Martin-Luther Universität Halle-Wittenberg
Departement of Physics

Bachelor Thesis

Machine Learning Prediction of the Stability of Perovskites

Author: Jonathan Schmidt
Email:jonathan.schmidt@student.uni-halle.de

Main Supervisor: Professor Miguel A.L. Marques
Second Supervisor: Dr. Carlos L. Benavides-Riveros

Submission date: 30.8.2016

Table of contents

Acknowledgement

I would like to thank Professor Miguel Marques for allowing me so much freedom in pursuing this topic and for the ideas and help he offered. Furthermore, I want to thank Carlos Benavides-Riveros, Conrad Schuster and especially Mário Marques for the help and support they offered during the work for my thesis.

1 Motivation

Often technical solutions for humanity's problems already exist in theory, the only problem beeing that either the materials needed to implement the solutions are too toxic, rare or expensive to use or they do not exist yet. Every new material that is discovered offers a minuscule chance to solve one of these problems. Unfortunately, the classic way of finding such new compounds is extremely slow. Classically experimental physicists or chemists start from known structures and try to synthesize similar materials without concrete knowledge on the stability of the new material.

New theoretical methods like Density Functional theory (DFT), better algorithms and the exponential rise of computing power paved the way for theoretical structure prediction methods. Nowadays, by using DFT, one can predict the stability of a new compound by calculating the formation energy of the material and all competing phases of the compound and finding the global minimum. Even though this process is orders of magnitude faster than classical methods it is still slow, because DFT calculations and finding minima of a high-dimensional surface are computationally expensive and need millions of hours of CPU time.

If one could somehow presort the set of potential new materials to reduce its size without excluding stable materials, one could greatly accelerate the process of discovering new materials. In order to do this one has to build a different stability prediction model that is faster but not as accurate as DFT.

The idea behind this thesis to train and tests machine learning models on existing DFT-data for perovskites and evaluate their stability-prediction abilities.

2 Density Functional Theory

As the data we used was obtained with DFT, we give a brief overview of this theory.

The Hamilton operator of a condensed matter system can be written as (in atomic units):

$$\hat{H} = -\sum_j \frac{\nabla_j^2}{2M_j} + -\sum_i \frac{\nabla_i^2}{2} + \sum_{j\neq i} \frac{z_j z_i}{|\mathbf{R}_j - \mathbf{R}_i|} + \sum_{j\neq i} \frac{1}{|\mathbf{r}_j - \mathbf{r}_i|} + \sum_{j,i} \frac{z_j}{|\mathbf{R}_j - \mathbf{r_i}|} \tag{1}$$

In (2) the first two terms describe the kinetic of the nuclei and the electrons, the third and fourth term describe the potential energy due to the Coulomb interaction between each other and between the electrons respectively. The last term describes the Coulomb interaction between nuclei of charge z_i and electrons.

The corresponding stationary Schr*o*edinger equation can be written as:

$$\hat{H}\,\Psi(\mathbf{R}_i, \mathbf{r}_j) = E\Psi(\mathbf{R}_i, \mathbf{r}_j) \tag{2}$$

If one could solve the Schrödinger equation, one could, in theory, know everything about the system. Unfortunately, this Schrödinger equation can only be solved either in very simple cases or by doing several simplifications.

2.1 Born-Oppenheimer approximation

One of these important simplifications is the Born-Oppenheimer approximation [2]. This approximation is based on the fact that the mass of the nuclei is several orders of magnitude larger than the electron mass, while the forces are of the same order of magnitude, which causes the nuclei to move extremely slowly in comparison to the electrons. Therefore, one can approximate the nuclei as stationary when calculating the wavefunctions for the electrons. When using this approximation, one separates the wavefunction $\Psi(\mathbf{R}_i,\ \mathbf{r}_i)$ into the product of a nuclear $\Phi(\mathbf{R}_i)$ and an electronic wavefunction $\Upsilon_{\mathbf{R}_i}(\mathbf{r}_i)$:

$$\Psi(\mathbf{R}_i, \mathbf{r}_j) = \Phi(\mathbf{R}_i)\Upsilon_{\mathbf{R}_i}(\mathbf{r}_i) \tag{3}$$

and one also gets two separate Schrödinger equations:

$$\left[-\sum_i \frac{\nabla_i^2}{2} + \sum_{j\neq i} \frac{1}{|\mathbf{r}_j - \mathbf{r}_i|} + \sum_{j\neq i} \frac{z_j z_i}{|\mathbf{R}_j - \mathbf{R}_i|} + \sum_{j,i} \frac{z_j}{|\mathbf{R}_j - \mathbf{r_i}|}\right] \Upsilon\left(\mathbf{r}_i\right) = V(\mathbf{R}_i)\Upsilon(\mathbf{r}_j) \tag{4}$$

$$\left[-\sum_j \frac{\nabla_j^2}{2M_j} + V(\mathbf{R}_i)\right] \Phi\left(\mathbf{R}_i\right) = E\Phi\left(\mathbf{R}_i\right) \tag{5}$$

the first for the electrons and the second for the nuclei. In (4) $\Upsilon_{\mathbf{R}_i}(\mathbf{r}_j)$, the wavefunction of the electrons depends parametrically on the positions of the nuclei.

2.2 Hohenberg-Kohn Theorems

After doing the Born-Oppenheimer approximation the electronic system is easier to solve but it is still depending on $3N$ variables, where N is the number of electrons. Density Functional Theory (DFT) aims to reduce this number by replacing the many-body electronic wavefunction with the electronic density as central variable. This is made possible by the Hohenberg-Kohn theorems[11].

The electron density is defined so that $n(\boldsymbol{r})d^3\mathrm{r}$ is the probability of finding an electron in the volume element $d^3\mathrm{r}$ at the position $\mathbf{r}$ [24]. According to the first Hohenberg-Kohn theorem [11], this electron density uniquely describes the ground state of a system, because it uniquely determines a potential V([n],$\mathbf{r}$) (except for a constant) which in turn defines a unique Hamiltonian. Therefore, one can unambiguously describe an N-electron system using just the electron density or the spin-resolved electron densities, which will be named $n_\downarrow(\boldsymbol{r})$ and $n_\uparrow(\boldsymbol{r})$ for electrons with spin down and spin up:

$$n(\boldsymbol{r}) = n_\downarrow(\boldsymbol{r}) + n_\uparrow(\boldsymbol{r}). \tag{6}$$

The second Hohenberg-Kohm-theorem [11] provides a way to actually find the ground-state wave function and ground-state energy for this Schrödinger equation. It states that a universal functional $F[n_\downarrow(\boldsymbol{r}),\ n_\uparrow(\boldsymbol{r})]$ exists such that it is independent of the external potential and that the minimum of the energy functional:

$$E[n_\downarrow(\boldsymbol{r}), n_\uparrow(\boldsymbol{r})] = \int n(\mathbf{r}) V_{\text{ext}}(\mathbf{r}) d^3r + F[n_\downarrow(\boldsymbol{r}), n_\uparrow(\boldsymbol{r})]$$

is the correct ground state electronic density of the system. Therefore, by minimizing the energy functional $E[n_\downarrow(\boldsymbol{r}), n_\uparrow(\boldsymbol{r})]$, while keeping the electron number N constant, one can calculate the ground state electron density.

2.3 Kohn-Sham Equations

In order to reduce the number of independent coordinates from 3N spatial coordinates of N electrons to the 3 spatial coordinates of the spin densities, one uses a fictitious system of non-interacting electrons. For this fictitious system, called the Kohn-Sham system, one only has to solve an one electron Schrödinger equation with an effective potential V_{eff}, which is only depending on the spin densities instead of the positions and spins of all N electrons [15].

$$\hat{H}\psi = \left(-\frac{\nabla^2}{2} + V_{\text{eff}}([n_\downarrow, n_\uparrow], \mathbf{r}) \right) \Psi_i = E_i \Psi_i \tag{7}$$

As solution one uses a single Slater determinant with the N wavefunctions Ψ_i with the lowest energy eigenvalues E_i. The electron density of the actual system is:

$$n(\mathbf{r}) = \sum_{i}^{N} |\Psi_i(\mathbf{r})|^2 \tag{8}$$

The effective potential $V_{\text{eff}}([n_\downarrow, n_\uparrow], \mathbf{r})$ is expressed as the sum of an external potential and two functionals of the electron density $n(\mathbf{r})$ or the spin densities:

$$V_{\text{eff}}(\mathbf{r}) = v(\mathbf{r}) + u([n], \mathbf{r}) + v^{\sigma}_{\text{xc}}([n_\downarrow, n_\uparrow], \mathbf{r}) \tag{9}$$

where:

- $v(\mathbf{r})$ is the external potential due to the nuclei in the Born-Oppenheimer approximation, and depends parametrically on the nucleus positions
- $u([n], \mathbf{r}) = \int \mathrm{dr}^3 r' \, \frac{n(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}$ is the classical Coulomb repulsion between the electrons
- $v^{\sigma}_{\text{xc}}([n_\downarrow, n_\uparrow], \mathbf{r}) = \frac{\delta E_{\text{xc}}}{\delta n_\sigma(\mathbf{r})}$ is the exchange-correlation potential

The total energy of the electronic system can we be written as:

$$E_{\text{total}} = T_s[n_\downarrow, n_\uparrow] + \int \mathrm{dr}^3 n(\mathbf{r})\, v(\mathbf{r}) + U[n] + E_{\text{xc}}[n_\downarrow, n_\uparrow] \tag{10}$$

where $T_s[n_\downarrow, n_\uparrow]$ is the non-interacting kinetic energy of the electrons,

$$\int \mathrm{dr}^3 n(\mathbf{r})\, v(\mathbf{r}) \tag{11}$$

is the energy of the electrons in the external potential,

$$U[n] = \frac{1}{2} \int d^3 r \int d^3 r' \frac{n(\mathbf{r}) n(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} \tag{12}$$

is the electrostatic repulsion between the electrons and $E_{\text{xc}}[n_\downarrow, n_\uparrow]$ is the exchange-correlation energy which describes all interactions not included in the other terms e.g. the Pauli repulsion. In theory, if one knew $E_{\text{xc}}[n_\downarrow, n_\uparrow]$ one could obtain exact solutions from DFT. However, in practice, $E_{\text{xc}}[n_\downarrow, n_\uparrow]$ can only be approximated. Approximating $E_{\text{xc}}[n_\downarrow, n_\uparrow]$ is one of the bigger difficulties one faces when using DFT, as there is no known analytic description for $E_{\text{xc}}[n_\downarrow, n_\uparrow]$ except for the free electron gas, and a few other model systems.

2.4 Approximation of $\mathbf{E_{xc}}$

The local spin density approximation(LSDA) [24] is a really simple approximation. When using LSDA the exchange-correlation energy in a volume element $d^3 r$ is estimated as:

$$(\varepsilon_x(n_\downarrow(\mathbf{r}), n_\uparrow(\mathbf{r})) + \varepsilon_c(n_\downarrow(\mathbf{r}), n_\uparrow(\mathbf{r})))n(\mathbf{r})\, d^3 r \tag{13}$$

Here $\varepsilon_x(n_\downarrow(\mathbf{r}),\ n_\uparrow(\mathbf{r}))$ and $\varepsilon_c(n_\downarrow(\mathbf{r}),\ n_\uparrow(\mathbf{r}))$ are respectively the exchange- and the correlation energy per particle of a homogeneous electron gas with the spin densities $n_\downarrow(\mathbf{r})$ and $n_\uparrow(\mathbf{r})$. By multiplying with $\mathrm{n}(\mathbf{r})\mathrm{d}^3r$ one gets the average energy in the volume element d^3r. Therefore, the total exchange correlation energy is:

$$E_{\mathrm{xc}}^{\mathrm{LSDA}}[n_\downarrow, n_\uparrow] = \int d^3r\, n(\mathbf{r})(\varepsilon_x(n_\downarrow(\mathbf{r}), n_\uparrow(\mathbf{r})) + \varepsilon_c(n_\downarrow(\mathbf{r}), n_\uparrow(\mathbf{r}))) \tag{14}$$

The LSDA was originally introduced by Kohn and Sham [15] and is still one of the most important approximation techniques. LSDA is exact for constant densities and still works very well for slowly changing densities. There are other methods like the generalized gradient approximation (GGA) that expand on LSDA by using the gradients of the densities or higher order derivatives in order to obtain better results for faster-changing densities.

$$E_{\mathrm{xc}}^{\mathrm{GGA}}[n_\downarrow, n_\uparrow] = \int d^3r\, f(n_\downarrow(\mathbf{r}), n_\uparrow(\mathbf{r}), \nabla n_\downarrow(\mathbf{r}), \nabla n_\uparrow(\mathbf{r})) \tag{15}$$

The DFT results used in this work were obtained with one such functional, namely the GGA of Perdew, Burke, and Ernzerhof [25].

2.5 Solving the Kohn-Sham Equations

When solving the Kohn-Sham equations one selects a starting electron density which can, for example, be a superposition of electron orbitals of the atoms. Then one calculates $V_{\mathrm{eff}}(\mathbf{r})$ (9) with the starting electron density and solves the Kohn-Sham equations while keeping the number of electrons constant. With the solutions $\Psi_i(\mathbf{r})$ of the Kohn-Sham equations one calculates the new electronic densities. This process is continued until self-consistency is achieved [15]. Several numerical methods can be used to solve the partial-differential equation. The results in this work were obtained using a plane-wave expansion of all relevant quantities with the code vasp [16],[17],[18]. This is the most adequate basic set for the case of solids, as it takes advantage of the periodicity of these systems.

3 Machine Learning

According to Arthus Samuel, "Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed"[30]. By deciding on a specific machine learning algorithm, one determines the shape of the model the computer will use, e.g. a neural network or a random forest. All these algorithms are based on statistical methods and need some form of training input data which is used to fit the parameters of the model to the problem.

There are two major types of machine learning, supervised machine learning and unsupervised machine learning. In supervised machine learning the training data includes the desired outputs for all inputs, while unsupervised learning aims to discover a hidden structure in the input data and the desired output is not yet known for the training data. This thesis uses solely supervised learning, specifically regression methods, where the desired output is a value. In the following chapters [4],[5],[6] and [7] different machine learning algorithms will be discussed.

4 Decision Trees, Random Forests and Extremely Randomized Trees

4.1 Decision Trees

In broad terms, a decision tree is a graph like structure in tree form. A more accurate definition can be found in [27] "A decision tree may be either a leaf identified by a [...][value], or a structure of the form:

$$\begin{aligned} C_1&\text{: } D_1 \\ C_1&\text{: } D_1 \\ &\vdots \\ C_n&\text{: } D_n \end{aligned}$$

where the C_is are mutually exclusive and exhaustive [binary] logic conditions and the D_is are themselves decision trees".

Classically when searching for the best logic condition C_i one uses a metric to determine the attribute and the splitting point at each node, e.g. the Gini impurity [5] or an information gain [26]. Unfortunately, these classic methods are very prone to overfitting. To reduce the variance the tree building process can be randomized as in algorithms like Random Forests [4] or Extremely Randomized Trees [9]. In order to keep the bias low, despite the randomization in the training process, ensembles of trees are built.

4.2 Random Forests and Extremely Randomized Trees

Random forests are a method to reduce the variance of decision trees. In this thesis the random forest implementation from the library scikit-learn [23] [4] was used.

A random forest regressor [4] is an ensemble of decision trees $\{h(\mathbf{x}_i, \Theta_k), k = 1, ...\}$. The Θ_k are independent random vectors which were used to randomize the tree-building process. In this implementation of random forest the random vector Θ_k is used for bootstrap aggregating [3] and picking random features to split a node. Given a training set, bootstrap aggregating (also called bagging) generates new training subsets by picking random samples with replacement out of the original set. Each new set is then used to train an individual tree. When searching for the best possible split, a random selection of features is chosen at each node. When training a random forest for regression one typically picks one-third of the features. The split is chosen by minimizing the mean squared error which corresponds to a variance minimization [31]. In the end, the result for an input vector $\mathbf{x}_i$ is the average of all predictions.

Extremely randomized trees (ERT) [9] is another algorithm to reduce the variance of decision trees but it differs in several ways from random forests. In this work the ExtraTree [9] algorithm in its classic version as implemented in the python library scikit-learn [23] was used. The ExtraTree algorithm does not use bootstrap aggregating but it randomizes the tree growing process by picking random splitting points. More specifically at each node N random attributes a_i are selected and for each of them a random cut-point $a_{i,c} \in \left[a_{i_{\min}}, a_{i_{\max}}\right]$ is drawn. The set of samples at each node is split into two parts by the binary condition $a_i < a_{i,c}$ and the Gini impurity (mean squared error for regressors) is computed to evaluate the splits. Afterwards the split with the best score is chosen. The node splitting process is repeated until one of the following conditions applies:

- The set is smaller than the minimum number of samples per set
- All samples in the set have the same value in all attributes
- All samples in the set result in the same output.

This procedure is repeated to produce an ensemble of trees. The output of the ensemble is set equal to the mean of the single-tree-outputs.

5 Neural Networks

5.1 Structure and function of Neural Networks

In general neural networks take inputs and connect them in a nonlinear fashion to calculate an output.

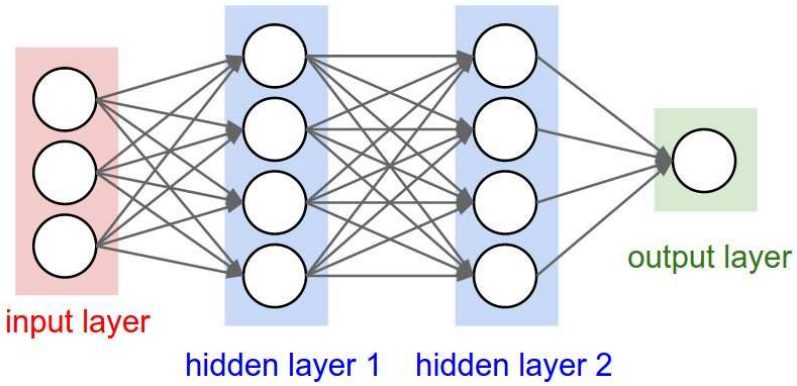


Figure 1. [14] A fully connected Neural Network with two hidden layers, three inputs and one output

Every neural network consists of at least one input layer and one output layer. Between those layers can be an arbitrary number of hidden layers which transform the input values. Each layer consists of a number of neurons, which can be represented by a vector $\mathbf{z} \in R^n$, with n beeing the number of neurons in the layer. Let $\mathbf{z}_i$ be the ith layer in a neural network and z_{ij} be the jth neuron in the ith layer with $j \in \{0, 1, ..., n_i\}$.

When one wants to compute the output of a neural network for a sample the input layer $\mathbf{z_0}$ is set equal to the feature vector representing the sample. The n_1 neurons in the next layer are connected to the input layer through activation functions a_{1j}:

$$\begin{aligned}\mathbf{z_1} &= [z_{10}, z_{11}, ..., z_{1n_1}]^T \\ &= a_{10}(z_{00}, z_{01}, ..., z_{0n_0}), a_{11}(z_{00}, z_{01}, ..., z_{0n_0}), ..., a_{1n}(z_{00}, z_{01}, ..., z_{0n_0})]^T \\ &= \mathbf{a}_1(\mathbf{z_0}) \end{aligned} \tag{16}$$

The network's designer decides which activation functions are used. In most cases the activation functions will be nonlinear functions φ using linear connections of the neurons in the previous layer as arguments e.g.:

$$z_{\mathrm{ij}} = \varphi\left(\sum_k^{n_i} z_{i-1,k}\,\theta_{\mathrm{jk}}\right) = \varphi(\mathbf{z_{i-1}}\theta_j) \tag{17}$$

$$z_i = \varphi([\theta_0 z_{i-1}, \theta_1 z_{i-1}, ..., \theta_{\mathrm{ni}} z_{i-1}]) = \varphi(\Theta \mathbf{z_{i-1}}) \tag{18}$$

In this case, the linear connection would be a scalar product with a weight vector θ, respectively the matrix product with the weight matrix Θ for the whole vector $\mathbf{z}_i$. Commonly used nonlinear functions are the ReLU (rectified linear unit) function, the sigmoid function or hyperbolic tangent.

To sum up, in most neural networks each layer is connected with the previous one by a weight matrix and a nonlinear function.

When training a neural network for classification or regression the last layer is usually a loss layer $\mathcal{L}$ which computes a measure of error for the output $\mathbf{z}_o(\mathbf{x}_i)$ by comparing it to the input values $\mathbf{y}_i$. In this thesis the Loss $\mathcal{L}$ was computed using a simple square loss function:

$$\mathcal{L} = \sum_i (\mathbf{z}_o(\mathbf{x}_i) - \mathbf{y}_i)^2 \tag{19}$$

In order to minimize the loss function the weight matrices of the neural networks are trained with backpropagation. When backpropagating the partial derivative of the loss function with respect to the weights is computed. Steepest descent or other more sophisticated methods are used to find the minimum of the loss function.

The derivative is computed by using the chain rule, therefore all activation functions must be differentiable. For the computation of the gradients with the chain rule the values in the neurons are still needed. Therefore, first all Δw_{ijk} are computed and only afterwards the weights are updated. A training algorithm looks roughly like this:

Initialize weights

do:
  for all training examples:
    propagate example forward
    compute loss
    propagate derivatives backward
  update weights
while error not small enough and number of training cycles $<$ max cycles

In order to reduce overfitting during the training process, a regularization term containing the norm of the weights can be added on top of the normal loss function e.g.:

$$\mathcal{L} = \text{MSE} + \lambda \sum_{i,j,k} |w_{\text{ijk}}| \, (L1 \, \text{regularization}) \tag{20}$$

Regularization causes the error to be smaller when using smaller weights which is giving preference to simpler hypothesis.

5.2 Activation functions

5.2.1 Sigmoid functions

The original idea behind Neural Networks was to emulate networks of biological neurons in a computer. The firing rate of biological neuron stays close to zero as long as no input is received. Once an input is received the firing rate first rises quickly and then approaches asymptotically one hundred percent. Classically normalized sigmoid functions were the most popular activation functions because they are a simple way to represent the firing potential of a biological neuron. All sigmoid functions fulfill these requirements but typically the logistic function (21) and the hyperbolic tangent (22) were used.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{21}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{22}$$

The logistic function has several disadvantages. Because the gradient of the logistic function is $f'(x) = f(x)(1 - f(x))$, the gradient is nearly nonexistent for f(x) close to 0 or 1 causing the logistic function to become very easily saturated. Once the upper layers of the network are close to being saturated they backpropagate near-zero gradients. This issue is known as vanishing gradient problem [1] and can cause either slow convergence or result in poor local minima. Secondly, the output of the sigmoid functions is non-zero-centered which results in slower convergence. On top of that, the exponential function is relatively expensive to compute. (There are however approximations like hard sigmoid which can be used to lower the computation time). Compared to the logistic function, the hyperbolic tangent has the advantage of beeing zero-centered [13].

5.2.2 Rectified Linear Units (ReLU)

With the introduction of rectified linear units many of these problems were solved[19].

$$f(x) = \max(0, x) \tag{23}$$

Rectified linear units have several advantages over sigmoid functions [19]. They are extremely easy to compute, and as long as x stays positive they do not saturate, meaning that their gradient does not approach zero as easily. In addition, neural networks with ReLU activation functions converge several times faster than similar networks with sigmoid functions [19]. Besides their output beeing non-zero-centered, the other disadvantage of ReLU activation functions is that they become saturated as soon as x becomes smaller or equal to zero. To bypass this problem one can use leaky ReLUs [20][10] which do not saturate:

$$f(x) = \max(0.001\,x, x) \tag{24}$$

5.3 Backpropagation methods

5.3.1 Steepest Descent

One simple method to backpropagate the gradient is gradient descent. In gradient descent a new weight $w_{\mathrm{ij}}^{\mathrm{new}}$ is computed as:

$$w_{\mathrm{ij}}^{\mathrm{new}} = w_{\mathrm{ij}}^{\mathrm{old}} - \alpha \frac{d\mathcal{L}}{\mathrm{dw}_{\mathrm{ij}}^{\mathrm{old}}} \tag{25}$$

where α is the learning rate. When calculating $\mathcal{L}$ one sums the errors of all the samples, therefore when computing the gradient, one has to use the whole training set which can be computationally expensive. One alternative to gradient descent which is more viable for big training sets is stochastic gradient descent.

5.3.2 Stochastic gradient descent (SDG)

In SDG one obtains an approximation of the gradient by calculating the gradient for a random subset (batch) of the training set. The updates are a linear combination of the gradient and the previous update.

$$w_{\mathrm{ij}_{\mathrm{new}}} = w_{\mathrm{ij}_{\mathrm{old}}} - \alpha \frac{d\mathcal{L}_{\mathrm{stochastic}}}{\mathrm{dw}_{\mathrm{ij}_{\mathrm{old}}}} + v\Delta w_{\mathrm{ij}_{\mathrm{old}}} \tag{26}$$

The batchsize and the momentum v are two more hyperparameters that need to be optimized. A small batchsize might not be representative of the training set which can prevent convergence. On the other hand, when the batch is too large only a negligible amount of computation time is saved.

6 Ridge Regression

The standard approach for linear regression is finding a vector $\mathbf{x}$ that minimizes the squared error loss function $\mathcal{L} = \sum_i (\mathbf{a}_i \mathbf{x} - \mathbf{y}_i)^2$. The vector $\mathbf{x}$ can be calculated by finding the root of the derivative of the loss function $\mathcal{L}$. This can be performed by solving the normal equation:

$$\mathbf{A}\mathbf{x} = \mathbf{y} \, (A = [a_1 \, a_2 ... a_n]). \tag{27}$$

Unfortunately, this method has several disadvantages. If the features are correlated the solution has a high variance and if they are linearly dependent A cannot be inverted and the normal equation has no solution. In order to counteract this behavior, and to favor one particular solution, one can include a regularization term in the loss function:

$$\mathcal{L} = ||\mathbf{A}\mathbf{x} - \mathbf{y}||^2 + \|\Gamma \mathbf{x}\|^2 \tag{28}$$

This method is called Tikhonov regularization or Ridge regression. In this work the Tikhonov matrix Γ will be chosen as $\Gamma = \lambda \mathbf{I}$ which is also known as L_2 regularization. This favors solutions with a smaller norm [21]. In order to minimize equation (29) one has to calculate the root of the derivative:

$$\mathbf{x} = (\mathbf{A}^T \mathbf{A} + \lambda \mathbf{I})^{-1} \mathbf{A}^T \mathbf{y} \tag{29}$$

7 AdaBoost

7.1 AdaBoost

Adaptive Boosting (AdaBoost) [6] produces ensembles of machines in order to reduce their variance compared to the single machine. Therefore, it has to be used in conjunction with another machine learning algorithm. The boosting algorithm picks a training subset and uses the main machine learning algorithm to build regressors/classifiers. Over the course of the boosting process the probability of each sample to be picked for training is changed as a function of the relative errors the regressor produces for the sample.

In order to calculate the error, different loss functions can be used as long as $L_j \in [0,1]$, e.g.[6]:

$$L_i^j = \frac{|\mathbf{y}_j(\mathbf{x}_i) - y_i|}{\sup_{\forall i} |\mathbf{y}_j(\mathbf{x}_i) - y_i|} \text{ (linear loss function) (jth machine, ith sample)}$$

$$L_i^j = \frac{|\mathbf{y}_j(\mathbf{x}_i) - y_i|^2}{\sup_{\forall i} |\mathbf{y}_j(\mathbf{x}_i) - y_i|} \text{ (square loss function)}$$

$$L_i^j = 1 - \exp\left(-\frac{|\mathbf{y}_j(\mathbf{x}_i) - y_i|}{\sup_{\forall i} |\mathbf{y}_j(\mathbf{x}_i) - y_i|} \right) \text{(exponential loss function)}$$

Samples with a high relative error have an increased likelihood of appearing in the training set so more time is spent training regressors on the difficult samples. After training, the results of the regressor ensemble are combined to boost the precision of the cumulative prediction. When combining the different regressors, the more confident regressors are weighted more heavily.

7.2 Adaboost according to Drucker [6]

In the following AdaBoost as proposed in [6], which is in turn a modification of AdaBoost.R [8], will be explained.

In every step of AdaBoost one regressor is trained on a subset of the training set. This subset is determined by choosing N samples randomly out of the original training set. The samples are picked with replacement, meaning that a sample can be picked multiple times. Initially, each training sample is assigned the same weight w_i and thus the same initial probability p_i of beeing picked for training.

$$w_i = 1 \quad \forall i \in \text{training set}$$

$$p_i = \frac{w_i}{\sum_i w_i}$$

Afterwards a random subset of N training samples is picked with replacement. These samples are used to build a regressor $\mathbf{y}_j$, which is in turn used to calculate a prediction $\mathbf{y}_j(\mathbf{x}_i)$ for every sample $\mathbf{x}_i$ from the whole training set. In the next step the loss function is calculated for the whole training set. Afterwards the average loss $\bar{L^j}$ is calculated in order to introduce β.

$$\bar{L^j} = \sum_{i=1}^{N_1} L_i^j p_i$$

$$\beta_j = \frac{\bar{L^j}}{1 - \bar{L^j}}$$

β is a measure of confidence in the predictor, where a low β corresponds to a confident prediction by the regressor.

After calculating β the weights w_i are updated as

$$w_i^{\text{new}} = w_i \, \beta_j^{\left[1-L_i^j\right]}.$$

In this case, a small β_j represents a high confidence. When the prediction is accurate the weight of the sample is reduced. A small loss L_i^j will also reduce the weight of the sample thus reducing the probability of the sample beeing picked again.

The combined prediction $\mathrm{y}(\mathrm{x}_i)$ of the boosted regressors for an input x_i is found by calculating a weighted median. The regressor's weight when calculating the median is directly connected to its confidence. In order to calculate the weighted median all regressors make a prediction $\mathbf{y}_j(\mathbf{x}_i)$, which is sorted so that $\mathbf{y}_1(\mathbf{x}_i) < \mathbf{y}_2(\mathbf{x}_i) < \mathbf{y}_3(\mathbf{x}_i) < .... < \mathbf{y}_N(\mathbf{x}_i)$. The indices of the β_j are changed accordingly keeping the connection to the $\mathbf{y}_j(\mathbf{x}_i)$ intact. Every prediction is weighted with $\log \frac{1}{\beta_j}$ and the weighted median is calculated so that $y(x_i)$ is the smallest y_t for which the sum of the weights of the previous y_j are bigger than or equal to half the total weight.

$$y(x_i) = \inf \left\{ \mathbf{y}_t : \sum_{j : y_j < y_t} \log \frac{1}{\beta_j} \geq \frac{1}{2} \sum_j \log \frac{1}{\beta_j} \right\}$$

8 Stability

In order to predict the stability of perovskites one first has to explain what "stability" means. One can simplify this problem by limiting the thermodynamic degrees of freedom by only discussing the stability at zero pressure and temperature. As a first idea one can look at the formation energy:

$$F_{\text{form}} = E_c - \sum_i E_i n_i$$

where E_c is the energy of the compound, E_i the energy of the constituents (in their most stable elementary phase) and n_i the number of times the ith constituent is found in the compound. However, the compound can still be unstable towards decomposition in other binary, ternary, or multinary phase. Then one can define a convex hull off stability as "the hypersurface in composition space that passes by all materials that are thermodynamically stable"[28]. When calculating the free energies of new materials, unstable materials will have a positive distance to the convex hull, while the distance to the convex hull will be 0 for thermodynamically stable compounds. In practice the distance to the convex hull can be negative for new compounds as they were not included in the convex hull at the time of calculating the distance. In the following, compounds with a predicted distance to the convex hull (of stability) smaller than $10 \frac{\text{meV}}{\text{atom}}$ will be treated as potentially stable.

9 Test- and Training set

Perovskites are a well-known class of materials with a great variety of applications in modern electronics. Their basic chemical structure is of the form ABX_3. The data used in this work was obtained using plane-wave DFT with the code vasp [16][17][18]. The data contained the distances to the convex hull of stability and the formation energy for 236473 different combinations of ABX_3. These were almost all combinations of ABX_3 using most elements from hydrogen until bismuth. Excluded were all noble gasses and most lanthanides. All compounds with a distance to the convex hull smaller than -0.5 meV/atom or larger than 3 meV/atom were treated as outlier in the DFT calculation and were removed from the training and the test set. In the following, every time a Machine Learning algorithm was trained 20000 of the 236473 compounds were randomly chosen for training while the rest was used for testing.

10 Feature Selection

Besides choosing the optimal algorithm for our problem, the most important decision when using Machine Learning is the selection of the feature vector representing the problem. Considering the problem of stability prediction, the feature vector at least has to uniquely describe every compound.

As starting point 119 features ranging from basic elemental properties like atomic numbers to higher level properties like boiling points were chosen to represent the perovskites (39 features per element and 2 combinations; see also 15.1). The majority of the data was collected using the python library pymatgen [22].

In the following Adaptive Boosting was used on Extremely Randomized Trees and Random Forests to calculate the distances to the convex hull and gradually reduce the number of features. It was expected that the number of features could be reduced greatly as a lot of them are very closely related e.g. the atomic number and the atomic mass. In order to measure the quality of the regressors, the mean absolute error (MAE) of the test set was used. All the following MAEs are averaged over twenty training runs with randomly chosen training- and test sets (see also 9). The starting selection of features resulted in an MAE of 130 meV/atom for the test and 1.7 meV/atom for the training set. It might be noted that one can already see the tendency of decision trees to overfitting even after using AdaBoost and Extremely Randomized Trees/Random Forests to reduce the variance. In order to find the least influential features and remove them, the attribute .feature_importances_ of the regressor class in pymatgen was used. This attribute estimates the importance of the features by calculating out of bag errors. When calculating out of bag errors, one attribute is varied randomly for a random subset of the test set, while all other attributes are kept constant. Then the error of the inputs with the varied attribute is compared to the errors of the original inputs. The .feature_importances are the relative errors originating from this process.

The number of features was reduced step by step to 11 features per element which resulted in the lowest error of all combinations tried. In the following the number of features was further reduced to 5 and then 3 features per element. We found that removing the atomic number, Pauling electronegativity, most common oxidation state, polarizabilities, number of s and p valence electrons and the number of f and d valence electrons had almost no effect one the MAE. The differences in the MAEs when removing the other features are presented in table 1 and 2 for different numbers of features.

test RT	training ERT	test RF	training RF	Removed feature
132	1.7	146	43	Nothing removed
134	1.8	143	43	Pauling electronegativity
130	1.8	134	40	Average ionic radius
137	1.9	148	45	Number of Valence electrons
144	2.4	152	46	Row in the periodic table
162	2.5	160	49	Group in the periodic table

Table 1. MAE in meV/atom using Random Forest (RF) and Extremely Randomized Trees (ERT) in combination with AdaBoost. For the first row, 5 features per element were used in order to train the regressor. In each following row, 1 feature was removed from the training process. All errors are rounded up. (First 2 rows Extremely Randomized Trees, third and fourth Random Forest)

test RT	training ERT	test RF	training RF	Removed feature
133	2.0	136	41	Nothing removed
140	2.2	135	40	Number of Valence electrons
440	354	433	362	Row in the periodic table
322	115	308	170	Group in the periodic table

Table 2. MAE in meV/atom using random forest (RF) and extremely randomized trees (ERT) in combination with AdaBoost.For the first row, 3 features per element were used in order to train the regressor. In each following row, 1 feature was removed from the training process. All errors are rounded up. (First 2 rows Extremely Randomized Trees, third and fourth Random Forest)

Surprisingly, as one can see in Table 2 the location of the elements in the periodic table is sufficient to predict the distance to the convex hull with an MAE of 140. By adding the average ionic radius, the Pauling electronegativity and the number of valence electrons one can decrease the MAE to around 131meV/atom. If one tries to

remove the group or row in the periodic table as feature the MAE rises drastically. The presumed reason is that the algorithm does not have enough information to completely distinguish all elements and therefore cannot identify all unique perovskites. The MAE increase when removing the row is larger than when removing the group. This is the case because the number of valence electrons is closely related to the group, therefore, more information is included when the row and the valence electrons are used than when the group and the valence electrons are used.

11 Hyperparameter optimization

Hyperparameters are all parameters of the machine learning model that are determined before the learning process. After deciding on an algorithm and a feature vector the different hyperparameters of the algorithm have to be fitted to the problem in order to optimize the learning process.

11.1 Random Forest and Extremely Randomized Trees

The most important hyperparameters that have to be fitted for random forests and extremely randomized trees are the number of trees in the ensemble and the percentage of features that is picked randomly to split a node. The values in all the following figures are averaged over 5 runs with randomly selected training- and test sets. The fitting was done for the whole feature set of 139 features and for the set of 33 features (table 1).

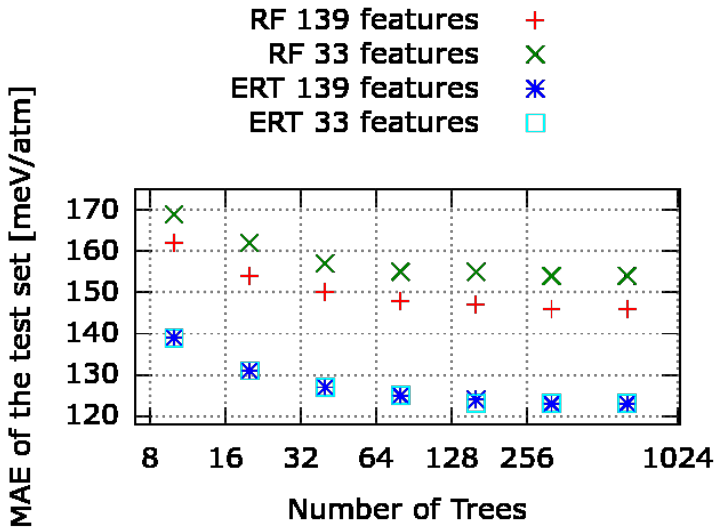


Figure 2. The MAE of the test set for different numbers of trees with Random Forest and Extremely Randomized Trees with the full set of 139 features and with eleven features

As expected the error decreases with a larger number of trees. For both algorithms as well as both feature sets the error converges at around 250 trees (Figure 2) and does not decrease significantly when using more than 350 trees. Therefore in order to keep the computation time low, the number of trees was limited to 350.

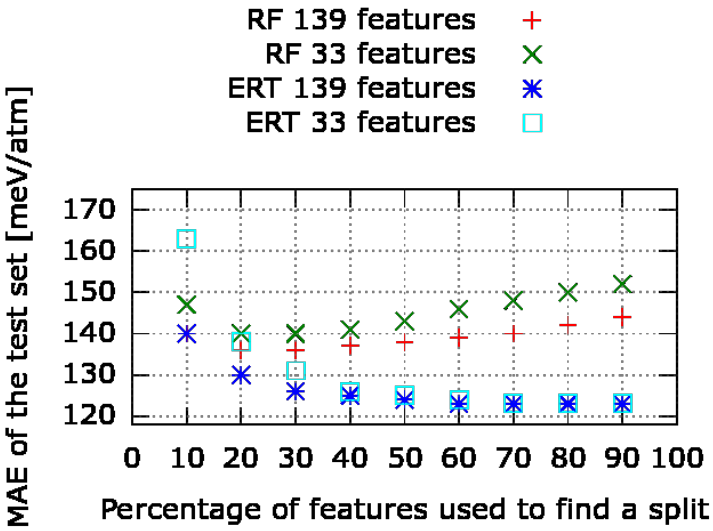


Figure 3. The mean absolute error of the test set for Extremely Randomized Trees and Random Forest with different numbers of features plotted over the percentage of features used at each node to determine the best possible split

When using random forest regressors it is recommended to use one-third of the features to determine the best split at each node which holds true for both feature sets as can be seen in Figure 3. In the case of extremely randomized trees using all features to find the best split was optimal.

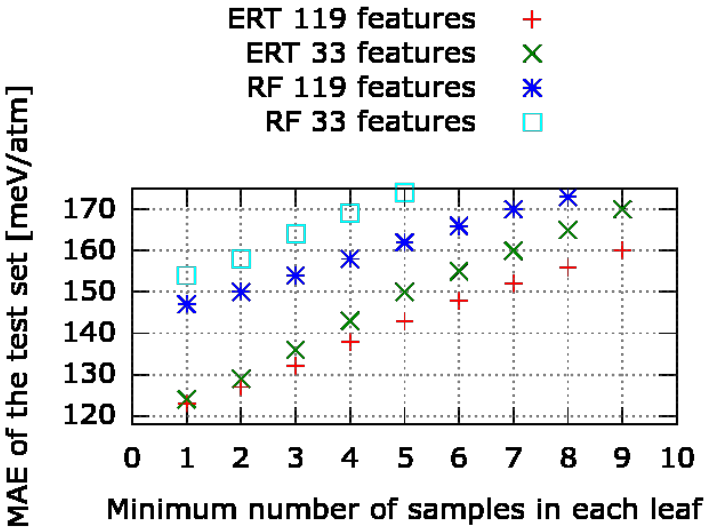


Figure 4. The mean absolute error of the test set for Extremely Randomized Trees and Random Forests for different numbers of minimum samples per leaf

Another question that arises when using decision tree algorithms is, whether to use early stopping criteria. In order to solve this question, the MAE of the test set was calculated for training runs with different minimum numbers of samples per leaf, meaning that splits where a leaf ended up with fewer samples than this minimum number were disregarded. In all cases, early stopping criteria increased the testing error and were therefore not used.

11.2 Adaptive Boosting

AdaBoost was used on random forests and extremely randomized trees. In both cases, it was tested whether the hyperparameters should be changed in comparison to the sole algorithms without AdaBoost.

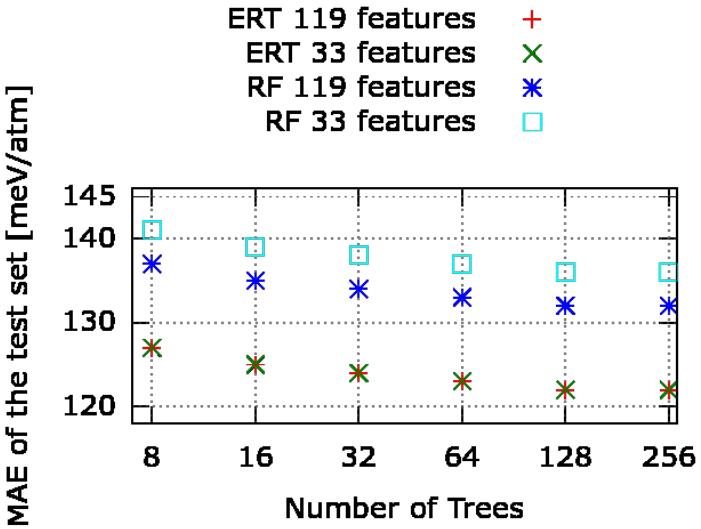


Figure 5. The mean absolute error of the test set for AdaBoost used on Extremely Randomized Trees and Random Forest for different numbers of trees per Random Forest/ERT-ensemble

The number of trees per Random Forest/ERT-ensemble was reduced to 250 as the MAE is converging faster when using AdaBoost. Of course, the actual number of trees is still larger because each boosted Regressor itself contains multiple Random Forests/ERT-ensembles.

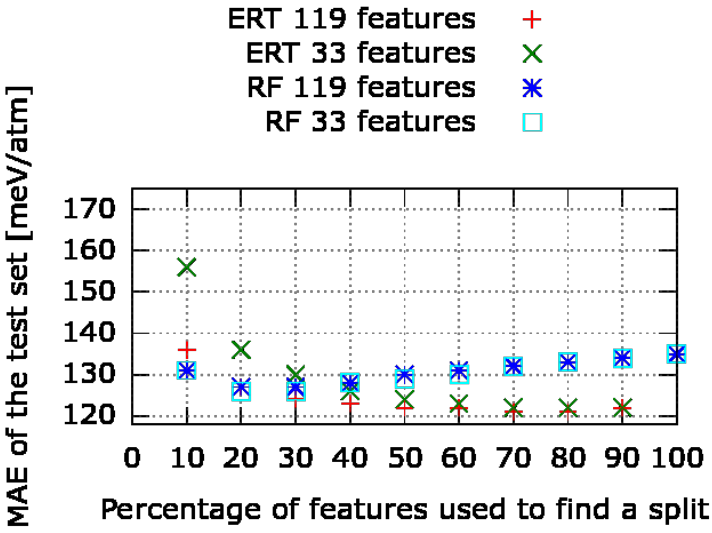


Figure 6. The mean absolute error of the test set for AdaBoost used on Extremely Randomized Trees and Random Forest with different numbers of features plotted over the percentage of features that was used at each node to determine the best possible split

The percentage of features used to find a split was left unaltered as the results

for AdaBoost are qualitatively equal to the ones of the sole algorithms. The same holds true for early stopping criteria as one can see in figure 7.

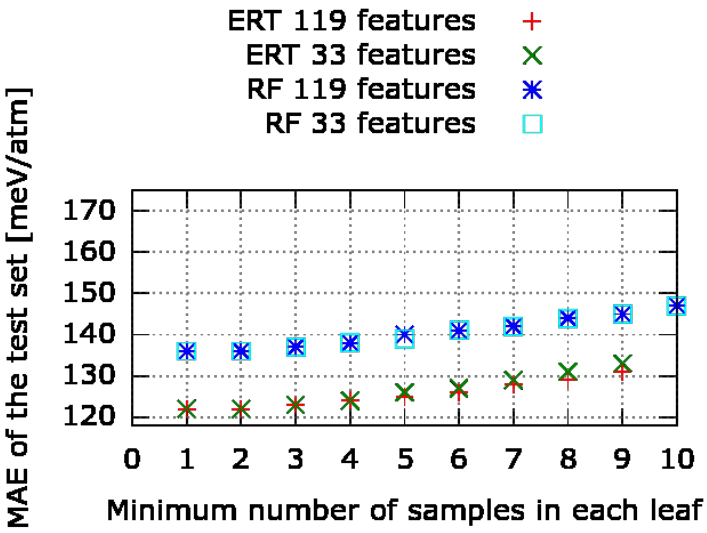


Figure 7. The mean absolute error of the test set for AdaBoost used on Extremely Randomized Trees and Random Forest for different numbers of minimum samples per leaf

11.3 Ridge Regression

As Ridge Regression is a relatively simple algorithm it was implemented in python instead of using another library. At first, different numbers of features were tried.

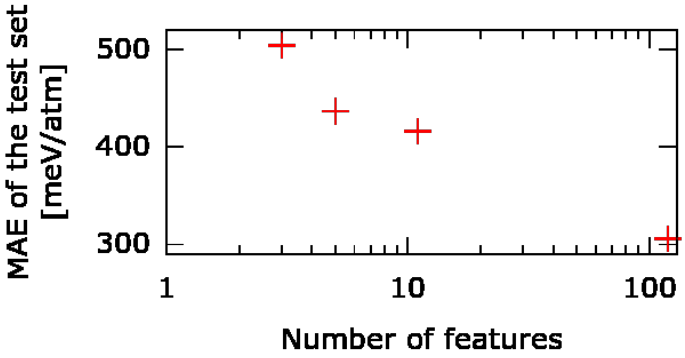


Figure 8. MAE of the test set for different numbers of features, for 11, 5 and 3 features the same features as in table 1,2 and 3 were chosen.

As one can see in Figure 2 the error decreases with a rising number of features. This is simply because every new feature is another coefficient for the linear regression enabling a more complicated model. The regularization parameter λ was also optimized but it was more or less irrelevant as long as it was not zero. When λ was set equal to zero the matrix was not invertible which proves that at least two features were linearly dependent. The MAE for the training set (304.12 ± 1.4 meV/atom) and the test set (306.01 ± 0.3 meV/atom) are almost the same therefore overfitting was not a problem. However, the high MAE of the training set implies that this is a high bias problem. This can be counteracted by using a more complex model e.g using higher order polynomial regression instead of a linear regression. When

using linear and squared feature terms the MAE of the training set decreases to 282.72±1.7 meV/atom and the error of the test set decreases to 285.75±0.6 meV/atom. If one tries terms of higher polynomial order the regularization parameter λ has to be raised drastically or else the matrix becomes non-invertible or the error diverges. Unfortunately higher polynomial orders do not result in lower errors than the regression with first and second order terms. Therefore, in the following ridge regression with first and second order polynomial terms is used.

11.4 Neural Networks

All neural networks discussed in this thesis were built using the python interface of the library caffe [12]. The same data as for the other machine learning applications was used but the feature vectors were normalized meaning that all their feature values were transformed to the interval $[-1, 1]$. The distance to the convex hull was normalized to a range of -0.5 to 0.5.

The final neural network that was used starts with a data layer which is a 33-dimensional feature vector. The data layer was connected to an inner-product layer which executes a matrix multiplication with a weight matrix $W_1 \in R^{20\times 33}$. The resulting 20-dimensional vector was input into a leaky ReLU activation function. This process was repeated two more times ($W_2 \in R^{64\times 20}$, $W_3 \in R^{7\times 64}$). The 4th inner product layer combines 7 neurons into one value. A hyperbolic tangent is used to calculate the distance to the convex hull from this value. The hyperbolic tangent is used in this case because it is zero-centered and can, therefore, output both the positive and the negative values needed for the prediction of the distance to the convex hull. The neural network was trained using a squared error loss function and stochastic gradient descent.

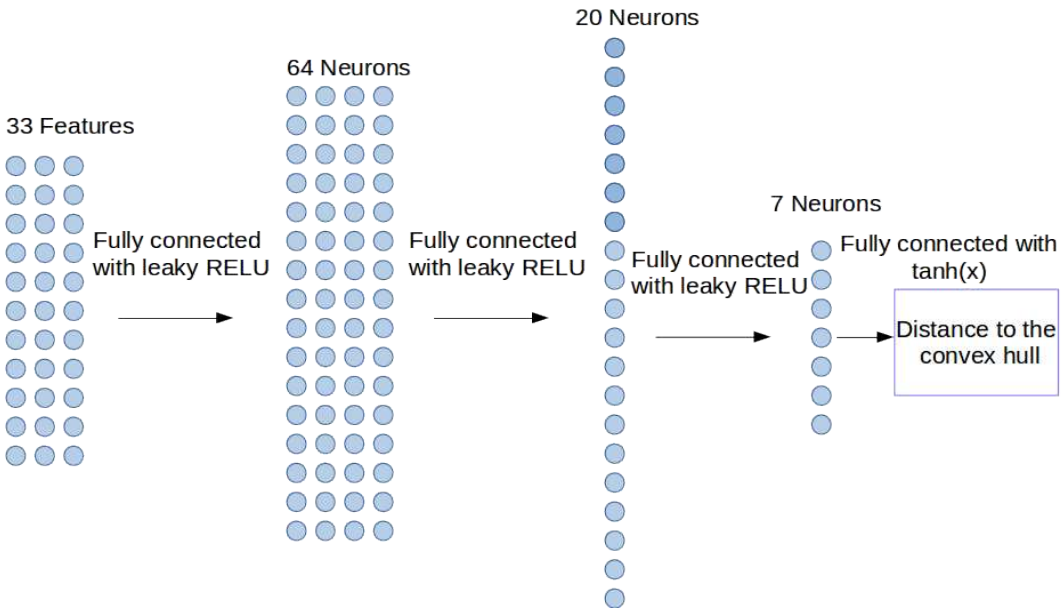


Figure 9. Architecture of the Neural Network

In order to be able to train the neural network optimally several hyperparameters had to be fitted. First, different learning rates and different number of epochs for each learning rate were tried out. The learning rate was gradually reduced from 0.1

to 0.00081.

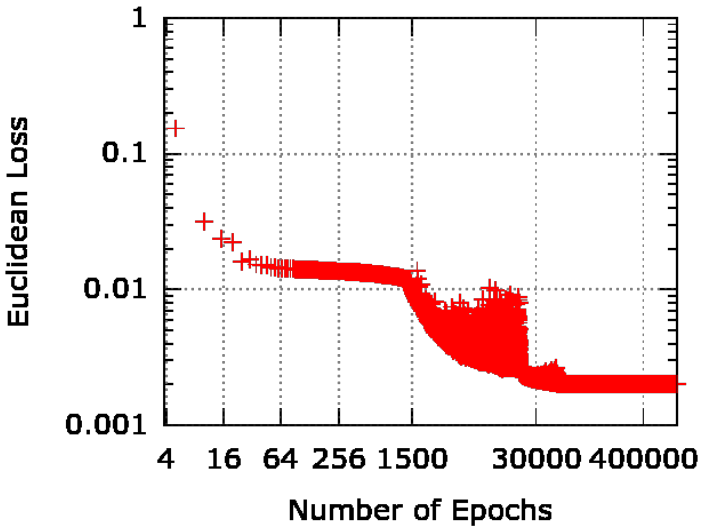


Figure 10. The Euclidean loss plotted over the number of epochs for a base learning rate of 0.1 which is gradually reduced by a factor of 0.3

In Fig 10 the learning curve for the base learning rate 0.1 is plotted. This learning rate is reduced by a factor of 0.3 after a certain number of epochs. As one can see, for each learning rate the error converges to a new smaller level after a relatively small number of epochs and than alternates around this level until a new learning rate is introduced. Once a new learning rate is introduced the error decreases drastically. The error converges at every learning rate because once the error is small enough the weight changes Δw_{ij} are too large, so the gradient constantly changes direction but the error stays close to constant. The following neural networks were trained using these results.

As second hyperparameter the regularization parameter was optimized.

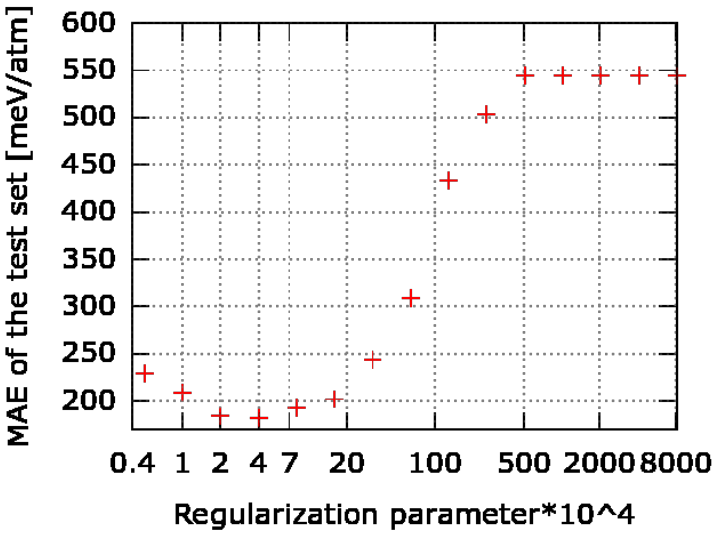


Figure 11. The MAE of the test set is plotted for neural networks trained with different regularization parameters

Neural networks were trained with 15 different values for λ:

$$\lambda = 0.0005 * (2^i), i \in \{0, 1, ..., 14\}$$

In figure 11 the MAE of the test for the different neural networks is plotted against λ and one can find a minimum at $\lambda = 0.004$. For the moment the default value of 0.9 was used.

12 Comparison of the models

12.1 MAEs

Table 3 shows the MAEs and their standard deviations for all machine learning models averaged over twenty training and testing runs with random training and testing sets. The hyperparameters of the algorithms were all set according to the results of chapter 11.

MAE	±	SD	Machine Learning Modell
285.8		0.6	Ridge Regression
167.4		4.9	Neural Network
139.4		0.7	RF 33 Features
137.2		0.8	RF 139 Features
127.2		0.6	AdaBoost/RF 33 Features
125.8		0.6	AdaBoost/RF 139 Features
123.3		0.7	ERT 139 Features
123.2		0.6	ERT 33 Features
122.3		0.7	AdaBoost/ERT 139 Features
121.7		0.6	AdaBoost/ERT 33 Features

Table 3. The MAEs and standard deviations in meV/atom for all tested machine learning models averaged over 20 training and testing runs, sorted from largest to smallest error

As expected Ridge Regression produces by far the worst predictions, because the model is just not complex enough.

Neural Networks ended up in second place from behind, although better results were expected. There are several possible explanations for this. First of all, only a very basic Neural Network structure was used and only a few topologies were tried. Therefore it is entirely possible that the results can be improved upon by changing the structure of the Neural Network. Furthermore training Neural Networks for regression is not an easy task and with enough time and knowledge one might be

able to improve the training procedure as well. At last the training set was relatively small compared to the size of the Neural Network. Therefore if the size of the dataset is increased by at least one order of magnitude the Neural Network might do better in comparison to decision tree algorithms.

When comparing the decision tree algorithms, Extremely Randomized Tree Ensembles consistently do better than Random Forests. Extremely Randomized Tree Ensembles are favored due to the high variance nature of the problem, as in comparison to Random Forests they are better in reducing variance [9]. In this case, they also had a far lower training errors (Table 1). Introducing Adaptive Boosting decreases the MAE of the random forests by around 10 percent, while only bringing slight improvements for extremely randomized trees. This implies that further variance reduction measures will most likely not bring any more improvements for Extremely Randomized Trees.

After fitting all the hyperparameters, the differences of the MAEs for different numbers of features are smaller than three SDs for random forest and smaller than one SD for extremely randomized trees, therefore we can regard them as basically equal. From this result one can guess that the further information in the dataset with 139 features either was already included through correlations in the 33 feature dataset or was useless for the task in question.

The MAE can be decreased by increasing the size of the training set.

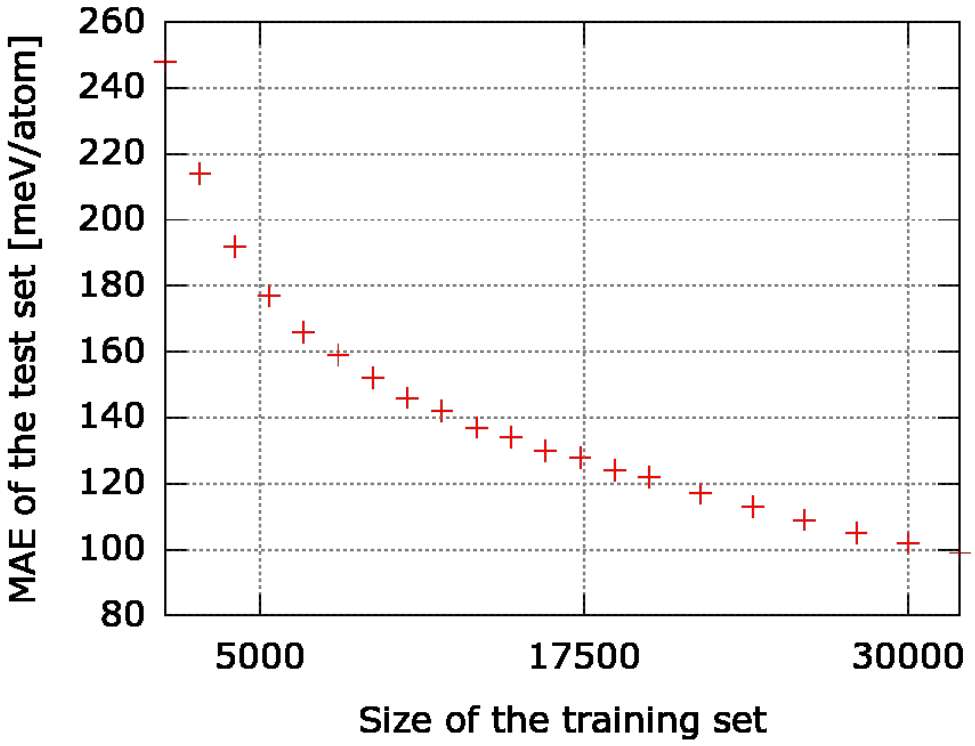
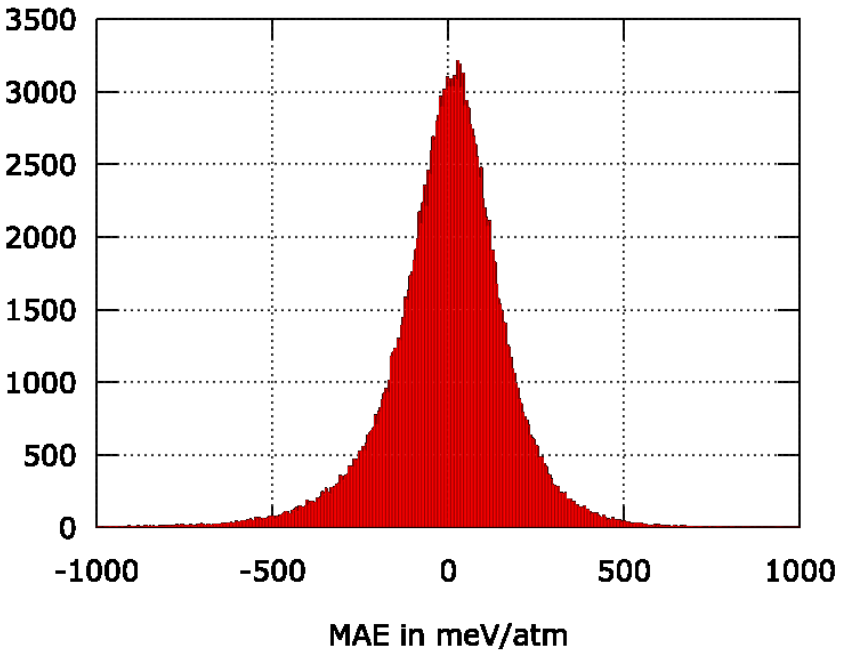


Figure 12. MAE of the test set for AdaBoost used on ERT in meV/atom plotted over the size of the training set.

Depending on which is more important a small training set or a small MAE one should change the training set size.

As before all the MAEs are calculated in comparison to the DFT results, therefore one does not know the actual error of the machine learning with respect to experimental results. Unfortunately one cannot assume that the errors are uncorrelated as the training of the machine learning models is based on the DFT predictions, therefore it is hard to give a correct estimate of the actual error.

On top of this the error for the different algorithms is not distributed in a perfect Gaussian.



As

Figure 13. Histogram of the error for Extremely Randomized Trees

But as the MAE of the DFT predictions is estimated to be around 250 $\frac{\text{meV}}{\text{atom}}$ the actual MAE of the best machine learning results (training set size of 20000) could be estimated as $\sqrt{250^2+(122\frac{\text{meV}}{\text{atom}})^2} = 278$ meV/atom and one can safely assume that 250+122 = 372 meV/atom is an upper boundary for the MAE.

12.2 Stability prediction

As already discussed in 13.1 the actual MAEs for even the best machine learning methods researched in this thesis are still higher than the MAE for DFT or at least we have to assume that they could be. Unfortunately, we do not know how these

two errors add up, therefore using solely machine learning algorithms does not seem feasible.

However, machine learning methods can still be incredibly helpful in decreasing the computation time of global structure prediction methods by limiting the number of compounds one has to research with DFT.

In order to research a new material class for stability one first has to use DFT to get a training set anyway. Afterwards one trains a regressor on the DFT dataset and uses the trained regressor to predict the distance to the convex hull for all other compounds in the material class. As discussed before these results are not good enough to directly to decide on the stability of the materials but one can do a preselection of the compounds that are worth to further research with DFT.

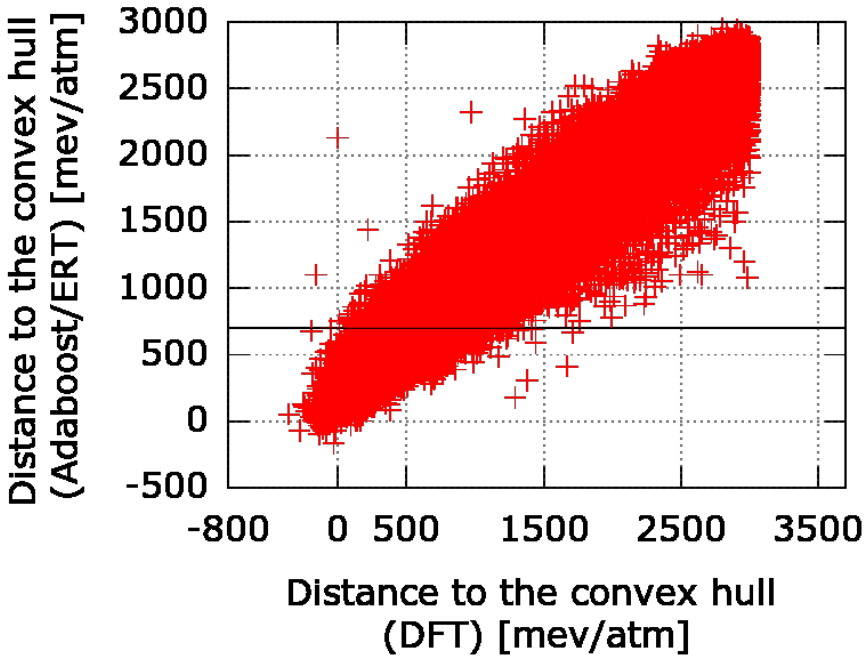


Figure 14. In this figure the distances to the convex hull predicted with AdaBoost/ERT are plotted over the distances to the convex hull predicted by DFT. By dividing the data at 699 meV/atom the set below the cutoff point includes 99.6% of the compounds that are stable according to DFT

In order to do this one decides on a cutoff point for the distance to the convex hull and uses DFT to calculate the stability of the compounds below the cutoff point.

One example for this can be seen in figure 14, were the cutoff point is set at 699 meV/atom in which case 99.6% of the compounds that are stable according to DFT (distance to the convex hull $<$10 meV/atom) are included in the set below the cutoff point. Unfortunately, around 40000 compounds are included in this dataset, which is obviously not ideal.

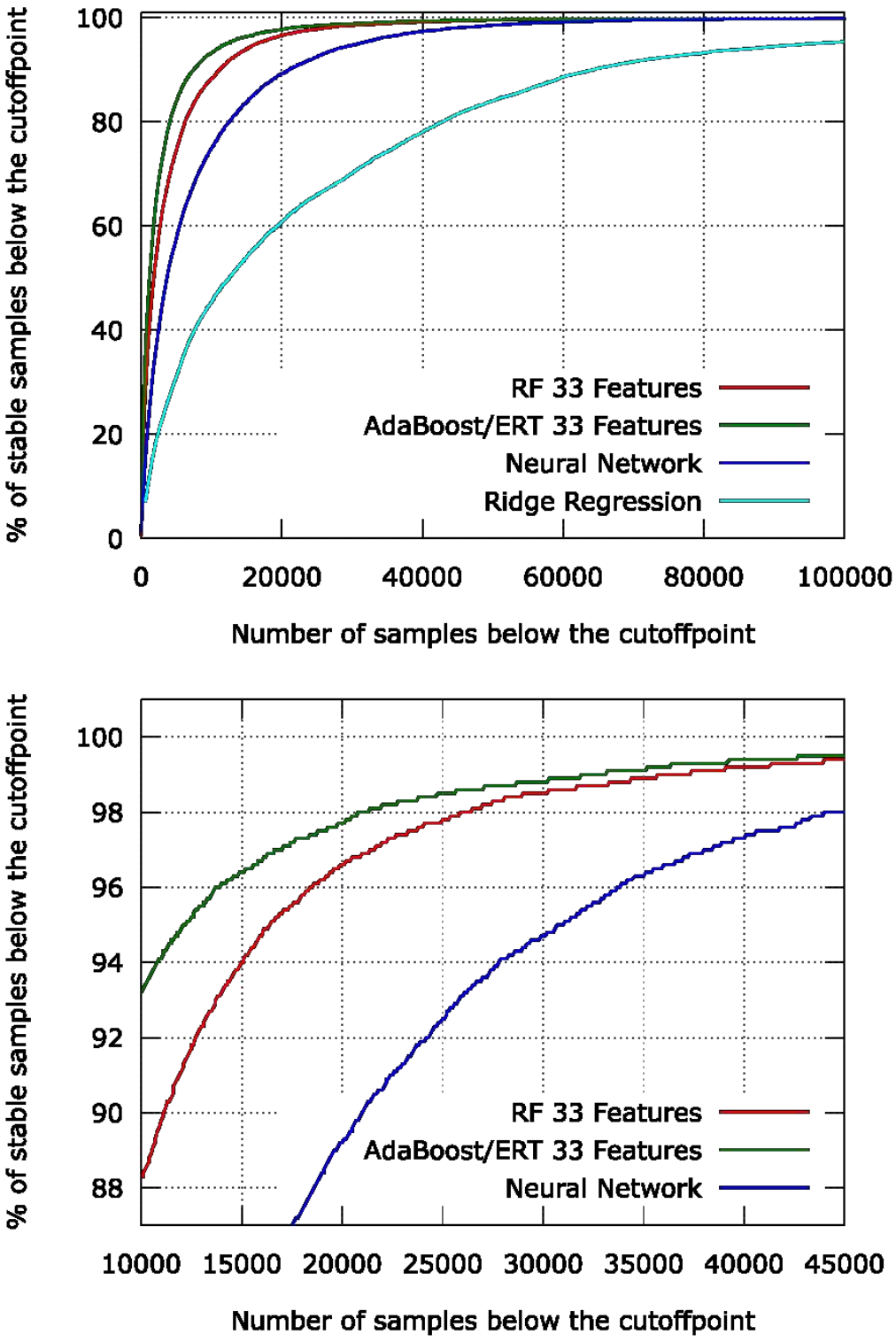


Figure 15. The percentage of samples that are potentially stable according to DFT plotted against the number of samples that have to be researched again with DFT in order to find them. The percentages are averaged over twenty runs with random training/ test sets

As expected Ridge Regression yields by far the worst results and only finds all stable samples if basically the whole test set is below the cutoff point. Also as expected the Neural Network does not do as well as the decision tree algorithms. The Random Forests algorithm is worse than the other decision tree algorithms and AdaBoost mostly does better than the sole algorithms. The decision tree algorithms have a difficult time reaching 100 percent because one outlier (NRhMn) with a distance to the convex hull of -0.0844 meV/atom consistently gets predicted with a distance to the convex hull of around 2300 meV/atom by all of them. Extremely randomized trees in combination with adaptive boosting produced the best results for percentages below 97 percent. Above 97 percent Extremely Randomized Tree Ensembles produced the best results.

In order to find 99.5 percent of the compounds potentially stable according to DFT one has to research another 41000 compounds with DFT, for 99% one only has to research 30000 compounds and in order to find 95% 13000 compounds are enough. Depending on the accuracy one wants to achieve one might want to change the size of the training set to increase or decrease the MAE in order to minimize the total number of DFT calculations:

$$n_{\text{total}} = \text{size}_{\text{TrainingSet}} + n_{\text{BelowCutoff}}$$

which is the sum of the size of the training set and the number of compounds below the cutoff point.

When using a training set size of 20000 one can reduce the computational effort by e.g. 73%, 78% or 85% when aiming for 99.5, 99, 95 percent. By choosing a different training set size one might be able to further optimize these numbers.

13 Correlation between the MAE and the constituents of a compound

In order to find out whether there is a correlation between the ability of the Machine Learning algorithms to predict the distance to the convex hull of a compound and the constituents of a compound the mean of the MAE for the prediction of the convex hull of a compound containing a specific element was computed for all elements. On top of this, it was researched whether the relationship between the MAE and the elements would be different if the specific element was in the third position of the formula ABX_3 but there was no difference.

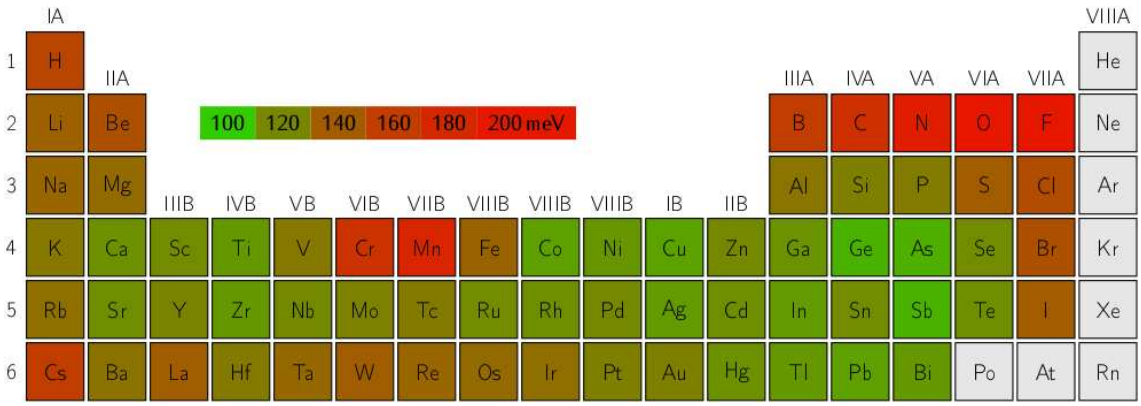


Figure 16. The MAE of all compounds containing the particular element for AdaBoost used on ERT, grey elements were not included in any compound

As one can see in figure 16 the compounds containing elements from the first and second row of the periodic table have an increased error, especially the compounds with oxygen and fluor. This increased error is probably caused by the "first-row anomaly". This phenomenon describes the difficulties in predicting properties of compounds containing elements from the second row (Li to Ne)[7]. Besides the first two rows, the error is more or less constant with the exception of cesium, chromium and manganese which also cause a higher than average MAE. The higher error for cesium could possibly be explained by the DFT calculations. The pseudopotential used for cesium had some problems and therefore the DFT results for all compounds containing cesium are not as reliable, which might be the reason for the higher MAE. The higher MAEs for chromium and manganese might be due to magnetic effects. These results are qualitatively the same for Random Forests with AdaBoost and Extremely Randomized Tree Ensembles with and without AdaBoost. The standard deviations over twenty training runs are smaller than 3 percent of the MAEs. Assuming one would want to use these results to reduce the MAE one could remove the 4 elements with the highest error from the training- and test set. Afterwards one is left with 194931 compounds and the MAE of the test set is decreased to around 106 meV/atom (AdaBoost with ERT). Of course one cannot guarantee whether

all these correlations between the error of a compound and the elements in the compound hold true for other crystal structures than perovskites. But one can safely assume that at least the first-row anomaly is not singular to perovskites.

13.1

14 Conclusion

The idea behind this thesis was to evaluate the ability of different Machine Learning methods to predict the stability of perovskites. As discussed in chapter 14 it is not yet possible to directly predict the stability of perovskites using only machine learning but one can use machine learning methods to speed up global structure prediction methods by up to 85% while still beeing able to find 95% of the compounds that are potentially stable according to DFT-calculations. In this case AdaBoost used on Extremely Randomized Trees produced the best results of all the different methods used in this thesis. However, it should be noted that Extremely Randomized Trees without AdaBoost produce only slightly worse results while beeing significantly faster which can make a difference for bigger data sets.

In order to generalize the results of this thesis for different crystal structures ,the same Machine Learning methods would have to be tested on a similar dataset for another crystal structure.

Bibliography

[1] Yoshua Bengio, Patrice Simard and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994.

[2] Max Born and Robert Oppenheimer. Zur quantentheorie der molekeln. *Annalen der Physik*, 389(20):457–484, 1927.

[3] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[4] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[5] Leo Breiman, Jerome H Friedman, Richard A Olshen and Charles J Stone. Classification and regression trees belmont. 1984.

[6] Harris Drucker. Improving regressors using boosting techniques. In *ICML*, volume 97, pages 107–115. 1997.

[7] Thom H Dunning Jr, David E Woon, Jeff Leiding and Lina Chen. The first row anomaly and recoupled pair bonding in the halides of the late p-block elements. *Accounts of chemical research*, 46(2):359–368, 2012.

[8] Yoav Freund, Robert E Schapire et al. Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156. 1996.

[9] Pierre Geurts, Damien Ernst and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. Delving deep into rectifiers: surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[11] Pierre Hohenberg and Walter Kohn. Inhomogeneous electron gas. *Physical review*, 136(3B):0, 1964.

[12] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama and Trevor Darrell. Caffe: convolutional architecture for fast feature embedding. 2014.

[13] I. Kanter, Y. LeCun and S. Solla. Second-order properties of error surfaces: learning time and generalization. In R. Lippmann, J. Moody and D. Touretzky, editors, *Advances in Neural*

Information Processing Systems (NIPS 1990), volume 3. Denver, CO, April 1991. Morgan Kaufman.

[14] Andrej Karpathy. Cs231n convolutional neural networks for visual recognition. may 2016.

[15] Walter Kohn and Lu Jeu Sham. Self-consistent equations including exchange and correlation effects. *Physical review*, 140(4A):0, 1965.

[16] G. Kresse and J. Hafner. *Ab initio* molecular dynamics for liquid metals. *Phys. Rev. B*, 47:558–561, Jan 1993.

[17] Georg Kresse and Jürgen Furthmüller. Efficiency of ab-initio total energy calculations for metals and semiconductors using a plane-wave basis set. *Computational Materials Science*, 6(1):15–50, 1996.

[18] Georg Kresse and Jürgen Furthmüller. Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set. *Physical Review B*, 54(16):11169, 1996.

[19] Alex Krizhevsky, Ilya Sutskever and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105. 2012.

[20] Andrew L Maas, Awni Y Hannun and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, page 1. 2013.

[21] Andrew Y Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

[22] Shyue Ping Ong, William Davidson Richards, Anubhav Jain, Geoffroy Hautier, Michael Kocher, Shreyas Cholia, Dan Gunter, Vincent L Chevrier, Kristin A Persson and Gerbrand Ceder. Python materials genomics (pymatgen): a robust, open-source python library for materials analysis. *Computational Materials Science*, 68:314–319, 2013.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay. Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[24] John P Perdew, Stefan Kurth, C Fiolhais, F Nogueira and M Marques. Density functionals for non-relativistic coulumb systems in the new century. *Density Functionals: Theory and Applications*, :1–55, 2003.

[25] John P. Perdew, Kieron Burke and Matthias Ernzerhof. Generalized gradient approximation made simple. *Phys. Rev. Lett.*, 77:3865–3868, Oct 1996.

[26] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[27] J. Ross Quinlan. Simplifying decision trees. *International journal of man-machine studies*, 27(3):221–234, 1987.

[28] Rafael Sarmiento-Perez, Tiago FT Cerqueira, Sabine Ko<08>rbel, Silvana Botti and Miguel AL Marques. Prediction of stable nitride perovskites. *Chemistry of Materials*, 27(17):5957–5963, 2015.

[29] Peter Schwerdtfeger. Table of experimental and calculated static dipole polarizabilities for the electronic ground states of the neutral elements (in atomic units). *Ctcp. massey. ac. nz/Tablepol2014. pdf (accessed Feb. 8, 2014)*, , 2014.

[30] Phil Simon. *Too Big to Ignore: The Business Case for Big Data*, volume 72. John Wiley &

Sons, 2013.

[31] Roman Timofeev. Classification and regression trees (cart) theory and applications. 2004.

15 Appendix

15.1 List of features

Features (from all three elements):	Source:
Electronic structure	pymatgen [22]
Number of valence electrons	calculated from the electron structure that is given by pymatgen
Atomic number	pymatgen
Pauling electronegativity	pymatgen
Maximum oxidation state of the element	pymatgen
Minimum oxidation state of the element	pymatgen
Tuple of all common oxidation states	pymatgen
Atomic mass	pymatgen
Atomic radius	pymatgen
Average ionic radius	pymatgen
All ionic radii of the element	pymatgen
Row in the periodic table	pymatgen
Group in the periodic table	pymatgen
Block in the periodic table	pymatgen
Molar volume	pymatgen
Melting point	pymatgen
Boiling point	pymatgen
Polarizability	[29]
Ionization energy for one oxidation state	http://www.periodictable.com/Properties/A/IonizationEnergies.html (07.03.2016)
Number of electrons in the last s, p and d orbitals	calculated from the electron structure that is given by pymatgen
(separate features)	
Differences in Pauling electronegativity between the elements	Calculated from the Electronegativities given by pymatgen

Declaration of academic integrity

I hereby confirm that this is the result of my own independent scholarly work, and that in all cases material from the work of others is acknowledged, and quotations and paraphases are clearly indicated. No material other than the listed has been used. This written work has not previously or not yet been published.

Name, Date