JCTC Journal of Chemical Theory and Computation

Article

pubs.acs.org/JCTC

Assessment and Validation of Machine Learning Methods for Predicting Molecular Atomization Energies

Katja Hansen,*[†] Grégoire Montavon,[‡] Franziska Biegler,[‡] Siamac Fazli,[‡] Matthias Rupp,[¶] Matthias Scheffler,[†] O. Anatole von Lilienfeld,[§] Alexandre Tkatchenko,[†] and Klaus-Robert Müller*[‡,∥]

[†]Fritz-Haber-Institut der Max-Planck-Gesellschaft, Berlin, Germany

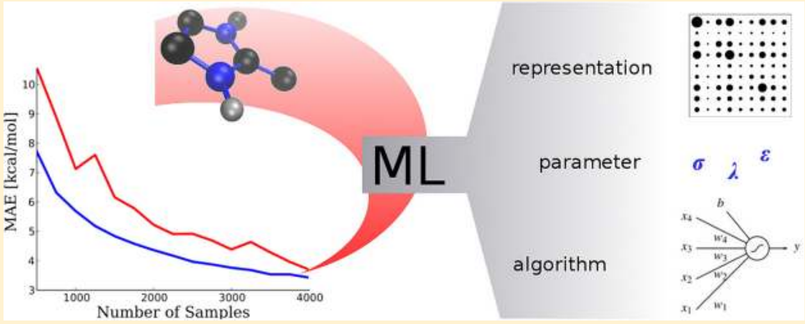
[‡]Machine Learning Group, TU Berlin, Germany

[¶]Institute of Pharmaceutical Sciences, ETH Zurich, Switzerland

[§]Argonne Leadership Computing Facility, Argonne National Laboratory, Lemont, Illinois, United States

[∥]Department of Brain and Cognitive Engineering, Korea University, Korea

S *Supporting Information*



ABSTRACT: The accurate and reliable prediction of properties of molecules typically requires computationally intensive quantum-chemical calculations. Recently, machine learning techniques applied to *ab initio* calculations have been proposed as an efficient approach for describing the energies of molecules in their given ground-state structure throughout chemical compound space (Rupp et al. *Phys. Rev. Lett.* **2012**, *108*, 058301). In this paper we outline a number of established machine learning techniques and investigate the influence of the molecular representation on the methods performance. The best methods achieve prediction errors of 3 kcal/mol for the atomization energies of a wide variety of molecules. Rationales for this performance improvement are given together with pitfalls and challenges when applying machine learning approaches to the prediction of quantum-mechanical observables.

1. INTRODUCTION

The accurate prediction of molecular properties in chemical compound space (CCS) is a crucial ingredient toward rational compound design in chemical and pharmaceutical industries. Therefore, one of the major challenges is to enable quantitative calculations of molecular properties in CCS at moderate computational cost (milliseconds per molecule or faster). However, currently only high level quantum-chemical calculations, which can take up to several days per molecule, yield the desired "chemical accuracy" (e.g., 1 kcal/mol for molecular atomization energies) required for predictive *in silico* rational molecular design. Therefore, more efficient algorithms that can predict properties of molecules quickly and reliably would be a powerful tool in order to sample and better understand CCS.

Throughout this paper, we focus on atomization energies of molecules in their ground-state equilibrium geometry. The atomization energy is an essential molecular property that determines the stability of a molecule with respect to the atoms that compose it. Atomization energies are also measurable experimentally and are frequently used to assess the performance of approximate computational methods. Though we focus on atomization energies the methods described in this paper could also be applied to predict total molecular energies or different quantum-mechanical properties.[1]

Recently, a machine learning (ML) approach has been demonstrated to predict atomization energies of various small molecules in their given ground-state geometry.[2] The method uses the same input as electronic-structure calculations, namely nuclear charges and atomic positions, and learns from a training set of *ab initio* molecular energies. Though the authors show that their proposed kernel ridge regression approach (details will be discussed below) outperforms the semiempirical PM6[3] and a simple bond-counting[4] scheme, the question arises whether other molecular descriptors or machine learning methods, e.g. neural networks, which have been successfully

Received: March 12, 2013
Published: July 11, 2013

applied for potential-energy surface (PES) description,[5] are also or even better suited for predicting atomization energies.

Let us briefly comment on the terminology. The term "model" here refers to a function trained on a set of molecules (training set) that returns a property value for a given molecule.[6] The term "prediction" refers to the fact that such a model is able to predict properties of molecules that were not used to fit the model parameters. We note, however, that the presented models are neither derived from nor explicitly based on physical laws. Purely data-driven machine learning algorithms are used to generate them. Thus, for molecules that behave significantly different than those of the training set, the prediction is likely to fail; the model may however assess the probability for own errors.[7] For example, when the training set would not contain first-row elements, the prediction of properties of first-row elements may not work, and, if the training set does not contain 3d transition metals, a failure for these elements is also to be expected. On the other hand predictions can in principle work even when the underlying physical laws are (still) unknown. Thus, the methods described below enable us to generate predictions for properties of a huge number of molecules that are unknown to the algorithm, but they must not be qualitatively distinct from the molecules of the training set.

In this work we show how significant improvements in the prediction of atomization energies can be achieved by using more specific and suitable ML approaches compared to the one presented by Rupp et al.[2] We review several standard ML techniques and analyze their performance, scaling, and handling of atomization-energy prediction with respect to different representations of molecular data. Our best methods reduce the prediction error from 10 kcal/mol[2] to 3 kcal/mol.

These explicit ML models for learning molecular energies have only been introduced recently. We therefore provide in this paper comprehensive instructions for the best practical use of this novel tool set. If model selection and validation are not carried out properly, overly optimistic or in the worst case simply useless results may be obtained. A number of common pitfalls are outlined to help avoid this situation.

In cheminformatics, ML has been used extensively to describe experimentally determined biochemical or physicochemical molecular properties such as in quantitative structure−activity relationships and quantitative structure−property relationships (e.g., refs 8−10). Since the 1990s neural networks have been proposed to model first-principles calculations for variable geometry.[5,11−20] Most of them are limited to fixed molecular compositions or systems with only a few different atom types. More recently, Gaussian processes and kernel ridge regression models were also applied to predict atomic multipole moments,[21] the PES of solids,[22] transition-state dividing surfaces,[23] and exchange-correlation functionals.[24] Hautier et al.[25] used machine learning techniques to suggest new crystal structures in order to explore the space of ternary oxides, while Balabin and Lomakina[26,27] applied neural networks and support vector machines to predict energies of organic molecules. The latter considered molecular descriptors and DFT energies calculated with small basis sets to predict DFT energies calculated with large basis sets. Most of these models either partition the energy and construct separate models for local atom environments or represent the whole molecule at once.

In this work we explore methods which consider whole chemical compounds at once to learn atomization energies of molecules across chemical compound space. Much work has been done (with and without ML) to describe nonequilibrium geometries and understand potential-energy surfaces of various molecular systems. However, due to the unmanageable size of CCS it is impossible to do QM calculations on large molecular databases. One also needs methods that can extend the accuracy of first-principles QM methods across CCS. Our work is aiming toward this perspective.

Note that we therefore restrict ourselves in this attempt to ground-state geometries and focus on enlarging the number and diversity of included systems. The incorporation of nonequilibrium geometries is the subject of ongoing work.

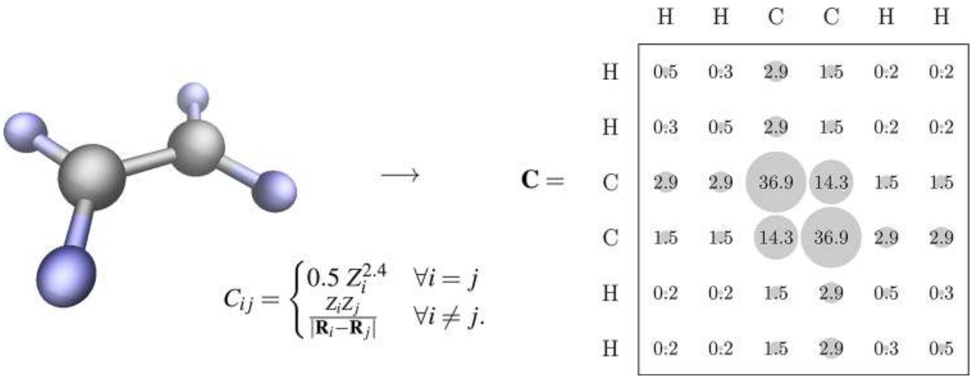
2. DATA SET AND DATA REPRESENTATION

In this section we describe the data set that is used to build and validate the ML models for atomization energy prediction. The quality and applicability of ML prediction models inherently depend on the size and diversity of this underlying data set. Moreover, the numerical representation of the included molecular structures is a critical aspect for model quality. Three different representations are introduced in this section and further discussed in Section 5.

2.1. Data Set. The chemical database GDB-13 contains all molecules obeying simple chemical rules for stability and synthetic feasibility up to 13 first- and second-row atoms of C, N, O, S, and Cl (970 million compounds).[28] In this work, as in Rupp et al.,[2] the subset formed by all molecules up to seven first- and second-row atoms of C, N, O, and S is extracted from the GDB-13. This data set contains 7165 structures with a maximal size of 23 atoms per molecule (including hydrogens). The GDB-13 gives bonding information in the form of SMILES[29] strings. These are converted to Cartesian coordinates of the ground-state structure using the OpenBabel implementation[30] of the force-field method by Rappé et al.[31] The atomization energies, which range from −800 to −2000 kcal/mol, are then calculated for each molecule using the Perdew-Burke-Ernzerhof hybrid functional (PBE0).[32,33] These single point calculations of geometry optimization were performed with a well converged numerical basis, as implemented in the FHI-aims code[34] (tight settings/tier2 basis set).

2.2. Data Representation. In order to apply machine learning, the information encoded in the molecular three-dimensional structure needs to be converted into an appropriate vector of numbers. This vectorial representation of a molecule is very important for the success of the learning approach. Only if all information relevant for the atomization energy is appropriately encoded in the vector will the machine learning algorithm be able to infer the relation between molecules and atomization energies correctly. Our representation should be solely based on atomic coordinates $\mathbf{R}_i$ and nuclear charges Z_i, as we want to pursue an approach from first principles that can deal with any stoichiometry and atomic configuration.

Different system representations that include internal coordinates, system-specific variables, and complex projections of local atom densities have been proposed in the context of potential-energy prediction.[22,5] Finding the optimal representation for molecules is the subject of ongoing research, and an in-depth discussion of all of them would be beyond the scope of this work. Therefore, we focus on three representations derived from the *Coulomb matrix* $\mathbf{C}$, a simple matrix representation introduced by Rupp et al.[2] (Figure 1).



$$C_{ij} = \begin{cases} 0.5\, Z_i^{2.4} & \forall i = j \\ \frac{Z_i Z_j}{|\mathbf{R}_i - \mathbf{R}_j|} & \forall i \neq j. \end{cases}$$

$$\mathbf{C} = \begin{array}{c|cccccc} & \mathrm{H} & \mathrm{H} & \mathrm{C} & \mathrm{C} & \mathrm{H} & \mathrm{H} \\ \hline \mathrm{H} & 0.5 & 0.3 & 2.9 & 1.5 & 0.2 & 0.2 \\ \mathrm{H} & 0.3 & 0.5 & 2.9 & 1.5 & 0.2 & 0.2 \\ \mathrm{C} & 2.9 & 2.9 & 36.9 & 14.3 & 1.5 & 1.5 \\ \mathrm{C} & 1.5 & 1.5 & 14.3 & 36.9 & 2.9 & 2.9 \\ \mathrm{H} & 0.2 & 0.2 & 1.5 & 2.9 & 0.5 & 0.3 \\ \mathrm{H} & 0.2 & 0.2 & 1.5 & 2.9 & 0.3 & 0.5 \end{array}$$

Figure 1. Coulomb matrix representation of ethene: A three-dimensional molecular structure is converted to a numerical Coulomb matrix using atomic coordinates $\mathbf{R}_i$ and nuclear charges Z_i. The matrix is dominated by entries resulting from heavy atoms (carbon self-interaction $0.5 \cdot 6^{2.4} = 36.9$, two carbon atoms in a distance of 1.33 Å result in $((6.6)/(1.33/0.529)) = 14.3$). The matrix contains one row per atom, is symmetric, and requires no explicit bond information.

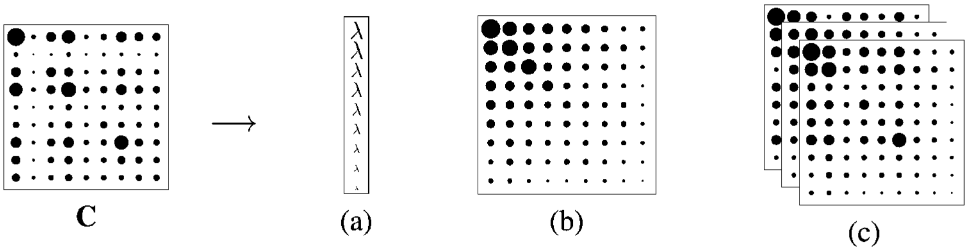


Figure 2. Three different permutationally invariant representations of a molecule derived from its Coulomb matrix **C**: (a) eigenspectrum of the Coulomb matrix, (b) sorted Coulomb matrix, (c) set of randomly sorted Coulomb matrices.

The main diagonal of the Coulomb matrix $0.5\ Z_i^{2.4}$ consists of a polynomial fit of the nuclear charges to the total energies of the free atoms,[2] while the remaining elements contain the Coulomb repulsion for each pair of nuclei in the molecule. Except for homometric structures (not present in the data set) the Coulomb matrix is a unique representation of molecules.

The fact that rotations, translations, and symmetry operations such as mirror reflections of a molecule in 3D space keep the total energy constant is reflected by the invariance of the Coulomb matrix with respect to these operations.

However, there are two problems with the representation of molecules by their Coulomb matrices, which make it difficult to use this representation in a vector-space model. First, different numbers of atoms d result in different dimensionalities of the Coulomb matrices, and second there is no well-defined ordering of the atoms in the Coulomb matrix; therefore, one can obtain up to $d!$ different Coulomb matrices for the same molecule by simultaneous permutation of rows and columns, while the energies of all these configurations remain unchanged.

In order to solve the first problem we introduce “dummy atoms” with zero nuclear charge and no interactions with the other atoms. In the Coulomb matrix representation this is achieved by padding each matrix with zeros,[2] which causes all matrices to have size $d \times d$ (where d is the maximal number of atoms per molecule).

The ambiguity in the ordering of the atoms is more difficult as there is no obvious physically plausible solution. To overcome this problem we investigate three candidate representations derived from the Coulomb matrix. They are depicted in Figure 2: (a) the eigenspectrum representation consisting of the sorted eigenvalues of **C**, (b) a sorted variant of the Coulomb matrix based on a sorting of the atoms, and (c) a set of Coulomb matrices, which all follow a slightly different sorting of atoms. All of them are explained in more detail below.

2.2.1. Eigenspectrum Representation. In the eigenspectrum representation the eigenvalue problem $\mathbf{Cv} = \lambda \mathbf{v}$ for each Coulomb matrix **C** is solved to represent each molecule as a vector of sorted eigenvalues $(\lambda_1,...,\lambda_d)$, $\lambda_i \geq \lambda_{i+1}$. This representation (first introduced by Rupp et al.[2]) is invariant with respect to permutations of the rows and columns of the Coulomb matrix.

Computing the eigenspectrum of a molecule reduces the dimensionality from $(3d-6)$ degrees of freedom to just d. In machine learning, dimensionality reduction can sometimes positively influence the prediction accuracy by providing some regularization. However, such a drastic dimensionality reduction can cause loss of information and introduce unfavorable noise (see Moussa[35] and Rupp et al.[36]), like any coarse-grained approach.

2.2.2. Sorted Coulomb Matrices. One way to find a unique ordering of the atoms in the Coulomb matrix is to permute the matrix in such a way that the rows (and columns) C_i of the Coulomb matrix are ordered by their norm, i.e. $||C_i|| \geq ||C_{i+1}||$. This ensures a unique Coulomb matrix representation. As a downside, this new representation makes the problem much higher-dimensional than it was when choosing only eigenvalues. The input space has now dimensionality N_{atoms}^2 compared to N_{atoms} for the eigenspectrum representation. Also, slight variations in atomic coordinates or identities may cause abrupt changes in the Coulomb matrix ordering, thereby impeding the learning of structural similarities.

2.2.3. Random(-ly Sorted) Coulomb Matrices. The problem of discontinuities due to abrupt changes in the matrix ordering can be mitigated by considering for each molecule a set of

Table 1. Selection of Loss Functions for Regression Tasks[a]

parameter	comments
squared error loss	$l_{\mathrm{se}}(y_i, f(\mathbf{x}_i)) = (y_i - f(\mathbf{x}_i))^2 \quad (2)$
absolute error loss	$l_{\mathrm{ae}}(y_i, f(\mathbf{x}_i)) = \lvert y_i - f(\mathbf{x}_i)\rvert \quad (3)$
ε-insensitive loss	$l_\varepsilon(y_i, f(\mathbf{x}_i)) = \lvert f(\mathbf{x}_i) - y_i\rvert_\varepsilon = \begin{cases} 0 & \mathit{if} \lvert f(\mathbf{x}_i) - y_i\rvert \le \varepsilon \\ \lvert f(\mathbf{x}_i) - y_i\rvert - \varepsilon & \mathit{otherwise} \end{cases} \quad (4)$

[a] The squared error loss is the most commonly used. The absolute error compared to the squared error is more robust to outliers but not differentiable. The ε-insensitive loss leaves errors up to ε unpenalized and has the effect of introducing some slack (or looseness).

Coulomb matrices rather than a single sorted Coulomb matrix.[37] To generate these randomly sorted Coulomb matrices we construct the Coulomb matrix based on a random ordering of the atoms and compute the row norms $\|\mathbf{C}\|$ (i.e., a vector containing the norm of each row of C_i). We add random noise $\varepsilon \sim \mathcal{N}(0,\sigma I)$ to disturb the vector $\|\mathbf{C}\|$ and determine the permutation P that sorts $\|\mathbf{C}\|+\varepsilon$. Finally, the rows and columns of the Coulomb matrix $\mathbf{C}$ are permuted according to this permutation, i.e. $\mathbf{C}_{\text{random}} = \text{permuterows}_P(\text{permutecols}_P(\mathbf{C}))$. (Note that for no noise ($\sigma = 0$) this equals the sorted Coulomb representation described above.)

This procedure corresponds to an approximate sampling from the conditional distribution of all possible valid Coulomb matrices given a specific molecule.[37] Similar approaches have been used in a variety of contexts, for example, feeding elastically distorted handwritten digits to a neural network[38,39] or a support vector machine,[40] leading to dramatic performance improvements.

Note that the increased number of samples caused by considering a set of random Coulomb matrices for each molecule helps to overcome the high-dimensionality of the input space but also considerably increases the computational costs for some ML methods. We discuss this problem in Section 4.2.2.

3. MACHINE LEARNING METHODS

Machine learning (ML) seeks to infer dependencies from data using computing systems with learning capability. This subfield of artificial intelligence evolved in the 1950s from the intersection of computer science, statistics, and neuroscience and gave rise to various learning algorithms commonly used in bioinformatics, computer vision, speech recognition, and finance.[41−47]

In this paper we focus on the description of atomization energies of molecules in their ground-state structures. From a mathematical point of view this is a regression task: We seek to find a function or model $f \in \mathcal{F}$ that maps an input vector $\mathbf{x} \in \mathbb{R}^d$ (representing the molecule: nuclear number and position of all atoms) onto the corresponding continuous label value $y \in \mathbb{R}$ (here the atomization energy). The space of functions $\mathcal{F}$ depends on the employed machine learning method and incorporated additional information and assumptions. Since we consider a training data set $\{(\mathbf{x}_1,y_1),...,(\mathbf{x}_n,y_n)\}$ in order to find $f \in \mathcal{F}$, the task falls into the category of *supervised learning*. The ML problem is formulated as a minimization problem of the form

$$\min_f \sum_{i=1}^{n} l(f(\mathbf{x}_i), y_i) + \lambda r(f) \text{ with } f \in \mathcal{F} \tag{1}$$

The first term of the objective is the empirical risk described by a loss function l which measures the quality of the function f. A specific case is the squared loss $l(\hat{y}, y) = (\hat{y} - y)^2$. Some common loss functions are given in Table 1. The second term of the objective in eq 1 is a regularization term which measures the complexity or roughness of the function f. In general $r(f)$ is a norm of f or its derivatives, e.g. $\int f''(x)^2 \mathrm{d}x$ can be used to favor functions that vary slowly over small regions of input space. The interplay between these two terms can be interpreted as follows: *Among all functions f that predict outputs y from inputs x well, choose the one that is the least complex.*

In addition to the need to carefully design the complexity penalty $\lambda r(f)$, we also need to make sure that the space of functions $\mathcal{F}$ contains enough functions that can approximate the physical dependencies between molecular structures and atomization energies. The atomization energy results from the quantum mechanical interactions between many electrons. Thus we expect a function that reasonably approximates atomization energies to be complex and highly nonlinear. In this work the effect of the considered space of functions on the quality of the approximation is illustrated by moving from the space of linear functions to nonlinear differentiable functions.

The simultaneous need for sophisticated function classes $\mathcal{F}$ and appropriate regularizers $r(f)$ underlies the design of all ML algorithms. Learning algorithms that implement them in one way or another are described in the following sections. For the sake of notational simplicity we will assume in the formulas that the data are centered, i.e. $(1/n)\Sigma_{i=1}^{n} y_i = 0$ and $(1/n)\Sigma_{i=1}^{n}\mathbf{x}_i = \mathbf{0}$. The main differences of the regression methods discussed in the following lie in two aspects, namely the set of candidate functions for f that are taken into account and the criteria applied to select the best candidate functions, i.e. the choice of functionals l and r.

3.1. Linear Ridge Regression. One of the simplest and most popular regression models is *least-squares linear regression*, where the unknown function is approximated by the hyperplane that minimizes the squared distance between the predicted and the true function value. In ridge regression the objective function of the least-squares linear regression model is extended by a *regularizer* to make the model less sensitive to outliers. It reads in analogy to eq 1 as

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i))^2 + \lambda \cdot \|\mathbf{w}\|^2 \text{ with } f(\mathbf{x}) = \mathbf{x}^T\mathbf{w} \tag{5}$$

for a given $\lambda > 0$. The minimization problem of eq 5 can be solved in closed form with $\mathbf{w} = (\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$, where $\mathbf{y}$ is the training label vector, and $\mathbf{X}$ refers to the matrix of all input vectors. This approach, in contrast to the following ones, is limited to the set of linear functions, i.e. only linear

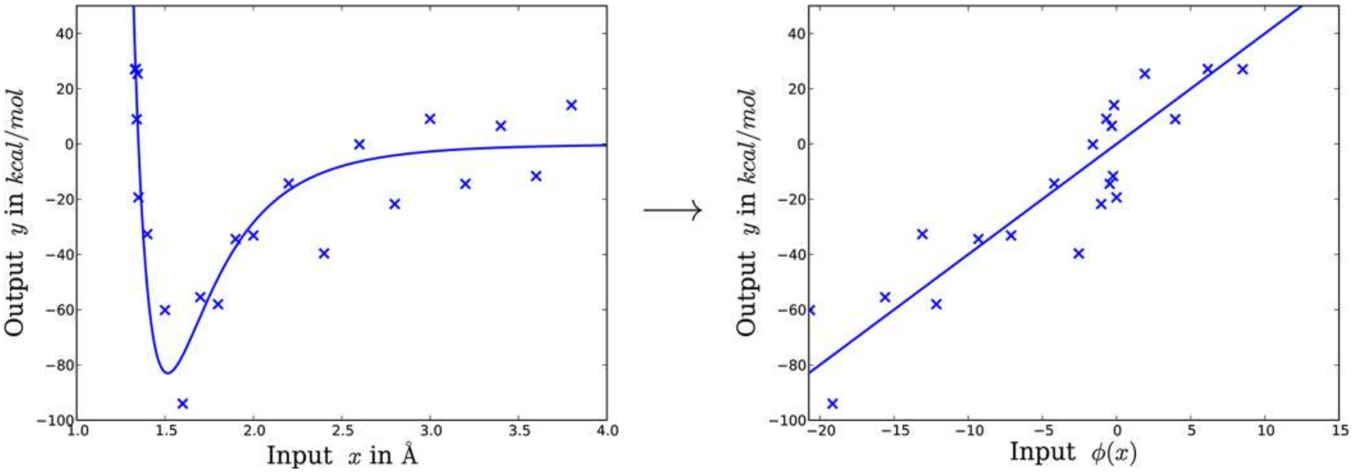


Figure 3. Basic idea of kernel ridge regression illustrated for the task of learning a Lennard-Jones potential function: The data points are mapped into feature space in such a way that the learning problem becomes linear. The linear solution in feature space corresponds to a nonlinear function in input space.

dependencies between molecules represented as vectors $\mathbf{x}_i$ and their corresponding energies are captured.

The regularization parameter λ controls the balance between the quality of the fit (here measured using the squared loss) and the complexity of the function. The parameter λ is a so-called "hyperparameter". It is not determined within training (i.e., solving the optimization problem eq 5) and needs to be chosen separately within a so-called "model selection" procedure (cf. Section 4.1). In general, regularization is needed in order to work with noisy data, such as experimental measurements of molecular energies. However, in this work we aim to reproduce the results of an electronic structure calculation, and these calculated molecular energies include no noise up to numerical precision. The concept of regularization is still beneficial in order to focus on less complex solutions and to cope with ambiguities in the representation of molecules. For example, based on the molecular representations introduced in Section 2.2 two homometric molecular structures of different energy values are mapped onto exactly the same input vectors $\mathbf{x}_i$. From an algorithmic perspective this situation could also result from two noisy measurements of the same input. This ambiguity can be handled using regularization.

3.2. Kernel Ridge Regression. Kernel ridge regression generalizes the linear ridge regression model toward nonlinear functions.[44] The nonlinearity is incorporated by mapping the data from the input space into a different space called "feature space", aiming to transform the original, nonlinear regression task into a linear task in feature space. This idea is illustrated for a simple one-dimensional example in Figure 3: there is no linear relation between the distance between two atoms and the corresponding energy as depicted by the Lennard-Jones potential in the left plot. However, if we apply an appropriate mapping function to the distances x we get a new variable $\phi(x)$ which lives in a new one-dimensional space called feature space and is perfectly correlated with the energy. In general it is not that obvious how to choose the mapping function ϕ. Often the most suitable functions are rather complex and result in infinite dimensional feature spaces. Since many learning algorithms only require dot products between the data vectors, this can be handled using the so-called *kernel trick*: Instead of mapping the data and computing the inner product in feature space, we can use a kernel function $k(x,x')$ to do it in one operation and leave the mapping function and feature space completely implicit. According to Mercer's Theorem[48] any symmetric positive semidefinite function allows for an inner product representation $k(x,x') = <\phi(x),\phi(x')>$.

Some commonly used kernels are listed in Table 2. Here $\|x\|$ corresponds to a Euclidean L^2-norm, while $|x|$ corresponds to a Manhattan, or city block, L^1-norm. Note that now a kernel function needs to be chosen instead of a complex mapping function. The kernel function facilitates treating nonlinear problems by mapping into infinite-dimensional feature spaces. Mathematically they allow us to move our problem from a d-dimensional input space into an n-dimensional space spanned by the n data points. There we can focus on the task relevant subspace to solve our problem.[49] Often several kernel functions yield good results for the same data set — however, the optimal choice cannot be assessed beforehand and needs to be determined in statistical tests.

Table 2. Selection of Commonly Used Kernels

linear kernel	$k(x, x') = x{\cdot}x'$	(6)
polynomial kernel	$k(x, x') = (x{\cdot}x' + c)^d$	(7)
Gaussian kernel	$k(x, x') = \exp\left(-\frac{1}{2\sigma^2}\|x - x'\|^2\right)$	(8)
Laplacian kernel	$k(x, x') = \exp\left(-\frac{1}{\sigma}|x - x'|\right)$	(9)

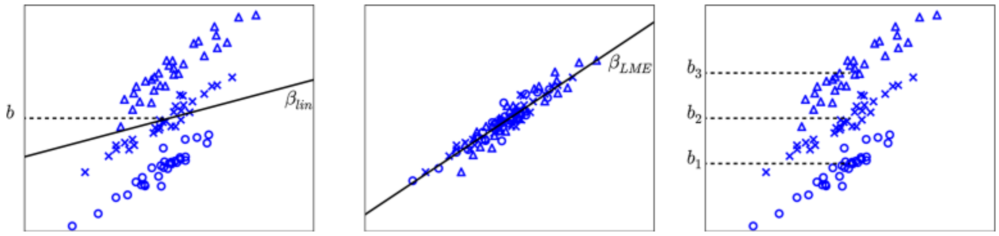
In this paper both the Gaussian as well as Laplacian kernels showed promising results in preliminary experiments and will be used to train our models.

Kernel ridge regression[44] uses a quadratic constraint on the norm of the weights α_i and solves the nonlinear regression problem

$$\min_{\alpha} \sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda \cdot \sum_{i,j} \alpha_i k(x_i, x_j)\alpha_j$$
$$\text{with } f(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x) \qquad (10)$$

The minimization problem has the closed form solution $\alpha = (\mathbf{K} + \lambda \cdot I)^{-1}\mathbf{y}$, where $\mathbf{y}$ is the training label vector and $\mathbf{K}$ is the kernel

Scheme 1. Illustration of Linear Mixed-Effects Models Assuming That the Molecules Are Grouped into Three Clusters (Differentiated by Symbols) and within Each Cluster the Dependency between Molecular Structure (Here Represented As One-Dimensional Input, x-Axis) and Atomization Energy (y-Axis) Are of Similar Slope but Different Bias[a]



[a] Left: Using linear regression on the whole data set does not model the data appropriately. Center: With a linear mixed-effects model, the fixed-effect β_{LME} is estimated first from the decorrelated data by linear regression. Right: In a second step, the random effects b_1, b_2, and b_3 are recovered.

matrix. The regularization parameter λ is again a hyperparameter, as are any kernel dependent parameters, such as, in the case of the Gaussian and Laplacian kernel, the kernel width σ. They need to be chosen separately within a model selection procedure (cf. section 4.1).

One main drawback of kernel-based methods such as kernel ridge regression is that they are in general not well suited for large data sets. In our work this issue only arises when many random Coulomb matrices are used to represent one molecule (see Section 4.2.2 for further discussion).

3.3. Support Vector Regression.

Support vector regression (SVR)[46,50,51] is a kernel-based regression method, which can be depicted as a linear regression in feature space — like kernel ridge regression. Unlike kernel ridge regression, SVR is based on an ε-insensitive loss function (eq 4), where absolute deviations up to ε are tolerated.

The corresponding convex optimization problem has a unique solution, which cannot be written in a closed form but is determined efficiently using numerical methods for quadratic programming (see Chapter 10 of Schölkopf and Smola[46] and Platt[52]). One key feature of SVR is sparsity, i.e. only a few of the data points contribute to the solution. These are called *support vectors*. Though the number of support vectors is generally small it may rise dramatically for very complex or very noisy problems.[43]

For a given data set with n data points, SVR solves the optimization problem

$$\max_{\alpha_i,\alpha_i^*} -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)k(x_i,x_j) - \varepsilon\sum_{i=1}^{n}(\alpha_i+\alpha_i^*)+\sum_{i=1}^{n}y_i(\alpha_i-\alpha_i^*), \qquad (11)$$

subject to the following constraints

$$\sum_{i=1}^{n}(\alpha_i-\alpha_i^*)=0 \text{ and } 0\le\alpha_i,\alpha_i^*\le C \qquad (12)$$

where C is a regularizing hyperparameter. The regression function takes the form

$$f(x)=\sum_{i=1}^{n}(\alpha_i-\alpha_i^*)k(x_i,x) \qquad (13)$$

Based on preliminary experiments the Gaussian and Laplacian kernel were selected for the support vector regression employed in this study. Thus two hyperparameters need to be optimized, namely the kernel width σ and the parameter C.

3.4. Mixed Effects.

The space of relaxed geometries of small chemical compounds we are considering is not equally populated and exhibits some intrinsic structure, e.g. clusters of compounds with the same stoichiometry surrounded by cavities due to not chemically feasible stoichiometries. The forming and characteristics of these clusters depend on the metric, which is used to measure the distance between compounds. In Section 5 and Figure 10 we discuss the shape of clusters generated by different metrics used in our work.

Mixed-effects models can be used to account for the cluster structures in compound space. They are intended for the analysis of such grouped or clustered data.[53] Mixed-effects models divide the sources of variation of atomization energy into two categories, namely within-group variation (called *random effects*) and between-group variation (called *fixed effects*). Let us focus on l_1-penalized linear mixed-effect models, which assume a linear target function of the form

$$y_i = \mathbf{x}_i^F\beta + \mathbf{x}_i^M b_i + \varepsilon_i \quad i=1,...,N \qquad (14)$$

for each group i. The first part of this model ($\mathbf{x}_i^F\beta$) describes the fixed effects, i.e. the linear effect of the input features $\mathbf{x}_i$ on the atomization energy y_i that is *independent* of the group structure. The second part ($\mathbf{x}_i^M b_i$) captures the group-dependent contributions (random effects), and ε_i is an independently and identically distributed (iid) random error term.

For our data set of 7165 molecules the most stable cluster structure (with respect to different clustering methods and data partitioning) was reached based on eight cluster centers. These clusters mainly differed in terms of molecular size and atomic composition (especially whether sulfur is present or not). Since it is impossible to visualize the mixed effect models in our input space of more than 20 dimensions we illustrate the idea in Scheme 1: after assigning the molecules into the different clusters the data are decorrelated in such a way that the covariances between input dimensions becomes zero and the variance in each dimension one. This way the molecular representation and atomization-energy information causing the grouping is partially removed. Afterward linear regression is used to estimate the fixed effect β. In a last step the random effects b_i are recovered for each cluster.

The l_1-penalized linear mixed-effects model allows for high-dimensional input data and efficient selection of the fixed-effects variables by implementing a Lasso-type concept[54] (see Schelldorfer et al.[55] and Fazli et al.[56] for details). In this work l_1-penalized linear mixed-effects models are used as a nonlinear method. To incorporate the nonlinearity the kernel trick is applied, and the mixed effect is integrated into a kernel matrix by adding a block-wise diagonal matrix of group memberships to the original kernel matrix in a kernel ridge regression model.[56]

3.5. Multilayer Neural Networks. In multilayer neural networks, the function f that maps inputs to outputs is implemented by a directed set of interconnected units, which are called *neurons*.[41,57,58] Each neuron implements a basic computation, typically $y = \tanh(\sum_i w_i x_i + b)$, where $\{x_i\}$ are the inputs to the neuron, $\{w_i\}$ are the weights, b is a constant, and y is the neuron output. A neuron is depicted in Figure 4 (left).

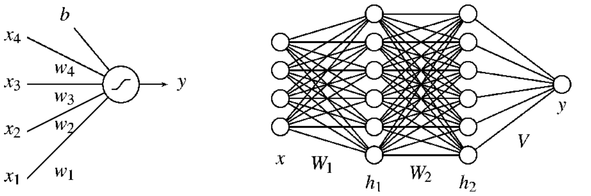


Figure 4. Left: example of a single neuron annotated with its inputs, weights, bias, and output. Right: multilayer neural network with its multiple layers and parameters.

The neurons are placed in a layered architecture as shown in Figure 4 (right). The mapping between inputs and outputs is implemented by the following computation:

$$\mathbf{h}_i = \tanh(\mathbf{W}_i \cdot \mathbf{h}_{i-1} + \mathbf{b}_i) \quad \text{for } 1 \leq i \leq L, \text{ and } \mathbf{h}_0 = \mathbf{x}$$
$$y = \tanh(\mathbf{V} \cdot \mathbf{h}_L)$$

The matrices $\mathbf{W}_1,...,\mathbf{W}_L,\mathbf{V}$ and vectors $\mathbf{b}_1,...,\mathbf{b}_L$ are the model parameters and have to be learned from data. Neural networks are generally trained one sample at a time, using stochastic gradient descent.[59] Typically the training procedure requires a large number of iterations but scales well with the number of samples. Thus neural networks are able to absorb a large number of data points such as several random Coulomb matrices per molecule, and as we will see later, they strongly benefit from these additional data points. An advantage of neural networks is that they are able to automatically extract, layer after layer, the representation necessary for solving the task.[57,60−62] As a downside, neural networks algorithms typically have multiple local minima, i.e. if a parameter configuration where any small change to this parameter set only reduces the model quality is discovered, we have in general not reached the optimal parameter set. This implies that successful training requires an experienced choice of parametrizations, learning rates, and initializations in order to find the best parameter set.[57,58]

3.6. *k*-Nearest Neighbor. A further well-known nonlinear algorithm is the *k*-nearest neighbor method (KNN).[44] For each new sample $\mathbf{x}$ the k nearest training samples are selected, and the predominant label among those neighbors determines the label in classification tasks; for regression tasks the (weighted) mean label value is taken into account. Despite its simplicity the KNN approach often works well, if a reasonable distance measure is applied (typically the Euclidean distance). Only one hyperparameter k needs to be determined, e.g. by cross-validation. An illustration of a KNN prediction for ethanol is given in Figure 5. Note that this approach fails on our molecular data set especially for very small molecules, where few compounds of similar size and structure are available.

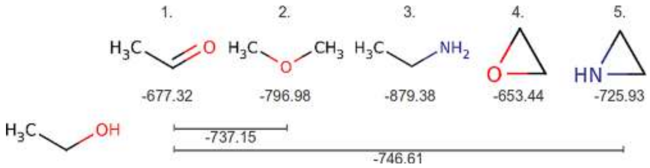


Figure 5. *k*-nearest neighbor prediction for ethanol with $k = 5$ ($k = 2$): The five (two) closest neighbors of ethanol with respect to Euclidean distances of the sorted Coulomb matrix representations are calculated, and the average over the corresponding atomization energies (given below the structures in kcal/mol) is used as predicted value (true value −808 kcal/mol).

4. METHODOLOGY

4.1. Concepts of Model Selection and Validation.

4.1.1. Model Assessment. The overall goal of machine learning is to establish high-quality prediction models. The quality of a model is measured by the *generalization error* − the expected prediction error on new data. As the error on new data cannot be known, one must approximate it by an inference procedure, which allows one to judge how well the learning machine generalizes.

The simplest procedure separates the data set into a training and a test set (also called the hold-out set). The model is then built on the training set, and the average loss over the test set (*test error*)

$$err_{\text{test}} = \frac{1}{n}\sum_{i=1}^{n} l(y_i - f(\mathbf{x}_i)) \tag{15}$$

serves as an approximation of the generalization error. Averaging over the absolute error loss function results in the mean absolute error (MAE); for the squared error loss function the mean squared error (MSE) is calculated. Instead of the latter, the root mean squared error (RMSE), which estimates the standard deviation of an unbiased estimator, is commonly reported.

It is important to note that the training error (err_{train}) does not provide an unbiased estimate of the generalization error since it decreases for models that adapt too close to the training data, i.e. overfit (Figure 6).

To build an optimal model and assess its quality accurately following this simple procedure requires large data sets, which in practice generally exceed the data available.[63] However, this procedure is often applied in the context of potential-energy surfaces fitting. Here the ML models are frequently evaluated on a single set of randomly selected structures or extensive molecular dynamics runs.[64,65] One of the standard procedures to estimate the generalization error on limited data sets is *k*-fold cross-validation, which is depicted in Figure 7.

In *k*-fold cross-validation the data set is randomly split into *k* equally sized subsets (splits). Each of these splits serves as a test set once, while the remaining splits are used for training. In this way, *k* models are built, and one prediction is generated for each available data point. The errors of the *k* different models are averaged into one cross-validation error. Note that the cross-validation error is still a random variable, which depends on the initial splitting of the data into *k* splits. To reduce its variance the whole process of cross-validation may be repeated several times.

The number of folds *k* should be chosen with respect to the number of available data points and computational capacities. Small data sets and complex tasks require larger values of *k* to

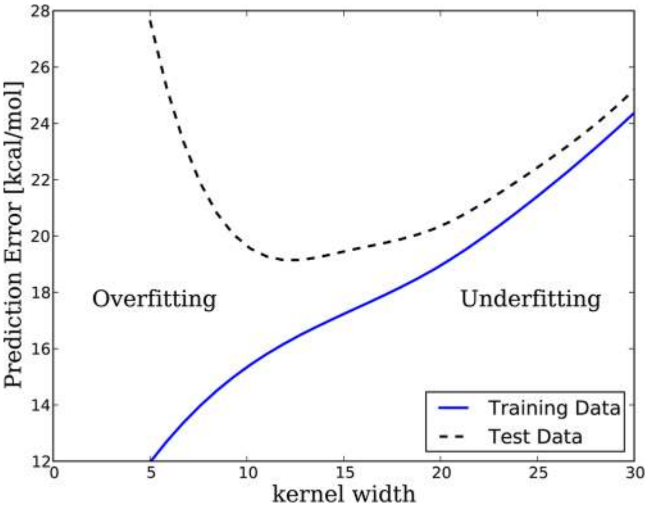
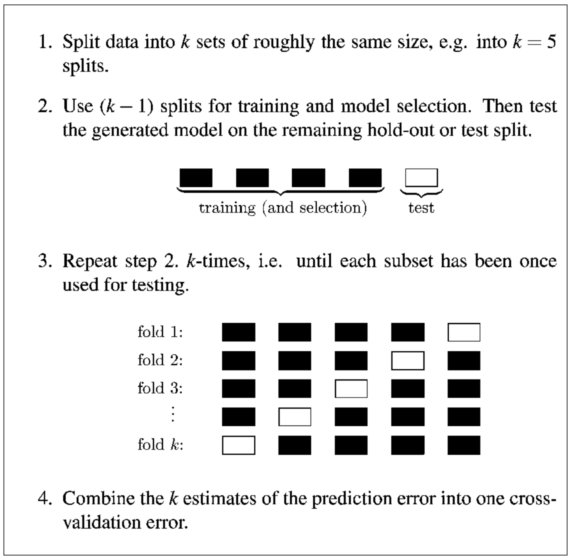


Figure 6. Trends of training and test error for rising model complexity illustrated for kernel ridge regression with Gaussian kernel, eigenspectrum representation and λ set to 0.01. A smaller kernel width results in a more flexible or complex model. This explains why the model overfits (i.e., the training error decreases while the test error increases) for kernel widths chosen too small.



1. Split data into k sets of roughly the same size, e.g. into $k = 5$ splits.
2. Use $(k - 1)$ splits for training and model selection. Then test the generated model on the remaining hold-out or test split.

3. Repeat step 2. k-times, i.e. until each subset has been once used for testing.

4. Combine the k estimates of the prediction error into one cross-validation error.

Figure 7. Model assessment using k-fold cross-validation. This cross-validation scheme is frequently used to estimate how accurately predictive models will perform in practice. Multiple rounds of cross-validation may be performed using different partitions in order to reduce variability of the calculated validation error.

ensure sufficiently large training sets. For $k = N$ we attain the maximal possible number of folds. This cross-validation scheme is called *leave-one-out* (LOO) cross-validation and has high computational cost, since N models must be constructed. For a few algorithms the LOO cross-validation error may be derived analytically. In general a k value of 5 or 10 is recommended by Breiman and Spector.[66]

4.1.2. Model Selection. To construct a learning machine, we not only need to optimize the loss function with respect to the model parameters (as done by learning algorithms), but the hyperparameters need to be chosen with care in order to regularize the optimization problem appropriately. A carefully designed implementation of this crucial set of procedures is what makes machines learn successfully. One of the standard procedures to select hyperparameters is again k-fold cross-validation: First a number of candidate models, i.e. models with different hyperparameter settings, are specified. Then the data set assigned as the "train and model selection" set is split into k equally sized subsets. Each candidate model is trained on $(k-1)$ subsets and evaluated on the remaining validation subset. As before, the training and evaluation is repeated by cycling around the validation subset (see Figure 7 Step 3). Finally the candidate model that performed best in the majority of the k repetitions is selected and trained on all k subsets. It is important that the model selection is done separately from the quality assessment. Using cross-validation for both tasks results in a nested cross-validation scheme (Figure 8).

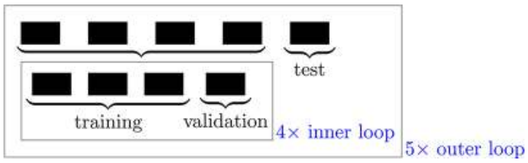


Figure 8. The nested cross-validation procedure that was implemented to select the hyperparameters of the kernel-based models and to estimate the generalization error.

4.2. Evaluation of Models and Data Representations on GDB Molecular Data Set.

4.2.1. Model Validation. In this study, model validation is done by stratified 5-fold cross-validation. For the stratification, we follow the approach of Rupp et al.[2] The 7165 molecules are stratified into 1433 buckets of five molecules each, by level of atomization energy. Then, folds are produced by randomly picking one molecule from each bucket. The stratified sampling procedure ensures that each fold contains a representative set of atomization energies and, therefore, reduces the variance of the estimator of the test error. In addition to this cross-validation analysis, saturation curves are calculated to illustrate how the prediction accuracy changes with an increasing number of training samples. Training sets from 500 to 7000 data points were sampled identically for the different ML methods.

4.2.2. Parameter Selection and Implementation Details. Choice of Parameters for Kernel Methods. The kernel algorithms were trained using a Gaussian and a Laplacian kernel. No further scaling or normalization of the data was done to preserve the meaning of the data in chemical compound space. To determine the hyperparameters for each method we used a grid search with an inner loop of cross-validation. More specifically, we applied a nested cross-validation or double cross-validation[67,68] scheme and performed a 4-fold cross-validation to determine the optimal hyperparameters on each training set of the 5-fold cross-validation (see Figure 8). The mean and variance of the resulting five sets of hyperparameters are listed for each model in the Supporting Information.

Learning on Random Coulomb Matrices Using Kernel Methods. Kernel-based methods such as kernel ridge regression are notoriously difficult to scale to larger data sets. This makes the learning using several random Coulomb matrices per molecule as proposed in Section 2.2 difficult. Large-scale kernel methods have been devised;[69] however, they are either application-specific or induce some performance vs computation trade-off[70] that may outweigh the benefits of using more samples. We consider two different approaches:

(a) *maximal kernel matrix:* this first approach consists of generating as many data points from the Coulomb matrix distribution as the kernel learning algorithm can afford. Due to the scaling issues of most kernel algorithms, it is in practice difficult to handle more than a few tens of thousands of data points. In the case of a data set of 5000 molecules, this equals using no more than 10 permutations per molecule.

(b) *average over kernels:* an alternative solution to the scaling problem of the previous solution is to redefine the kernel operator to account for the multiple permutations of the Coulomb matrix. The kernel between two molecules i and j becomes

$$k(x_i, x_j) = \frac{1}{2}\sum_{l=1}^{L} \kappa(x_i, P(x_j)) + \kappa(P_l(x_i), x) \tag{16}$$

The kernel κ is typically a simple Gaussian or Laplacian kernel. $P_l(x)$ is the l^{th} permutation of the Coulomb matrix x. This way, the kernel matrix remains of size $N_{\text{molecules}} \times N_{\text{molecules}}$ and remains scalable for all employed kernel-based methods.

Choice of Parameters for the Neural Network. Given the large number of parameters in a neural network, most of them are usually selected heuristically: We first expand the raw Coulomb matrix as $\mathbf{x} = [..., \tanh((C-\theta)/\theta), \tanh(C/\theta), \tanh((C+\theta)/\theta),...]$, where the hyperparameter θ determines the granularity of the expansion. We choose $\theta = 1$ as it offers a good trade-off between representational power and computational efficiency. The resulting expanded input $\mathbf{x}$ has between 1500 and 2000 dimensions. We choose a neural network with two hidden layers of 400 and 100 hidden units, respectively. Neural network weights are initialized as $w \sim \mathcal{N}\left(0, \frac{1}{\sqrt{a_{\text{in}}}}\right)$ where a_{in} is the number of incoming connections to a neuron.[57] We use stochastic gradient descent with mini batches of size 25 and maintain a moving average of neural network parameters throughout training. When training the neural network on random Coulomb matrices, we compute predictions for out-of-sample molecules based on 10 realizations of the random Coulomb matrix, and taking the average of the associated network outputs.

For selecting the optimal number of iterations of the gradient descent, we use a simple hold-out validation procedure where 90% of the training data is used for training the neural network and 10% for validation. The number of iterations is selected using early stopping (where the gradient descent stops once the error on the validation set has reached its minimum).

4.3. Pitfalls, Causes of Overfitting, and Practical Hints.

In this section we review common mistakes that may arise when implementing ML methods. Not all of these pitfalls apply to each situation. We explain under which circumstances each pitfall becomes relevant, and, where appropriate, we underline the pitfalls with examples related to our task of atomization-energy prediction.

4.3.1. Pitfalls Concerning Data Distribution and Constitution. Aggregation of Data Sets. The makeup of the data set is a critical issue in machine learning. Often data are assembled from different sources, e.g. different research groups, levels of approximation, or experiments. Though all of them describe the same data attributes, differences in accuracy and data density may introduce an unfavorable bias. Let us consider two data sets − one very small and highly accurate and a larger one of high diversity but lower accuracy. If they are merged naively, the accuracy of the small set is easily lost, and the resulting model predicts with an overall low accuracy. With the knowledge of different data sources, a different level of accuracy can be assigned to (or learned for) each subset of data, and the resulting model will focus on the highly accurate data, wherever it is available.[71,72]

Regions of Low Data Density. Even if the data are taken from a single source, an explorative analysis of the data distribution is recommended. This can be done, for example, by plotting the data or some projections of the data, e.g. using principal component analysis (PCA).[73,74] The chemical space of druglike molecules contained in the GDB-13 is not equally populated, and there are regions of low data density. If the data are partitioned randomly for cross-validation, the splits might vary highly regarding the concentration of data points from low density regions. This unbalanced partitioning leads to a high variance of the test error, which is a good indicator of a poor data distribution. The phenomenon is diminished by stratified sampling of the cross-validation splits, i.e. by distributing the data to the different cross-validation splits according to their labels in such a way that data from low density regions are present in roughly equal proportions in each split.

As an example, we consider kernel ridge regression on our data with a Gaussian kernel and the eigenspectrum representation. With stratified cross-validation, we achieve a mean absolute error of 10.04 ± 0.25 kcal/mol. If the cross-validation splits are generated without stratification we achieve, for example, the following results: 10.14 ± 0.84 kcal/mol, 11.08 ± 3.44 kcal/mol, 13.06 ± 7.17 kcal/mol, or, worst, 64.94 ± 110.95 kcal/mol.

Clusters. A different sampling strategy has to be applied to clustered data, if the prediction task requires the interpolation between clusters. Consider for example a data set where several conformational isomers are given for each molecule and the main interest lies in the prediction of new molecules (rather than additional conformers). A model trained in a standard cross-validation yields an overoptimistic generalization error, i.e. the model performs poorly on new molecules compared to the cross-validation results. A cluster analysis (or physical knowledge) may help to detect such clusters and a clustered cross-validation, where the compounds are distributed clusterwise over splits, ensures an unbiased model selection in this situation.[75]

Scaling of Inputs. Most machine learning methods are not invariant under scaling of the input. Typically each input dimension (feature) is standardized to have zero mean and unit variance. The standardization ensures that all inputs are penalized equally strong within regularization, which typically produces better results. Moreover, the standardization generally improves the condition of the optimization problem and alleviates the selection of good model hyperparameters (e.g., kernel width, learning rate, initial solution). However, it may sometimes distort the internal structure and relations between input dimensions.

The general advice is to run experiments with the standardized inputs if no prior knowledge is available. Otherwise, it is worth checking whether an adjusted scaling (or no scaling) that better reflects the importance and relations of the features improves the results; e.g. in this study the input dimensions are related and scaling proved to be disadvantageous for all learning methods.

4.3.2. Pitfalls Concerning Modeling. Underfitting. When implementing a machine learning method, the first problem that may arise is that the learning algorithm performs poorly, is converging very slowly, or does not converge at all. This could be due to an unsolvable problem (the molecules are not appropriately represented, the data set is too small) or to an inadequate implementation of the modeling procedure. The following checklist may be useful to avoid the latter cause:

• Try normalizing the data to have mean 0 along each dimension and variance 1. Also check that the inputs are reasonably distributed by plotting a two-dimensional PCA of data (see previous section on scaling of inputs).

• Make sure that the initialization of all parameters is reasonable in the case of gradient-based learning algorithms. For a neural network, initial parameters at each layer are typically drawn iid from a Gaussian distribution of scale $1/(a_{in})^{1/2}$ where a_{in} is the number of incoming nodes that are connected to each unit at the output of the layer. [For more detailed practicalities concerning neural networks see LeCun et al.[57] and Montavon et al.[58]] This is the heuristic we use in our atomization-energy prediction model. Unreasonable initial parameters, like zero-valued weights (no learning signal at all), very large weights (sigmoid saturated), or correlated weights can cause the algorithms to either diverge or converge slowly.

• Try different learning rates for neural networks and other gradient-based learning algorithms. The learning rate is a very important parameter. A too small learning rate will lead to slow convergence. On the other hand, a large learning rate may cause the learning algorithm to diverge. A rule of thumb is to try different learning rates logarithmically spaced between 10^{-2} and 10^{-6}.

• Check regularization strength. Regularization may improve the condition of the optimization problem, especially in the case of correlated inputs. In the context of quantum chemistry where there are limited accurate theoretical values but no noise on the calculated labels **y**, a small value of regularization must be applied in order to reflect this low-noise assumption. Slightly too strong regularization limits the flexibility of the function and results in high prediction errors. This problem is illustrated for a kernel ridge regression model on our data set in Figure 9. The contour lines describe the test error in kcal/mol for different kernel widths σ and regularization strengths λ. For a low regularization of 10^{-5} the test error reaches values around 9.5 kcal/mol. It quickly increases to values above 15 kcal/mol if the regularization parameter is increased to 0.001 atomic units.

•Try different kernels for kernel methods (e.g., a linear kernel, Gaussian kernel, Laplacian kernel). If the kernel is linear but the problem is nonlinear, then the learning algorithm will simply not work. This problem can be detected in a support vector machine by counting the number of obtained support vectors. A very small number of support vectors (ranging from one to ten) is suspicious. Considering the relevant dimensionality of the kernel (i.e., the number of kernel principal components that correlate with the task)[49] can also be helpful in order to select the most appropriate kernel.

• Check the grid for grid-based hyperparameter selection. It is important to make sure that the grid spans a sufficiently large number of hyperparameter values, while, at the same time, being sufficiently fine-grained for containing the best hyperparameters. Starting from a large and coarse grid iteratively choose smaller and finer-grained grids, while always making sure that the optimal hyperparameter values lie within the grid and not on the edge of it. In the presence of multiple hyperparameters, for example, the kernel width σ and the regularization parameter λ in kernel ridge regression, care must be taken. The two parameters are not independent of each other and need to be optimized simultaneously. Figure 9 illustrates this dependency for a kernel ridge regression model with Gaussian kernel and eigenspectrum representation. The test error indicated by the contour lines is very sensitive to changes of the regularization parameter. The optimal kernel width lies above a value of 25, if it is optimized for a fixed regularization value of 10^{-8}. For a higher regularization value, e.g. 10^{-5}, the optimal kernel width is much smaller (around 12).

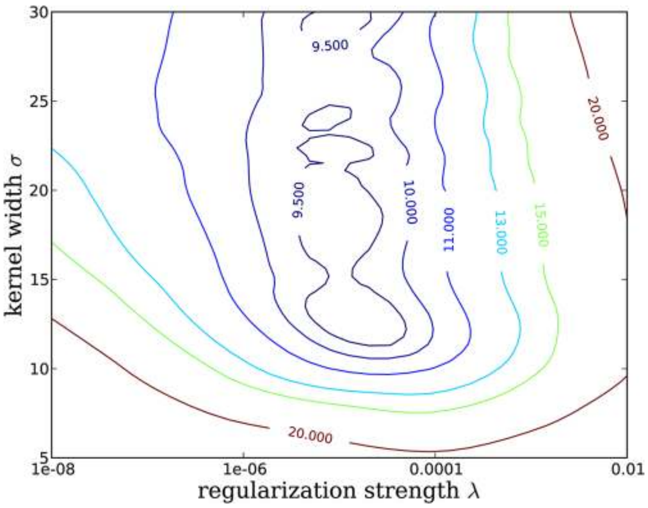


Figure 9. Illustrating the effect of hyperparameter selection for kernel ridge regression with Gaussian kernel and the eigenspectrum representation. The contour lines describe the test error in kcal/mol for different kernel widths σ and regularization strengths λ in atomic units. Note that λ and σ live on different scales. The plot illustrates that the two hyperparameters are not independent of each other and need to be optimized together.

Overfitting. The reduction of the prediction performance caused by too close adaptation of the model to the training data is one of the most common problems in machine learning. In particular, models with many degrees of freedom, such as neural networks, tend to overfit. Three main causes of overfitting − typical pitfalls − are explained below:

• *Model selection on validation data:* Suppose a model with various sets of parameters is trained on some data and subsequently evaluated on a separate test data set. Then the model with the best performing sets of parameters is reported together with the error on the test data. Unfortunately, this model has been chosen (among all possible models) to minimize the error on the test set. Thus the error on this test set is not a fair estimate of the performance to be expected from the model on future data. To generate realistic generalization errors, it is absolutely necessary to do model selection as part of the training procedure−separately from model validation. Though this might sound trivial, there are a large number of studies violating this rule by applying feature selection, parameter selection, or scaling on the basis of the *whole* data set. To reiterate, for the selection of features or standardization of features only the training data from the current fold may be used, *not* the whole set. Practically hyperparameter selection together with unbiased performance estimation can be implemented via the *nested cross-validation* scheme sketched in Figure 8. Note that this model selection scheme can be easily parallelized to reduce computation time.[76]

Table 3. Prediction Errors (In Terms of Mean Absolute Error and Root Mean Squared Error ± Standard Deviation) for Several Algorithms and Representations[a]

	algorithm	molecule representation	MAE [kcal/mol]	RMSE [kcal/mol]
basic methods	mean predictor	none	179.02 ± 0.08	223.92 ± 0.32
	k-nearest neighbors	eigenspectrum	70.72 ± 2.12	92.49 ± 2.70
		sorted Coulomb	71.54 ± 0.97	95.97 ± 1.45
	linear regression	eigenspectrum	29.17 ± 0.35	38.01 ± 1.11
		sorted Coulomb	20.72 ± 0.32	27.22 ± 0.84
methods with Gaussian kernel	mixed effects	eigenspectrum	10.50 ± 0.48	20.38 ± 9.29
		sorted Coulomb	8.5 ± 0.45	12.16 ± 0.95
	kernel support vector regression	eigenspectrum	10.78 ± 0.58	19.47 ± 9.46
		sorted Coulomb	8.06 ± 0.38	12.59 ± 2.17
	kernel ridge regression	eigenspectrum	10.04 ± 0.25	17.02 ± 2.51
		sorted Coulomb	8.57 ± 0.40	12.26 ± 0.78
		random Coulomb (2)	8.46 ± 0.21	11.99 ± 0.73
		random Coulomb (5)	7.10 ± 0.22	10.43 ± 0.83
		random Coulomb (8)	6.76 ± 0.21	10.09 ± 0.76
		average random Coulomb (250)	7.79 ± 0.42	11.40 ± 1.11
methods with Laplacian kernel	mixed effects	eigenspectrum	9.79 ± 0.37	13.18 ± 0.79
		sorted Coulomb	4.29 ± 0.12	6.51 ± 0.56
	kernel support vector regression	eigenspectrum	9.46 ± 0.39	13.26 ± 0.85
		sorted Coulomb	3.99 ± 0.16	6.45 ± 0.71
	kernel ridge regression	eigenspectrum	9.96 ± 0.25	13.29 ± 0.59
		sorted Coulomb	4.28 ± 0.11	6.47 ± 0.51
		random Coulomb (2)	4.02 ± 0.07	5.98 ± 0.35
		random Coulomb (5)	3.29 ± 0.08	5.10 ± 0.39
		random Coulomb (8)	**3.07 ± 0.07**	**4.84 ± 0.40**
		average random Coulomb (250)	4.10 ± 0.14	6.16 ± 0.65
multilayer neural network	backpropagation	eigenspectrum	14.08 ± 0.29	20.29 ± 0.73
		sorted Coulomb	11.82 ± 0.45	16.01 ± 0.81
		random Coulomb (1000)	3.51 ± 0.13	5.96 ± 0.48
previous	PM6	atoms and coordinates	4.9	6.3
	bond counting	covalent bonds	10.0	13.0

[a] The ML algorithms are grouped into four categories: basic machine learning methods which serve as baselines, kernel methods using Gaussian kernel, kernel methods using Laplacian kernel, and neural networks. For comparison results for bond counting and PM6 (adjusted to this data set) as reported by Moussa[35] are given. (Some of the ML results are also included in a preliminary conference contribution.[37])

- *Hyperparameters:* Like with underfitting, inappropriate selection of hyperparameters, especially the regularization strength, may cause overfitting. As an example Figure 6 illustrates how an unfavorable selection of the kernel width σ can cause overfitting of a kernel ridge regression model on our data.
- *Neglect of baselines:* Overfitting is also defined as the violation of Occam's Razor by using a more complicated model than necessary.[77] If a complex model is directly applied without considering simple models, then the solution found may be more complex than the underlying problem and more intuitive relations between input and output data will remain undetected. Thus, it is desirable to additionally report results on simple baseline models like, the mean predictor, linear regression, or KNN in order to determine the need for complex models.

5. RESULTS AND DISCUSSION

Here, we apply the techniques described in the first part of the paper (learning algorithms, data representations, and methodology) to the problem of predicting atomization energies from raw molecular geometries. We run extensive simulations on a cluster of CPUs and compare the performance in terms of cross-validation error of the learning algorithms shown in Section 3 and data representations described in Section 2. We also discuss computational aspects such as training time, prediction speed, and scalability of these algorithms with the number of samples.

Cross-Validation Study. The cross-validation results for each learning algorithm and representation are listed in Table 3. The algorithms are grouped into five categories: basic machine learning methods which serve as baselines, kernel methods using a Gaussian kernel, kernel methods using a Laplacian kernel, neural networks, and two physically motivated approaches reported by Moussa.[35]

We first observe that ML baseline models in the first category are clearly off-the-mark compared to the other more sophisticated ML models such as kernel methods and multilayer neural networks. This shows that the problem of predicting first-principles molecular energetics is a complex one. Among the set of baseline methods, linear regression performs significantly better (best MAE 20.72 kcal/mol) than k-nearest neighbors on this data set (best MAE 70.72 kcal/mol). This indicates that there are meaningful linear relations between physical quantities in the system and that it is insufficient to simply look up the most similar molecules (as k-nearest neighbors does). The k-nearest neighbors approach fails to create a smooth mapping to the energies.

Next, the kernel methods with Gaussian kernel are compared to the methods that use the Laplacian kernel (instead of the

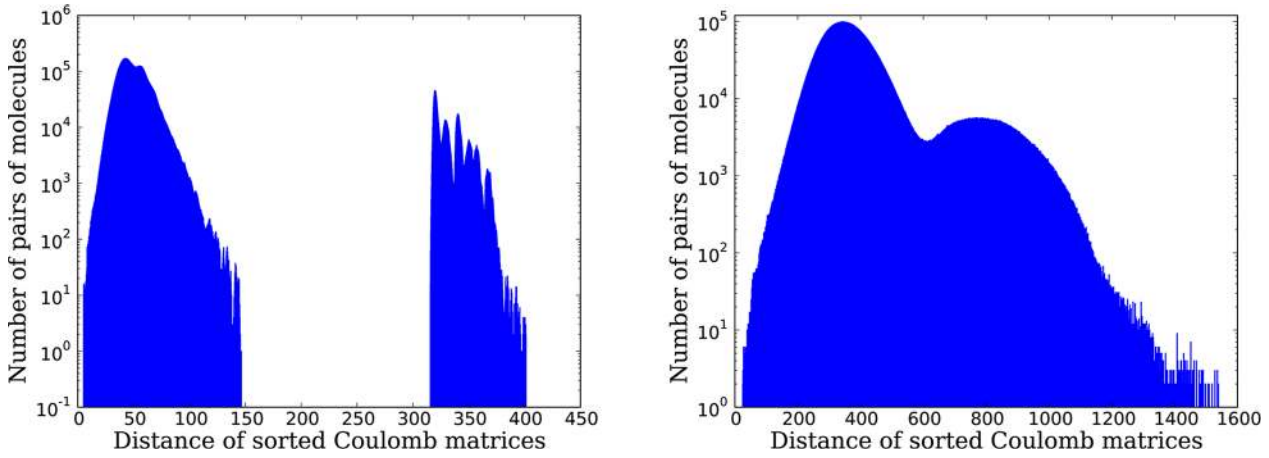


Figure 10. Distribution of pairwise distances within the GDB data set based on the sorted Coulomb representation of molecules (in hartree). The left plot illustrates the distribution of Euclidean distances between molecules. The cluster between 320 and 410 consists of distances between molecules with different number of sulfur atoms. The other cluster includes pairs of molecules having both none or both one sulfur atom. For the Manhattan distance metric (right plot) these clusters are less pronounced, and the distance values show a much larger variety. This may aid the prediction task.

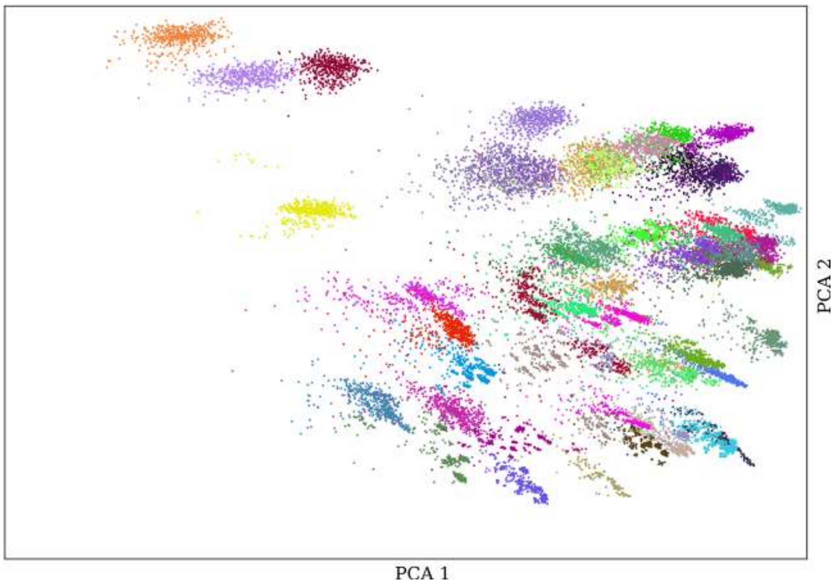


Figure 11. Two-dimensional PCA of the distribution of random Coulomb matrices illustrated for 50 molecules (calculated over the set of all molecules). The Manhattan distance between Coulomb matrices is given as input to PCA. Each cloud of same-colored points represents one molecule, and each point within a cloud designates one random Coulomb matrix associated with the molecule.

Gaussian). The Laplacian kernel seems to be better suited for the prediction problem than the Gaussian kernel, as it improves results for all kernel methods. In order to gain insight into the kernel functions we compare the effect of the Manhattan distance metric and the Euclidean distance metric on the distribution of distances between molecules in Figure 10. (Note that the main difference between Laplacian and Gaussian kernel is the use of the Manhattan distance instead of the Euclidean distance.) The Euclidean distances lie in two narrow separate groups, while the Manhattan distance spreads a much larger range of distances and shows larger variety with respect to different stoichiometries of the molecules. One might speculate that the Laplacian kernel with its Manhattan distance metric better encodes the approximately additive nature of atomization energy than the Gaussian kernel with its Euclidean distance metric. Additionally, the longer tails of the Laplacian kernel and its nondifferentiable peak at the center help to model piecewise-smooth functions (i.e., composed of cliffs and linear plateaus). Such piecewise smoothness may arise as a result of the highly complex nature of the learning problem or possibly due to suboptimal molecular representations.

The results on all ML methods illustrate the impact of the molecule representation on the prediction performance. For kernel ridge regression with Laplacian kernel the trend is the most distinct: the random Coulomb matrix representation performs best (MAE down to 3.07 kcal/mol), followed by the sorted Coulomb matrix (MAE 4.28 kcal/mol) and the eigenspectrum (MAE 9.96 kcal/mol). This ordering correlates with the amount of information provided by the different representations: The randomly sorted Coulomb matrix representation is the richest one as it is both high-dimensional and accounting for multiple indexing of atoms. This is best illustrated in Figure 11 where Coulomb matrix realizations form "clouds" of data points with a particular size, orientation, or

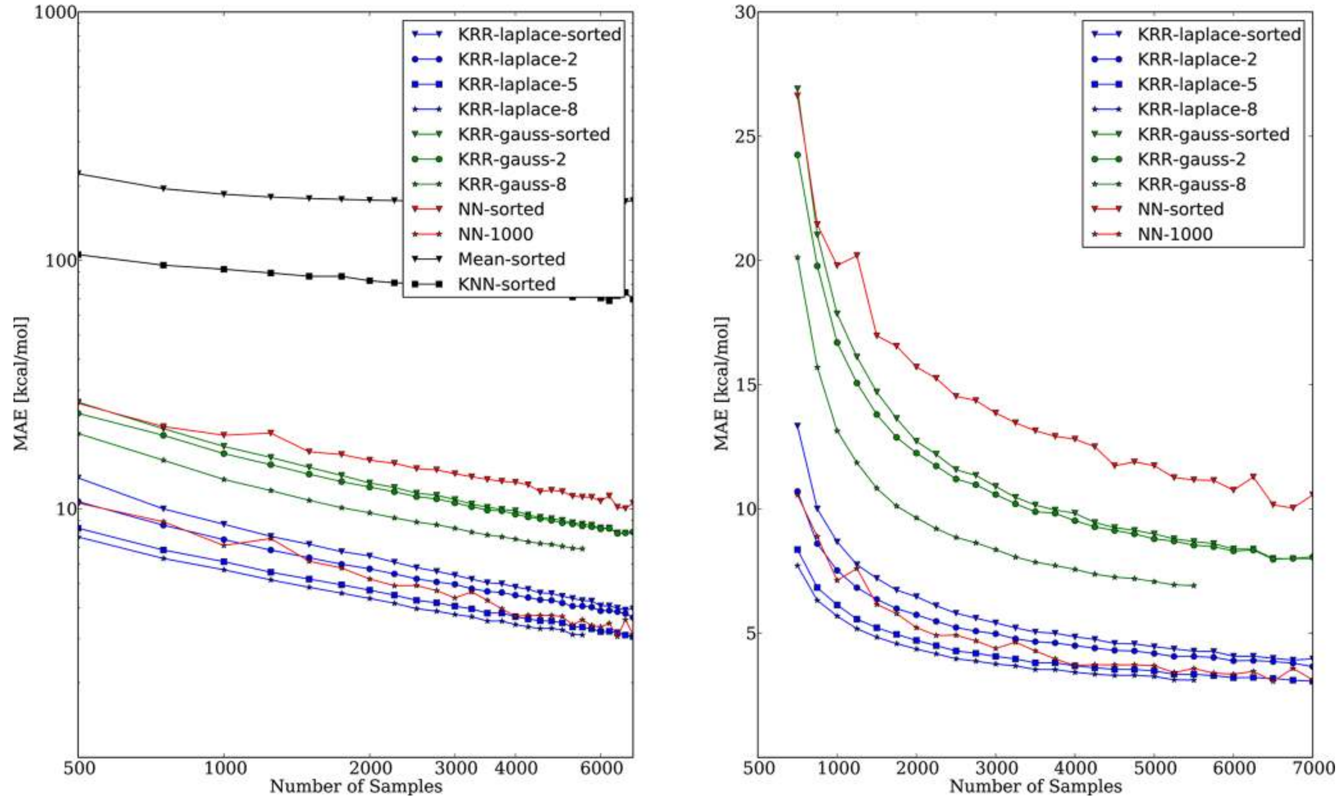


Figure 12. Saturation curves for various ML models and representations (KRR = kernel ridge regression, NN = multilayer neural network, Mean = mean predictor, KNN = *k*-nearest neighbors, laplace = with Laplacian kernel, gauss = with Gaussian kernel, sorted = using sorted Coulomb representation, 2/5/8/1000 = using 2/5/8/1000 random Coulomb matrices to represent one molecule). Left: Log–log plot where the slope of the line reflects the learning rate of the algorithm. Right: nonlogarithmic learning curves for kernel ridge regression and neural network models. Kernel ridge regression with Laplacian kernel and 8 random Coulomb matrices per molecule performs best. However, due to the scaling problem of kernel methods, the representation of 8 random Coulomb matrices per molecule could not be used with kernel ridge regression for data sets larger than 5500 molecules.

shape for each molecule. This cloud-related information is missing in the sorted Coulomb matrix representation, as each molecule is represented by only one data point. The eigenspectrum representation has the lowest performance in our study, in part, because different Coulomb matrices may result in the same eigenspectrum and information is lost in this mapping. This reduction of information has a particularly dramatic impact on the performance of complex models (e.g., kernel ridge regression or neural networks), as they are no longer able to exploit the wealth of information available in the previous representations.

The last group of ML method presents results for multilayer neural networks. Interestingly, these methods do not perform better than kernel methods on the eigenspectrum representation (neural networks MAE 14.08 kcal/mol, while kernel methods are below 10.5 kcal/mol). Moving from the eigenspectrum to the sorted Coulomb matrices neural networks improve significantly (MAE 11.82 kcal/mol), and finally neural networks are almost on par with the kernel methods considering the random Coulomb matrix representation. Note that using many randomly permuted Coulomb matrices (typically more than 1000 per molecule) is crucial for obtaining good performance in the order of MAE 3.5 kcal/mol with a neural network (while for kernel ridge regression models five permutations are sufficient). Figure 11 shows how random permutations of the Coulomb matrix help to fill the input space with data points and, thus, allow for learning complex, yet statistically significant decision boundaries.

The last category includes results on bond counting and the semiempirical method PM6[3] taken from Moussa.[35] Bond energies are refit to the given data set, and PM6 is converted to an electronic energy using a per-atom correction in order to allow for a fair comparison to data driven ML methods. His validation setup slightly deviates from our study (static training set of 5000 compounds instead of 5-fold cross-validation). This may introduce a bias in the MAE. However, this will not affect the qualitative results: simple ML models are clearly inferior to bond counting or PM6 methods that have been adjusted to the data set. However, bond counting uses explicit information about covalent bond orders. This information is not explicitly included in the Coulomb matrix. Given that, it is important to note that the best kernel ridge regression model achieves a MAE of 3.1 kcal/mol compared to 4.9 kcal/mol (PM6) and 10.0 kcal/mol (bond counting).

Saturation Study. The results of the saturation study are summarized in Figure 12. The learning curves (cf. ref 78) illustrate the power-law decline of the prediction error with increasing amount of training data. Each curve can be characterized by the initial error and the "learning rate", i.e. the slope of the learning curve in the log–log plot. The curves for the mean predictor and the KNN model are rather flat, indicating low learning capacity. The steepest learning curve is obtained by the neural network based on 1000 randomly permuted Coulomb matrices per molecule. This may be attributed to the fact that the neural network gradually learns the data representation (in its multiple layers) as more data become available. The neural networks still do not perform better than the kernel ridge regression model with eight randomly permuted Coulomb matrices and a Laplacian kernel. The latter already yields good results on smaller data sets and demonstrates the gain of providing the learning algorithm with a good similarity measure (or kernel). However, for the representation of eight random Coulomb matrices per molecule the learning curve of kernel methods is incomplete. The calculation failed for more than 5500 molecules (i.e., a kernel matrix of $(5500\cdot8)^2 = 1936 \times 10^6$ entries). These scaling problems together with calculation times are discussed in the next section. In summary, the saturation study confirms our previous observations: The baseline methods cannot compete with sophisticated ML algorithms, the Laplacian kernel yields better results than the Gaussian kernel, and among the three different molecular representations the random Coulomb matrices perform best.

The saturation study also illustrates the limits of the presented approaches: The molecular representation and applied algorithm define the maximal reachable accuracy. Even large data sets can barely improve the performance of the best models below 3 kcal/mol. Note that the accuracy of our energy calculation is estimated to be on the order of 3 kcal/mol. Even a perfect model that reflects chemical reality would probably yield errors in this order of magnitude. Only a more accurate or larger data set could clarify which further enhancements of the algorithms and chemical representations are recommendable in order to explore chemical compound space.

Runtime Comparison. Training and prediction times of most ML methods depend on the size *n* of the training set and the maximal size of a molecule or more precisely the dimensionality of the vector representing each molecule *d*. Runtime obviously depends on the machine as well as the implementation used. For this reason, the numbers given here are only meant to provide generic guidance.

In general, baseline methods, e.g. the *k*-nearest neighbors or linear regression, are fast to train. The *k*-nearest neighbors approach does not require any training and takes less than 1 s to predict 1000 compounds using either a sorted Coulomb or eigenspectrum representation. However, the prediction time of *k*-nearest neighbors scales with the number of training samples ($O(n)$), while the prediction time of linear regression is independent of *n*. (It only scales with the dimensionality *d* of the input.)

Kernel ridge regression as well as mixed effects models take only a few seconds to train on 3000 samples using the eigenspectrum or sorted Coulomb representation for one set of hyperparameters. However, training times for mixed effects models as well as kernel ridge regression models scale with $O(n^3)$ and require $O(n^2)$ memory space. Prediction on our data set is fast (less than 1 s for 1000 samples) but scales linearly with *n*. Kernel ridge regression with 3000 training samples on a stochastic Coulomb representation with eight permutations still takes only a matter of minutes.

Support vector regression is implemented using an iterative approach. When a fixed number of iterations is assumed, support vector regression training and testing scale with $O(n\cdot n_{sv} + n_{sv}^3)$ and $O(n_{sv})$, respectively, where n_{sv} is the number of support vectors. For 3000 training samples, support vector regression training times varied from a few seconds to about 1 h depending on the cost parameter and representation. Prediction is fast and takes less than 1 s for 1000 samples. In general, support vector regression trades training speed for prediction speed as the learning algorithm seeks to minimize the number of data points that contribute to the prediction function.

For kernel methods the influence of the input dimensionality *d* on the runtime depends on the used kernel. Generally computation times of kernels grow slowly with rising dimensionality which makes kernel methods a favorable tool for small high-dimensional data sets.

The only algorithm that requires more than a few minutes for training is the multilayer neural network, which took about 15 min to train on 3000 compounds using the sorted Coulomb representation and about 10 h to train on the stochastic Coulomb representation with 1000 permutations. While this is slow compared with the other methods, the multilayer neural network offers the advantage of (almost) linear scaling in $O(kn)$ for training, where *k* is the size of the network, which grows with *n* and *d*. Unlike training, prediction in a neural network is fast (less than 1 s for 1000 samples) and does only depend on the size of the network.

6. CONCLUSIONS

Algorithms that can predict energies of ground-state molecules quickly and accurately are a powerful tool and might contribute substantially to rational compound design in chemical and pharmaceutical industries. However, state-of-the-art *ab initio* calculations that achieve the "chemical accuracy" of 1 kcal/mol are typically subject to prohibitive computational costs. This makes the understanding of complex chemical systems a hard task and the exploration of chemical compound space unfeasible. Semiempirical models on the other side trade accuracy for speed. Here machine learning can provide an interesting contribution barely compromising in this trade-off. Specifically, we show that machine learning allows one to drastically accelerate the computation of quantum-chemical properties, while retaining high prediction accuracy.

All ML algorithms surveyed in this work achieve the first goal of making quantum-chemical computation a matter of milliseconds rather than hours or days when using *ab initio* calculations.

With respect to prediction accuracy, our results improve over the 10 kcal/mol MAE using kernel ridge regression that was reported recently by Rupp et al.[2] While simple baseline ML methods such as linear regression and *k*-nearest neighbors are not suited for this task, more sophisticated methods such as kernel-based learning methods or neural networks yield prediction errors as low as 3 kcal/mol achieved out-of-sample. This substantial improvement results from a combination of multiple factors such as a choice of model (such as the kernel function) and an appropriate representation of the physical properties and invariance structure of molecules (random Coulomb matrices). These factors interact in a complex manner; our analysis is a first step and further investigation should follow.

The focus of this work was on the prediction of ground-state atomization energies for small molecules. However, all methods also provide derivatives which can be used as forces in atomistic simulations. Whether these forces are (given an appropriate training set) of similar accuracy as the predicted atomization energies is a point of further research.[24] Moreover, alternative representations of molecules should be elaborated for this different task—especially when moving to large systems like proteins.

One practical contribution of this work was to establish a best-practice use of ML methods. Having done this we clearly point to erroneous modeling and model selection (which is unfortunately still quite common in applications of ML methods). We reiterate the fact that ML techniques can be error prone if the modeling does not adhere to a strict methodology, in particular, training and validation procedures. Failing to select hyperparameters appropriately or to keep test data aside will likely lead to overly optimistic assessment of the model's generalization capabilities.

Modeling and model assessment are always limited to the available data regime; thus predictions beyond the space sampled by the training data carry uncertainty. For future studies it will be important to further explore, understand, and quantify these limits as well as to enlarge the investigated compound space. Many ML methods are not designed to extrapolate. However, their interpolation in a nonlinearly transformed space is much different from common linear interpolations in the data space (see Figure 3).

There is no learning algorithm that works optimally on *all* data sets. Generally, we recommend a bottom-up approach, starting from simple (linear) baseline algorithms and then gradually increasing the complexity of the ML method for further improving model accuracy.

Concluding, ML methods implement a powerful, fast, and unbiased approach to the task of energy prediction in the sense that they neither build on specific physical knowledge nor are they limited by physical assumptions or approximations. Future studies will focus on methods to decode the trained nonlinear ML models in order to obtain a deeper physical understanding and new insights into complex chemical systems.

ASSOCIATED CONTENT

Supporting Information

The data set and the used splits plus a Python package including the scripts to run one of the used algorithms can be found at http://quantum-machine.org/. A table summarizing the parameters selected for each method in cross-validation. This material is available free of charge via the Internet at http://pubs.acs.org.

AUTHOR INFORMATION

Corresponding Author

*E-mail: hansen@fhi-berlin.mpg.de (K.H.), klaus-robert.mueller@tu-berlin.de (K.R.M.).

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

This work is supported by the European Research Council (ERC Starting Grant VDW-CMAT), by the World Class University Program through the National Research Foundation of Korea funded by the Ministry of Education, Science, and Technology, under Grant R31-10008, the Einstein Foundation, and by the U.S. Department of Energy, Basic Energy Sciences, Office of Science, under contract # DE-AC02-06CH11357. The work of Franziska Biegler is funded, in part, by the Natural Sciences and Engineering Research Council of Canada. The authors also acknowledge partial support by DFG (MU 987/17-1). Matthias Rupp acknowledges support by FP7 programme of the European Community (Marie Curie IEF 273039).

REFERENCES

(1) Montavon, G.; Rupp, M.; Gobre, V.; Vazquez-Mayagoitia, A.; Hansen, K.; Tkatchenko, A.; Müller, K.-R.; von Lilienfeld, O. A. *New J. Phys.* **2013**, accepted.

(2) Rupp, M.; Tkatchenko, A.; Müller, K.-R.; von Lilienfeld, O. A. *Phys. Rev. Lett.* **2012**, *108*, 058301.

(3) Stewart, J. J. P. *J. Mol. Model.* **2007**, *13*, 1173−1213.

(4) Benson, S. W. *Bond energies*; 1965. http://www.wiredchemist.com (accessed June 8, 2011).

(5) Behler, J. *Phys. Chem. Chem. Phys.* **2011**, *13*, 17930−17955.

(6) Breiman, L. *Stat. Sci.* **2001**, *16*, 199−231 Mathematical Reviews number (MathSciNet): MR1874152.

(7) Montavon, G.; Braun, M. L.; Krueger, T.; Müller, K.-R. *Signal Processing Magazine, IEEE* **2013**, *30*, 62−74.

(8) Selassie, C. The History of Quantitative Structure Activity Relationships. In *Burger's Medicinal Chemistry and Drug Discovery*, 6th ed.; Abraham, D., Ed.; John Wiley and Sons Publishers: New York, NY, 2003; Vol. *1*, Chapter 1, pp 1−48.

(9) Müller, K.-R.; Rätsch, G.; Mika, S.; Sonnenburg, S.; Grimm, M.; Heinrich, N. *J. Chem. Inf. Model.* **2005**, *45*, 249−253.

(10) Le Bailly de Tilleghem, C.; Govaerts, B. *A review of quantitative structure-activity relationship (QSAR) models*; Technical Report 07027; Universite catholique de Louvain, 2007.

(11) Sumpter, B. G.; Noid, D. W. *Chem. Phys. Lett.* **1992**, *192*, 455−462.

(12) Blank, T. B.; Brown, S. D.; Calhoun, A. W.; Doren, D. J. *J. Chem. Phys.* **1995**, *103*, 4129.

(13) Lorenz, S.; Groß, A.; Scheffler, M. *Chem. Phys. Lett.* **2004**, *395*, 210−215.

(14) Lorenz, S.; Scheffler, M.; Gross, A. *Phys. Rev. B* **2006**, *73*, 115431.

(15) Manzhos, S.; Carrington, T. *J. Chem. Phys.* **2006**, *125*, 084109.

(16) Hu, L.; Wang, X.; Wong, L.; Chen, G. *J. Chem. Phys.* **2003**, *119*, 11501.

(17) Zheng, X.; Hu, L.; Wang, X.; Chen, G. *Chem. Phys. Lett.* **2004**, *390*, 186−192.

(18) Behler, J.; Parrinello, M. *Phys. Rev. Lett.* **2007**, *98*, 146401.

(19) Handley, C. M.; Popelier, P. L. A. *J. Chem. Theory Comput.* **2009**, *5*, 1474−1489.

(20) Behler, J.; Martonák, R.; Donadio, D.; Parrinello, M. *Phys. Rev. Lett.* **2008**, *100*, 185501.

(21) Mills, M. J.; Popelier, P. L. *Comput. Theor. Chem.* **2011**, *975*, 42−51.

(22) Bartók, A. P.; Payne, M. C.; Kondor, R.; Csányi, G. *Phys. Rev. Lett.* **2010**, *104*, 136403.

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

(24) Snyder, J. C.; Rupp, M.; Hansen, K.; Müller, K.-R.; Burke, K. *Phys. Rev. Lett.* **2012**, *108*, 253002.

(25) Hautier, G.; Fisher, C. C.; Jain, A.; Mueller, T.; Ceder, G. *Chem. Mater.* **2010**, *22*, 3762−3767.

(26) Balabin, R. M.; Lomakina, E. I. *J. Chem. Phys.* **2009**, *131*, 074104.

(27) Balabin, R. M.; Lomakina, E. I. *Phys. Chem. Chem. Phys.* **2011**, *13*, 11710−11718.

(28) Blum, L. C.; Reymond, J.-L. *J. Am. Chem. Soc.* **2009**, *131*, 8732−8733.

(29) Weininger, D. *J. Chem. Inf. Model.* **1988**, *28*, 31−36.

(30) Guha, R.; Howard, M. T.; Hutchison, G. R.; Murray-Rust, P.; Rzepa, H.; Steinbeck, C.; Wegner, J.; Willighagen, E. L. *J. Chem. Inf. Model.* **2006**, *46*, 991−998.

(31) Rappé, A. K.; Casewit, C. J.; Colwell, K. S.; Goddard, W. A.; Skiff, W. M. *J. Am. Chem. Soc.* **1992**, *114*, 10024−10035.

(32) Perdew, J. P.; Burke, K.; Ernzerhof, M. *Phys. Rev. Lett.* **1996**, *77*, 3865.

(33) Ernzerhof, M.; Scuseria, G. E. *J. Chem. Phys.* **1999**, *110*, 5029−5036.

(34) Blum, V.; Gehrke, R.; Hanke, F.; Havu, P.; Havu, V.; Ren, X.; Reuter, K.; Scheffler, M. *Comput. Phys. Commun.* **2009**, *180*, 2175.

(35) Moussa, J. E. *Phys. Rev. Lett.* **2012**, *109*, 059801.

(36) Rupp, M.; Tkatchenko, A.; Müller, K.-R.; von Lilienfeld, O. A. *Phys. Rev. Lett.* **2012**, *109*, 059802.

(37) Montavon, G.; Hansen, K.; Fazli, S.; Rupp, M.; Biegler, F.; Ziehe, A.; Tkatchenko, A.; von Lilienfeld, O. A.; Müller, K.-R. Learning Invariant Representations of Molecules for Atomization Energy Prediction. *Advances in Neural Information Processing Systems* **2012**, *25*, 449−457.

(38) LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. *Proc. IEEE* **1998**, *86*, 2278−2324.

(39) Ciresan, D. C.; Meier, U.; Gambardella, L. M.; Schmidhuber, J. *Neural Comput.* **2010**, *22*, 3207−3220.

(40) DeCoste, D.; Schölkopf, B. *Mach. Learn.* **2002**, *46*, 161−190.

(41) Bishop, C. M. *Pattern Recognition and Machine Learning*, 1st ed.; Springer: New York, NY, 2011.

(42) Duda, R. O.; Hart, P. E.; Stork, D. G. *Pattern classification*; Wiley: New York, NY, 2001.

(43) Vapnik, V. N. *The nature of statistical learning theory*; Springer-Verlag: New York, NY, 1995.

(44) Hastie, T.; Tibshirani, R.; Friedman, J. *The Elements of Statistical Learning*; Springer Series in Statistics; Springer: New York, NY, 2001.

(45) Müller, K.-R.; Mika, S.; Rätsch, G.; Tsuda, K.; Schölkopf, B. *IEEE Neural Networks* **2001**, *12*, 181−201.

(46) Schölkopf, B.; Smola, A. J. *Learning with Kernels*; MIT Press: Cambridge, MA, 2002.

(47) Rasmussen, C.; Williams, C. *Gaussian Processes for Machine Learning*; MIT Press: Cambridge, MA, 2006.

(48) Mercer, J. *Philos. Trans. R. Soc. London, Ser. A* **1909**, *209*, 415−446.

(49) Braun, M. L.; Buhmann, J.; Müller, K.-R. *J. Mach. Learn. Res.* **2008**, *9*, 1875−1908.

(50) Vapnik, V. *Statistical Learning Theory*; Wiley: New York, NY, 1998; pp 443−492.

(51) Cristianini, N.; Shawe-Taylor, J. *An Introduction to Support Vector Machines*; Cambridge University Press: Cambridge, UK, 2000; pp 112−120.

(52) Platt, J. C. In *Advances in kernel methods*; Schölkopf, B., Burges, C. J. C., Smola, A. J., Eds.; MIT Press: Cambridge, MA, 1998; Chapter Fast training of support vector machines using sequential minimal optimization, pp 185−208.

(53) Pinheiro, J. C.; Bates, D. M. *Mixed-Effects Models in S and S-Plus*; Springer: New York, NY, 2000; pp vii−viii.

(54) Tibshirani, R. *J. R. Stat. Soc. Ser. B Stat. Methodol.* **1996**, 267−288.

(55) Schelldorfer, J.; Bühlmann, P.; van de Geer, S. *Scand. J. Stat.* **2011**, *38*, 197−214.

(56) Fazli, S.; Danóczy, M.; Schelldorfer, J.; Müller, K.-R. *NeuroImage* **2011**, *56*, 2100−2108.

(57) LeCun, Y.; Bottou, L.; Orr, G. B.; Müller, K.-R. Efficient BackProp. *Neural Networks−Tricks of the trade LNCS 1524, Berlin Heidelberg* **1998**, 5−50.

(58) *Neural Networks: Tricks of the Trade, Reloaded*, 2nd ed.; Montavon, G., Orr, G. B., Müller, K.-R., Eds.; Springer: Berlin, Heidelberg, 2012; Vol. *7700*.

(59) Bottou, L. Stochastic Gradient Learning in Neural Networks. *Proceedings of Neuro-Nîmes 91*; Nimes, France, 1991; pp 687−706.

(60) Hinton, G. E.; Osindero, S.; Teh, Y.-W. *Neural Comput.* **2006**, *18*, 1527−1554.

(61) Bengio, Y. *Foundations and Trends in Machine Learning* **2009**, *2*, 1−127.

(62) Montavon, G.; Braun, M. L.; Müller, K.-R. *J. Mach. Learn. Res.* **2011**, *12*, 2563−2581.

(63) Amari, S.; Murata, N.; Müller, K.-R.; Finke, M.; Yang, H. *IEEE Trans. Neural Networks* **1997**, *8*, 985−996.

(64) Jose, K. V. J.; Artrith, N.; Behler, J. *J. Chem. Phys.* **2012**, *136*, 194111.

(65) Handley, C. M.; Popelier, P. L. A. *J. Phys. Chem. A* **2010**, *114*, 3371−3383.

(66) Breiman, L.; Spector, P. *Int. Stat. Rev.* **1992**, *60*, 291−319.

(67) Cawley, G. C.; Talbot, N. L. C. *J. Mach. Learn. Res.* **2010**, *11*, 2079−2107.

(68) Stone, M. *J. R. Stat. Soc. Ser. B Stat. Methodol.* **1974**, *36*, 111−147.

(69) *Large-Scale Kernel Machines (Neural Information Processing)*; Bottou, L., Chapelle, O., DeCoste, D., Weston, J., Eds.; MIT Press: Cambridge, MA, 2007.

(70) Rahimi, A.; Recht, B. Random Features for Large-Scale Kernel Machines. In *Advances in Neural Information Processing Systems 20*; Platt, J., Koller, D., Singer, Y., Roweis, S., Eds.; MIT Press: Cambridge, MA, 2008; pp 1177−1184.

(71) Sugiyama, M.; Suzuki, T.; Kanamori, T. *Density Ratio Estimation in Machine Learning*; Cambridge University Press: New York, NY, 2012; pp 119−214.

(72) Kersting, K.; Plagemann, C.; Pfaff, P.; Burgard, W. Most likely heteroscedastic Gaussian process regression. *Proceedings of the 24th international conference on Machine learning*; New York, NY, 2007; pp 393−400.

(73) Jolliffe, I. T. *Principal Component Analysis*, 2nd ed.; Springer: New York, NY, 2002.

(74) Ivosev, G.; Burton, L.; Bonner, R. *Anal. Chem.* **2008**, *80*, 4933−4944.

(75) Hansen, K.; Rathke, F.; Schroeter, T.; Rast, G.; Fox, T.; Kriegl, J. M.; Mika, S. *J. Chem. Inf. Model.* **2009**, *49*, 1486−1496.

(76) Lemm, S.; Blankertz, B.; Dickhaus, T.; Müller, K.-R. *NeuroImage* **2011**, *56*, 387−399.

(77) Hawkins, D. *J. Chem. Inf. Comput. Sci.* **2004**, *44*, 1−12.

(78) Müller, K.-R.; Murata, N.; Finke, M.; Schulten, K.; Amari, S. *Neural Comput.* **1996**, *8*, 1085−1106.