École Nationale Supérieure des Mines de Nancy

*

High-accuracy materials modelling by machine learning quantum simulations

by Lucas LHERBIER

Laboratoire de Physique et Chimie Théoriques
Department of Applied Mathematics
3rd year project

November 5, 2020

Contents

List of Figures

List of Tables

Introduction

This report is part of the third year academic program of the Ecole des Mines de Nancy. The project is supervised by Dario ROCCA, associate professor of Physics at the *Université de Lorraine* and the *Centre national de la recherche scientifique* (CNRS), and it is done in collaboration with Fabien PASCALE, research engineer at the *Université de Lorraine*. The project is realized in association with the Laboratoire de Physique et Chimie Théoriques of the Université de Lorraine.

From the virtual assistant SIRI Virtual to self-driving cars, artificial intelligence is everywhere in our society. The field of artificial intelligence begun with Alan Turing: he proposed changing the question from whether a machine was intelligent to "whether or not it is possible for machinery to show intelligent behaviour" [7]. Since then, many mathematicians and researchers allowed the quick expansion of artificial intelligence. It is now defined as the creation of machines, as well as computer hardware and software, aiming at reproducing in part the intelligent behavior of human beings.
Indeed, AI covers various aspects of human behavior: creativity, planning, writing, learning, auditing and natural language processing. But the concept of artificial intelligence is in continuous evolution: new advanced systems are developed once the use of machines with specific clever features becomes widespread. Thus, AI is revolutionizing every sector: medical diagnosis, image recognition in photographs, autonomous drones, self-driving cars, predicting flight delays, creating art, playing games like Chess, search engines with Google, spam filtering, prediction of judicial decisions, targeting online advertisements...The non-exhaustive list allows us to be aware that artificial intelligence surrounds us and all the more every day.

Consequently, Machine learning - which will be defined in the first chapter - has also been applied to materials science. As well, it is used for several different applications: solving quantum equations with interpolations, performing expensive experiments by computer simulations... For instance, with the consideration of sustainable development, researchers try to create new revolutionary solar panels designed from organic molecules. As explained in the *The Harvard Clean Energy Project*, these panels would make it possible to avoid the rare-earth elements that are essential today, and to achieve better yields energy. The principal problem is to find new molecules presenting optimal properties for use on solar panels. Thus, machine learning algorithms can be used to calculate the molecular properties much faster and with a reasonable level of accuracy.

This report will be divided in four parts. First, the context of the project will be presented. Then, the diverse aspects of the problem such as the provided data will be explained in the second chapter. Next the machine learning theory used to solve the problem will be discussed in the third chapter. Finally, the third chapter will involve the practical work and the obtained results.
All the algorithmic implementations and the slides are available here.

Chapter 1

Scientific context and motivation

1.1 Machine learning in materials science

First of all, it is important to explain the differences between artificial intelligence and machine learning. Both terms are used very frequently when the subject deals with Big Data, analytics or more generally with the broader technological change which are sweeping through our world. They can be defined as :

- **Artificial Intelligence** is the broader concept of producing machines that have some of the capacities of the human brain, such as problem solving and learning.
- **Machine Learning** is a current application of AI. It is based on the idea that computer systems are able to use algorithms and statistical model to perform a specific task, by learning for themselves from data, without human explicit instruction. According to the American computer scientist Tom M. Mitchell, *Machine learning is the study of computer algorithms that allow computer programs to automatically improve through experience.* [8]

Nowadays, machine learning has several applications and benefits in materials science. More precisely, they are revolutionizing the way research is realized seeking to discover new molecules or materials with better abilities. Let's see two diverse applications.

1.2 New solar panels

As briefly explained in the introduction, researchers try to create new revolutionary solar panels designed from organic molecules. The Harvard *Clean Energy Project* (CEP) aims to study a theoretical search for the next generation of organic solar cell materials [9]. It uses distributed computing to identify the most likely molecules to discover the next generation of organic photovoltaics and fuel cells. In addition to finding specific candidates, the goal of the project is to understand the structure relations in the domain of organic electronics. Such ideas can pave the way to a systematic design of future high-performance materials.

These panels could allow to improve yield and to save a lot of money with the production facility. The main problem is to find the best molecules with the characterization of 3.5 million [10] of molecular motifs using first-principles quantum chemistry. Then, molecular properties calculations require the solution of the Schrödinger equations which is not known analytically. The development of density functional theory (DFT) [1] allowed to compute the stats energies from the ground state. But the computing time to study all the molecular structures is consequent. It explains why the CEP works with a crowd computing network which supply huge calculation power.

[1] Density functional theory is a computational quantum mechanical modelling method used in materials science to investigate the ground state of many-body systems, such as atoms or molecules. It uses functions of another function, called functionals, which in this case is the spatially dependent electron density, to determine the properties of a many-electron system. Source: wikipédia.

In this way, machine learning algorithms can be used in order to reduce the computational cost: from a molecular set for whom the Schrödinger equations has been solved, the goal is to predict the atomization energies thanks to the DFT.

1.3 Materials for environmental remediation

Slowly, people became aware of the human responsibility in global warming, mostly due to the greenhouse gas emissions. Researchers showed that CO_2 global emissions increased from 2 billion tonnes of carbon dioxide in 1900 to over 36 billion tonnes 115 years later [11]. It entails global warming and all the natural disasters that appear today. Materials science researchers are trying to develop new materials for environmental remediation, such as carbon dioxide capture.

The *in silico* methods are commonly used. These approaches are based on computer simulations which allow to predict new molecules and materials, and then test their properties. Specifically, accurate quantum-mechanical approximations with the inclusion of temperature effects are required to model the properties of technologically relevant materials. This can be achieved through *ab initio* molecular dynamics (AIMD), first introduced in the seminal work of CAR and PARRINELLO - *Unified Approach for Molecular Dynamics and Density-Functional Theory* [12] -. The *ab initio* methods attempt to solve the electronic Schrödinger equation considering only basic information on a given system, such as the positions and types of atoms: this allows to obtain useful information on the properties of the system (including the adsorption energies which are our interests here) to solve the Schrödinger equations. However, the numerical cost to solve the Schrödinger equation is huge and accordingly, AIMD calculations are typically performed using numerically inexpensive approximations. For instance, the bigger is the system, the larger is the computational cost and the lower the precision. Consequently, these widely used approximations do not always reach chemical accuracy and are affected by several known shortcomings [13].

Thanks to the use of machine learning, it is possible to overcome the shortcomings in material simulations by bridging molecular dynamics with more sophisticated correlated approximations [14]. This will be the main focus of this project. Only a small but optimal number of geometric configurations of a material will be used to train machine learning algorithms, in order to predict quantum-mechanical properties of different structures. Consequently, the high number of expensive calculations required in an AIMD simulation will be significantly reduced. They will be replaced by the predictions of the machine learning models.

Chapter 2

The chabazite problem

This chapter presents the problem and the provided data. It explains also how the data sets for the machine learning algorithm are created.

2.1 Explanation of the problem

In materials science, they are two main uses of machine learning:

- replacing experiments. The research of new materials needs to realize expensive experiments and performing them by computer allows significant savings.
- solving quantum equations. One of the principal problems of solving quantum equations is processing time, often too high even with a huge computing power.

My project deals with the second use.

In order to model the properties of technological relevant materials, the main problem is to predict ground state correlation energy. In this work, we will consider the random phase approximation (RPA), which is an approximation method in condensed matter physics and in nuclear physics. For a long time, physicists had been trying to incorporate the effect of microscopic quantum mechanical interactions between electrons in the theory of matter. In this context, the RPA is a method that has recently attracted a lot of interest as a method that could overcome some of the shortcomings of more widespread quantum approximations (e.g. based on density functional theory) : however, the RPA comes at a very high cost.
The project will be focused on a single example: *the finite-temperature RPA enthalpy of adsorption of molecules in zeolites*. Those micro-porous minerals are commonly used as commercial adsorbents and catalysts. Zeolites occur naturally but are also produced industrially on a large scale. One of the more common mineral zeolites are chabazite: chabazite is a solid material whose primitive cell is composed of 42 atoms. It is a periodic crystal, whose constituents are regularly assembled, modeled by an infinite network. Here we will consider the adsorption of CH_4 in chabazite, because CH_4 takes part of model systems.

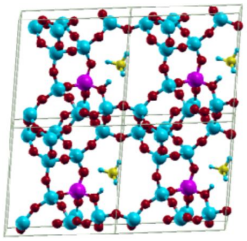

Figure 2.1: CH_4 in protonated chabazite

Using the *ab initio* molecular dynamics (AIMD) to directly perform a brute force RPA simulation would require several tens of millions of CPU hours, around 7000 years :

$$5_{hours} * 64_{cores} * 200000_{steps} = 64 * 10^6_{CPU\ hours}$$

In this work, we will consider the *machine learning thermodynamic perturbation theory* (MLPT), which involves only a small number of expensive RPA calculations and requires only a few tens of thousands of CPU hours. With this approach, it is possible the considerably reduce calculation time to reach 4 computational months:

$$5_{hours} * 64_{cores} * 10 training\ structures = 3.2 * 10^3_{CPU\ hours}$$

2.1.1 Machine learning thermodynamic perturbation theory (MLPT)

This method associates both the *ab initio* molecular dynamics, the perturbation theory and a machine learning model [15].
Actually, a machine learning algorithm is used for the interpolation of diverse physical proprieties such as the energy surfaces or energy bands. This report is focused on the interpolation of the energy surfaces. As the figure 2.2 shows, some computations are done at several points over the potential energy surface (the red points on the figure). With those points, the machine learning algorithm can be trained in order to predict and approximate the energy surface.

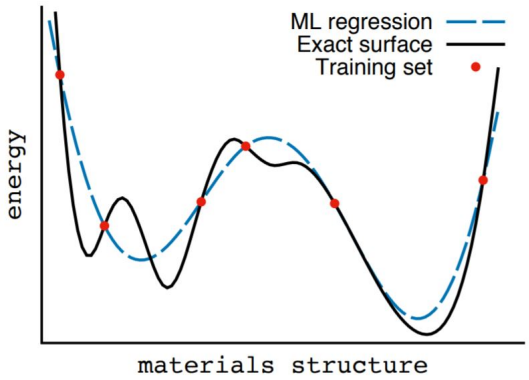

Figure 2.2: Interpolation of an energy surface

2.2 Presentation of the provided data

2.2.1 *Mlpos*

This file is an XYZ file in the ASE[1] format. It contains 19000 lines and each line corresponds to a specific configuration of the primitive cell at different simulation time. The 19000 configurations corresponds to a $PBE + D2$ [2] trajectory. For each configuration of the file, the data are divided in three sections:

- the first one is a line indicating the number of atoms in the molecule, here 42
- the second one presents several characteristics of the material
- the last one shows all the atoms of the primitive cell, their positions to a reference atom and the value of a molecular force in a Cartesian coordinate system (X,Y,Z).

[1] The *Atomic Simulation Environment*, ASE, is a set of tools and Python modules for setting up, manipulating, running, visualizing atomistic simulations.
[2] $PBE + D2$ is a method in optimization of various crystalline systems.

```
42
Lattice="7.94875485 -4.997e-05 4.90143315 -3.97456407 6.88389431 4.90125479 -3.97443877 -6.88377714
4.90124571" Properties=species:S:1:pos:R:3:forces:R:3 pbc="T T T" energy=-311.78921226 stress="0.0182252587392
-0.000259929994355 0.00562047564113 -0.000259929994355 0.0203235315241 -0.0056400132936 0.00562047564113 -
0.0056400132936 0.0212252419793" free_energy=-311.78921226
Al       2.06431498       0.32946941       6.64619220      -1.17472396       0.80257011      -0.22558447
C        0.45354344      -2.90930619       4.56610161      -0.83449173      -2.03091242      -1.94967959
H        0.07848447      -2.05679382       5.01956828      -1.06179303       1.79798802       1.13108555
H        0.05888788      -3.80077473       5.09647066       0.34425125       0.33989819      -0.15657845
H        0.09200188      -2.92158328       3.48958824       0.48064516      -0.02820056       0.96858562
H        1.52802207      -2.91033876       4.58321789       0.95774100      -0.34158574      -0.03286544
H        1.72699016      -1.20484931       4.84686198      -0.35959329       1.79322103      -0.28971321
O        1.88028602      -1.16309027      12.36146257       0.11573832      -0.40377797      -0.48546922
O       -1.03151502       5.09719259       4.83865887       0.26283745       0.61415297      -0.32906421
O       -4.30058622      -3.25193201       7.74011310      -0.13452195       1.69100409      -0.06533371
O       -3.47592729      -0.74026586       7.74991547       0.50170257       0.53640309       0.21984736
```

Figure 2.3: Extract of the *Mlpos* file

2.2.2 *Corrections*

This file is composed of 200 lines and 2 columns. The first column refers to the number that identifies the configuration in the *Mlpos* file and the second column corresponds to the energy of the configuration.
The energy is the target value for our machine learning models. It has been computed by performing a molecular dynamics simulation at 300 K using the $PBE + D2$ functional. For those 200 atomic configurations, the Schrödinger equations has been solved using the RPA to get the energy. The goal is therefore to use this 200 target value to train our machine learning algorithms in order to predict the adsorption RPA energy for the 19000 configurations.

Chapter 3

Machine learning models

This chapter presents the theory of the diverse machine learning algorithms used for the predictions. Three models are considered: the random forest, the kernel ridge regression and the multi-layer perceptron.

3.1 Random Forests

3.1.1 Basic concept

The **ensemble methods** combine multiple individual learning models to produce an aggregate model that is more powerful. This idea is similar to the concept of *wisdom of crowd*: if you want to know the birth date of Alan Turing, you can ask mathematicians, it seems likely that the mean of their answers tends to the exact solution (1912). However, if I ask a unique person randomly chosen, I would have no way of knowing if he was sure or if he randomly answered. Similarly for an ensemble method, each individual model might overfit to a different part of the data. Consequently, by combining different individual models into an ensemble, we can average out their individual mistakes to reduce the risk of overfitting, while maintaining strong prediction performance.
Random forests are an example of the ensemble idea applied to decision trees.

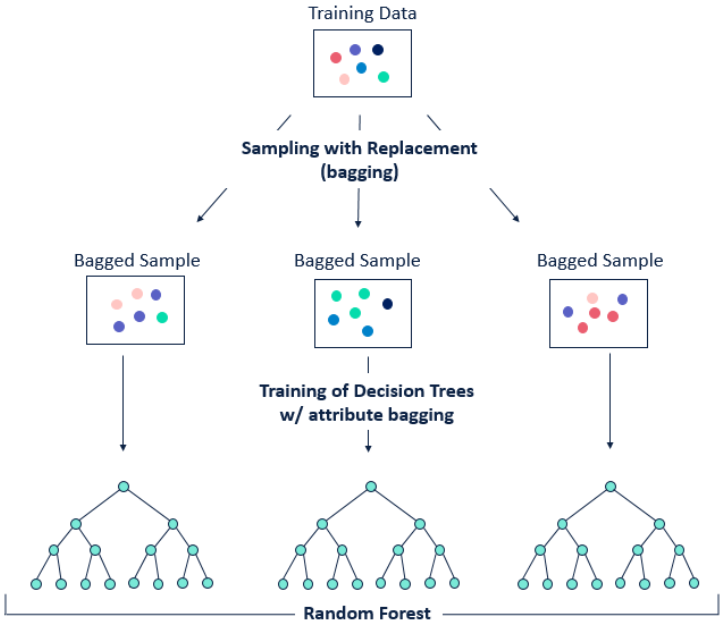

Figure 3.1: Example of random forest. Source: [1]

3.1.2 Detailed model

Random forests create lots of individual decision trees on a training set: each of the individual trees does well at predicting the target values, but should also be constructed to be different in some way from the other trees in the forest.This is accomplished by introducing random variation into the process of building each decision tree:

- **tree bagging** or **bootstrap sample**: random selection of the data used to build each tree. If the training set has n instances, a bootstrap sample of size b is created by picking one of the n data-set rows at random with replacement.This random selection process is repeated b times.
- **feature sampling**: random selection of the features chosen in each split sample. Briefly, for decision trees, the split variable is very important: it is the one who separates the data. The goal is to divide one class from the others, maximizing the purity (i.e. homogeneity) of the groups. With feature sampling, picking the best split for a node is found within a random subset of features instead of all the features.

The bootstrap sample and the random subsets of the features of the split virtually guarantees that all of the decision trees will be different. In our case of regression, the target value will be the mean of the individual tree predictions.

3.1.3 Key parameters

There are key parameters for using random forests.

- **n_estimators:** sets the number of trees to use. The default value for $n\_estimators$ is 10: increasing it for larger data sets improves the performance because ensembles can average over more trees will reduce overfitting. That will entail the rise of the computational cost of training, using more time and more memory
  $\nearrow\ n\_estimators\ \Rightarrow \swarrow$ overfitting & $\nearrow$ computing time and memory
- **max_features:** has a strong effect on performance and influences the diversity in the forest. The default setting is $\log_2(p)$ − is p the total number of the features − for regression.
- **max_depth:** controls the depth of each tree. The default is none, i.e. the nodes in a tree will continue to be split until all leave contain same class or have fewer samples than the minimum sample split parameter value, which is 2 by default.
- **n_jobs:** sets the number of cores to use in parallel during training. A negative one will use all the cores on the system, and if it is more than the number of cores on the system, they will not have any additional effect.

Consequently, if we want to avoid overfitting, it is possible to increase $n\_estimators$ or decrease $max\_features$. The second solution limits the split on the few feature that were selected randomly, entailing that the trees will be very different.

3.1.4 Advantages and drawbacks

On the positive side, random forests are widely used because they give excellent prediction performance on a many problem. Likewise, they do not require careful scaling of the feature data: without extensive parameter tuning, they achieve very good test set performance on the data set. And even though building many different trees requires a corresponding increase in computation, building random forests is easily paralyzed across multiple CPU's.

On the negative side, even if random forests inherit many of the benefits of decision trees, a random forests model can be very difficult for people to interpret. i.e. it is tricky to see the predictive structure of the features or to know why a particular prediction was made.

3.2 Kernel Ridge Regression

Kernel ridge regression [16] combines Ridge regression with the kernel trick. We will briefly summarize the regression process and then explain the concept of the Ridge regression.

3.2.1 Ridge Regression

A multi-linear regression is a model which expresses the target output value in terms of a sum of weighted input features, as follows:

$$y = w_0 + w_1 x_1 + w_2 x_2 + .. + w_p x_p + \epsilon \tag{3.1}$$

They are often fitted using the least squares approach. This approach uses the *mean squared error* (MSE) as loss function. The MSE is the total sum of the squared differences between the predicted target value $\hat{y_i}$ and the actual target value y_i for all the points in the training set. The least squares approach changes and finds the optimal parameters to minimize the MSE, thanks to gradient descent.
The ridge regression works in the same way as linear regression, but adds a penalty term to the loss function: the effect, controlled by the λ hyper-parameter, is to reduce the complexity of the final estimated model. The new objective is to minimize :

$$\sum_{i=1}^{n}(y_i - \hat{y_i})^2 + \lambda \sum_{j=1}^{p} w_j^2 \tag{3.2}$$

n is the number of instances in the data-set.
p is the number of features.
y_i the target value for the i^{th} instance.
$\hat{y_i}$ the estimated value for the i^{th} prediction.

It is a way to prevent overfitting and thus improve the generalization performance of a model by restricting the models possible parameter settings : if ridge regression finds two possible linear models, it will prefer the linear model that has a smaller overall sum of squared feature weights.

Besides, ridge regression is a solution to the bias–variance trade-off: a simple model would have high-bias and low-variance, or vice versa. In statistics, the model error is the addition of:

- the bias error comes from erroneous assumptions in the learning algorithm: the model fails to fit the training set. High bias can entail under-fitting due the fact that the algorithm misses the relevant relations between features and target outputs.
- and the variance error comes from sensitivity to small fluctuations in the training set: the model fails to fit the testing set. High variance can entail overfitting cause because the algorithm models the random noise in the training data, rather than the intended outputs.

Actually, the higher is the number of features p, the higher is variance error and the lower is the bias error. ridge regression allows to get the optimal value of p in order to find the minimum error of the model, as presented in the below figure.

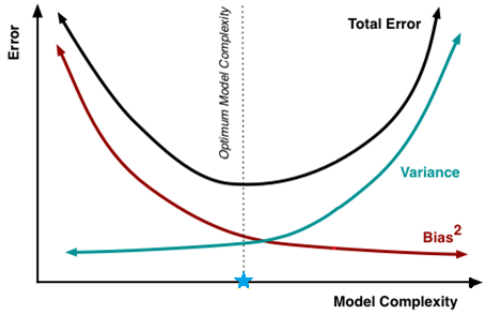

Figure 3.2: Optimization of the total error model. Source: [2]

3.2.2 Kernel Trick

The **kernel trick** is a way of mapping observations from a general set S into an inner product space V, without ever having to compute the mapping explicitly. In other words, it is to transform the input data points to a new feature space where a linear model is used, in the hope that the observations gain meaningful linear structure in the new space.
The trick to avoid the explicit mapping is to use learning algorithms that only require dot products between the vectors in V and choose the mapping such that these high-dimensional dot products can be computed within the original space, by means of a kernel function. In fact, the algorithm doesn't have to perform this actual transformation on the data points: the transformed feature representation is implicit. This makes it practical when the transformed feature space is complex or even infinite dimensional.
For all $\mathbf{x}$ and $\mathbf{x}'$ in the input space S, a certain function $K(x, y)$, called kernel, can be expressed as an inner product. The goal is to construct:

$$\phi : S \to V \text{ such that } K(x, y) = \langle \phi(x), \phi(y) \rangle_V$$

The kernel function points out the similarity of two original points in the new space:

- the **RBF** - radial basis function - kernel is: $K_{RBF}(x, y) = exp\left[-\gamma ||x - y||^2\right]$
- the polynomial kernel, for degree d, is: $K(x, y) = (x^T y + c)^d$.
  In 2D, we have $K(x, y) = \left(\sum_{i=1}^{n} x_i y_i + c\right)^2$

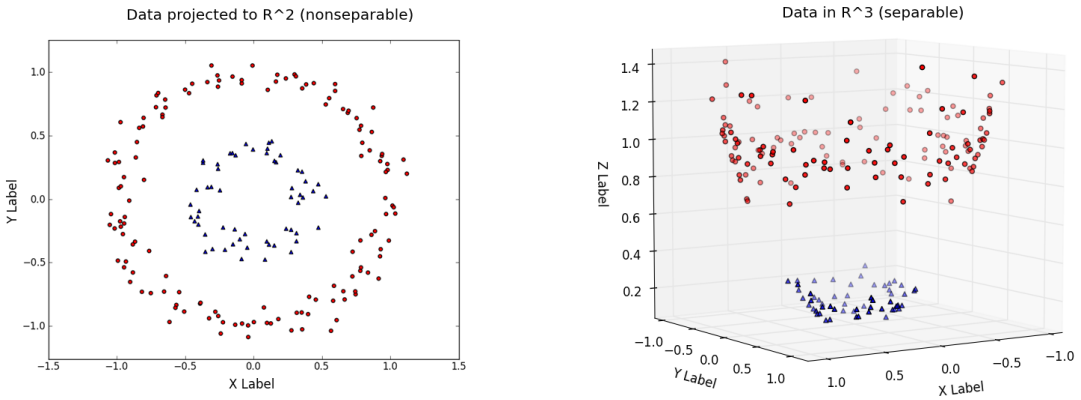

Figure 3.3: Kernel trick. Source: [3]

Mathematical demonstration

For the ridge regression, the goal is to map data to a higher dimensional space and perform linear regression in the embedded space. The kernel trick is applicable to the ridge regression.

Proof
Let us recall that the prediction of the ridge regression is:

$$f(x) = \langle x, \beta_{Ridge} \rangle \tag{3.3}$$

with $X \in \mathbb{R}^{n \times (p+1)}$, λ the parameter of ridge regression and

$$\beta_{Ridge} = (\lambda I + X^T X)^{-1} X^T y$$

Then by multiplying on the left side by $(\lambda I + X^T X)$, we get :

$$\begin{aligned} \lambda I \beta_{Ridge} + X^T X \beta_{Ridge} = X^T y \Leftrightarrow \beta_{Ridge} &= \frac{1}{\lambda} X^T (y - X\beta_{Ridge}) \\ \Leftrightarrow \beta_{Ridge} &= X^T \alpha \end{aligned}$$

where $\alpha = \frac{1}{\lambda}(y - X\beta_{Ridge})$
Thus, we have:

$$\lambda\alpha = y - X\beta_{Ridge} \Leftrightarrow \lambda\alpha = y - XX^T\alpha \Leftrightarrow \alpha = (\lambda I + XX^T)^{-1} y$$

The equation 3.3 can be written as:

$$f(x) = xX^T\alpha = xX^T(\lambda I + XX^T)^{-1} y \tag{3.4}$$

Let $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ a kernel. It exists an Hilbert space $\mathcal{H}$ and a application $\phi : \mathcal{X} \mapsto \mathcal{H}$ such as:

$$\forall x, y \in \mathcal{X} \times \mathcal{X},\ k(x, y) = \langle x, y \rangle_{\mathcal{H}}$$

Then applying the ridge regression to the observations in $\mathcal{H}$ is the same as:

$$f_\phi(x) = \phi(x)\Phi^T(\lambda I + \Phi\Phi^T)^{-1} y$$

where Φ is the image of x^i (the i^{th} data input) by ϕ in $\mathcal{H}$. Finally, we get:

$$_\phi(x) = \kappa(\lambda I + K)^{-1} y \tag{3.5}$$

$\kappa \in \mathbb{R}^n$ with $\kappa_i = \langle \phi(x), \phi(x^i) \rangle_{\mathcal{H}} = k(x, x^i)$
$K \in \mathbb{R}^{n \times n}$ with $K_{il} = \langle \phi(x^i), \phi(x^l) \rangle_{\mathcal{H}} = k(x^i, x^l)$

It is not necessary to use explicitly ϕ and K to train a ridge regression in the new feature space $\mathcal{H}$ defined by a kernel.

3.3 Multi-layer Perceptron (MLP)

3.3.1 Basic concept

A **multi-layer perceptron** is a class of feed-forward artificial neural network. A neural network is composed of layers of nodes which activate at various levels depending on the previous layer's nodes.

MLP is used for supervised learning with labeled data and refers to a neural network with at least three layers of nodes:

- an input layer: the way the network takes in data
- one or several intermediate layers: the computational machine of the network which transforms the input to the output
- an output layer: the way that results are obtained

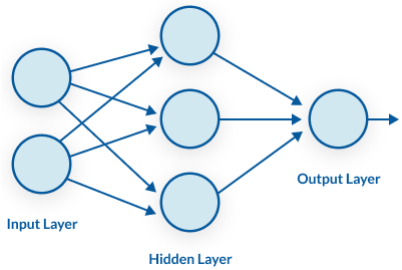

Figure 3.4: Example of Multi-layer perceptron. Source: [4]

3.3.2 Detailed model

For each training sample, nodes activate at different levels depending on a weighted sum of each connection of the previous. This is done by the **activation function**, which is really important for an Artificial Neural Network to learn a non-linear complex functional mappings between the inputs and response variable. It introduces non-linear properties to the network by converting an input signal of a node to an output signal, which will be used as an input in the next layer in the stack. In practice, the activation function could be the function :

- **sigmoid**: $f(x) = \frac{1}{1+e^{-x}}$
- **tanh**: $f(x) = tanh(x)$
- **ReLu**: $f(x) = max(0, x)$
- **identity**: $f(x) = x$

A neural network without activation function would be a linear regression model. They are considered universal function approximators: they can approximate continuous functions on compact subsets of $\mathbb{R}^n$, under assumptions on the activation function [17]. Simply put, any process can be represented as a functional computation in neural networks.
Besides, when training a neural network, the expected output is a level of activation for each node in the output layer. But the weights of each connection is not perfectly accurate during the training. In fact, MLP use the method called **back-propagation** to learn from training data. And to apply this method, the activation function should be differentiable.

Back-propagation is a way of computing gradients of expressions through recursive application of chain rule. You will find the proof below.
The core problem is: given some function $f(x)$ where x is a vector of inputs, how can we compute the gradient of f at x, i.e. $\nabla f(x)$. For the MLP, f corresponds to the loss function (in our problem it will the MSE, defined in 3.2) and x consists of the training data and the neural network weights.

Proof

Notations:

- $(X_k, Y_k)_{k=1,...,n}$ the data
- $X_k \in \mathbb{R}^{n \times p}$
- $Y_k = (y_{k1}, ..., y_{ks}) \in \mathbb{R}^s$ with s output layers
- Total Cost function: $E(w) = \frac{1}{n} \sum_{k=1}^{n} E_k(w)$
- Individual cost function: $E_k(w) = \frac{1}{2} \|Y_k - \hat{Y}_k\|^2$
- $w_{ji}(n)$ is the weight of the i^{th} neuron (layer n) to the j^{th} neuron (layer $n+1$)
- $a_j = \sum_i w_{ji} \sigma(a_i)$ is the signal arriving at the j^{th} neuron

  With $z_i = \sigma(a_i)$, we have: $a_j = \sum_i w_{ji} z_i$

With gradient descent, we get:

$$\Delta w_{ji}(k) = -\eta \frac{\partial E}{\partial w_{ji}} = -\frac{\eta}{n} \sum_{k=1}^{n} \frac{\partial E_k}{\partial w_{ji}}$$

Using the definition of a_j, we have

$$\frac{\partial E_k}{\partial w_{ji}} = \frac{\partial E_k}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = z_i \frac{\partial E_k}{\partial a_j} = z_i \delta_j$$

with $\delta_j = \frac{\partial E_k}{\partial a_j}$

For the output layers, δ_k can be calculated by the outputs of the MLP:

$$\delta k = Y_n k - \hat{Y}_n k$$

For the hidden layers, we get :

$$\delta j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial}{\partial a_j} \sum_i w_{ki} \sigma(a_i) = \sum_k \frac{\partial E_n}{\partial a_k} w_{kj} \sigma'(a_j)$$

So $\delta j = \sigma'(a_j) \sum_k w_{kj} \delta_k$
Consequently, all the data on the network can be calculated by back-propagation of the error: learning occurs in the perceptron by changing connection weights after each piece of data is processed, based on the amount of error in the output compared to the expected result.

Chapter 4

Machine learning application

The machine learning models implementation has been done on several *Jupyter Notebook* [1] files. They can be found on this link. To compute machine learning on *Jupyter Notebook*, we used *Scikit-learn* [2], one of the most popular library to apply machine learning for the Python programming language. In this report, we only present the most important results.

4.1 Objective

The main goal of the project is to use machine learning algorithms instead of solving Schrodinger equations in order to considerably reduce calculation time.

The process is to use the *Corrections* files to get the adsorption energy values which have been computed for 200 configurations of the primitive cell. With that data, we can train machine learning algorithms in order to predict the activation energy for the 19000 configurations. In practice, we will only use the 200 computations to compare the model performance. However in practice, the machine learning models are trained with a difference of energy and not exactly the adsorption energies of the *Corrections* file. The new target is precisely:

$$\epsilon_{i-target} = \epsilon_{i-RPA} - \epsilon_{PDE+D2} \tag{4.1}$$

$\epsilon_{i-target}$ is the energy target for the $i-th$ configuration.
ϵ_{i-RPA} is the adsorption energy for the $i-th$ configuration. It is already computed and available in the *Corrections* files.
ϵ_{PDE+D2} is the energy of the $PDE+D2$ trajectory, i.e. the mean of the potential energies of all the configurations. The potential energy is computed thanks to the ASE library.

This choice is based on the idea that with many energies, the fluctuations of the difference 4.1 are lower than those of the ϵ_{i-RPA}. Therefore, the predictions are better and more accurate. And if we want to have the predicted adsorption energy of a configuration, we have only to add the $PDE+D2$.

In a first step, we will separate our data set:

- a **training set**: it contains a known output. The model learns on this data in order to be generalized to other unseen data.
- **test set**: it is used in order to test the models' predictions on this subset.

[1] *Jupyter Notebook* is a web-based interactive computational environment. It allows to create JSON documents containing an ordered list of input/output cells which can contain text, code, mathematics, rich media and plot.

[2] On *Scikit-learn*, most of the models are already implemented. For example, they are various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means.

4.2 Position descriptors

The descriptors of the machine learning algorithm were initially the positions of the atoms in each configuration, expressed in a Cartesian coordinate system (X,Y,Z). Those descriptors are a good starting point as they scale linearly and can be trivially obtained from the provided files. However, they are not invariant by rotation and translation. This will motivate the subsequent work on descriptors based on atomic distances.

4.2.1 First results

We test with the three machine learning algorithm described in the previous chapter. The error column presents the **mean squared error (MSE)**, chosen as the loss function. It is the total sum of the squared differences between the predicted target value and the actual target value for all the points in the training set.
The MSE function is defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2 \tag{4.2}$$

with y_i the target value and $\hat{y_i}$ the estimated target value.

Here, we test manually three times the performance of the algorithm with a random choice for the parameters: the means of the results of the three simulations results are presented in the following table 4.1. A really good performance is to reach 0.3 $kcal/mol$as the test error.

	Train set RMSE [1]	Test set RMSE	Processing time (s) [2]
Random Forest	0.3210	0.6748	0.69
Kernel Ridge Regression	0.3451	0.7658	0.43
MLP Regression	0.7467	1.2982	0.47

[1] Error expressed in kcal/mol
[2] Intel Core i5-7200U CPU, 2.5 GHz 2.61 GHz

Table 4.1: Results with a random choice for the parameters. The features are the positions of the atoms.

The train set errors have been computed in order to see the minimum error of our models, in other words the best accuracy that we can reach. To do that, we overfit on the train set.
The proportion of the number of samples of the train set is set to 50 % of the total number: this rate, usually around 75 %, is low because our data-set contains only 200 instances.
For each algorithm, some parameters have been changed in order to improve the model. After the basic tests, we can note that the processing time is quite similar to the three machine learning algorithms. It seems that the random forest provides the best results in term of the prediction error for the test set. Nevertheless, the tests are made with a random choice of parameters: now, the setting of the parameters has to be done.

4.2.2 Improve the performance

In fact, we have split the data into a training and test sets. But what if the split we make is not random ? For instance, if one subset of our configurations has only similar position for all the atoms. This will result in overfitting, which has been done deliberate previously. Now we want to avoid it. Moreover, when we manually tune the hyper-parameters [3] values, there is still a risk of overfitting. Thus, cross validation and grid search are two essential ways to optimize hyper-parameters for a model in order to get the best performance.

[3] A hyper-parameter is a parameter whose value is set before the learning process begins. The algorithm training can not learn it: the number of hidden layers in the multi-layer perceptron is an example of hyper-parameter.

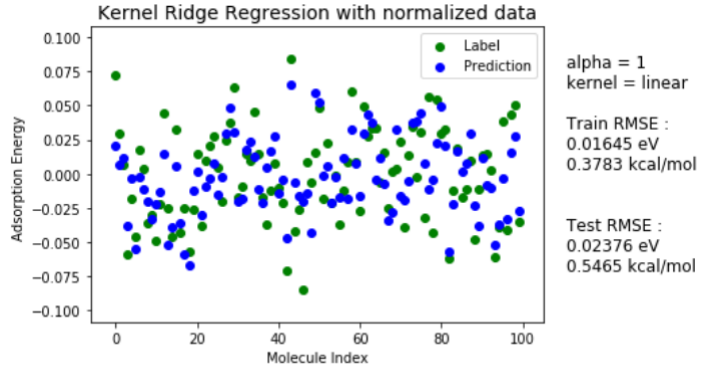

Figure 4.1: Example of a kernel ridge regression prediction with atoms positions as features.

Cross validation

In fact, the goal of a cross validation is to estimate model performance. The process idea, explained in the figure 4.2.2, is to divide the train set in k-folds and to create a new subset called validation set. The training proceeds on the training set, after which the validation set allows to evaluate the model, and then predictions or final evaluation can be done on the test set. The following procedure is followed for each of the k-folds:

1. a model is trained using $k-1$ of the folds as training data
2. the resulting model is validated on the remaining part of the data

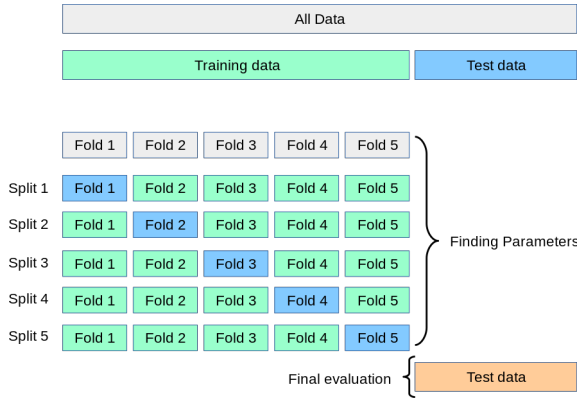

Figure 4.2: Cross validation process. Source: [5]

Grid search

The objective of a grid search is to optimize hyper-parameters in order to improve the model performance. Instead of manually testing some hyper-parameters values, a grid search algorithm is an exhaustive searching through a provided subset of the hyper-parameters space. In other words, we have to specify the list of possible hyper-parameters values that we want to test, and then the algorithm will train models with every possible combination of the specified hyper-parameters values.
A grid search must also be led by a performance metric which allows to assess the performance of each trained model. This performance is often measured by cross-validation on the training set.

It can be also extremely computationally expensive because it builds a model on each parameter combination possible: it iterates through every parameter combination and stores a model for each combination.
The figure 4.2.2 represents a grid search example of nine trials for optimizing a function with two hyper-parameters.

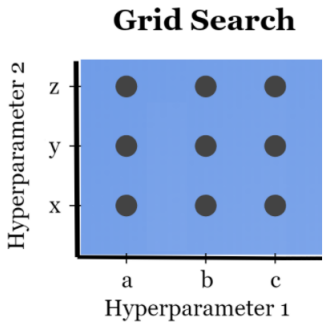

Figure 4.3: Grid search. Source: [6]

Results with optimization

We make a grid search in order to tune the hyper-parameters for the three machine learning models. For the random forests, we decided to focus on those presented in the subsection 3.1.3.
For the kernel ridge regression, we try to tune the following parameters:

- the λ parameter in the equation 3.2
- the parameter γ in the expression of the RBF kernel
- the type of kernel

For the multilayer perceptron, we try to tune the following parameters :

- the number of hidden layer sizes
- the maximum number of iterations. The solver iterates until convergence or this number of iterations.
- the type of solver for weight optimization

The table 4.2 presents the results after the cross-validation in the grid search: as the first table, the results are the mean of three tests. The grid search needs a lot of computational time such as several minutes for some models: the more parameters we test, the longer the execution is. By comparison with the figure 4.1, we note that the performance of the multi-layer perceptron increased by 50 %. The grid search allows also to decrease the test error of the random forest. We note that the error decreases for the three algorithms as well, which means that the grid search builds are efficient.

	Train set RMSE	Test set RMSE	Processing time (s) [2]
Random Forest	0.2366	0.6433	103.43
Kernel Ridge Regression	0.5361	0.6784	2.24
MLP Regression	0.5212	0.7853	73.81

[2] From now on, the processing time corresponds to the grid search and the predictions

Table 4.2: Errors with cross-validation and grid search. The descriptors are the positions of the atoms.

However, physically speaking, the model is not really accurate because as the primitive cell moves, all the positions change. But if the trajectory is a translation, the position will move as much: then by focusing on the atoms positions, it is impossible to determine if the primitive cell deformed or no. A solution to solve this problem is to change the features of the data-sets.

4.3 Distance descriptors

In order to avoid the issues with direct use of the coordinates as descriptors (lack of roto-translational invariance), we develop a second type of descriptors for the machine learning algorithms. The new ones are the distances of the closest neighbors of a specific atom of reference. The atoms of reference are the 42 atoms for each configuration. Nonetheless, to create the data set with the features, several tasks have to been done.

4.3.1 Get the neighbors of one atom

Let's see for a given configuration. Firstly, we take one atom as the atom of reference. For this specific atom, we search its neighboring atoms in a sphere of given radius. Like the configuration represents a primitive cell, it can be represented by an infinite network: then, if the radius is high, the number of neighbours can be superior to 41. Secondly, we compute the distance of the neighboring atoms from the atom of reference. The distances and the radius are expressed in Ångström [4]. We choose a radius of 4 Å.

4.3.2 Get the neighbors for one configuration

Let's see for a given configuration. We want to get the neighboring atoms from all the atoms of the configuration. In fact, we iterate the search of the neighbors of a specific atom for all the atoms of the configuration taken as atom of reference.

4.3.3 Sort the neighbors for all the data set

After getting the neighbors for all the configurations, the question is: how can we create a data frame whose columns are the same for all the configurations and all the atoms of reference ? For example, if the first atom of reference has one and ten neighbors for respectively the first and the second configuration, the number of columns will be different and it will be impossible to interpret. Actually, we sort the neighboring atoms by their type. The atom types for the primitive cell are Aluminium, Carbon, Hydrogen, Oxygen and Silicon.
Then, we compute the maximum number for each type, for all the atoms of all the configurations. Those numbers will be the number of columns for each atom.
Finally for each atom of reference, we create a list of the neighbor distances: it is sorted alphabetically for the atom type, and in ascending order for the distances for the same atom type. If there are less neighbors for one atom type that the maximum number, we set 0.

The figure 4.3.3 shows the process. The letters $[Al, C, H, O, Si]$ are the symbol of the atoms. The numbers corresponds to the distance between them and the atom of reference.

[4] 1 Å$= 10^{-10}$m. The Ångström is often used in molecular theory because it refers to the size of atoms.

```
[1]: List of the neighbors for one spectific atom of reference:
     [['O', 5], ['O', 7], ['Si', 6], ['Al', 3], ['O', 1], ['Si', 10]]

[2]: Number of neighbors for each atom type:
     {'O': 3, 'Al': 1, 'Si': 2}

[3]: Maximun numbers of neighbors for each atom type for all the atoms:
     {'Al': 1, 'C': 1, 'O': 3, 'Si': 4}

[4]: Neighbors grouped by atom type and sorted in ascending order for the distance:
     {'Al': [3], 'O': [1, 5, 7], 'Si': [6, 10]}

[5]: List of the distance order by the previous dictionnary:
     [3, 0, 1, 5, 7, 6, 10, 0, 0]

[6]: Dataframe for a configuration with 2 atoms:
```

	A-1	C-1	O-1	0-2	0-3	Si-1	Si-2	Si-3	Si-4	A-1	C-1	O-1	0-2	0-3	Si-1	Si-2	Si-3	Si-4
config 1	3	0	1	5	7	6	10	0	0	etc.	etc.	etc.	etc.	etc.	etc.	etc.	etc.	etc.

atom of reference 1 atom of reference 2

Figure 4.4: Procedure of the creation of the distance descriptors

4.3.4 Results

Once the data-frame created, we apply to it the machine learning algorithms. We make a cross validation with a grid search to optimize the tuning of the hyper-parameters and to get the best models performance. The results are the means of three simulations and are presented in the following table.

	Train set RMSE	Test set RMSE	Processing time (s)
Random Forest	0.2182	0.6008	147.14
Kernel Ridge Regression	0.0822	0.6090	9.97
MLP Regression	0.5103	2.769	530.05

Table 4.3: Mean of RMSE error with grid search. The descriptors are the distances of the neighboring atoms as descriptors.

The first significant evolution is the processing time: the new data-set containing more than 1000 features, the exhaustive search through the list of the provided parameters needs a lot of computational time. By comparison with the figure 4.2, we notice that the change of the features is efficient because some results improved. For both the random forest and the kernel ridge regression, there is a little drop of the test errors. Those errors respectively decreased by 6.6 % and 10.2 %. However, the multi-layer perceptron has disappointing results. Precisely, two of the three simulations provided similar results with train errors inferior to 0.2 kcal/mol and test errors superior to 5 kcal/mol. Such a high difference between the two errors means overfitting: the model corresponds too closely to the train set. It captures more local variations rather than trying to find the more global trend, so it does not generalize with unknown data. It can be explained by the fact that neural networks perform best with a lot of data points: here, we trained the model with only 100 points which are not enough. Historically, the multi-layer perceptrons were popular in the 1980s in the field of image recognition: the input images were transformed to pixel representations and many picture trained the neural networks.

In this section, the features are the distances of the closest neighbors of a specific atom of reference. They are sorted alphabetically for the atom type, and the atom type is sorted in ascending order

for the distance. And if there are less neighbors for one atom type that the maximum number, we set 0. However, it is not really logical to sort them in ascending order and then to add 0. The figure 4.3.3 emphasizes this idea for the silicon with the respective distance $[6, 10, 0, 0]$. It would be more coherent to compute the inverses of the distances in the data-frame. The idea is to keep for the descriptors data linked to the distances of the neighboring atoms: the distances allow to know if the movement of the primitive cell in the trajectory is just a translation for instance or if the primitive cell deformed.

4.4 Inverse of the distance descriptors

In this section, we apply the idea to compute the inverses of the distances. By taking for initial data frame the one in the figure 4.3.3, the new data frame is:

[1]: Dataframe for a configuration with 2 atoms:

	A-1	C-1	O-1	0-2	0-3	Si-1	Si-2	Si-3	Si-4	A-1	C-1	O-1	0-2	0-3	Si-1	Si-2	Si-3	Si-4
config 1	0.333333	0	1	0.2	0.142857	0.166667	0.1	0	0	etc.	etc.	etc.	etc.	etc.	etc.	etc.	etc.	etc.

Figure 4.5: New data-frame with the inverse distances

As previously, we make the same process: cross validations with grid search to optimize the tuning of the hyper-parameters and to get the best models performance. And the presented results are the mean of three tests.

	Train set RMSE	Test set RMSE	Processing time (s)
Random Forest	0.1383	0.5674	110.345
Kernel Ridge Regression	0.6969	0.6774	8.80
MLP Regression	0.0246	0.7160	419.34

Table 4.4: Mean of RMSE error with grid search. The features are the inverses of the distances of the neighboring atoms.

These results are quite similar to those in the table 4.3 in term of processing time. We observe that the results improves for two of the three algorithms. The random forest reaches the lowest test error for all the simulations. The multi-layer perceptron is the lowest with the inverse distances data as well. But we note that there is still overfitting of the train set with the significant difference between the errors.
For the kernel ridge regression, the results were better with the previous features. We remark an unusual fact: the train error is superior to the test error. It is quite confusing but possible. For instance, it happens if the test set has the similar statistical properties.

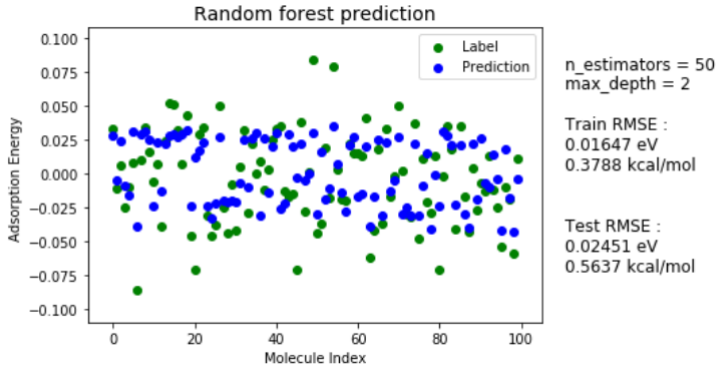

Figure 4.6: Random forest prediction which the inverse distances as descriptors

Conclusion

In the past decade, artificial intelligence has transformed our world, from big tech companies such as Facebook to musical application such as Spotify. The technological evolution appears in science materials by replacing expensive experiments and by solving quantum equations. From now on, all the computations taking many years to be executed are not necessary. With interpolations and predictions, quantum equations can be solved by machine learning methods. The project was divided between theory and practice.

The theory was focused on the random forest, the kernel ridge regression and the multi-layer perceptron. They are used for a regression in a supervised learning, i.e. with targeted samples. The random forest provides brilliant prediction performance without any standardization of the data. Its drawbacks are that we can not interpret it because of the vast number of decision trees used. The kernel ridge regression allows to prevent overfitting with the addition of a penalty term. And the kernel trick transforms the input data points to a new feature space in order to apply a linear model. The multi-layer perceptron is good with data involving many features. Once trained, the predictions are pretty fast as well.

The problem of this project was to test those three models with data in relation to materials science. With a data-set containing 200 samples for which the adsorption energy has been computed, the goal was to be able to create a machine learning model to predict the adsorption energy for other configurations. In practice, we the data-set was divided in a train and a test set with 100 samples. The models learned on the train set and me evaluate them with the test set. We created three types of features for the data-set:

- the position descriptors referring to the position between the atoms and an atom of reference
- the distance descriptors corresponding to the distance between the neighboring atoms and the atoms of reference
- the inverse of the distance between the neighboring atoms and the atoms of reference

The best results were reached with a random forest. On average, this method provides the best results, followed by the kernel ridge regression. However, the processing time to apply kernel ridge regression is very lower than the others. The multi-layer perceptron gives often overfitting as well, due the lack of inputs. The idea to use as descriptors the inverse distance between the neighboring atoms and the atoms of reference brings the best results.
In a future work, it would be interesting to modify the features. We search the neighboring atoms in a sphere of radius 4 Ångström: for instance, run the algorithms with a bigger radius in order to compare the results. Moreover, the features does not distinguish the type of atom. Thus, add properties of the atom such as the atomic number could be an idea.

Bibliography

[1] Sydney F. Seeing the forest for the trees: An introduction to random forest, August 2019. *Alteryx Community*.

[2] Kyoosik Kim. Ridge regression for better usage, January 2019. *Towards Data Science*.

[3] Harish Kandan. Understanding the kernel trick, August 2017. *Towards Data Science*.

[4] Perceptrons and multi-layer perceptrons: The artificial neuron at the core of deep learning, Neural Network Concepts. *MissingLink.ai*.

[5] Scikit learn developers (BSD License). Cross-validation: evaluating estimator performance.

[6] Jack Stalfort. Hyperparameter tuning using grid search and random search: A conceptual guide, June 2019. *Towards Data Science*.

[7] B Jack Copeland. *The essential turing*. Clarendon Press, 2004.

[8] T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997.

[9] Johannes Hachmann, Roberto Olivares-Amaya, Sule Atahan-Evrenk, Carlos Amador-Bedolla, and Alan Aspuru-Guzik. The harvard clean energy project: High-throughput screening of organic photovoltaic materials using first-principles electronic structure theory. *Bulletin of the American Physical Society*, 57, 2012.

[10] Pierre Gauthier. Algorithme d'apprentissage en chimie quantique et application au screening (sélection) de cellules photovoltaïques. *Institut Elie - Cartan*, 2019.

[11] Hannah Ritchie and Max Roser. co_2 and greenhouse gas emissions. *Our World in Data*, 2017.

[12] Richard Car and Mark Parrinello. Unified approach for molecular dynamics and density-functional theory. *Physical review letters*, 55(22):2471, 1985.

[13] Aron J Cohen, Paula Mori-Sánchez, and Weitao Yang. Challenges for density functional theory. *Chemical reviews*, 112(1):289–320, 2011.

[14] Filipp Furche. Molecular tests of the random phase approximation to the exchange-correlation energy functional. *Physical Review B*, 64(19):195120, 2001.

[15] Dario Rocca, Bilal Chehaibou, Michael Badawi, Tomas Bucko, and Timur Bazhirov. Computing rpa adsorption enthalpies by machine learning thermodynamic perturbation theory. *Bulletin of the American Physical Society*, 2020.

[16] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[17] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.