Machine Learning In Materials Science and Engineering
FYS-STK4155 - Project 3

Brage Brevig
(Dated: December 21, 2023)

In this project, two machine learning algorithms are deployed to learn and predict the thermal lattice conductivity coefficient κ_L of inorganic materials. In one instance we use a deep neural network (DNN) to find that the model makes predictions with a mean absolute error (MAE) committing to 0.332 with a CPU time of approximately 123 seconds. This is compared to a gradient boosted decision tree algorithm which makes predictions of MAE 0.526 at a much hastier CPU time of 21 seconds. Inconclusively, we can say that depending on the intent of use each algorithm has their own pros and cons. Further trialling and inspection is needed to make better statements about the two algorithms.

I. INTRODUCTION

The lattice thermal conductivity parameter κ_L is for non-metals a material - dependent property which determines its ability to conduct heat. In turn, the ability to conduct heat determines for which application a material may or may not be devised. Materials with high κ_L are sought after in technologies such as thermal energy storage and battery temperature management [1] while materials with lower κ_L are deployed in thermoelectric generation and thermal insulation coating [1]. In light of this, material engineers are naturally motivated in their efforts to determine this parameter value during material design.

Today, high-throughput computations via first principle calculations such as Density Functional Theory are employed to create data sets describing materials and their properties. Such efforts, however great, are incredibly expensive in terms of computation and further underlines the need for fast and efficient approximation of both known and novel materials.

II. NUMERICAL METHODS, ALGORITHMS AND IMPLEMENTATIONS

In this project, machine learning algorithms have been implemented in order to train on data sets containing material features and in return make models that predict the κ_L value for a given material. It is evident from literature [2] that Gradient Boosted Decision Trees and Deep Neural Networks rank as the two most popular machine learning algorithms deployed in materials science. In this study we shall therefore implement the two methods, train them on the same data set and compare their performance.

A. Data Mining with `matminer`

An integral part of this study has been to create data sets which are suitable for training the two algorithms. Although thermal conductivity is a well studied property of materials, in literature it will often be reported in low dimensional formats containing maybe as little as the material formulae and their κ_L values. `matminer` [3] is an open source toolkit library building mainly on the `pymatgen` package for python. This library does an outstanding job at '*featurizing*' a data frame just from material formulae by mining information from each constituent element in a material's composition. In this project, the initial data set contains material formulae, structure and the target variable κ_L. After 'featurizing' the data frame with `matminer` and trimming descriptors, the final data set has 139 features. A Jupyter Notebook tutorial describing the process of making the data set(s) for this project can be found in the same GitHub destination with the rest of the project code.

Further pre-processing of the data set includes dropping columns which do not contain real valued numbers, containing it to inorganic, non-metallic materials only and scaling the target values by taking their log value and then centering them with `StandardScaler`. All data features are also scaled by `StandardScaler`. The target values are scaled by log as the original data is in units of a transport coefficient on order E10.

B. Deep Neural Networks with `Keras`

In this project, one of two implemented algorithms is a feed forward neural network with multiple hidden layers. This is considered a *deep* neural network (DNN), and is in this case implemented via the python package `Keras`. A semi-goal in this study is to tune the regressor models as the data they are training on is unordered and complex. Tuning the hyperparameters of the DNN model means in practice that several configurations of the input parameters are trialled a set number of times. The configuration for which the model makes prediction with the lowest error score is returned and saved. In the DNN model the key hyperparameters are

- `learning rate` : The learning rate determines the step size in the stochastic gradient engine. In this

study ADAM has been deployed as it has proven to be a soft and fast gradient descent method in previous studies.

- `units` : The number of nodes in each layer. For regression it is common practice that the first hidden layer consists of as many nodes as there are input nodes, and that for additional layer before the output layer the number of nodes lie between the number of nodes in the first hidden layer and output layer.

- `activation` : The activation function used in a hidden layer. In this study this parameter is limited to `relu`, `elu` and `tanh`.

Next to the highlighted parameters to the DNN model are boolean parameters such as whether or not to use dropout layers or an additional third hidden layer. Since the code searches for configurations which include up to three hidden layers, each hidden layer has its own declaration for the highlighted parameters. This leads to many possible configurations and in that opens up the possibility of describing what can be considered very non linear problems like the conductivity data. We limit the number of configuration trials in this study to 200.
The author notes that the use of dropout layers in this project is intended to reduce overfitting by dropping nodes with close to zero weights. There are several other node-trimming algorithms and similar, but that is outside the scope of this study.

C. Gradient Boosted Decision Trees with `XGBoost`

`XGBoost` (E**X**treme **G**radient **Boost**ing) is an open source code base for decision tree based machine learning algorithms. This study takes advantage of the `XGBRegressor` object, a gradient boosted decision tree regressor. As with the basic principle of decision tree algorithms, the algorithm creates a set of weak learners and builds a strong learning model from which predictions can be made. `XGBoost` was chosen as a contender in this study due to its many citations in literature, and its flexible and robustly optimized design. As with the DNN model, a 'half-way goal' in this study was to properly tune the algorithm before making predictions on unseen data and analyzing its performance. The `XGBRegressor` object takes a large amount of parameters. In this study the hyperparameter space was limited to 11 distinct parameters. Of these parameters the most important ones for general performance are

- `n_estimators` : the number of weak learners. Much like a high epoch value can 'overtrain' a DNN model, a large number of weak learners can force the strong learner to overfit the training data.

- `gamma` : This parameter controls for what error in a leaf (node) a leaf will be partitioned. A higher value causes the weak learners to be more conservative. Naturally then, this parameter is in need of tuning to find a balance between under - and overfitting the training data.

- `max_depth` : controls how large a tree can grow.

Next to the highlighted parameters are `subsample`, `learning rate` and `reg_alpha` which all control the sampling - and step sizes when growing the trees. Most importantly, this regressor object uses the `booster` called `dart` (Dropouts meet Multiple Additive Regression). In this call, the regressor performs a 'leaf pruning' by dropping weak leaves. This has proven to ensure the stability of predictions when using multiple trees.

III. RESULTS

A. Benchmarking and Code Testing

In the initial code development stage, each machine learning algorithm trained on a virtually created data set consisting of Taylor series features up to the fifth order in order to approximate the target function

$$f(x) = e^{-x^2} + 2e^{-(x-2)^2} + \epsilon \tag{1}$$

where ϵ describes random deviations in the target function f. The tuning of each model was done on the same hyperparameter space as we will employ for the models when training on conductivity data.

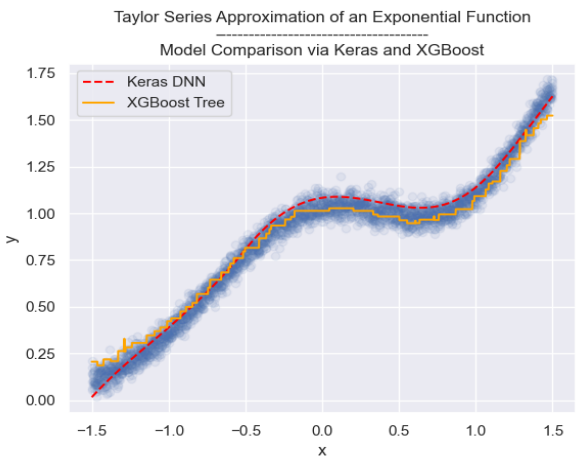

Figure 1. Linear fitting of an exponential function using Taylor series features in a DNN and a decision tree.

Figure 1 is a visual confirmation that the DNN and tree models are both functioning as intended. This figure was made running the code as it stands now with the virtual training data set.

Before moving forward, we state here that the metrics used throughout the project to assess performance and viability of the results obtained are mean absolute error (MAE) and the R2 score. MAE was chosen as the loss function used in both regression models as it is less sensitive to outliers in data sets (which the conductivity data has plentiful of), and was therefore a natural choice for the performance metric, too. The author has also implemented a function `similarity_score` which is visualized in many of the figures. This metric is quite simply computed through 1 - MAE for each predicted data point as to portray the 'accuracy', or similarity, between each true and predicted data point.

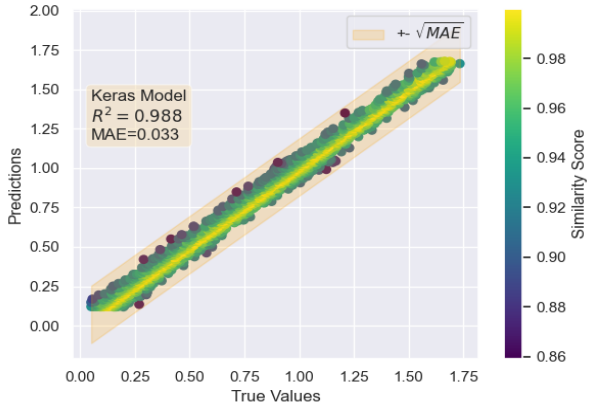

Figure 2. Model analysis of a DNN used to fit a linear function.

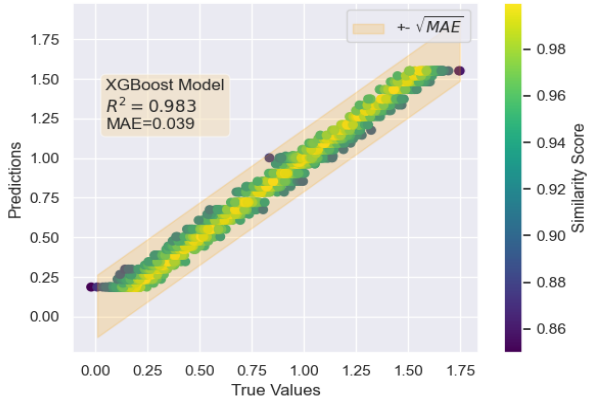

Figure 3. Model analysis of a XGBoost tree used to fit a linear function.

Figures 2 and 3 are used to analyse the performance of the two models by comparing the target function and respective predictions. The solid plane (orange hue) encapsulating the scattered points in the plots tells us what portion of the predicted data set that lies within the range $(-\sqrt{\text{MAE}}, \sqrt{\text{MAE}})$ of a perfect fit between the target - and predicted values.

B. Training and Predicting on Conductivity Data

Moving on, we shall present results obtained from training and making predictions on the conductivity data set.

Table I. Tabular overview of randomized search space and performance metrics for each model.

Model	**DNN**	**Decision Tree**
# Trials	200	200
Randomized	Yes	Yes
CV folds	0	5
Dimension	11	11
Objects	67	45
Combinatorial order	E12	E10
CPU time	123.0 s	21.0 s
MAE (best trial)	0.34	0.53
R2 (best trial)	0.67	0.44

Table I gives an overview of how the two algorithms in this project have been used, what the hyperparameter space looks like and how they perform when trained on the thermal conductivity data. As shown in the table, both algorithms have been tuned with 200 random trials, using subsets of the hyperparameter space to determine which configuration gives the best results. Seeing as the combinatorial orders of each hyperparameter space are so large, it is obvious that the searches have been done on very small partitions of said spaces. This was an intentional effort to save time during the debugging stage of the code development process. The hyperparameter tuning of the decision tree model includes cross-validation - this is to ensure stability across the trials. For the DNN model, stability in the hyperparameter tuning is ensured by having each model learn for a set amount of epochs with a stopping threshold. This applies in each trial.

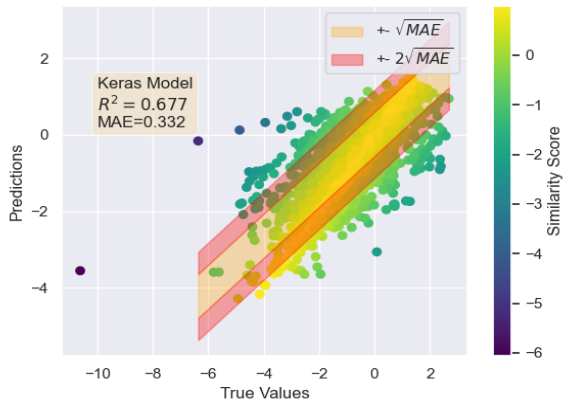

Figure 4. Model analysis of DNN used to predict thermal lattice conductivity from compositional and structural material features.

Figure 5. Model analysis of XGBoost used to predict thermal lattice conductivity from compositional and structural material features.

Figures 4 and 5 are results obtained from running the tuning codes on regressors that learn the conductivity data. Like before, the orange hued plane describes all data in the range $(-\sqrt{\text{MAE}}, \sqrt{\text{MAE}})$. The additional red plane includes all data points in the range $(-2\sqrt{\text{MAE}}, 2\sqrt{\text{MAE}})$.

IV. DISCUSSION

As shown in Figures 2 and 3, we are confident that the DNN and tree models are performing as intended. Just by visual inspection in Figure 1 it looks like the DNN model returns a generalized model that captures trends in the original data set. As expected, the tree model displays its stair - like properties as is common with decision trees. Under the same scope, this model seems to capture more of the noise in the original data set than the DNN model does, and simultaneously less of the general trend.

On further inspection in I, we start seeing some interesting results for each of the models. The DNN model is much more computationally expensive than the decision tree made via XGBoost. This is not surprising, as we are computing and updating changes in the weights across roughly 50 000 node connections. With the built-in gradient descent modules `Keras` offers, it is only reasonable that the total CPU time grows much larger than that of the time taken to make trees with XGBoost. It is worth noting here that XGBoost offers work task distribution on both multithread and cloud based services, which improves CPU time even further. `Keras` also offers multithreaded engines in their source code. However, we appreciate at this point that although the DNN model runs with a longer CPU time it also commits a lower MAE and better R2 score. As shown in Figures 4 and 3, a larger portion of the predicted data in both cases lie within the error boundaries of +- $2\sqrt{\text{MAE}}$ and in light of this commits decent accuracy given the very many degrees of freedom in the data set. Finally, we see that even though the tree model has a higher MAE compared to the DNN model, almost the entire set of predicted data lies within the said error boundaries. It also seems like there is an even distribution of data above and below the first error boundaries which indicates that the model is neither under-, nor overfitting the training data.

As this study is in its core a consideration of each models performance, we state here that XGBoost has a much lower 'barrier to entry' than Keras. Considering development time as an important factor in the total coding time (run - and development time), XGBoost quickly becomes a very serious contender to Keras and DNN models where they are suitable. As mentioned in the Methods section, XGBoost is also an 'open' model where the feature weights are extractable. In a study like this, such a property is very valuable as we are interested in retrieving intrisic information about the nature of the problem. Feature importance considerations are not in the scope of this project, but is noted as the natural next step in further improvement of the code library.

V. CONCLUSION AND SUMMARY REMARKS

The search for a high-throughput way of determining the thermal lattice conductivity of materials continues. It may be understood as the leading factor in development of technology in the fields of thermoelectrics, insulation and thermal energy storage. In this project we have seen that both DNN and decision tree models holds a high promise for large scale materials engineering applications. Although DNN models are more computationally expensive, it has during this study produced the most accurate predictions. On the other side, the

CPU time as shown by the decision tree algorithm is outstanding and an important factor in said large scale applications. Another important factor to consider under the results obtained from the `XGBoost` algorithm is that the internal workings of the final learner is extractable. One can study feature importance and from that extract important findings about the true nature of the underlying problem. Without conclusion, the author notes here that further inspection, development and testing is necessary to determine which of the algorithms perform better overall.

With the future in mind, the author wishes to state that both algorithms implemented in this project would be extraordinary candidates for large remote server computation as both modules have been optimized from multithreaded computation. With a (hopefully) much decrease CPU time, many more random trials may be performed to find even better model candidates. Going from there, it would be necessary to start looking into trimming algorithms to balance both the neural network and tree method in terms of null-nodes and 'dead' leaves.

---

[1] S. G. T. X. P. G. K. N. Taishan Zhu, Ran He and J. C. Grossman, Energy Environ. Sci. (2021).
[2] P. J. S. P. e. a. Pyzer-Knapp, E.O., npj Comput Mater 8, 84 (2022).
[3] "Matminer: An open source toolkit for materials data mining." https://hackingmaterials.lbl.gov/matminer/, accessed: 2023-12-01.

A. Appendix

GitHub: hyperlink