Predicting the Critical Temperature for known Superconductors with Machine Learning

Claxton JB., Materne L., Steinnes L.

Institute of Physics, University of Oslo

j.b.claxton@fys.uio.no, lukasmat@student.matnat.uio.no, lassst@student.matnat.iuo.no

December 13, 2019

Machine learning models: Support Vector Machines, Adaboosting, XGBoost and Linear Regression are applied to the chemical properties of the elemental make up of superconductors in order to predict the critical temperature. The $\sigma(T_c)$ found is ± 9.25 and R2 score 0.93. This is an improvement on previous papers modelling the same data.

I Introduction

Currently fossil fuels play a big role in our global economy, but will soon be finished. As we come to rely more and more on renewable sources of energy how will we transport energy efficiently. Society has an ever increasing desire for more energy, however the current grid cannot sustain this. The voltage and current that can be passed through overhead cables and pylons is limited; also approximately 10% of power travelling through the cables is dissipated as heat. Liquid hydrogen and superconductors offers a brilliant solution. Burning hydrogen is clean and produces no CO_2, the only by-product is water. A pipe made of superconducting material transporting liquid hydrogen would transport electricity without energy loss and transport fuel, the liquid hydrogen. Therefore, superconductors are essential for creating future smart grids. This paper looks at known superconductors with the aim of predicting their critical temperatures.

The theory of superconductivity is described briefly with an explanation to which features are likely to be most relevant for superconductivity. Next the methods of Adaboost and Support Vector Machines (SVM) are presented, although more methods are implemented in the report. Thirdly, the dataset and the pre-processing performed on the data is explained. This report creates a benchmark by making predictions with linear regression and additionally uses bayesian optimisation, for finding the optimal hyperparameters for model training (Appendix VIII.I). The results of linear regression, Adaboost, SVM and XGBoost are presented and discussed.

II Theory

II.I Superconductors

A superconductor is a material which when cooled below the critical temperature, will conduct electricity without resistance and expel magnetic flux (usually critical temperature $T_c \leq 20$ K), $\mathrm{T_c}$ Thus, electron currents can flow freely with no dissipation of energy.

The property of resistance free currents was first discovered by Heike Kamerlingh Onnes in 1911 from experiments with mercury (Hg), earning him a Nobel Prize. In the following years a lot of research has focused on superconductors and their attributes.

In 1933, the Meissner effect was invented. In the superconducting state, a superconductor expels magnetic flux, behaving diamagnetically. From this, one can categorize superconductors into two types. Type I superconductors revert to a normal conductor in presence of a weak magnetic field above a certain threshold, $\mathrm{H_c}$. In contrast, type II superconductors have two critical magnetic fields, H_{c1} and H_{c2}. Upon reaching the first critical magnetic field, the normal and superconducting states co-exist; flux penetrates the superconductor in fluxoids. Above the seconds critical magnetic field (H_{c1}) the superconductor becomes normal.

Superconductivity is a phenomenon which cannot be explained by perfect conductivity. Electrical resistance occurs due to impurities, grain boundaries and lattice vibration, which scatter conduction electrons. In principle, a perfect conductor would be without these properties and conduct electricity without resistance. If a magnetic field is applied to a material and subsequently became a perfect conductor, any applied magnetic field would be screened by surface currents and the magnetic field within the perfect conductor is unchanged. In another scenario if material becomes perfect conducting in zero magnetic field, the perfect conductor would again oppose any change in magnetic flux to keep the flux inside equal to zero. This explains the fascinating properties of superconductivity: no matter the history of the material it will always expel magnetic flux.

So far, theory only explain low temperature superconductors well, while high temperature ones ($80 \leq \mathrm{H_c}$ remains unaccounted for. John Bardeen, Leon Cooper and John Schrieffer laid the groundwork for today's understanding of low temperature superconductors, and were later awarded with the Nobel Prize in 1972 for the BCS-theory. In short, they hypothesised that conducting electrons in a superconductor form "Cooper pairs". Low-energy interactions with a negatively charged electron makes the positive ion-lattice move on a microscopic scale, attracting another electron in such a way that electrons meet no resistance. At high temperatures, this interaction is broken by thermal energy.

As a consequence of superconductors' unique properties, new technological advancement and discoveries can be made. Superconductors have great potential for usage within energy-storage, computers, particle accelerators, motors, high sensitivity sensors and more.

III Methods

III.I Adaboost

Adaboost is an iterative method using an ensemble of weak regressors or weak classifier. After each iteration the fitting is an improvement of the previous iteration. This improvement comes from re-weighting the training data based upon the mistakes of the weak-regressor or weak-classifier; larger weights are given to mis-classified data, or bigger MSE for the regression case.

In this report, the Adaboost is implemented following the explanations from H. Drucker [2], which is a modification of `Adaboost.R` from Y. Freund and R. Schapire [3]. The Adaboost is implemented as follows:

In the function `AdaBoost.training`, firstly the weights are initialised equal to 1 for each sample.

$$w_i = 1 \quad and \quad i = 1, 2, ..., N$$

The probability of each sample in the training data is:

$$p_i = \frac{w_i}{\sum_{i=1}^{N} w_i}$$

The training data is then fitted to a decision tree. The weights are passed to the sklearn decision tree function using the parameter `sample_weights`. A prediction is then made using the (weighted) decision tree on the test sample. The training loss is found from a particular loss function: linear, square law or exponential. For example in the square loss function:

$$L_i = \frac{(y_i^p - y_i)^2}{argmax(y_i^p - y_i)^2}$$

where w_i^p is the prediction and y_i is the training data. The average loss $\bar{L}$ is calculated by multiplying the individual losses with their respective probability and making a summation.

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

The parameter β is defined as:

$$\beta = \frac{\bar{L}}{1 - \bar{L}}$$

The parameter α is defined as:

$$\alpha = log \frac{1.0}{\beta}$$

which will be used later in making the ensemble prediction.

The weights are then updated using β and the loss L_i.

$$w_i = w_i \beta^{1-L_i}$$

The current iteration is now finished and the weights w_i are fed into the next iteration. The predictions, decision trees, α and β are stored after each iteration.

Once all the iterations are completed, the function `AdaBoost.evaluate` is run. This function calculates the ensemble prediction. Firstly all the predictions are stacked together in a matrix.

$$\begin{vmatrix}
y_0^1 & y_0^2 & y_0^3 & y_0^4 & y_0^5 & y_0^6 & \cdots & y_0^M \\
y_1^1 & y_1^2 & y_1^3 & y_1^4 & y_1^5 & y_1^6 & \cdots & y_1^M \\
y_2^1 & y_2^2 & y_2^3 & y_2^4 & y_2^5 & y_2^6 & \cdots & y_2^M \\
y_3^1 & y_3^2 & y_3^3 & y_3^4 & y_3^5 & y_3^6 & \cdots & y_3^M \\
y_4^1 & y_4^2 & y_4^3 & y_4^4 & y_4^5 & y_4^6 & \cdots & y_4^M \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
y_N^1 & y_N^2 & y_N^3 & y_N^4 & y_N^5 & y_N^6 & \cdots & y_N^M
\end{vmatrix}$$

where N is the number of samples in the prediction and M is the number of iterations. Each column represents the predictions from one iteration of the Adaboost.

Next, the predictions are put in ascending order. For example, the prediction of the 0th sample from iteration 5 (y_0^5) could have a value less than the prediction from iteration 1 (y_0^1) and hence need to be ordered. An example is shown below if there are 5 samples and 6 iterations of the adaboost. In reality within the code, it is a matrix of the indices which are put in ascending order.

$$\begin{vmatrix} y_0^5 & < y_0^6 & < y_0^2 & < y_0^3 & < y_0^4 & < y_0^1 \\ y_1^2 & < y_1^6 & < y_1^5 & < y_1^3 & < y_1^4 & < y_1^1 \\ y_2^1 & < y_2^6 & < y_2^2 & < y_2^3 & < y_2^5 & < y_2^4 \\ y_3^5 & < y_3^1 & < y_3^2 & < y_3^3 & < y_3^4 & < y_3^6 \\ y_4^4 & < y_4^5 & < y_4^3 & < y_4^1 & < y_4^2 & < y_4^6 \end{vmatrix}$$

Using the matrix of indices from above the parameters α are ordered, leading to a matrix as follows:

$$\begin{vmatrix} \alpha^5 & \alpha^6 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^1 \\ \alpha^2 & \alpha^6 & \alpha^5 & \alpha^3 & \alpha^4 & \alpha^1 \\ \alpha^1 & \alpha^6 & \alpha^2 & \alpha^3 & \alpha^5 & \alpha^4 \\ \alpha^5 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^6 \\ \alpha^4 & \alpha^5 & \alpha^3 & \alpha^1 & \alpha^2 & \alpha^6 \end{vmatrix}$$

Next, a cumulative matrix is created of the one above. Hence, the last element in each row is the sum of all α. The index of the median value in each row indicates the index of the ensemble prediction. In the first row for example, if $\alpha^5 + \alpha^6 + \alpha^2$ is equal or greater than the median, then index 2 (y_0^2) is the ensemble prediction for the 0th sample. Doing the same for each sample results in the ensemble prediction for the data set.

It should be noted that literature on Adaboosting for the regression case are not well written. The authors of this report found difficulty understanding content from [2] and [3] due to the poor and sometimes complicated explanations.

III.II XGBoost

XGBoost like Adaboost is an iterative method which improves at each iteration of the model. The paper by T. Chen and C. Guestrin introduces the XGBoost [1].

III.III Support-Vector Machines

Data points in a p-dimensional space that can be separated by a surface in the p-1 dimensional affine subspace - a hyperplane - are well suited for statistical learning using support vector machines (SVMs). SVMs are widely used for classification, but are also applicable to solving regression problems, which uses similar properties as the classifier.

For a matrix **X** with dimension n $\times$ p, where n is the number of observations and p the number of features, the hyperplane of row vector $\mathbf{x}_i$ is

$$f(\mathbf{x}_i) = \mathbf{w}^T\mathbf{x}_i + b = 0. \tag{1}$$

In the above equation **w** represent the weights and b is the intercept. Any vector $\mathbf{x}_i$ that does not satisfy the above expression either lies above or below the hyperplane. In problems of two classes, f(**x**) > 0 and f(**x**) < 0 can be used to separate datapoints into their respective category.

Any two points on the hyperplane must satisfy

$$\mathbf{w}^T(\mathbf{x}_i - \mathbf{x}_j) = 0, \tag{2}$$

thus the distance to a point **x** from the hyperplane will be

$$\delta = \frac{1}{||\mathbf{w}||}(\mathbf{w}^T\mathbf{x} + b) = 0. \tag{3}$$

Instead of defining a cost-function with a set of all misclassified points and doing a gradient descent optimization, the standard way of optimizing SVMs is through defining a margin M, and maximizing the margin distance using Lagrangian multipliers, λ.

The margin must satisfy the condition

$$\frac{1}{||\mathbf{w}||}y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq M \; \forall i = 1, 2, ..., n. \tag{4}$$

By scaling $||\mathbf{w}|| = 1/M$, one can find the maximum M by minimizing $||\mathbf{w}|| = \mathbf{w}^T\mathbf{w}$ while upholding 4. To find an extremal value a function f (df = 0) which variables are under constraints ϕ_k, the following equation must hold

$$\frac{\partial f}{\partial x_i} + \sum_k \lambda_k \frac{\partial \phi_k}{\partial x_i} = 0 \tag{5}$$

The Lagrangian loss-function is then defined as

$$\mathscr{L} = \sum_i \lambda_i - \frac{1}{2}\sum_{ij}^{n} \lambda_i \lambda_j y_i y_j \mathbf{x_i}^T \mathbf{x_j}, \tag{6}$$

when $\lambda_i \geq 0$ and $\sum_i \lambda_i y_i = 0$. In addition, the Karush-Kuhn-Tucker condition must be met, as it generalizes the Lagrangian multipliers method to include inequality constraints,

$$\lambda_i(y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1) \ \forall i \tag{7}$$

If $\lambda_i > 0$, then $y_i(\mathbf{w}^T \mathbf{x}_i + b) = 1$ and the vector $\mathbf{x}_i$ is on the boundary, being a support vector defining the margin M. When $\mathbf{x}_i$ is not on the boundary, $\lambda_i = 0$. Minimizing 6 with respect to λ, the problem reduces to

$$min_\lambda \frac{1}{2}\lambda^T \mathbf{P}\lambda + \mathbf{q}^T\lambda, \tag{8}$$

subject to

$$\mathbf{G}\lambda \preceq \mathbf{h} \wedge \mathbf{A}\lambda = f. \tag{9}$$

In the above equations $P_{ij} = y_i y_j K(\mathbf{x}_i \mathbf{x}_j)$, where the kernel (K) is the transformation for non-linear classification (i.e. basis functions), and **G** is a matrix collecting all inequalities, while **A** describes the equality constraints. To read more in detail about this convex optimalization problem, see Jensen's notes on the subject, or (insert some other source eg. Hastie). Finding λ_i under these constraints results in the following weights and intercept of the hyperplane

$$\mathbf{w} = \sum_i \lambda_i y_i \mathbf{x}_i \tag{10}$$

$$b = \frac{1}{y_i} - \mathbf{w}^T \mathbf{x}_i. \tag{11}$$

Thus any observations $\mathbf{x}_i$ can be classified by

$$y_i = sign(\mathbf{w}^T \mathbf{x}_i + b) \tag{12}$$

While modelling the behavior of complex datasets, working with hard margins would be unpractical, since the model should be able to predict unseen data. Thus, an error-insensitive - or soft- classifier is practical in this regard. The slack variable $\xi = (\xi_1, ..., \xi_n)$ introduce an error term on the margin, so that

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1 - \xi_i, \tag{13}$$

where $\xi_i \geq 0$. The sum over all ξ_i's gives information about the total violation, and by selecting a value C, this error can be bounded. Misclassification will then occur when $\xi_i > 1$. Introducing $\boldsymbol{\xi}_\mathbf{i}$ gives a new expression for the loss-functions and modified conditions to be met (see citation).

The regression case builds on a similar logic to that of classification. For simplicity, lets say we have linear regression where $f(x) = x^T\beta + \beta_0$, which can be generalized to non-linear models using kernels. In this case, estimating β entails minimizing

$$\sum_{i=1}^{N} V(y_i - f(x_i)) + \frac{\lambda}{2}||\beta||^2, \tag{14}$$

where

$$V_\varepsilon(r) = \begin{cases} 0 & \text{if}|r| < \varepsilon \\ |r| - \varepsilon, & \text{otherwise.} \end{cases} \tag{15}$$

For positive values of parameters $\widehat{\alpha}_i$ and $\widehat{\alpha}_i^*$, the optimization problem becomes

$$\min_{\widehat{\alpha}_i, \widehat{\alpha}_i^*} \varepsilon \sum_{i=1}^{N} (\widehat{\alpha}_i + \widehat{\alpha}_i^*) - \sum_{i=1}^{N} y_i(\widehat{\alpha}_i - \widehat{\alpha}_i^*) + \frac{1}{2} \sum_{i,j=1}^{N} (\widehat{\alpha}_i - \widehat{\alpha}_i^*)(\widehat{\alpha}_j - \widehat{\alpha}_j^*) \langle x_i, x_j \rangle . \tag{16}$$

Here $\langle x_i, x_j \rangle$ is the inner product, λ is a regularization parameter and ε is a parameter of the loss function. Equation 16 is subject to the constraints

$$0 \leq \widehat{\alpha}_i,\ \widehat{\alpha}_i^* \leq 1/\lambda, \tag{17}$$

$$\sum_{i=1}^{N} (\widehat{\alpha}_i - \widehat{\alpha}_i^*) = 0, \tag{18}$$

$$\widehat{\alpha}_i \widehat{\alpha}_i^* = 0. \tag{19}$$

Under these constraints, it can be proven that the solution is

$$\widehat{\beta} = \sum_{i=1}^{N} (\widehat{\alpha}_i - \widehat{\alpha}_i^*) x_i \tag{20}$$

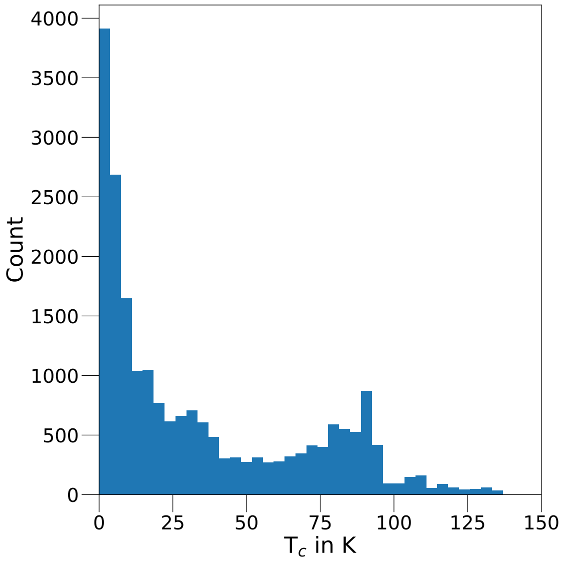


Figure 1: **Distribution of T_c**: The distribution of the target values in the regression.

$$\widehat{f}(x) = \sum_{i=1}^{N} (\widehat{\alpha}_i - \widehat{\alpha}_i^*) \langle x_i, x_j \rangle + \beta_0. \tag{21}$$

In this way, only data points with non-zero $(\widehat{\alpha}_i - \widehat{\alpha}_i^*)$ will contribute to the regression, and are therefore called support vectors.

III.IV Implementation

III.IV.1 Dataset

The data set contains 21264 samples with each 80 features. The distribution of critical temperatures is illustrated in figure 1. The main group of targets is at low temperature, i.e. moste likely type I superconductors. A smaller group of super conductors is around 80 K, which probably are type II superconductors.

The 80 features are constructed from the following eight atomic properties of the constituent elements of the superconductor: Atomic mass, first ionization energy (energy to remove one valence electron, `fie`), atomic radius, density (at standard temperature and pressure), electronic affinity (energy required to add electron to neutral element), fusion heat (or latent heat of fusion, i.e. energy to change from solid to liquid phase), thermal conductivity and the number of valence electrons.

From these features ten different averages and deviations are extracted. More details on the eight basic features and how to construct all 80 features is found in [4].

The critical temperature is separated into the target vector $\boldsymbol{y}$, while the design matrix $\boldsymbol{X}$ contains the 80 features. The design matrix is scaled such that the mean in each column is zero and the standard deviation of the column is one. This is done with `StandardScaler` from `sklearn.preprocessing`. The target values are mapped between [0,1] by setting $\tilde{\boldsymbol{y}} = \boldsymbol{y}/y_{\max}$ as new target values. The value of $y_{\max}$ is stored. This allows for a back transformation to real temperatures.
Finally, the samples are shuffled and a column of ones might be added to the design matrix at the beginning.

III.IV.2 Linear Regression

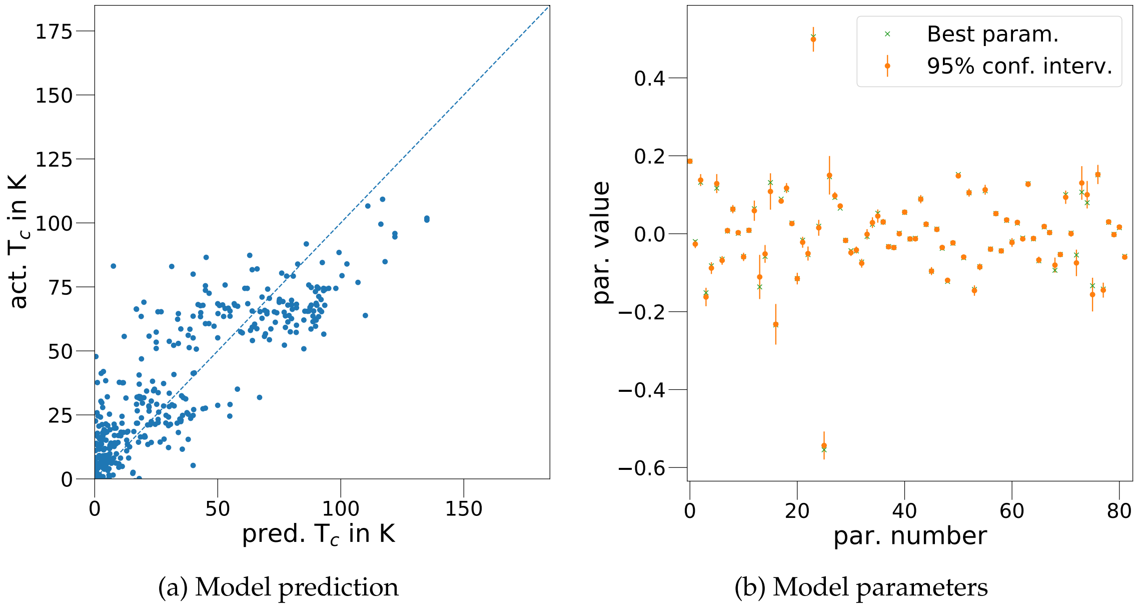
Three different types of linear regression are used: ordinary least square (OLS), Ridge and LASSO regression. The functionality of `sklearn.linear_model` is used. 10-fold cross-validation determines the best linear regression model and the average results of the regression model. Because Ridge and LASSO regression have a regularization parameter λ, a nested 10-fold-5-fold cross-validation is used. The inner (5-fold) cross-validation splits again the training data from the outer (10-fold) cross-validation. From the model's performance on the inner test data the optimal λ is picked. The model's performance are then again tested on the outer test data set (evaluation data set). These results are quoted.

In order to find the best λ two search strategies are used: A plain grid-search or a Bayesian optimization. The plain grid-search tests the model on $\lambda = n \times 10^{-m} \leq 1$, where $n = \{1,2,4,6,8\}$ and $m = \{0,1,2,3,4,5,6\}$. The Bayesian optimization searches for the optimal λ in the interval [0,1] by maximizing $-MSE$ of the model on the inner test data set. Possible λ values are sampled randomly. 50 initial guesses and 50 updates of the Bayesian model are performed, before the best λ is returned on the current fold. In order to account for the randomness of the sampling, in each fold 10 bootstraps are performed on top. More details on the Bayesian optimization is found in appendix VIII.I.

IV Results

IV.I Linear Regression

The results of the cross-validation are listed in table 1. OLS seems to give the overall best model with an R2 score of 0.755 and an uncertainty on the critical temperature $\sigma(T_c) = \sqrt{MSE} y_{\max} = \pm 16.90\,\mathrm{K}$. Contrary, the average $\sigma(T_c) = \pm(17.61 \pm 0.41)\mathrm{K}$ is the same



(a) Model prediction

(b) Model parameters

Figure 2: **Best Linear Regression Model**: From table 1 the model with the average least T_c uncertainty is picked. This is the LASSO regression with $\lambda = 10^{-6}$. The prediction from the model is plotted against the true critical temperature in figure (a) for randomly selected sample points. In figure (b) the model parameter values are shown. The three parameters with the largest absolute value are `std_FusionHeat`, `mean_fie` and `wtd_std_fie`. They are sorted from importance rank 3 to 1, where 1 is the most important feature.

as found in [4] for linear regression. The uncertainty on the average is related to the standard deviation of the MSE in different folds (with the same regularization strength if applicable). In other words, it measures the variability of the expected critical temperature uncertainty of the models prediction.

The best model will have a uncertainty on the critical temperature as small as and a minimal variability of it. Therefore, the best linear regression model is LASSO regression with $\lambda = 10^{-6}$ with $\sigma(T_c) = \pm(17.555 \pm 0.064)$K. The best LASSO regression has then an R2-score of 0.7373 and $\sigma(T_c) = 17.436\,$K. The prediction of the model and the respective parameters are shown in figure 2.

Grid-search and Bayesian optimization based search give not the same results as can seen in table 1, but the MSE curve based on both searches coincide as figure 3 demonstrates. The Bayesian optimization search is more nosy due to the random nature of the sampling of λ but also more centered around a region of minimal MSE.

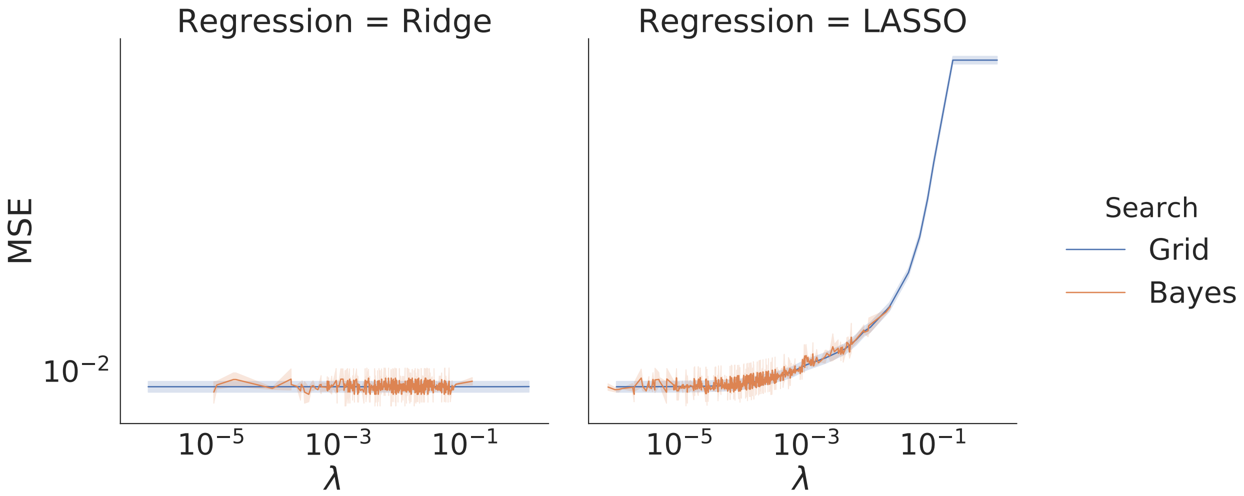


Figure 3: **Regularization Parameter Search**: Compared are the two search strategies which are used to find the best regularization parameter λ in Ridge and LASSO regression.

Table 1: **Results of Linear Regression**: The results of 10-fold cross-validation are presented. The regularization parameter λ in Ridge and LASSO regression are tuned with a nested cross-validation with 10-folds and 5-folds. For Bayesian optimization 10 bootstraps are applied within each nested cross-validation.
The model performance is measured on a part of the data set which was not used for training or λ selection.

Model	Search		λ	MSE	R2	unc. T_c in K
OLS		best		0.00835	0.755	16.90
		mean		0.00907 ± 0.00043	0.735 ± 0.011	17.61 ± 0.41
Ridge	Grid	best	1E-6	0.00888	0.74	17.45
		mean	1E-6	0.00908 ± 0.00042	0.735 ± 0.011	17.63 ± 0.40
	Bayes	best	0.031	0.00918	0.7369	17.72
		mean	0.015 ± 0.015	0.00908 ± 0.00041	0.7351 ± 0.0073	17.63 ± 0.23
LASSO	Grid	best	1E-6	0.008882	0.7373	17.436
		mean	1E-6	0.009005 ± 0.000066	0.7372 ± 0.0017	17.555 ± 0.064
	Bayes	best	9.0E-6	0.0092	0.737	17.73
		mean	0.0005 ± 0.0016	0.00956 ± 0.00084	0.721 ± 0.023	18.07 ± 0.75

V Adaboosting

The results of 10-fold cross validation are shown in tab. 2. The best overall model has an R2 score of 0.93 and an uncertainty in the critical temperature $\sigma(T_c) = \sqrt{MSE} y_{max} = \pm 9.252\,\mathrm{K}$. This is a better result than in [4].

Initially it was believed that adaboosting with a weak decision tree of maxdepth = 1 (the weakest possible) would most appropriate as most sources state to start with a regressor/classifier no better than guessing. However, it was found that increasing the max depth of the decision tree to approximately 15 improved the MSE and R2 score. An example of the hyper-parameter search can be seen in Fig. 6. The predicted critical temperature as a function of the actual critical temperature is plotted in Fig. 4. The same hyper-parameters are used to fit using the sklearn Adaboost (Fig. 5).

Table 2: **Results of Adaboost**: The results of 10-fold cross-validation are presented. The results in the table are from out-of-sample set which is unused in the parameter search. The sklearn implementation was not run with a GridSearch for best hyper-parameters; the hyper-parameters from the own implementation are used and then sklearn predicts using these hyper-parameters.

Model	Depth	Loss function	no. iterations	MSE	rMSE	R2
Own Adaboost	15	exponential	50	85.595	9.252	0.93
Sklearn Adaboost	15	exponential	50	98.568	9.928	0.92

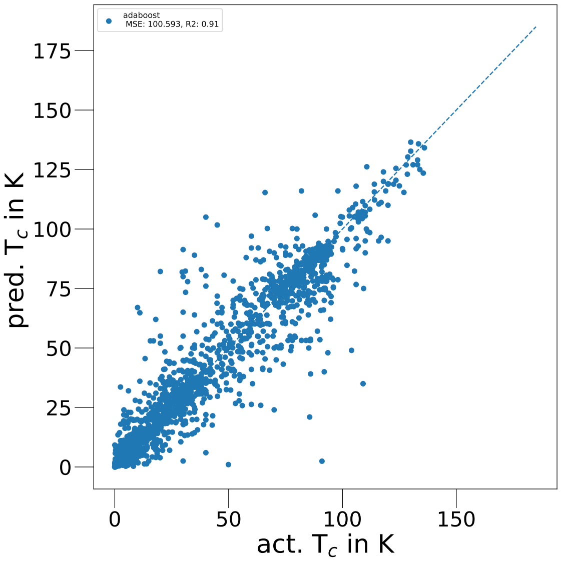


Figure 4: **Own Adaboost:** Predicted critical temperature as a function of actual critical temperature. A line y = x is plotted to show ideal result. Prediction is on out-of-sample data. The mse is 100.59 and r2 score 0.91.

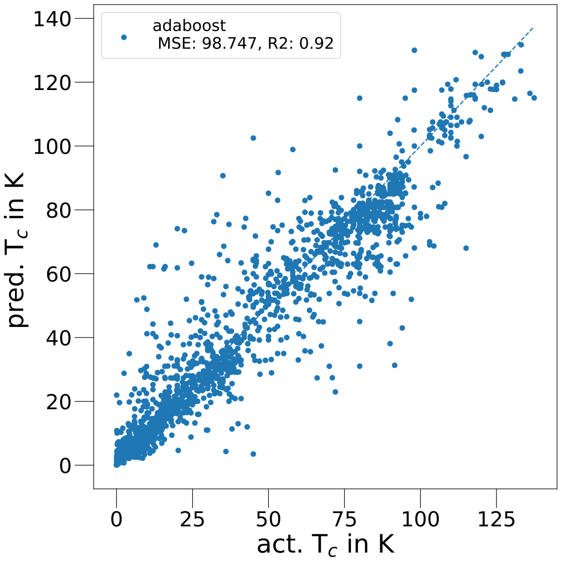


Figure 5: **Sklearn Adaboost:** Predicted critical temperature as a function of actual critical temperature.

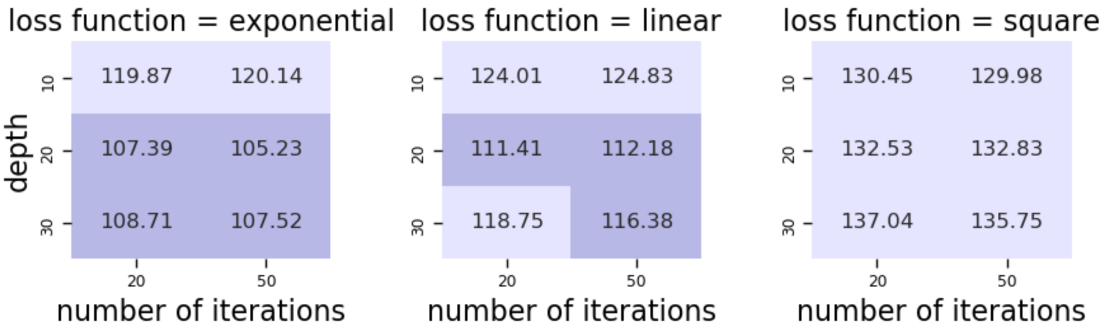


Figure 6: **Hyper-parameter Search:** Grid of heatmaps showing the test MSE of a parameter search. Results are 10 fold cross-validated. The lowest test MSE (105.23) in the figure results from the exponential loss function, a depth of 20 nodes and 50 iterations.

VI Discussion

VI.I Bayesian Optimization

The usage of Bayesian optimization to tune the regularization parameter λ in Ridge and LASSO regression is not paying of. The grid search gives faster and better results. Figure 3 might give a hint why this is. The grid search (blue) gives a smooth curve of $MSE(\lambda)$. This is due to the interpolation between the sampled grid points, which have a stepsize large enough to not resolve the full detail of $MSE(\lambda)$.
The sampling from the Bayesian optimization reveals the rugged structure of $MSE(\lambda)$. Obviously, $MSE(\lambda)$ is sensitive to small changes in λ. Such small changes lead to wildly varying MSE values. This makes it hard for the Gaussian process to capture the function $MSE(\lambda)$. Maybe with more updates, the result would be better.

The question is, if this approach to Bayesian optimization, with random sampling, initial guesses, absolute number of updates and fixed exploration parameter, can handle well such noisy functions. Maybe substituting random sampling for the initial guesses with a grid search might give the Gaussian process model a better chance modeling the average function behavior.
The number of updates could be also tied to an abortion criteria, like n-rounds of no improvement.
Last, but not least, the exploration parameter ξ could vary throughout the updating process, i.e. decaying from a large value to a smaller value. This might lead to a better exploration of the search space in the beginning and then narrowing down towards a most likely maxima.

Overall, Bayesian optimization has the potential to search higher dimensional hyperparameter spaces more efficiently than brute-force grid search. However, in addition to defining the search space, Bayesian optimization introduces at least three more hyperparameter, initial guesses, number of updates and exploration parameter. This list does not include, that the surrogate function and acquisition function itself are subject to many input parameters and options.
The user must carefully evaluate if the Bayesian optimization introduces unnecessary complexity to the hyperparameter search or if it might be a promising efficiency improvement. In the case of linear regression, a plain grid search is sufficiently fast and gives good results. Therefore, Bayesian optimization is not needed for this kind of problem.

VI.II Adaboost

The Adaboost model implemented in this report achieves an rMSE ($\sigma(T_c)$) of 9.2 compared to 9.5 in [4] using XGBoost. These results were found using 10-fold cross validation will one set held out as a final evaluation set. The achieved result of $\sigma(T_c) = \pm 9.2$ K is predicted from the evaluation set. To perform the iterative process on the evaluation set, the model which performed the best (lowest test MSE) from within the best averaging model in the 10-fold was stored. To improve on this an inner cross-validation could have been used to ensure each fold has an opportunity of being the evaluation set.

VII Conclusion

References

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[2] Harris Drucker. Improving regressors using boosting techniques. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 107–115, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[3] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*, ICML'96, pages 148–156, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[4] Kam Hamidieh. A data-driven statistical model for predicting the critical temperature of a superconductor. *Computational Materials Science*, 154:346 – 354, 2018.

[5] Jason Brownlee. How to Implement Bayesian Optimization from Scratch in Python, October 2019. Accessed: November 2019.

[6] Martin Krasser. Gaussian processes, March 2018. Accessed: November 2019.

[7] Martin Krasser. Bayesian optimization, March 2018. Accessed: November 2019.

VIII Appendix

VIII.I Bayesian Optimization

One fundamental problem with grid-search based hyperparameter optimization is, that the model to optimize is only evaluated on predefined grid points. If the search wants a higher resolution, usually the step size between suggested hyperparameter values is decreased. However, this becomes quickly computationally expensive if the model needs to be evaluated on many grid points. Moreover, uninteresting regions are also sampled at the same frequency as interesting regions of hyperparameter space. This is a waste of computation power.

One possible solution is to use Bayesian optimization techniques. The main idea is to use Bayes theorem to optimize a scalar function $f = f(d)$ given some data d

$$p(f|d) \propto p(d|f)p(f). \tag{22}$$

In Bayesian statistic terminology $p(f|d)$ is the posterior probability of a function value f given the data d, $p(d|f)$ the likelihood of the d given f and $p(f)$ is the prior probability of f [5]. With out loss of generality, it is assumed that f is to maximize.

The goal is now to estimate the posterior probability with an surrogate function S. This is often done by using Gaussian processes (GP) [6] to fit the data d, because GP's are fast to evaluate, fast to fit and give an estimation of model uncertainty. After each evaluation of the function f the GP model is updated to the new data.

With the help of an acquisition function A, a new data point is selected based on the surrogate function. This involves suggesting a number of data point candidates d_i to the surrogate function and take its prediction $\mu(d_i)$ for f and its uncertainty $\sigma(d_i)$ into the acquisition function. Then select the data point candidate which maximizes A. Then evaluate the the model on this data point and update S.

In this paper, the expected improvement [7] is used as the acquisition function

$$A = EI(d_i) = (\mu(d_i) - f_{\max} - \xi)\mathbf{\Phi}(Z(d_i)) + \sigma(d_i)\phi(Z(d_i)). \tag{23}$$

$\mathbf{\Phi}$ and ϕ are the cumulative distribution and the probability density function of a normal distribution, respectively. The value of $Z(d_i)$ is defined as

$$Z(d_i) = \frac{\mu(d_i) - f_{\max} - \xi}{\sigma(d_i) + \varepsilon}, \tag{24}$$

where $f_{\max}$ is the current maximal function value, $\varepsilon = 10^{-9}$ is a small constant chosen to avoid division by zero and ξ is the exploration parameter usually set to 0.01.
The first part of the sum in eq. (23) describes regions where the GP model predicts large values for the function f. The second part of the sum is related to regions with large GP uncertainty. ξ then regulates if regions of higher uncertainty (large ξ) or regions of high value predictions of GP model for f (small ξ).
To sum up, in algorithm 1 all the relevant steps are collected.

To illustrate the outcome (and test the implementation of the Bayesian maximization), a simple 1-dimensional optimization is performed. The used function is

$$f(x) = x^2 \sin(5\pi x)^6$$

with additional Gaussian noise ($\mu = 0,\ \sigma = 0.05$). The results of the Bayesian maximization can be seen in figure 7. 50 initial guesses and 50 updates are used. The GP model does not quiet capture the function f. However, from the histogram one can see, that most of the points at which f is evaluated are near the maximum of f. The found $x_{\max}$ is close to the true $\hat{x}_{\max}$. Differences are due to the noise on f and the random nature of the algorithm.

In order to use Bayesian maximization on hyperparameter tuning of a regression model, a few changes are made to f:

$$f \to -MSE(\boldsymbol{X}, \boldsymbol{y}|\boldsymbol{\lambda}) \tag{25}$$

In other words, the negative mean squared error MSE of the regression model $r(\boldsymbol{X}, \boldsymbol{\lambda})$, depending on the input data $\boldsymbol{X}$ and the set of hyperparameters $\boldsymbol{\lambda}$, and the target values $\boldsymbol{y}$ is maximized with respect to $\boldsymbol{\lambda}$.

Algorithm 1 Bayesian Maximization: The basic outline for a Bayesian optimization algorithm. Inputs are the function to optimize f, the search space D and the exploration parameter ξ

```
function BayesMax($f$, $D$, $\xi$)
    $\boldsymbol{d}, \boldsymbol{f}_i \leftarrow [\,], [\,]$    ▷ Initialize data $d$ and evaluated function values $f_i$
    for # init. samples do    ▷ Initial guesses
        $d_g \leftarrow \text{random}(D)$    ▷ Draw a random point from search space
        $\boldsymbol{d}, \boldsymbol{f}_i \leftarrow \boldsymbol{d}.\text{append}(d_g), \boldsymbol{f}_i.\text{append}(f(d_g))$
    end for
    $GP \leftarrow \text{GaussianProcess.fit}(d, f_i)$    ▷ Initalize GP
    $d_{\max}, f_{\max} \leftarrow \text{argmax}(\boldsymbol{f}_i), \max(\boldsymbol{f}_i)$
    for # updates do    ▷ Bayesian Optimization
        $\boldsymbol{d}_i, \boldsymbol{EI} \leftarrow [\,], [\,]$
        for # tries do    ▷ Find next point $d_g$ to evaluate $f$
            $d_t \leftarrow \text{random}(D)$
            $\mu(d_t), \sigma(d_t) \leftarrow GP(d_t)$    ▷ Evaluate GP at suggested point $d_t$
            $\boldsymbol{d}_i, \boldsymbol{EI} \leftarrow \boldsymbol{d}_i.\text{append}(d_t), \boldsymbol{EI}.\text{append}(EI(d_t))$    ▷ Use eq. (23)
        end for
        $d_g \leftarrow \text{argmax}(\boldsymbol{EI})$    ▷ Suggest point with maximal EI
        $\boldsymbol{d}, \boldsymbol{f}_i \leftarrow \boldsymbol{d}.\text{append}(d_g), \boldsymbol{f}_i.\text{append}(f(d_g))$    ▷ Evaluate $f$ at suggested point $d_g$
        $d_{\max}, f_{\max} \leftarrow \text{argmax}(\boldsymbol{f}_i), \max(\boldsymbol{f}_i)$
        $GP \leftarrow GP.\text{fit}(d, f_i)$    ▷ Update GP
    end for
    return $d_{\max}, f_{\max}$
end function
```

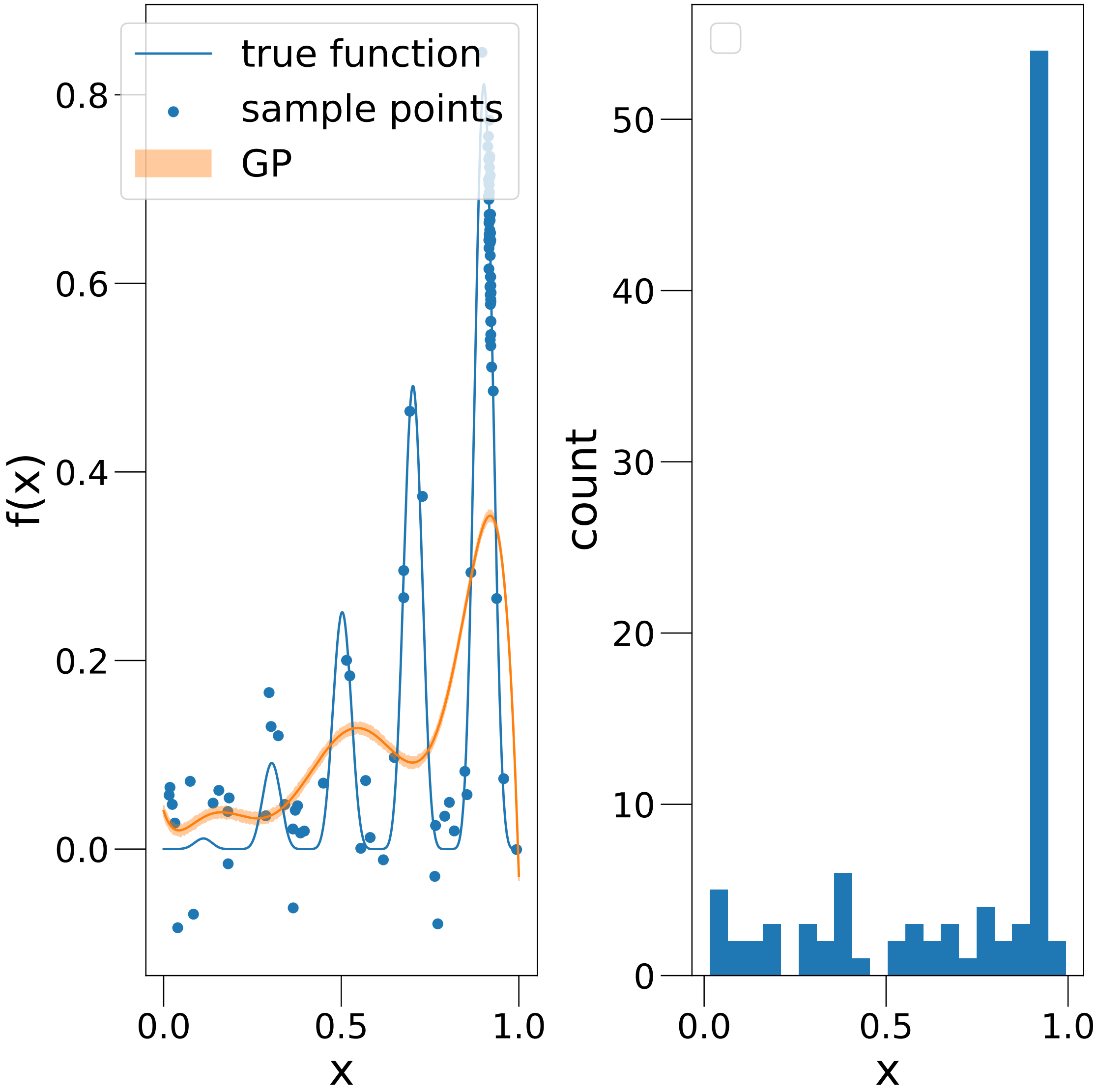


Figure 7: **1D Bayesian Maximization**: To test functionality, this 1D toy example is used. The goal is to find x which maximizes $f(x)$. The band around the curve of the Gaussian process (GP) illustrates the uncertainty of the GP model at this point.