do when one requires the accuracy and wide applicability of higher level QM methods for a large system, or, a large number of small systems? Linear-scaling QM methods[3] might offer an alternative by exploiting locality for an excellent $O(n)$ asymptotic runtime, but are not applicable to all systems and can have large prefactors. Another strategy is to use machine learning for fast and accurate approximations of QM solutions.

Table 1: *Acronyms and notation.* Vectors and matrices are typeset in bold font.

Term	Meaning
Acronyms	
ANN	artificial neural network
DFT	density functional theory
GPR	Gaussian process regression
KRR	kernel ridge regression
MAE	mean absolute error
ML	machine learning
PES	potential energy surface
QM	quantum mechanics (& chemistry)
RMSE	root mean square error
General mathematics	
$\mathbb{R}$	real numbers
$\mathbb{N}$	natural numbers
$\langle \cdot, \cdot \rangle$	inner product
$\boldsymbol{x}^T$	Transpose of vector, row $\leftrightarrow$ column
$\boldsymbol{M}^T$	Transpose of matrix, $\boldsymbol{M}^T_{ij} = \boldsymbol{M}_{ji}$
$\boldsymbol{I}$	identity matrix, $(\boldsymbol{I})_{ij} = \delta_{ij}$
δ_{ij}	Kronecker delta, 1 if $i = j$, 0 if $i \neq j$
δ_{cond}	Kronecker delta, 1 if cond is true
$\nabla_{\boldsymbol{v}}$	gradient with respect to $\boldsymbol{v}$
$\|\cdot\|$	Euclidean norm, $\|\boldsymbol{x}\| = \sqrt{\sum_i \|x_i\|^2}$
R	Pearson's correlation coefficient
$O(\cdot)$	Landau notation for scaling
$\mathbb{E}[X]$	Expectation value of X

Term	Meaning
Machine learning	
n	number of training examples
$\tilde{n}$	number of prediction examples
$\boldsymbol{x}_i$	i-th training input, $\boldsymbol{x}_i \in \mathbb{R}^d$
$\tilde{\boldsymbol{x}}_j$	j-th prediction input, $\tilde{\boldsymbol{x}}_j \in \mathbb{R}^d$
$\tilde{\boldsymbol{x}}$	a prediction input, $\tilde{\boldsymbol{x}} \in \mathbb{R}^d$
$\boldsymbol{X}$	matrix of training inputs (rows)
$\tilde{\boldsymbol{X}}$	matrix of prediction inputs (rows)
$\mathcal{X}$	input space, here vector space $\mathbb{R}^d$
$\boldsymbol{y}$	training outputs (labels), $\boldsymbol{y} \in \mathbb{R}^n$
$\tilde{\boldsymbol{y}}$	prediction outputs (labels), $\tilde{\boldsymbol{y}} \in \mathbb{R}^{\tilde{n}}$
$\mathcal{Y}$	output space, here $\mathcal{Y} = \mathbb{R}$
k	kernel function, $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$
ϕ	map from input to feature space
$\mathcal{H}$	feature space, a Hilbert space
$\boldsymbol{K}$	kernel matrix, $k_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$
$\boldsymbol{L}$	kernel matrix, $l_{ij} = k(\boldsymbol{x}_i, \tilde{\boldsymbol{x}}_j)$
$\boldsymbol{M}$	kernel matrix, $m_{ij} = k(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_j)$
$\boldsymbol{\alpha}$	regression coefficients, $\boldsymbol{\alpha} \in \mathbb{R}^n$
$\boldsymbol{\beta}$	regression coefficients, $\boldsymbol{\beta} \in \mathbb{R}^d$
λ	regularization strength, $\lambda \in \mathbb{R}$
σ	length scale, $\sigma \in \mathbb{R}$
L	loss function, $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$

Table 2: *Hierarchy of numerical approximations to Schrödinger's equation.* Runtimes (scalings) have illustrative character, and depend on implementation. Note that the used Landau notation describes asymptotic behaviour for $N \to \infty$ only; in particular, prefactors are not shown, but can have substantial impact in practice. Here, N is system size, e.g., number of atoms, electrons, or basis functions.

Abbrev.	Method	Runtime
FCI	Full Configuration Interaction (CISDTQ)	$O(N^{10})$
CC	Coupled Cluster (CCSD(T))	$O(N^7)$
FCI	Full Configuration Interaction (CISD)	$O(N^6)$
MP2	Møller-Plesset second order perturbation theory	$O(N^5)$
QMC	Quantum Monte Carlo	$O(N^3)$ – $O(N^4)$
HF	Hartree-Fock	$O(N^3)$ – $O(N^4)$
DFT	Density Functional Theory (Kohn-Sham)	$O(N^3)$
TB	Tight Binding	$O(N^3)$
MM	Molecular Mechanics	$O(N^2)$

Machine learning (ML)[4–8], a subfield of artificial intelligence, studies algorithms whose performance improves with data ("learning from experience").[4] Its main concerns are the systematic identification and exploitation of regularity (non-randomness) in data for prediction or analysis. It has been successfully applied in a wide variety of fields, including brain-computer interfaces[9], recommender systems[10], robotics[11], and chemistry[12], in particular cheminformatics[13]. Widely known algorithms include artificial neural networks[14–16] and support vector machines[17]. ML can be used in a variety of settings; this tutorial focuses on the *supervised learning problem*, where one learns a mapping (function) from inputs $\boldsymbol{x}$ to labels y, given a *training set* of n reference pairs $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$. Examples of such problems in QM are ab initio molecular dynamics, where one learns the potential energy surface, mapping system configurations to their energy; orbital-free density functional theory, where one learns the map from electronic densities to their kinetic energy; and, molecular property prediction, mapping molecules to property values. At its core, much of this is interpolation between data points, often in high-dimensional spaces, based on some notion of similarity between inputs (cf. the "similarity principle"[18] in cheminformatics). As such, ML is inherently data-driven and highly empirical. Some ML methods, in particular modern ones like kernel-based learning, also have strong theoretical underpinnings (e.g., learning theory[19]). In this sense, ML is empirical in a principled way.

1.2 The key idea

QM/ML models exploit redundancy in a series of QM calculations by using ML to interpolate between reference calculations (Fig. 1). When doing the same QM calculation for a series of similar systems, these calculations contain redundant information due to the similarity of the systems (the same calculation is repeated for slightly different inputs, yielding correlated outputs). Examples are running a molecular dynamics simulation (conformational changes), and, calculating a property for a series of molecules with common scaffold or substituents

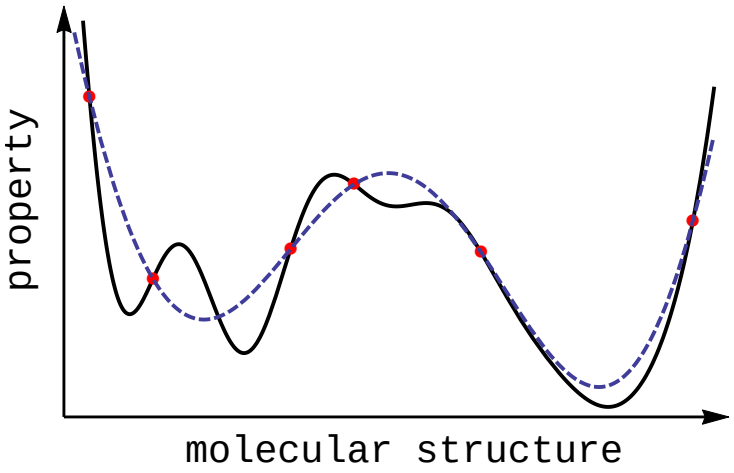


Figure 1: *Sketch illustrating the idea of QM/ML models* for prediction of molecular properties. Results of computationally demanding QM calculations ("ground truth", black line) are approximated by interpolating (ML, dashed line) between reference calculations (training data, dots).

(changes in chemical space[20,21]). This redundancy can be exploited by doing only some of the QM calculations and interpolating between them to obtain approximate solutions for the remaining systems.* The usefulness of this approach depends on the error incurred due to the approximation and the computational cost of obtaining it.

1.3 Related work

Interpolation of ab initio potential energy surfaces (PES) has a long history, dating back maybe as far as the middle of the last century, when computers were first used for QM. The topic is related to, among other subjects, parametrization of force fields, with the major difference being the used functional form, and, cheminformatics, in particular quantitative structure-activity/property relationships (QSAR/QSPR),[13] where outcomes of experimental measurements are interpolated, with the main difference being the large uncertainties in the reference values fitted to.

In the early 1990s, artificial neural networks (ANN) started being used for interpolation of PES of single systems, and have since developed into powerful tools for large-scale molecular dynamics simulations.[27–30] A variety of other approaches, including Shepherd interpolation[31–33], cubic splines[34], moving least-squares[35,36], and symbolic regression[37], were used as well. Predating the introduction and formalization of the kernel learning framework,[38,39] concepts like regularization were used early on, e.g., by H. Rabitz[40–42]. Interpolation between QM results for different systems, e.g., molecular property estimates, started roughly a decade later with usage of ANN to predict correlation energies[43] and bond dissociation enthalpies[44]. Later, other methods such as support vector machines were used as well.[45,46] The last years have witnessed publication of and application to large datasets[47–50], the in-

*An alternative to ML for exploiting such *alchemical changes*[22,23] is to use gradient information, e.g., in the form of Taylor series expansions in chemical space.[24–26]

troduction of further kernel-based ML methods, such as Gaussian process regression (GPR), e.g., for estimating multipoles[51] and PES interpolation[52], as well as new applications of QM/ML models, such as mapping electron densities to kinetic energy for orbital-free density functional theory[53–55] and optimization of transition state theory dividing surfaces[56].

One of the most important aspects of a QM/ML model is how a system, be it molecular or periodic, is numerically represented for interpolation. A wide variety of representations has been used, including symmetry functions,[27] ad hoc descriptors,[57–59] Fourier expansions of radial basis functions,[60] smooth overlap of atomic positions,[52,61] and the Coulomb matrix[50,62–66].

2 Machine learning

At its core, much of ML is interpolation between data points. So, is ML just a fancy word for fitting? Not quite. ML encompasses a wide variety of formal problems and algorithms for their solution. For QM/ML models, the most important problem is regression, a supervised learning problem. An example of another problem relevant in a chemical context is dimensionality reduction,[67] an unsupervised learning problem, related to, e.g., identification of reaction coordinates (collective variables).[68]

Regression is a *supervised learning* problem: Given a set of n observations (the training data) $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, consisting of inputs $\boldsymbol{x}_i \in \mathcal{X}$ and corresponding outputs $y_i \in \mathcal{Y}$ (the labels), predict the label $\tilde{y}$ for new inputs $\tilde{\boldsymbol{x}}$. For $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \mathbb{R}$, this is ordinary multiple regression. Note that all observations $(\boldsymbol{x}_i, y_i)$, $(\tilde{\boldsymbol{x}}, \tilde{y})$ are assumed to be independent and identically distributed, an assumption often violated in practice.

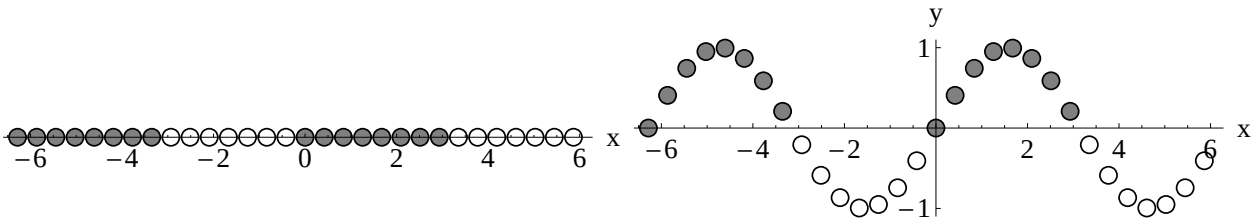
The problem of learning a function from a finite sample of its values has no unique solution (there are infinitely many functions that are "compatible" with the training data), and additional assumptions have to be made. In ML, one usually assumes smoothness, which leads to regularization.[69] Essentially, one chooses the simplest model that is compatible with the data (Occam's razor).

2.1 Learning with kernels

Kernel-based ML methods, the focus of this tutorial, have become widely popular since their introduction in the 1990s.* They have strong theoretical foundations, and tend to be somewhat easier to set up in practice than their main competitor, artificial neural networks; for an introduction to the latter, see the tutorial review by Jörg Behler[30]. Only concepts required for the tutorial are introduced here; for further information, consult Refs. 71–73.

The basic idea of kernel-based ML is to derive non-linear versions of linear ML algorithms in a systematic way. This is done by (implicitly) mapping the inputs into a higher-dimensional space and applying the linear algorithm there (Fig. 2). This approach has two immediate problems: Computational complexity, and how to find the right mapping.

* The support vector machine was the first widely successful kernel algorithm, introduced by Boser, Guyon & Vapnik in 1992.[38] The involved concepts had been investigated since the 1960s, e.g., the first use of kernels in ML by Aizerman, Braverman & Rozonoer in 1964.[70] (see chapter 6.5 in ref. 17).



Linear inseparability in input space Linear separability in transformed space

Figure 2: *Linear separability via a nonlinear mapping into a higher-dimensional space.* Left: In input space $\mathbb{R}$, samples from two classes (blank and grey disks) are not linearly separable. Right: The nonlinear function $x \mapsto (x, \sin x)$ maps the samples into the higher-dimensional space $\mathbb{R}^2$, where samples are linearly separable (by the x-axis).

Example Consider the mapping $\phi : \mathbb{R}^p \to \mathbb{R}^{p^d}$ which maps $\boldsymbol{x}$ to the space of all ordered monomials of degree d, e.g., for $p = d = 2$, $\phi\big((x_1, x_2)\big) = (x_1^2, x_1x_2, x_2x_1, x_2^2)$. The size p^d of the space mapped into (the *feature space*) depends polynomially on the size p of the input space. For $p = 120$ and $d = 3$, the size of the feature space is already $120^3 = 1\,728\,000$. Other mappings, e.g., from the Gaussian kernel, are into feature spaces of infinite dimension. For such mappings, explicit computations in feature space are computationally either infeasible or impossible.

The kernel trick offers a solution to this problem based on two observations: First, most linear ML algorithms can be rewritten to use only inner products between inputs (these contain information about norms, angles, and distances, i.e., about relations between inputs). This reduces the problem of arbitrary computations with feature space vectors to computing inner products between them. Second, functions called kernels operate on input space vectors, but yield the same results as inner product evaluations in feature space. In other words, one can replace evaluation of inner products in feature space by evaluations of a kernel function in input space. Combined, these two observations elegantly sidestep the problem of explicit computations in feature space.

2.2 Kernel functions

Kernel functions allow replacing computations in feature space by computations in input space giving identical results. This is achieved via *inner products*, which generalize geometric concepts like length, angle, and orthogonality. For a real vector space $\mathcal{V}$, a function $\langle \cdot\,, \cdot \rangle : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ is an inner product if and only if for all $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathcal{V}$, $\alpha \in \mathbb{R}$ holds

- $\langle \boldsymbol{a}, \boldsymbol{a} \rangle \geq 0$ (non-negativity) and $\langle \boldsymbol{a}, \boldsymbol{a} \rangle = 0 \Leftrightarrow \boldsymbol{x} = 0$,
- $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \langle \boldsymbol{b}, \boldsymbol{a} \rangle$ (symmetry),
- $\langle \boldsymbol{a} + \boldsymbol{b}, \boldsymbol{c} \rangle = \langle \boldsymbol{a}, \boldsymbol{c} \rangle + \langle \boldsymbol{b}, \boldsymbol{c} \rangle$ and $\langle \alpha\, \boldsymbol{a}, \boldsymbol{b} \rangle = \alpha \langle \boldsymbol{a}, \boldsymbol{b} \rangle$ (linearity).

The pair $(\mathcal{V}, \langle \cdot\,, \cdot \rangle)$ is called an *inner product space*. Two vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{V}$ are *orthogonal* if and only if their inner product is zero, $\boldsymbol{a} \perp \boldsymbol{b} \Leftrightarrow \langle \boldsymbol{a}, \boldsymbol{b} \rangle = 0$. In a real inner product space

$(\mathcal{V}, \langle\cdot\,,\cdot\rangle)$, the *angle* θ (measured in radians) between two non-zero $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{V}$ is defined as

$$\theta \in [0, \pi] \quad \text{such that} \quad \cos\theta = \frac{\langle \boldsymbol{a}, \boldsymbol{b} \rangle}{\|\boldsymbol{a}\| \; \|\boldsymbol{b}\|}. \tag{1}$$

An inner product $\langle\cdot\,,\cdot\rangle$ can be used to construct a norm (and corresponding metric) via $\|\boldsymbol{a}\|^2 = \langle \boldsymbol{a}, \boldsymbol{a} \rangle$. Conversely, given a norm $\|\cdot\|$ on $\mathcal{X}$, $\|\boldsymbol{a}\|^2 = \langle \boldsymbol{a}, \boldsymbol{a} \rangle$ if and only if the *parallelogram identity* $\|\boldsymbol{a} + \boldsymbol{b}\|^2 + \|\boldsymbol{a} - \boldsymbol{b}\|^2 = 2\big(\|\boldsymbol{a}\|^2 + \|\boldsymbol{b}\|^2\big)$ holds; the Euclidean norm is the only L^p-norm satisfying this identity.[74]

A *kernel* is a function that corresponds to an inner product in some feature space $\mathcal{H}$. Formally, $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel if and only if there exists a map $\phi : \mathcal{X} \to \mathcal{H}$ such that

$$\forall \boldsymbol{a}, \boldsymbol{b} \in \mathcal{X} : k(\boldsymbol{a}, \boldsymbol{b}) = \langle \phi(\boldsymbol{a}), \phi(\boldsymbol{b}) \rangle \,. \tag{2}$$

It is not necessary to know ϕ or $\mathcal{H}$ explicitly, their existence is sufficient. Using a kernel, inner products in high-dimensional feature spaces can be implicitly computed as kernel values in input space, thereby alleviating the computational complexity issue due to feature space dimensionality. Kernels are characterized by the positive definiteness property. A real symmetric matrix $\boldsymbol{K} \in \mathbb{R}^{n \times n}$ is *positive definite* if and only if

$$\forall \boldsymbol{c} \in \mathbb{R}^n : \boldsymbol{c}^T \boldsymbol{K} \boldsymbol{c} = \sum_{i,j=1}^{n} c_i c_j \boldsymbol{K}_{i,j} \geq 0. \tag{3}$$

$\boldsymbol{K}$ is *strictly positive definite* if and only if equality occurs only for $\boldsymbol{c} = \boldsymbol{0}$. Positive definite and strictly positive definite matrices are also called *positive semidefinite* and *positive definite* matrices, respectively; corresponding care has to be taken when consulting the literature.

The Gram matrix of vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ is the $n \times n$ matrix $\boldsymbol{K}$ of their inner products, $\boldsymbol{K}_{i,j} = \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that has (strictly) positive definite Gram matrix for all $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathcal{X}$, $n \in \mathbb{N}$ is called (strictly) positive definite. Inner products are positive definite due to

$$\sum_{i,j=1}^{n} c_i c_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left\langle \sum_{i=1}^{n} c_i \phi(\boldsymbol{x}_i), \sum_{j=1}^{n} c_j \phi(\boldsymbol{x}_j) \right\rangle \geq 0. \tag{4}$$

Vice versa, it can be shown that every positive definite function corresponds to an inner product in some inner product space (via reproducing kernel Hilbert spaces and the Moore-Aronszajn theorem[75]). Proper kernels are therefore characterized by positive definiteness.* Criteria for the positive definiteness of matrices other than Eq. 3 include Sylvester's criterion ($\boldsymbol{K}$ is strictly positive definite if and only if its leading principal minors are positive; it is positive semidefinite if and only if all of its principal minors are non-negative[77]), and, the eigenspectrum of a matrix: $\boldsymbol{K}$ is (strictly) positive definite if and only if all of its eigenvalues are non-negative (positive).

* Historically, kernels satisfying Mercer's theorem[76] were used. Such functions correspond to inner products, but not all functions corresponding to inner products satisfy the theorem's conditions.

2.3 Specific kernels

The *linear kernel* $k(\boldsymbol{x}, \boldsymbol{z}) = \langle \boldsymbol{x}, \boldsymbol{z} \rangle$ is arguably the simplest kernel, with identical input and feature space, $\phi(\boldsymbol{x}) = \boldsymbol{x}$. Using the linear kernel results in an equivalent of the original linear algorithm. For a new problem, this is the first kernel to try.

▷ *Linear kernel yields original model* Given a kernel-based ML model (Eq. 14) $f(\tilde{\boldsymbol{x}}) = \sum_{i=1}^{n} \alpha_i k(\boldsymbol{x}_i, \tilde{\boldsymbol{x}})$, where α_i are regression coefficients, $\boldsymbol{x}_i \in \mathbb{R}^d$ are training inputs, and $\tilde{\boldsymbol{x}} \in \mathbb{R}^d$ is the input to predict, show that for the linear kernel this yiels the linear regression model $f(\tilde{\boldsymbol{x}}) = \sum_{j=1}^{d} \beta_j \tilde{\boldsymbol{x}}_j$ (Eq. 8).

The *Gaussian kernel* (also squared exponential kernel, radial basis function kernel) is a popular default choice for non-linear kernel models:

$$k(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{z}\|^2}{2\sigma^2}\right), \tag{5}$$

where $\sigma > 0$ is a hyperparameter defining the length scale on which the kernel operates. The Gaussian kernel is a good kernel to try after the linear kernel, as it performs reasonably well for many problems. It maps into an infinite-dimensional feature space.[78] To understand the behaviour of the Gaussian kernel, consider the limiting cases

$$\lim_{\sigma \to 0} k(\boldsymbol{x}, \boldsymbol{z}) = \delta_{\boldsymbol{x}=\boldsymbol{z}} \qquad \text{and} \qquad \lim_{\sigma \to \infty} k(\boldsymbol{x}, \boldsymbol{z}) = 1. \tag{6}$$

In the first case, all inputs are mapped into different dimensions orthogonal to each other, leading to overfitting. In the second case, all inputs are mapped into a single point, leading to underfitting. For intermediate values of σ, the kernel value depends on $\|\boldsymbol{x} - \boldsymbol{z}\|$, approaching 1 for $\|\boldsymbol{x} - \boldsymbol{z}\| \to 0$, and 0 for $\|\boldsymbol{x} - \boldsymbol{z}\| \to \infty$. Samples that are close in input space are therefore correlated in feature space, whereas faraway samples are mapped to orthogonal subspaces. In this way, the Gaussian kernel can be seen as a local approximator, with scale dependent on σ (Fig. 3).

The *Laplacian kernel* performed better than the Gaussian kernel for prediction of molecular properties in some studies.[63–65] Like the Gaussian kernel, it is an exponential function,

$$k(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{z}\|_1}{\sigma}\right), \tag{7}$$

but uses the 1-norm $\|\boldsymbol{z}\|_1 = \sum_{i=1}^{d} |\boldsymbol{z}_i|$ instead of the Euclidean norm.

Fig. 4 presents plots of the linear, Gaussian, and Laplacian kernels as functions $k(0, x)$ of a single variable x (left column), similar to basis functions placed at the origin, providing some intuition about their shape (note the influence of the hyperparameter σ). The figure also shows functions randomly sampled from a stochastic process using the corresponding kernel as covariance function (right column). These provide an intuition about the shape of functions that can be modeled with each kernel.

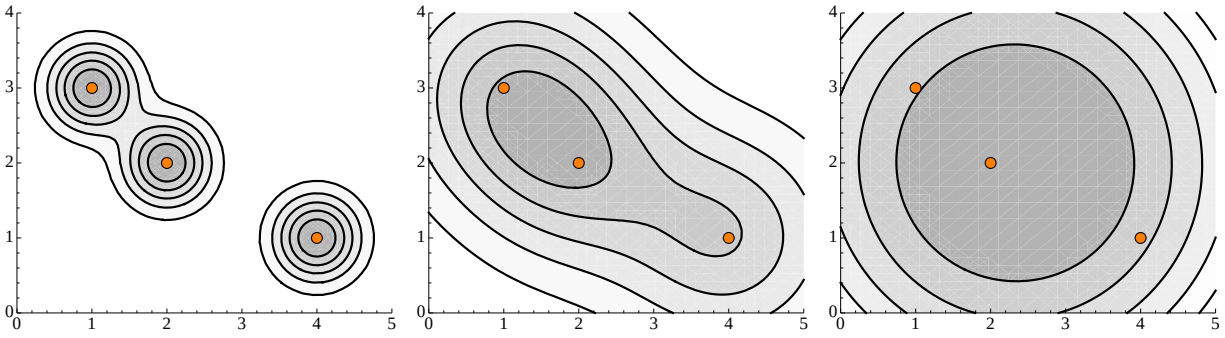


Figure 3: *Dependence of the Gaussian kernel on length scale hyperparameter* σ. Shown are 5 isolines (having different values in the three plots; solid lines) of a Gaussian mixture model $g(\boldsymbol{x}) = \frac{1}{3}\sum_{i=1}^{3} k(\boldsymbol{x}_i, \boldsymbol{x})$ for three data points $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, $\boldsymbol{x}_3$ (disks), with k from Eq. 5. Graylevels indicate the density of g, with darker values corresponding to higher densities.

2.4 Linear regression

Multiple linear regression A linear model in d dimensions is given by a linear combination of these dimensions, each weighted by a regression coefficient β_i:

$$f(\tilde{\boldsymbol{x}}) = \sum_{i=1}^{d} \beta_i \tilde{x}_i = \langle \boldsymbol{\beta}, \tilde{\boldsymbol{x}} \rangle \,. \tag{8}$$

Note that Eq. 8 contains no bias term $+b$, and thus can model only functions that pass through the origin. Having a bias term is equivalent to centering both inputs and labels, i.e., working with $\tilde{\boldsymbol{x}} - \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i$ and $\tilde{y} - \frac{1}{n}\sum_{i=1}^{n} y_i$ instead of $\tilde{\boldsymbol{x}}$ and $\tilde{y}$. From now on, this is assumed to be the case (without loss of generality, since training set means can be subtracted before training, then added for predictions).

Ideally, one would like to find coefficients $\boldsymbol{\beta}$ that minimize the *generalization error*, i.e., the average error on new inputs. However, the distribution of samples is usually not known, and one has access only to the finite training set $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$, where $(\boldsymbol{x}_i, y_i) \in \mathbb{R}^d \times \mathbb{R}$. Therefore, the *empirical error* is minimized instead, i.e., the error on the training set:

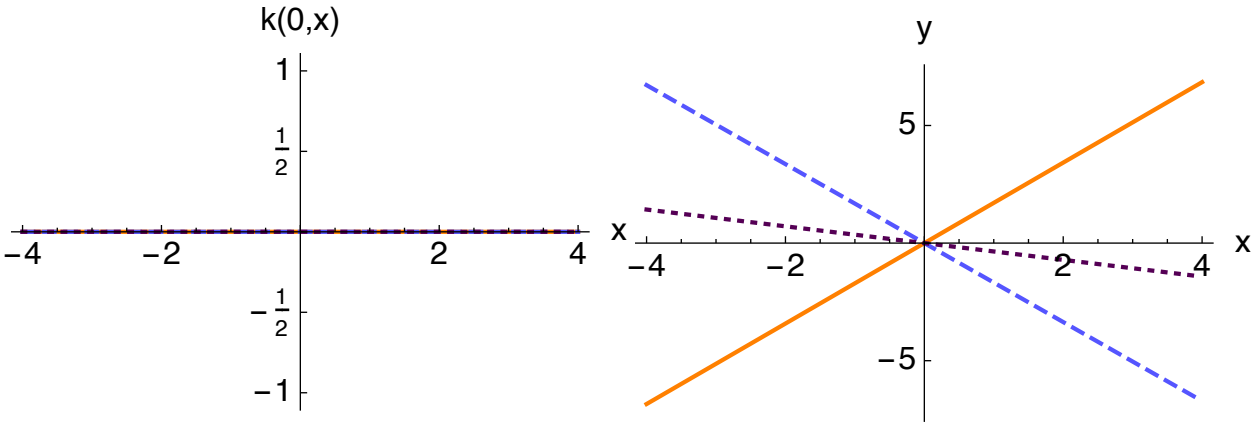
$$L(\boldsymbol{X}, \boldsymbol{y}) = \sum_{i=1}^{n} \big(f(\boldsymbol{x}_i) - y_i\big)^2, \tag{9}$$

where $\boldsymbol{X}$ is the input matrix with rows $\boldsymbol{x}_i$. Note that Eq. 9 constitutes a specific choice of error, or *loss*, namely the squared error. The resulting convex optimization problem

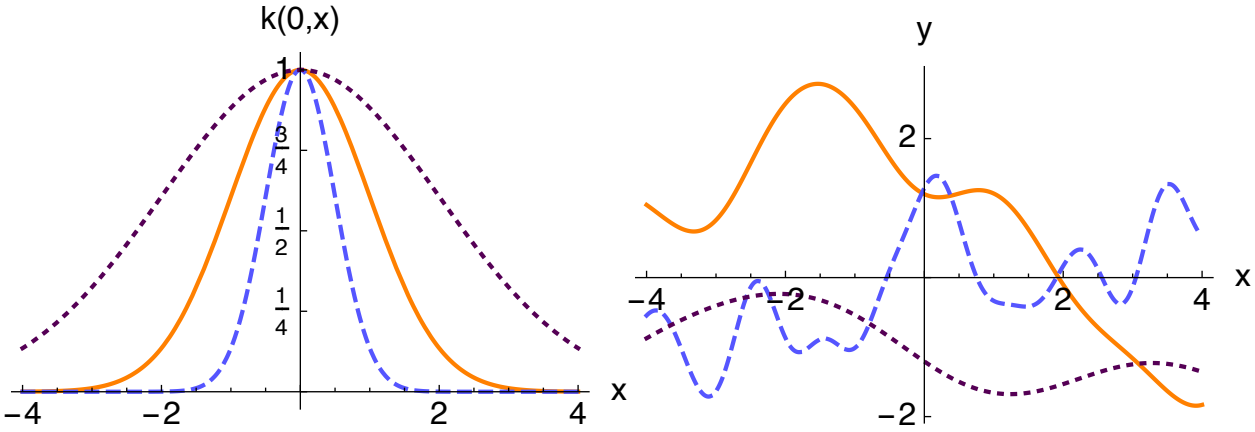
$$\operatorname*{arg\,min}_{\boldsymbol{\beta} \in \mathbb{R}^d} \sum_{i=1}^{n} \big(\langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle - y_i\big)^2 \tag{10}$$

is solved by setting its gradient to zero,

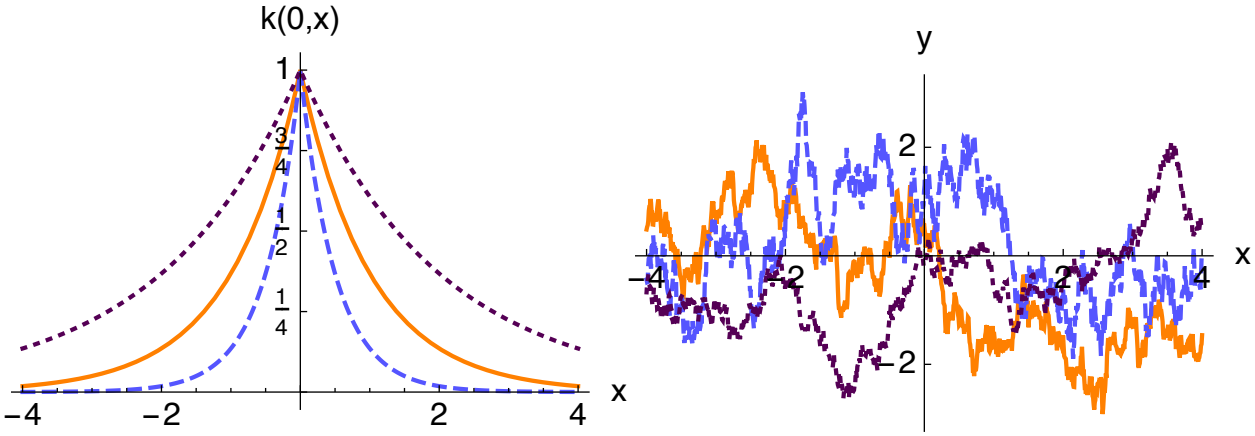
$$\begin{aligned} \nabla_{\boldsymbol{\beta}} \sum_{i=1}^{n} \big(\langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle - y_i\big)^2 = \boldsymbol{0} \quad &\Leftrightarrow \quad \sum_{i=1}^{n} \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle \, \boldsymbol{x}_i - \sum_{i=1}^{n} y_i \boldsymbol{x}_i = \boldsymbol{0} \\ \Leftrightarrow \quad \boldsymbol{X}^T \boldsymbol{X} \boldsymbol{\beta} = \boldsymbol{X}^T \boldsymbol{y} \quad &\Leftrightarrow \quad \boldsymbol{\beta} = \big(\boldsymbol{X}^T \boldsymbol{X}\big)^{-1} \boldsymbol{X}^T \boldsymbol{y}, \end{aligned} \tag{11}$$



(a) Linear kernel.



(b) Gaussian kernel with $\sigma = 0.5, 1, 2$ (dashed, solid, dotted lines).



(c) Laplacian kernel with $\sigma = 0.5, 1, 2$ (dashed, solid, dotted lines).

Figure 4: *Examples of different kernel functions k.* Shown are $k(0, x)$ as a function of x (left) and random samples from a Gaussian process with k as covariance function (right).

provided that the inverse exists. One problem of this approach is that the training data are fitted exactly. This means that label noise (e.g., numerical deviations in the calculated properties due to different implementations or settings, or meaningless last bits of y_i beyond the accuracy of the QM method) is fitted exactly as well. Exact fitting of such small differences often leads to large coefficients β_i that almost cancel on the training data, but cause large errors for new inputs. This is one form of overfitting.[79]

Ridge regression[8,80] is linear regression with added regularization to prevent overfitting. Regularization shrinks the regression coefficients towards each other and towards zero, mitigating the overfitting effect described before. It increases bias, but reduces variance (cf. bias-variance trade-off[81]). Ridge regression adds a penalty term to Eq. 10, resulting in

$$\underset{\boldsymbol{\beta}\in\mathbb{R}^d}{\arg\min} \sum_{i=1}^{n} \big(\langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle - y_i\big)^2 + \lambda \left\|\boldsymbol{\beta}\right\|^2, \tag{12}$$

where $\lambda \geq 0$ is a hyperparameter determining strength of regularization. The norm of the coefficient vector $\|\boldsymbol{\beta}\|$ is in some sense related to the smoothness and the complexity of the model f, with larger values of λ leading to smoother and simpler models. Analogous to the derivation of Eq. 11, solving for $\boldsymbol{\beta}$ yields

$$\boldsymbol{\beta} = \big(\boldsymbol{X}^T\boldsymbol{X} + \lambda \boldsymbol{I}\big)^{-1} \boldsymbol{X}^T\boldsymbol{y}, \tag{13}$$

where $\boldsymbol{I}$ denotes the identity matrix. Unlike in Eq. 11, the inverse always exists for $\lambda > 0$. Note that Eq. 13 determines all the parameters $\boldsymbol{\beta}$ of the model, but not the hyperparameter λ.

▷ *Ridge regression solution* Solve Eq. 12 to obtain the closed-form expression in Eq. 13.

2.5 Kernel ridge regression

Applying the kernel trick to ridge regression results in *kernel ridge regression* (KRR), a non-linear version of ridge regression, where the type of non-linearity is determined by the kernel. Note that it is only necessary to derive and implement the algorithm once; after that, using it with another kernel will effectively provide a different non-linear version of ridge regression.

The kernel trick can be applied to any linear ML algorithm that depends only on inner products of the inputs. Examples of other algorithms that have been "kernelized" include support vector machines,[17,38] principal component analysis,[39] Gaussian process regression,[82] and partial least squares.[83,84]

Kernel learning algorithms are implicitly carried out in feature space $\mathcal{H}$. Since feature vectors $\phi(\boldsymbol{x}) \in \mathcal{H}$ are not directly accessible (only their inner products are), kernel models are not expressed as a sum over dimensions, as in Eq. 8, but as a sum over training examples,

$$f(\tilde{\boldsymbol{x}}) = \sum_{i=1}^{n} \alpha_i k(\boldsymbol{x}_i, \tilde{\boldsymbol{x}}). \tag{14}$$

The *representer theorem*[85] guarantees that this is always possible. Intuitively, although the dimensionality of $\mathcal{H}$ can be high, the solution lives in the finite span of the projected training data, enabling a finite representation. The corresponding convex optimization problem is

$$\underset{\boldsymbol{\alpha}\in\mathbb{R}^n}{\arg\min} \sum_{i=1}^{n} \big(f(\boldsymbol{x}_i) - y_i\big)^2 + \lambda \left\|f\right\|_{\mathcal{H}}^2 \tag{15}$$

$$\Leftrightarrow \quad \underset{\boldsymbol{\alpha}\in\mathbb{R}^n}{\arg\min} \ \langle \boldsymbol{K}\boldsymbol{x} - \boldsymbol{y}, \boldsymbol{K}\boldsymbol{x} - \boldsymbol{y}\rangle + \lambda\, \boldsymbol{\alpha}^T \boldsymbol{K}\boldsymbol{\alpha}, \tag{16}$$

where $\left\|f\right\|_{\mathcal{H}}$ is the norm of f in $\mathcal{H}$, i.e., the complexity of the linear ridge regression model in feature space, and $\boldsymbol{K} \in \mathbb{R}^{n\times n}$, $\boldsymbol{K}_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the *kernel matrix* between training samples. As before, setting the gradient to zero yields an analytic solution for the regression coefficients:

$$\begin{aligned} \nabla_{\boldsymbol{\alpha}} \ \boldsymbol{\alpha}^T \boldsymbol{K}^2 \boldsymbol{\alpha} - 2\boldsymbol{\alpha}^T \boldsymbol{K}\boldsymbol{y} + \boldsymbol{y}^T\boldsymbol{y} + \lambda \boldsymbol{\alpha}^T \boldsymbol{K}\boldsymbol{\alpha} = \boldsymbol{0} \quad &\Leftrightarrow \quad \boldsymbol{K}^2\boldsymbol{\alpha} + \lambda \boldsymbol{K}\boldsymbol{\alpha} = \boldsymbol{K}\boldsymbol{y} \\ &\Leftrightarrow \quad \boldsymbol{\alpha} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\, \boldsymbol{y}. \end{aligned} \tag{17}$$

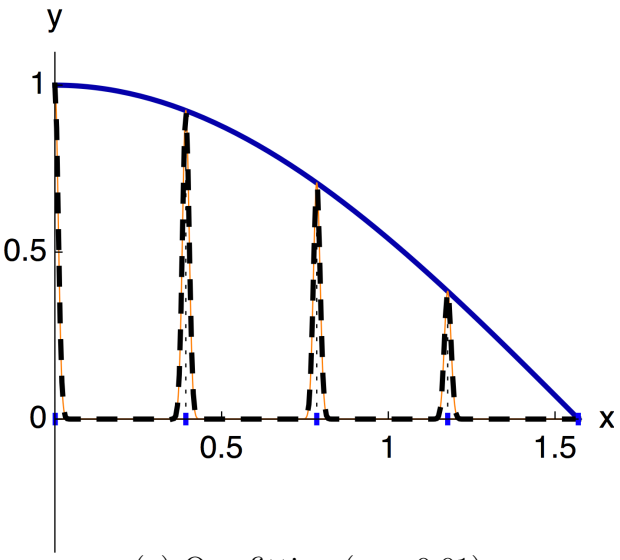
Fig. 5 presents an example of a KRR model with Gaussian kernel that demonstrates the role of the length scale hyperparameter σ. Note that although σ is not directly related to the regularization term in Eq. 15, it does control smoothness of the predictor, and therefore also effectively regularizes.[86]

2.6 Implementation

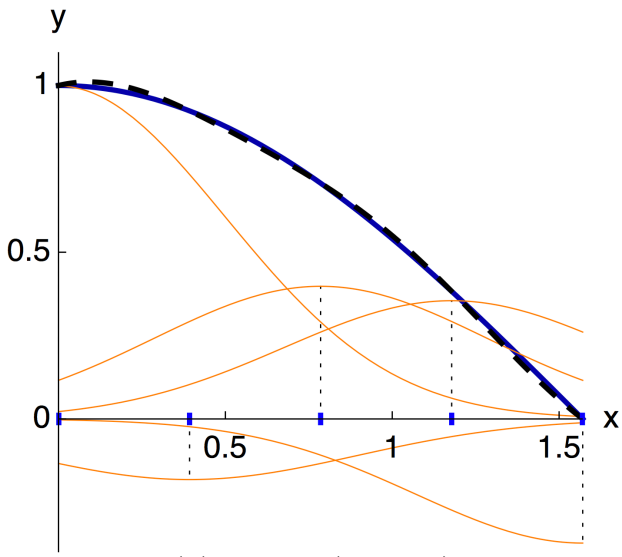
Implementation of kernel-based learning algorithms is discussed at the example of KRR.

Basic considerations Consider Eqs. 14 and 17. All information used to train a KRR model is contained in the matrix $\boldsymbol{K}$ of kernel evaluations between training data. Similarly, all information necessary to predict new inputs is contained in the kernel matrix of training versus prediction data. The kernel can thus be pictured as the "lens" through which the algorithm sees the data. Kernel matrices are therefore a natural interface choice for implementations of kernel learning algorithms: They contain exactly the required information about the data. An additional advantage of relying on kernel matrices as building blocks, as opposed to, say, individual kernel evaluations, is that this strategy plays along well with high-level interpreted languages (e.g., MatLab, `mathworks.com`; Mathematica, `wolfram.com`; Python, `python.org`), where few calls of optimized procedures, such as matrix and vector products, are preferable to many calls in interpreter-intensive code. A similar argument holds when using a low-level language (e.g., Fortran, C) with a numerical library (e.g., BLAS,[87] LAPACK[88]).

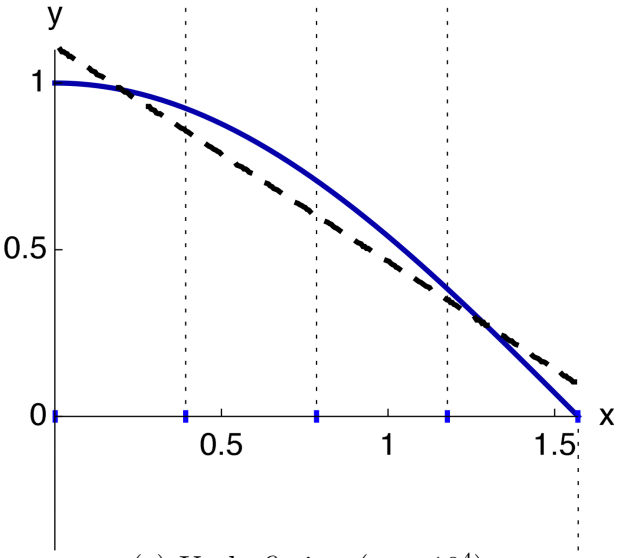
Kernel ridge regression Following Eq. 17, the regression coefficients $\boldsymbol{\alpha}$ are obtained by solving the linear system of equations $(\boldsymbol{K} + \lambda \boldsymbol{I})\boldsymbol{\alpha} = \boldsymbol{y}$, where $\boldsymbol{K} + \lambda \boldsymbol{I}$ is symmetric and strictly positive definite. One way to do this, suggested by Rasmussen and Williams[82] for numerical stability, is to use Cholesky decomposition,[89,90] $\boldsymbol{K} + \lambda \boldsymbol{I} = \boldsymbol{U}^T\boldsymbol{U}$, where $\boldsymbol{U}$ is upper triangular. One then solves $\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{\alpha} = \boldsymbol{y}$ by solving two linear systems of equations, first $\boldsymbol{U}^T\boldsymbol{\beta} = \boldsymbol{y}$, then $\boldsymbol{U}\boldsymbol{\alpha} = \boldsymbol{\beta}$. Since $\boldsymbol{U}^T$ is lower triangular and $\boldsymbol{U}$ is upper triangular,



(a) Overfitting ($\sigma = 0.01$)



(b) Fitting ($\sigma = 0.5$)



(c) Underfitting ($\sigma = 10^4$)

$\boldsymbol{x} = \{0, \frac{1}{8}\pi, \frac{2}{8}\pi, \frac{3}{8}\pi, \frac{4}{8}\pi\}$

	α_1	α_2	α_3	α_4	α_5
(a)	1.000	0.924	0.707	0.383	0.000
(b)	0.998	−0.182	0.398	0.355	−0.372
(c)	-10^{13}	10^{12}	10^{13}	10^{12}	-10^{12}

(d) Training points $\boldsymbol{x}$ and weights $\boldsymbol{\alpha}$

Figure 5: *Example of kernel ridge regression with Gaussian kernel and different length scales.* Shown are the learned function $\cos(x)$ (solid line), the ML model (dashed line), 5 training points (tick marks and dotted lines; (d), top), the weights $\boldsymbol{\alpha}$ ((d), bottom), and, the corresponding Gaussian basis functions (thin solid lines). In all cases, the regularization constant was set to the small value of $\lambda = 10^{-14}$. In (a), a too small σ results in exact reproduction of the training set ($\alpha_i = y_i$), with high error in-between, i.e., for new samples (test data). In (b), a good choice of σ leads to low error on training and test data. In (c), a too large σ results in close to linear behavior, with high error on training and test data. The basis functions are almost straight horizontal lines at heights given by $\boldsymbol{\alpha}$ (not shown). Small differences in the coefficients lead to the observed almost linear fit.

this requires only two straight-forward passes over the data called forward and backward substitution, respectively. For $\boldsymbol{U}^T\boldsymbol{\beta} = \boldsymbol{y}$, one obtains

$$\begin{aligned}
\boldsymbol{U}_{1,1}^T\beta_1 = y_1 \Leftrightarrow \beta_1 = y_1/u_{1,1}, \\
\boldsymbol{U}_{2,1}^T\beta_1 + \boldsymbol{U}_{2,2}^T\beta_2 = y_2 \Leftrightarrow \beta_2 = (y_2 - u_{1,2}\beta_1)/u_{2,2} \\
\dots \\
\sum_{j=1}^{i} \boldsymbol{U}_{i,j}^T\beta_j = y_i \Leftrightarrow \beta_i = (y_i - \sum_{j=1}^{i-1} u_{j,i}\beta_j)/u_{i,i}
\end{aligned} \tag{18}$$

For $\boldsymbol{U}\boldsymbol{\alpha} = \boldsymbol{\beta}$, the order is reversed. Once the model is trained, predictions can be made via Eq. 14. The prediction for a new input $\tilde{\boldsymbol{x}}$ is the inner product between the vector of coefficients and the vector of corresponding kernel evaluations. For several prediction samples $\tilde{\boldsymbol{x}}_1, \dots, \tilde{\boldsymbol{x}}_{\tilde{n}}$, this can be conveniently expressed, and efficiently computed, using matrices,

$$f(\tilde{\boldsymbol{x}}) = \sum_{i=1}^{n} \alpha_i k(\boldsymbol{x}_i, \tilde{\boldsymbol{x}}), \qquad f(\tilde{\boldsymbol{X}}) = \boldsymbol{L}^T\boldsymbol{\alpha}, \tag{19}$$

where $\tilde{\boldsymbol{X}} \in \mathbb{R}^{\tilde{n}\times d}$ is the prediction input matrix with rows $\tilde{\boldsymbol{x}}_1, \dots, \tilde{\boldsymbol{x}}_{\tilde{n}}$, and $\boldsymbol{L} \in \mathbb{R}^{n\times\tilde{n}}$ is the kernel matrix of training versus prediction inputs, $\boldsymbol{L}_{ij} = k(\boldsymbol{x}_i, \tilde{\boldsymbol{x}}_j)$. Alg. 1 provides pseudo-code for training and testing a KRR model using Cholesky decomposition.

2.7 What about other methods?

Two other popular choices for QM/ML modeling are Gaussian process regression[82] (GPR), sometimes called Kriging, and *artificial neural networks*[14–16] (ANN). GPR is the Bayesian equivalent of the frequentist KRR, and provides identical predictions, although other bells and whistles such as predictive variance differ. In GPR, the kernel plays the role of covariance between inputs. Fig. 6 presents a sketch of the basic idea of GPR. All models of form Eq. 14, including KRR and GPR, are *non-parametric** models, i.e., their number of parameters grows with the training data. ANN are *parametric* methods, having a fixed number of parameters (once network architecture has been chosen). A detailed analysis of ANN is beyond the scope of this tutorial. For further information, consult the reviews[28,29,92] and tutorial[30] on ANN for QM/ML by Jörg Behler.

2.8 Model selection and performance estimation

Model selection How does one choose the hyperparameters of a ML model? More generally, how does one choose between different ML models? The problem of choosing a model from a set of candidate models, given a dataset, is called *model selection.*[93] For KRR, parameters $\boldsymbol{\alpha}$

* Terminology from statistics: Parametric models summarize the training data in a predetermined number of parameters (e.g., fitting data with a normal distribution). For non-parameteric models, the number of parameters grows with the amount of training data.[91] Consequently, they require storing all or part of it (e.g., the $\boldsymbol{x}_i$ in Eq. 14). Due to their stronger assumptions, parametric models tend to be both computationally less demanding and less flexible. Note that non-parametric models do have parameters.

Algorithm 1: *Kernel ridge regression* training and predictions, based on Cholesky decomposition and forward-backward substitution.

Training:

Input: $\boldsymbol{K} \in \mathbb{R}^{n \times n}$, $\boldsymbol{y} \in \mathbb{R}^n$, $\lambda > 0$
Output: $\boldsymbol{\alpha} \in \mathbb{R}^n$

$\boldsymbol{K} \leftarrow \boldsymbol{K} + \lambda \boldsymbol{I}$
$\boldsymbol{U} \leftarrow \texttt{cholesky}(\boldsymbol{K})$ // $\boldsymbol{U}$ is upper triangular with $\boldsymbol{K} = \boldsymbol{U}^T \boldsymbol{U}$
Solve $\boldsymbol{U}^T \boldsymbol{\beta} = \boldsymbol{y}$ for $\boldsymbol{\beta}$, then $\boldsymbol{U} \boldsymbol{\alpha} = \boldsymbol{\beta}$ for $\boldsymbol{\alpha}$ // forward-backward substitution

---

Cholesky decomposition:

It is preferable to use an optimized implementation from a high-level language or a numeric library. This basic implementation follows the exposition by Golub and van Loan.[90]

for $i \leftarrow 1$ **to** n **do**
  $\boldsymbol{v} \leftarrow \boldsymbol{K}_{i \ldots n, i}$ // $\boldsymbol{v} \in \mathbb{R}^{n-i+1}$
  for $j \leftarrow 1$ **to** $i - 1$ **do** $\boldsymbol{v} \leftarrow \boldsymbol{v} - \boldsymbol{U}_{j,i} \boldsymbol{U}_{j, i \ldots n}$
  $\boldsymbol{U}_{i, i \ldots n} \leftarrow \boldsymbol{v} / \sqrt{v_1}$

Forward-backward substitution:

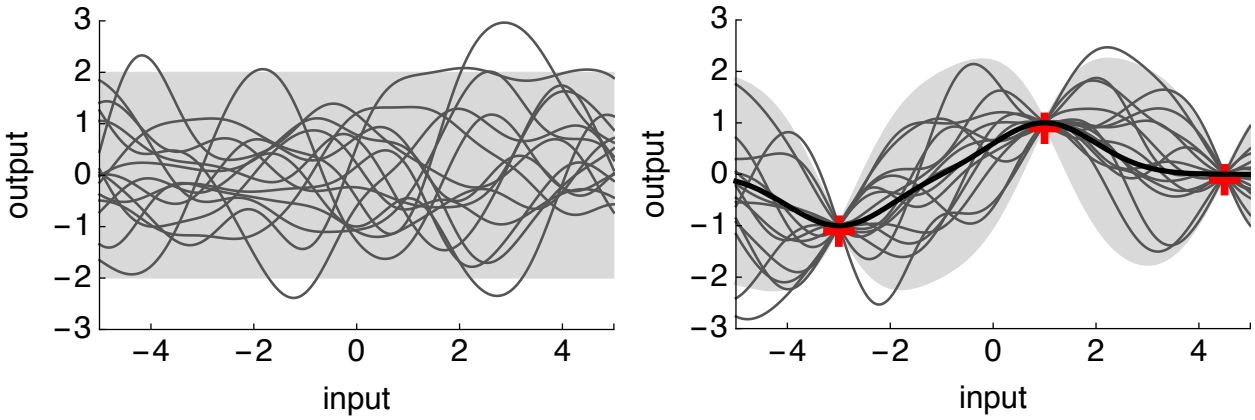
// Forward substitution $\boldsymbol{\alpha} = \text{ForwardSubstitution}(\boldsymbol{U}^T, \boldsymbol{y})$
for $i \leftarrow 1$ **to** n **do**
  $v \leftarrow y_i$
  for $j \leftarrow 1$ **to** $i - 1$ **do** $v \leftarrow v - u_{j,i}\, \alpha_j$
  $\alpha_i \leftarrow v / u_{i,i}$

// Backward substitution $\boldsymbol{\alpha} = \text{BackwardSubstitution}(\boldsymbol{U}, \boldsymbol{\alpha})$
for $i \leftarrow n$ **down to** 1 **do**
  $v \leftarrow \alpha_i$
  for $j \leftarrow n$ **down to** $i + 1$ **do** $v \leftarrow v - u_{i,j}\, \alpha_j$
  $\alpha_i = v / u_{i,i}$

Predictions:

Input: $\boldsymbol{L} \in \mathbb{R}^{n \times \tilde{n}}$, $\boldsymbol{\alpha} \in \mathbb{R}^n$
Output: $\tilde{\boldsymbol{f}} = (f(\tilde{\boldsymbol{x}}_1), \ldots, f(\tilde{\boldsymbol{x}}_{\tilde{n}}))^T \in \mathbb{R}^{\tilde{n}}$

$\tilde{\boldsymbol{f}} = \boldsymbol{L}^T \boldsymbol{\alpha}$



(a) Prior distribution. 15 samples (thin lines) drawn from a Gaussian process with zero mean and Gaussian covariance function with unit length scale.

(b) Posterior distribution with mean function (thick line). 15 samples (thin lines) drawn from the posterior distribution after conditioning on three training data (red crosses).

Figure 6: Idea of Gaussian process regression. Starting from the prior distribution (a), one conditions on the training data. Mean and variance of the posterior distribution (b) are used as predictor and confidence estimate. Shaded regions denote two standard deviations.

are determined via Eq. 17, given a training set. This leaves the choice of kernel k and regularization hyperparameter λ, plus any hyperparameters of the kernel, with the optimal choice depending on the dataset. A general guiding principle is *Occam's razor*,* which for our purposes states that among models with equal performance, the simplest one should be preferred. Many approaches to model selection are in use;[94] here, the focus is on performance estimation as selection criterion. As a specific example, given similar performance, for the models presented in this tutorial one should prefer (i) the linear kernel over the Gaussian and the Laplacian kernel, (ii) the Gaussian over the Laplacian kernel, (iii) higher regularization strengths, and (iv) larger length scales, the reason for (ii)–(iv) being smoothness of the estimator.

Estimating model performance. Ideally, we would like to know the error of our model on new data—predicting those is its purpose, after all. In *statistical learning theory*,[19,95,96] this is measured by the *risk* of the model f,

$$R(f) = \int L\big(y, f(\boldsymbol{x})\big)\, \mathrm{d}P(\boldsymbol{x}, y) = \mathbb{E}_P\big[L(y, f(\boldsymbol{x}))\big] \tag{20}$$

where P is the joint distribution of inputs and labels, and $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is a *loss function* measuring the error of a prediction. Eq. 20 is the expected error of f. Unfortunately, P is usually not known, and R has to be estimated from a finite set of training data as the *empirical risk*

$$R_n(f) = \frac{1}{n} \sum_{i=1}^{n} L\big(y_i, f(\boldsymbol{x}_i)\big). \tag{21}$$

* Attributed to William of Ockham (early 14th century), but already known to Aristotle and Ptolemy in classical antiquity.

The model f that minimizes the empirical error on the training set is usually a bad choice: If f is complex enough, it will essentially rote learn the training data and fail to generalize to new data (akin to a lookup table, which will perfectly reproduce the training set, and fail for all other inputs). This is called *overfitting*. One approach to counter overfitting is to use *regularization* by adding a term that penalizes model complexity,

$$\frac{1}{n}\sum_{i=1}^{n} L\big(y_i, f(\boldsymbol{x}_i)\big) + \lambda \left\|f\right\|^2 . \tag{22}$$

For quadratic loss $L(y, f(\boldsymbol{x})) = (y - f(\boldsymbol{x}))^2$, this is regularized regression as introduced earlier.

Choosing hyperparameters The preceding discussion provided a rationale for the choice of regression coefficients $\boldsymbol{\beta}$ and $\boldsymbol{\alpha}$ in Eqs. 13 and 17, given the hyperparameters. So how does one choose these? A simple strategy is to use a *hold-out* set (also *validation set*). This is a set $\{(\tilde{\boldsymbol{x}}_{\boldsymbol{i}}, \tilde{y}_i)\}_{i=1}^{\tilde{n}}$ of examples set aside in the beginning and not used for training. Given a set Θ of possible values for the hyperparameters $\boldsymbol{\theta}$, e.g., $\boldsymbol{\theta} \in \Theta = \{(\lambda, \sigma) \in \mathbb{R}^2 \mid \lambda, \sigma \geq 0\}$, one then optimizes the empirical risk over the hold-out set

$$\arg\min_{\boldsymbol{\theta}\in\Theta} \frac{1}{\tilde{n}}\sum_{i=1}^{\tilde{n}} L\big(\tilde{y}_i, f_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}}_i)\big) \tag{23}$$

to find the best hyperparameters. If not enough data is available to set aside a large enough hold-out set, methods like cross-validation[97] or bootstrapping[98] can be employed. These split the training data repeatedly in different ways, effectively re-using the data. The key to understanding the uses of hold-out sets, cross-validation, bootstrapping and similar statistical validation procedures is to follow the golden rule of model validation: Never estimate model performance on data that was used during training.

Depending on L, f, Θ, solving Eq. 23 can be hard. Grid search is an approach that is simple to implement, but computationally demanding and limited to at most two to three hyperparameters. In short, given a set of trial values for each hyperparameter, one sets up a grid $\Theta_1 \times \Theta_2 \times \ldots$ of all combinations of these values. For each grid entry, one trains a model on the training set using the corresponding hyperparameters and evaluates performance on the hold-out set. The hyperparameters resulting in best hold-out set performance are chosen. In practice, a logarithmic grid is often used.

Which *error statistics* should be reported? Default choices include the root mean square error (RMSE) and mean absolute error (MAE, also *unsigned error*),

$$\text{RMSE} = \sqrt{\frac{1}{\tilde{n}}\sum_{i=1}^{\tilde{n}}\big(\tilde{y}_i - f(\tilde{\boldsymbol{x}}_i)\big)^2} \quad \text{and} \quad \text{MAE} = \frac{1}{\tilde{n}}\sum_{i=1}^{\tilde{n}}\big|\tilde{y}_i - f(\tilde{\boldsymbol{x}}_i)\big|, \tag{24}$$

with the RMSE part of the optimization criteria in Eqs. 10 and 15, and the MAE the average magnitude of the model's error. Another useful and widely used statistic is the square of *Pearson's correlation coefficient* (also *product-moment correlation coefficient*). Let

$a_1, \ldots, a_k$, $b_1, \ldots, b_k$ be samples drawn from two random variables A, B. Then

$$R^2 = \frac{\operatorname{covar}^2(A,B)}{\operatorname{var} A \operatorname{var} B} = \frac{\left(n\sum_{i=1}^k a_i b_i - \sum_{i=1}^k a_i \sum_{j=1}^k b_j\right)^2}{\left(n\sum_{i=1}^k a_i^2 - \left(\sum_{i=1}^k a_i\right)^2\right)\left(n\sum_{i=1}^k b_i^2 - \left(\sum_{i=1}^k b_i\right)^2\right)}, \tag{25}$$

where var and covar denote variance and covariance. For linear models (one variable taking the role of labels $\tilde{y}_i$, the other taking the role of predictions $f(\tilde{\boldsymbol{x}}_i)$), $R^2 \in [0,1]$ can be interpreted as the percentage of label variance explained by the model. Summarizing performance on a whole dataset in a single number always loses information. It is therefore good practice during model development to look at the distribution of errors, e.g., in the form of a scatterplot (see Fig. **??** for an example) or a histogram.

3 Predicting atomization energies

For the practical part of this tutorial, assume the following scenario: You are given a dataset of 7 k small organic molecules, and are asked to predict their atomization energies at the density functional level of theory. Your computing resources allow you to do 1 k reference calculations. How accurate can you estimate the atomization energies of the whole dataset?

This section provides a step-by-step walkthrough of how to answer this question using the methodology introduced before. The setting is modeled after recent studies on predicting atomization energies using ML.[50,62–66] The supplementary information contains (i) a dataset of 7 k small organic molecules with their with atomization energies, allowing any choice of training set, model, and performance estimate to be retrospectively evaluated; (ii) a basic reference implementation of the introduced algorithms, written in the Wolfram language[2]; and, (iii) a notebook with solutions to the exercises. Together, these should enable you to start experimenting right away. If you prefer another programming environment, implementing the presented algorithms should be straightforward and provide for a good exercise.

3.1 Dataset

The dataset contains 7 k small organic molecules, taken from the generated database GDB[99], with force field-relaxed geometries and DFT atomization energies (see appendix for details). The molecules are composed of elements H, C, N, O, S, with up to 7 non-H atoms (Table 3). Note that the ML model maps a molecule's geometry at the force field minimum to its energy in the DFT minimum. The ML model thus needs to compensate for a change in geometry as well. To see why this is necessary, consider using the DFT minimum as input for prediction: For a new molecule, one would have to do a DFT calculation to relax its structure, the very calculation the ML model is supposed to replace.

▷ *Obtain the dataset* by downloading the supplementary material for this tutorial. Load the molecular structures and atomization energies from the file `dsgdb7ae2.xyz`. The file is in extended XYZ format (Fig. 7).

▷ Count the non-H atoms in each molecule to reproduce Table 3.

Table 3: *Distribution of molecular size* measured by number of non-H atoms.

non-H atoms	1	2	3	4	5	6	7	$\sum$
molecules	1	3	12	43	157	935	5 951	7 102

▷ Visualize some of the molecules.

```
4
0004 -403.695
C   0.8951 -0.0226 0.0041   0.8992 -0.0226 0.0041
C   2.1004 -0.0226 0.0041   2.0963 -0.0226 0.0041
H  -0.1633 -0.0226 0.0041  -0.1656 -0.0226 0.0041
H   3.1588 -0.0226 0.0041   3.1611 -0.0226 0.0041
```

Figure 7: *Extended XYZ file format*, as used in this tutorial. This is a plain text file format, where a molecule with k atoms is stored as $k + 2$ consecutive lines, containing number of atoms k, molecule identifier and atomization energy y, and, k lines giving element type and coordinates / Ångström (left block force field, right block DFT coordinates) of each atom.

Training set The first step is to choose 1 k molecules for which to "calculate" atomization energies (here, it's just looking them up since they are provided with the dataset; these are the labels allowed to use for model building in this toy scenario). For large homogeneous datasets, randomly drawing the training set is sufficient. However, Table 3 shows that the provided dataset is inhomogeneous with respect to number of non-hydrogen atoms. Since there are too few examples of molecules with 4 or fewer non-H atoms for reliable prediction, these are all included in the training set (the equivalent of doing QM calculations for them). The remaining 941 molecules are drawn randomly from all molecules with 5 or more non-H atoms, stratified by size, which is known to correlate with atomization energy. Stratification ensures that the distribution of sizes, and thus indirectly of atomization energies, is similar for training and prediction sets. All molecules not in the training set are assigned to the prediction set.

▷ *Create a training set* Select all k molecules with 4 or fewer non-H atoms. Sort the remaining molecules by number of atoms and select every $\frac{7102-k}{1000-k}$-th one, rounding appropriately. Assign selected molecules to the training set, and all others to the prediction set.

Hold-out set For model selection, i.e., choice of kernel and hyperparameters, one needs to split the training set. This can be done several ways, e.g., using cross-validation. Here, there is enough data to afford the simpler approach of setting aside a hold-out set of 100 molecules, using the remaining ones as the training set proper. The hold-out set is used to estimate performance, i.e., it acts as a proxy for the 6 k prediction set, and should resemble it as closely as possible (in a distribution sense). Therefore, it should not include molecules with 4 or fewer non-H atoms, and be stratified by number of atoms.

▷ *Create a hold-out set* From the training set, select 100 molecules with 5 or more non-H atoms, stratified by number of atoms, and assign them to the hold-out set. The remaining 900 molecules constitute the training set proper.

3.2 Representation

The *Coulomb matrix* is a simple matrix representation encoding element types and internal distances in a molecule. It was used in Refs. 50,62–66 to numerically represent molecules to the ML algorithm for prediction of atomization energies. Its entries are given by

$$\boldsymbol{M}_{ij} = \begin{cases} 0.5 Z_i^{2.4} & i = j \\ \frac{Z_i Z_j}{\|\boldsymbol{R}_i - \boldsymbol{R}_j\|} & i \neq j \end{cases}, \tag{26}$$

where Z_i is the atomic number (nuclear charge) of atom i, and $\boldsymbol{R}_i$ is its position in atomic units (Bohr radii a_0). Note that $\boldsymbol{M}$ is symmetric and has as many rows and columns as there are atoms in the molecule. Intuitively, each row (and column) of $\boldsymbol{M}$ corresponds to an atom, and encodes how it interacts with the rest of the molecule. Off-diagonal elements contain scaled internal distances, encoding geometry, whereas main diagonal elements, where distance (of an atom to itself) is zero, are fits to free atom energies, encoding element types. Eq. 26 uniquely encodes a molecule, i.e., $\{(Z_i,\ \boldsymbol{R}_i)\}$ can be reconstructed from $\boldsymbol{M}$ up to translations and rotations.

While Eq. 26 is invariant to translation and rotation of the molecule, it is not invariant to reindexing its atoms. One remedy is to sort Coulomb matrices by descending row norm by simultaneously permuting rows and columns accordingly.

▷ *Compute Coulomb matrices* Calculate Eq. 26 for all molecules in the dataset. For each matrix, calculate the norm of it's rows and sort it by simultaneously permuting rows and columns. Pad each matrix to the right and bottom with zeros so they all have the same size, 23×23, which is the maximum number of atoms per molecule in this dataset. Keep only the non-redundant lower triangular part, including the diagonal, of each matrix, and rearrange it into a 276-dimensional column vector.

3.3 Model building

Basic model Start with a model from the publication that introduced the Coulomb matrix,[62] using KRR with the Gaussian kernel. For given hyperparameters, this requires computing kernel matrices $\boldsymbol{K}$, $\boldsymbol{L}$, application of Alg. 1, and performance estimation.

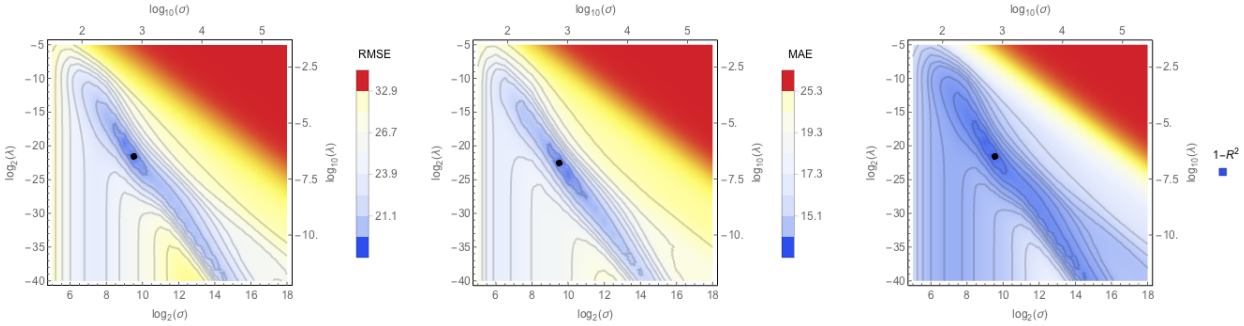
▷ *KRR with Gaussian kernel for given hyperparameters* Choose values for hyperparameters λ and σ. Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{900}$ denote the Coulomb matrices of the training set proper, and let $y_1, \ldots, y_{900}$ denote corresponding atomization energies. Calculate $\boldsymbol{K}$ using Eq. 5. Then, use Alg. 1 to compute regression coefficients $\boldsymbol{\alpha}$ using $\boldsymbol{K}$, $\boldsymbol{y}$, λ. For prediction on the hold-out set inputs $\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_{100}$, compute the kernel matrix $\boldsymbol{L}$, then use Alg. 1 to obtain predictions $\tilde{\boldsymbol{f}}$. Finally, compute performance statistics RMSE, MAE, $1 - R^2$ using $\tilde{\boldsymbol{f}}$ and $\tilde{\boldsymbol{y}}$.

Grid search Now that basic model building is established, determine the two hyperparameters λ and σ by performing a grid search: For each pair (λ, σ) of values in the base-2 logarithmic grid

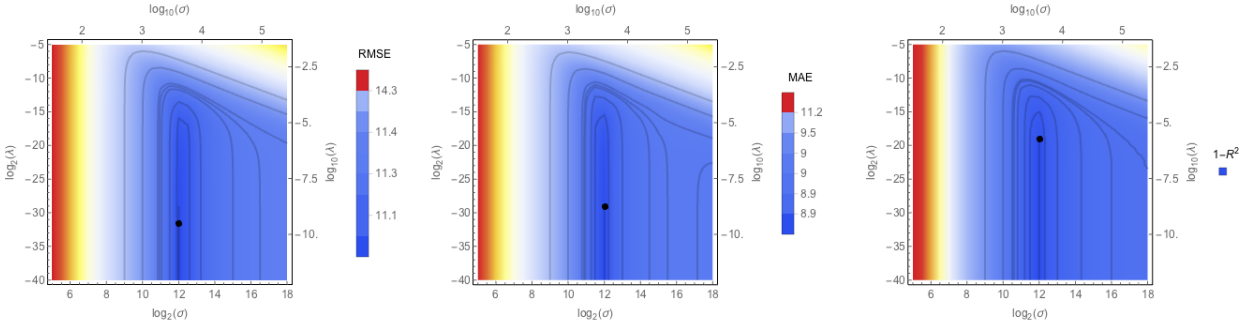
$$\left\{(2^i, 2^j) \;\middle|\; i = -40 : -5 : 0.5,\; j = 5 : 18 : 0.5\right\}, \tag{27}$$

where $a : b : c$ denotes the sequence $a, a + c, a + 2c, \ldots, b$ (from a to b with stepsize c), train a KRR model with Gaussian kernel on the training set proper and evaluate its performance on the hold-out set.

▷ *KRR with Gaussian kernel and grid search for hyperparameters* Build a model (as in the previous exercise) for each combination of hyperparameters in Eq. 27. Plot performance estimates as a function of λ and σ to reproduce Fig. 8. Determine optimal hyperparameters.



(a) Gaussian kernel.



(b) Laplacian kernel.

Figure 8: *Hold-out set performance as a function of hyperparameters*, evaluated on the base-2 logarithmic grid from Eq. 27. Shown are RMSE (left), MAE (middle), 1-R^2 (right) for Gaussian kernel (top) and Laplacian kernel (bottom). Optimal hyperparameters are shown as black dots.

Results Fig. 8 (top row) shows performance on the hold-out set as a function of hyperparameters. Table 4 presents numerical results. The optimal choice of hyperparameters is

$\lambda \approx 10^{-6.5}$ and $\sigma \approx 724$, yielding the values in Table 4. This is also the performance estimate for predicting the whole dataset. How well would the model have actually performed on the remaining 6 k molecules?

▷ Using the optimized hyperparameter values, train a model on the whole 1 k training set and predict the 6 k molecules in the prediction set.

Table 4: *Performance statistics for prediction of atomization energies.* RMSE and MAE in kcal mol^{-1}.

	hold-out set			prediction set		
Model	RMSE	MAE	R^2	RMSE	MAE	R^2
KRR, Gaussian kernel, grid search	19.2	13.5	0.993	17.7	12.5	0.993
KRR, Laplacian kernel, grid search	11.1	8.8	0.998	9.5	8.8	0.998

From Table 4, performance on the prediction set is better than estimated from the hold-out set. Note that it is generally desirable to overestimate rather than underestimate the error on new data.

Laplacian kernel The performance of the Gaussian kernel model is close to the one reported in the original publication[62]. Can one do better? A convenient approach to improve a kernel learning model is to use another kernel more suited to the problem. In subsequent publications, the Laplacian kernel often performed better for models using the Coulomb matrix as representation.

▷ Repeat the previous model building exercises, including the grid search, using the Laplacian kernel from Eq. 7 instead of the Gaussian kernel. Use the same training, prediction, proper training and hold-out sets.

For RMSE, MAE, and R^2, the optimal combination of hyperparameters is $\lambda \approx 10^{-12}$ and $\sigma \approx 10^{3.6}$, with performance shown in Fig. 8 (bottom row) and Table 4. Note the substantial improvement compared to the Gaussian kernel, reduced sensitivity to hyperparameters, particularly λ, and, closer agreement between performance estimate from the hold-out set and performance on prediction data.

4 What next?

Summary The purpose of this tutorial is to equip the reader with a basic understanding of kernel-based machine learning and its use in conjunction with computational quantum chemistry. The key idea is to use ML to interpolate between QM reference calculations, leading to substantial computational savings that can reach several orders of magnitude. For this, the decisive factor is control of the interpolation error, i.e., the ML approximation must be close enough to the QM reference to be able to act as a surrogate for it.

Table 5: *Free atom energies.*

	H	C	N	O	S
multiplicity	2	3	4	3	3
energy / E_h	-0.501036	-37.8054	-54.5438	-75.0186	-397.974

Further reading This tutorial is by necessity far from comprehensive. Many advanced topics, such as domain of applicability,[100,101] or, dynamic model retraining (learning on the fly),[102,103] have not been addressed. A cross-section of contemporary research on QM/ML models can be found in the special issue of International Journal of Quantum Chemistry where this tutorial was published. For more information on kernel ridge regression, see section 5.8 in the book by Hastie, Tibshirani and Friedman.[8] For a more in-depth treatment of kernel learning methods, consult any recent textbook on the topic, or the classic book by Schölkopf and Smola[71]. A review was published by Hofmann, Schölkopf and Smola in 2008.[73]

Acknowledgments I thank Felix Faber and Kuang-Yu "Samuel" Chang (張光宇) for testing the tutorial, Raghunathan Ramakrishnan for helpful discussions, and O. Anatole von Lilienfeld for helpful discussions and support via SNF grant PPOOP2_138932.

A Dataset

The tutorial's dataset is modeled after the one introduced in Ref. 62. The GDB-13 dataset[99] was downloaded from the website[104] of Jean-Louis Reymond in SMILES[105] format. A subset was selected by discarding all molecules with 8 or more non-H atoms and removing all molecules containing chlorine. Initial Cartesian coordinates were generated and subsequently relaxed using the Universal Force Field[106] as implemented in OpenBabel[107] (version 2.3.2). Structures were further relaxed and self-consistent field energies calculated at the density functional level of theory (DFT)[108] using the Perdew-Burke-Ernzerhof (PBE0)[109,110] functional with def2-TZVP basis set[111] as implemented in Gaussian[112] (version 09 rev. D.01). Atomization energies were then obtained by subtracting free atom energies computed in the same fashion (Table 5). Out of 7 173 calculations, 71 failed and were excluded. The remaining 7 102 molecules constitute the dataset.

References

[1] P. A. M. Dirac, Proc. Math. Phys. Eng. Sci. **123**, 714 (1929).

[2] Wolfram Research (`www.wolfram.com`), *Mathematica (version 10)* (2015).

[3] D. R. Bowler and T. Miyazaki, Rep. Progr. Phys. **75**, 036503 (2012).

[4] T. M. Mitchell, *Machine Learning* (McGraw Hill, 1997).

[5] R. Duda, P. Hart, and D. Stork, *Pattern Classification* (Wiley, New York, 2001), 2nd ed.

[6] D. MacKay, *Information theory, Inference, and Learning Algorithms* (Cambridge University Press, Cambridge, 2005).

[7] C. Bishop, *Pattern Recognition and Machine Learning* (Springer, Berlin, 2006).

[8] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning. Data Mining, Inference, and Prediction* (Springer, New York, 2009), 2nd ed.

[9] S. Lemm, B. Blankertz, T. Dickhaus, and K.-R. Müller, NeuroImage **56**, 387 (2011).

[10] F. Ricci, L. Rokach, and B. Shapira, in *Recommender Systems Handbook*, edited by F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor (Springer, 2011), pp. 1–35.

[11] J. Kober, J. A. Bagnell, and J. Peters, Int. J. Robot. Res. **32**, 1238 (2013).

[12] O. Ivanciuc, in *Reviews in Computational Chemistry*, edited by K. Lipkowitz and T. Cundari (Wiley, Hoboken, 2007), vol. 23, chap. 6, pp. 291–400.

[13] A. Varnek and I. Baskin, J. Chem. Inf. Model. **52**, 1413 (2012).

[14] C. Bishop, *Neural Networks for Pattern Recognition* (Oxford University Press, Oxford, 1996).

[15] S. O. Haykin, *Neural Networks and Learning Machines* (Pearson, 2008), 3rd ed.

[16] G. Montavon, G. B. Orr, and K.-R. Müller, eds., *Neural Networks: Tricks of the Trade*, vol. 7700 of *Lecture Notes in Computer Science* (Springer, Berlin, Germany, 2012), 2nd ed.

[17] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods* (Cambridge University Press, Cambridge, 2000).

[18] M. Johnson and G. Maggiora, eds., *Concepts and Applications of Molecular Similarity* (Wiley, New York, 1990).

[19] O. Bousquet, S. Boucheron, and G. Lugosi, in *Advanced Lectures on Machine Learning*, edited by O. Bousquet, U. von Luxburg, and G. Rätsch (Springer, Heidelberg, 2004), vol. 3176 of *Lecture Notes in Artificial Intelligence*, pp. 169–207.

[20] O. A. von Lilienfeld and M. E. Tuckerman, J. Chem. Phys. **125**, 154104 (2006).

[21] O. A. von Lilienfeld, Int. J. Quant. Chem. **113**, 1676 (2013).

[22] P. Geerlings, S. Fias, Z. Boisdenghien, and F. D. Proft, Chem. Soc. Rev. **43**, 4989 (2014).

[23] K. S. Chang and O. A. von Lilienfeld, CHIMIA Int. J. Chem. **68**, 602 (2014).

[24] O. A. von Lilienfeld, R. D. Lins, and U. Rothlisberger, Phys. Rev. Lett. **95**, 153002 (2005).

[25] M. Wang, X. Hu, D. N. Beratan, and W. Yang, J. Am. Chem. Soc. **128**, 3228 (2006).

[26] O. A. von Lilienfeld, J. Chem. Phys. **131**, 164102 (2009).

[27] J. Behler, J. Chem. Phys. **134**, 074106 (2011).

[28] J. Behler, Phys. Chem. Chem. Phys. **13**, 17930 (2011).

[29] J. Behler, J. Phys. Condens. Matter **26**, 183001 (2014).

[30] J. Behler, Int. J. Quant. Chem. **submitted** (2015).

[31] J. Ischtwan and M. A. Collins, J. Chem. Phys. **100**, 8080 (1994).

[32] R. Burkard and E. Çela, in *Handbook of Combinatorial Optimization*, edited by D. Du and P. Pardalos (Kluwer, 1999), vol. 1, pp. 75–149.

[33] K. Bennett and C. Campbell, SIGKDD Explor. **2**, 1 (2000).

[34] F. Ercolessi and J. B. Adams, Europhys. Lett. **26**, 583 (1994).

[35] G. G. Maisuradze, D. L. Thompson, A. F. Wagner, and M. Minkoff, J. Chem. Phys. **119**, 10002 (2003).

[36] Y. Guo, A. Kawano, D. L. Thompson, A. F. Wagner, and M. Minkoff, J. Chem. Phys. **121**, 5091 (2004).

[37] D. E. Makarov and H. Metiu, J. Chem. Phys. **108**, 590 (1998).

[38] B. Boser, I. Guyon, and V. Vapnik, in *Proceedings of the 5th Annual ACM Conference on Computational Learning Theory (COLT 1992), Pittsburgh, Pennsylvania, USA, July 27–29, 1992* (Association for Computing Machinery, 1992), pp. 144–152.

[39] B. Schölkopf, A. Smola, and K.-R. Müller, Neural. Comput. **10**, 1299 (1998).

[40] T. Ho and H. Rabitz, J. Chem. Phys. **89**, 5614 (1988).

[41] H. Heo, T.-S. Ho, K. K. Lehmann, and H. Rabitz, J. Chem. Phys. **97**, 852 (1992).

[42] T. Ho and H. Rabitz, J. Chem. Phys. **104**, 2584 (1996).

[43] G. M. E. Silva, P. H. Acioli, and A. C. Pedroza, J. Comput. Chem. **18**, 1407 (1997).

[44] S. Urata, A. Takada, T. Uchimaru, A. K. Chandra, and A. Sekiya, J. Fluor. Chem. **116**, 163 (2002).

[45] R. M. Balabin and E. I. Lomakina, Phys. Chem. Chem. Phys. **13**, 11710 (2011).

[46] A. Seko, T. Maekawa, K. Tsuda, and I. Tanaka, Phys. Rev. B **89**, 054303 (2014).

[47] *Materials genome initiative for global competitiveness*, White House whitepaper (2011), accessed 2015-01-15, URL `http://www.whitehouse.gov/mgi`.

[48] A. Jain, S. P. Ong, G. Hautier, W. Chen, W. D. Richards, S. Dacek, S. Cholia, D. Gunter, D. Skinner, G. Ceder, et al., APL Mater. **1**, 011002 (2013).

[49] R. Ramakrishnan, P. Dral, M. Rupp, and O. A. von Lilienfeld, Scientific Data **1**, 140022 (2014).

[50] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld, in preparation (2015).

[51] C. M. Handley, G. I. Hawe, D. B. Kell, and P. L. A. Popelier, Phys. Chem. Chem. Phys. **11**, 6365 (2009).

[52] A. P. Bartók, M. C. Payne, R. Kondor, and G. Csányi, Phys. Rev. Lett. **104**, 136403 (2010).

[53] J. C. Snyder, M. Rupp, K. Hansen, K.-R. Müller, and K. Burke, Phys. Rev. Lett. **108**, 253002 (2012).

[54] J. C. Snyder, M. Rupp, K. Hansen, L. Blooston, K.-R. Müller, and K. Burke, J. Chem. Phys. **139**, 224104 (2013).

[55] J. C. Snyder, M. Rupp, K. Hansen, K.-R. Müller, and K. Burke, in preparation (2014).

[56] Z. D. Pozun, K. Hansen, D. Sheppard, M. Rupp, K.-R. Müller, and G. Henkelman, J. Chem. Phys. **136**, 174101 (2012).

[57] G. Pilania, C. Wang, X. Jiang, S. Rajasekaran, and R. Ramprasad, Sci. Rep. **3**, 2810 (2013).

[58] J. Carrete, W. Li, N. Mingo, S. Wang, and S. Curtarolo, Phys. Rev. X **4**, 011019 (2014).

[59] V. Botu and R. Ramprasad, Int. J. Quant. Chem. **in press** (2015).

[60] R. Ramakrishnan, M. Rupp, A. Knoll, and O. A. von Lilienfeld, Int. J. Quant. Chem. **in preparation** (2015).

[61] A. P. Bartók, R. Kondor, and G. Csányi, Phys. Rev. B **87**, 184115 (2013).

[62] M. Rupp, A. Tkatchenko, K.-R. Müller, and O. A. von Lilienfeld, Phys. Rev. Lett. **108**, 058301 (2012).

[63] G. Montavon, K. Hansen, S. Fazli, M. Rupp, F. Biegler, A. Ziehe, A. Tkatchenko, O. A. von Lilienfeld, and K.-R. Müller, in *Advances in Neural Information Processing Systems 25 (NIPS 2012), Lake Tahoe, Nevada, December 3–6*, edited by P. Bartlett, F. C. Pereira, C. J. Burges, L. Bottou, and K. Q. Weinberger (MIT Press, 2012).

[64] G. Montavon, M. Rupp, V. Gobre, A. Vazquez-Mayagoitia, K. Hansen, A. Tkatchenko, K.-R. Müller, and O. A. von Lilienfeld, New J. Phys. **15**, 095003 (2013).

[65] K. Hansen, G. Montavon, F. Biegler, S. Fazli, M. Rupp, M. Scheffler, O. A. von Lilienfeld, A. Tkatchenko, and K.-R. Müller, J. Chem. Theor. Comput. **9**, 3543 (2013).

[66] M. Rupp, M. R. Bauer, R. Wilcken, A. Lange, M. Reutlinger, F. M. Boeckler, and G. Schneider, PLoS Comput. Biol. **10**, e1003400 (2014).

[67] L. van der Maaten, E. Postma, and J. van den Herik, Tech. Rep. TiCC TR 2009-005, Tilburg Centre for Creative Computing, Tilburg University (2009).

[68] P. Das, M. Moll, H. Stamati, L. E. Kavraki, and C. Clementi, Proc. Natl. Acad. Sci. USA **103**, 9885 (2006).

[69] Z. Chen and S. Haykin, Neural. Comput. **14**, 2791 (2002).

[70] M. Aizerman, E. Braverman, and L. Rozonoer, Autom. Rem. Contr. **25**, 821 (1964).

[71] B. Schölkopf and A. Smola, *Learning with Kernels* (MIT Press, Cambridge, 2002).

[72] J. Shawe-Taylor and N. Cristianini, *Kernel methods for pattern analysis* (Cambridge University Press, New York, 2004), 1st ed.

[73] T. Hofmann, B. Schölkopf, and A. Smola, Ann. Stat. **36**, 1171 (2008).

[74] C. Meyer, *Matrix Analysis and Applied Linear Algebra* (Society for Industrial and Applied Mathematics, Philadelphia, 2001).

[75] N. Aronszajn, Trans. Am. Math. Soc. **68**, 337 (1950).

[76] J. Mercer, Phil. Trans. Roy. Soc. Lond. **209**, 415 (1909).

[77] T. Kerr, J. Guid. Contr. Dynam. **13**, 571 (1990).

[78] I. Steinwart, D. Hush, and C. Scovel, IEEE Trans. Information Theory **52**, 4635 (2006).

[79] D. M. Hawkins, J. Chem. Inform. Comput. Sci. **44**, 1 (2004).

[80] A. Hoerl and R. Kennard, Technometrics **12**, 55 (1970).

[81] Bias Variance-Decomposition, in *Encyclopedia of Machine Learning*, edited by C. Sammut and G. I. Webb (Springer, 2010), pp. 100–101.

[82] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning* (MIT Press, Cambridge, 2006).

[83] R. Rosipal and L. Trejo, J. Mach. Learn. Res. **2**, 97 (2001).

[84] K. Bennett and M. Embrechts, in *Proceedings of the NATO Advanced Study Institute on Learning Theory and Practice, Leuven, Belgium, July 8–19*, edited by J. Suykens, G. Horváth, S. Basu, C. Micchelli, and J. Vandewalle (IOS Press, 2002), chap. 11, pp. 227–250.

[85] B. Schölkopf, R. Herbrich, and A. J. Smola, in *Proceedings of the 14th Annual Conference on Computational Learning Theory (COLT 2001) and 5th European Conference on Learning Theory (EuroCOLT 2001), Amsterdam, The Netherlands, July 16–19*, edited by D. Helmbold and B. Williamson (Springer, 2001), vol. 2111 of *Lecture Notes in Artificial Intelligence*, pp. 416–426.

[86] A. J. Smola, B. Schölkopf, and K.-R. Müller, Neural Networks **11**, 637 (1998).

[87] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, et al., ACM Trans. Math. Software **28**, 135 (2002).

[88] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, et al., *LAPACK Users' Guide* (Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999), 3rd ed.

[89] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes. The Art of Scientific Computing* (Cambridge University Press, Cambridge, 2007), 3rd ed.

[90] G. H. Golub and C. F. van Loan, *Matrix Computations* (John Hopkins Press, Baltimore, Maryland, 2013), 4th ed.

[91] K. P. Murphy, *Machine Learning. A Probabilistic Perspective* (MIT Press, 2012).

[92] J. Behler, in *Chemical Modelling Applications and Theory*, edited by M. Springborg (Royal Society of Chemistry Publishing, Cambridge, United Kingdom, 2010), vol. 7, pp. 1–41.

[93] I. Guyon, A. Saffari, G. Dror, and G. Cawley, J. Mach. Learn. Res. **11**, 61 (2010).

[94] I. Guyon, G. Cawley, G. Dror, and A. Saffari, eds., *Hands-On Pattern Recognition*, vol. 1 of *Challenges in Machine Learning* (Microtome Publishing, Brookline, Massachusetts, 2011).

[95] V. Vapnik, *Statistical Learning Theory* (Wiley, New York, 1998).

[96] V. Vapnik, IEEE Trans. Neural Networks **10**, 988 (1999).

[97] S. Arlot and A. Celisse, Stat. Surv. **4**, 40 (2010).

[98] B. Efron and R. Tibshirani, J. Am. Stat. Assoc. **92**, 548 (1997).

[99] L. C. Blum and J.-L. Reymond, J. Am. Chem. Soc. **131**, 8732 (2009).

[100] T. I. Netzeva, A. P. Worth, T. Aldenberg, R. Benigni, M. T. D. Cronin, P. Gramatica, J. S. Jaworska, S. Kahn, G. Klopman, C. A. Marchant, et al., Altern. Lab. Anim. **33**, 1 (2005).

[101] J. Jaworska, N. Nikolova-Jeliazkova, and T. Aldenberg, Altern. Lab. Anim. **33**, 445 (2005).

[102] G. Csányi, T. Albaret, M. C. Payne, and A. D. Vita, Phys. Rev. Lett. **93**, 175503 (2004).

[103] Z. Li, J. R. Kermode, and A. De Vita, Phys. Rev. Lett. (2015).

[104] *GDB-13 database*, accessed 2015-01-27, URL `http://www.gdb.unibe.ch/`.

[105] D. Weininger, J. Chem. Inform. Comput. Sci. **28**, 31 (1988).

[106] A. K. Rappé, C. J. Casewit, K. S. Colwell, W. A. Goddard III, and W. M. Skiff, J. Am. Chem. Soc. **114**, 10024 (1992).

[107] R. Guha, M. T. Howard, G. R. Hutchison, P. Murray-Rust, H. Rzepa, C. Steinbeck, J. Wegner, and E. L. Willighagen, J. Chem. Inf. Model. **46**, 991 (2006).

[108] K. Burke and L. O. Wagner, Int. J. Quant. Chem. **113**, 96 (2013).

[109] J. P. Perdew, K. Burke, and M. Ernzerhof, Phys. Rev. Lett. **77**, 3865 (1996).

[110] J. P. Perdew, K. Burke, and M. Ernzerhof, Phys. Rev. Lett. **78**, 1396 (1997).

[111] F. Weigend and R. Ahlrichs, Phys. Chem. Chem. Phys. **7**, 3297 (2005).

[112] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, B. Mennucci, G. A. Petersson, et al., *Gaussian 09 revision D.01*, Gaussian Inc., Wallingford CT 2009.