Predicting Superconducting Critical Temperatures with Supervised Machine LearningWork supported by the National Science Foundation

K. Kleinasser, Cornell University, Ithaca, NY*

CONTENTS

I. INTRODUCTION

I.1. Superconductors

Superconductors are materials that lose all electrical resistance at low temperatures. These materials have a critical temperature (T_C) at which they lose their resistance. Most have very low critical temperatures, but "unconventional superconductors" can have critical temperatures as high as room temperature under non-atmospheric conditions.

Electrons in superconductors form Cooper Pairs below their critical temperature. These pairs of electrons are held together with phonouns, which are atomic-level collective excitations. Phonouns are similar to photons in that they also have particle-like properties [1].

Unconventional superconductors are still not well understood and remain an open question in Physics. Understanding them could lead to the discovery of superconducting materials stable at room temperature under atmospheric conditions. Such a material would have large implications, such as super efficient electricity transfer and vast efficiency improvements for applications like particle accelerators and power lines.

I.2. Matminer

Most superconducter databases do not include enough information to train an effective machine learning model, but such data can be extracted from the data they do provide. We use matminer to produce our features from the provided material data. Matminer is a python library that generates data from various measured properties of a material. Matminer collects existing calculations into a machine learning friendly python package. Our matminer workflow is shown in Figure 1.

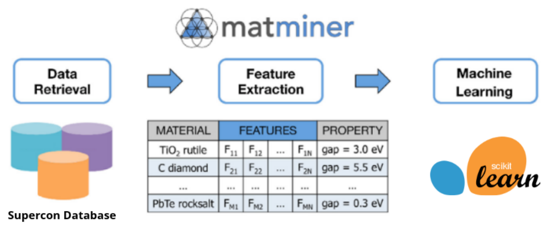

FIG. 1. Flowchart illustrating our matminer usage, modified from official matminer graphic [2].

Our database only provides the superconductor composition data. Matminer's featurizers can generate 53 features from the composition of a material. If we had band structure or other data, we could produce more information that we could use in our model.

I.3. Machine Learning

Previous papers have used random forest models to predict critical temperature [citation needed], but this paper will examine eight models before settling on two for further investigation. All models are implemented with Scikit-Learn, with the notable exception of a mlens superlearner [3, 4]. We will also use MAPIE models for uncertainty, discussed in Section II.3. These models are described below. Each model's hyperparameters[1] was optimized with Scikit-Learn's GridSearchCV, which tests combinations from a grid of hyperparameters and returns the best performing model based on a specified metric.

We started our model search with some linear models. Besides the base Linear Regression model, we used linear (and polynomial) Support Vector Regression (SVR) models. SVR uses decision boundaries, which are lines parallel to the regression line. The model aims to maximize the amount of data within the decision boundaries and has hyperparamters to modify sensitivity to prevent

* Lycoming College, Williamsport, PA; klekirk@lycoming.edu

[1] Hyperparameters are machine learning parameters that change how a model is trained.

overfitting.[2] We also trialed Elastic Net and Bayesian Ridge models. Elastic Net uses L1 and L2 penalties to stabilize the model, and Bayesian Ridge uses probability distributors instead of point estimates.

Additionally, we trialed Decision Tree and KNeighbors (KNN) models. Decision trees are very interpretable - they break predictions into nodes of the tree, eventually leading to a prediction value. These trees can be represented graphically and show how they produce results, unlike most machine learning models. KNN models are a little different, they store all the data and predict values based on a similarity measure. The model looks at a specified number of similar neighbors to produce a prediction.

Finally, we tried multiple ensemble models - Random Forest Regression (RFR), Extra Trees, and a superlearner. RFR models use numerous decision trees and subsamples the data with replacement. This means that the model replace data after using it in a subset. Extra Trees is like RFR, but it does not replace the data after use in a subset. The final ensemble model we tested is a superlearner, a model that can combine multiple high-scoring Scikit-Learn model predictions and sometimes improve the performance from the individual models.

II. METHODOLOGY

II.1. Code Structure

The source code used for this paper is available publicly on github at `https://github.com/sylphrena0/classe2022`. This repository also includes the source files for this latex paper, data files, images, and documentation files.

Our research uses numpy and pandas throughout our code to handle arrays and tabular data [5, 6]. We also use matplotlib and seaborns to generate our graphs [7, 8].

The code is split into multiple python files so processes could be completed in stages and to maintain readability in the code. Most of our testing and final training was completed in juypter notebooks, but some computations were highly computationally expensive and needed to be run remotely. For these jobs, we created simple python files and made bash scripts to run them on Cornell's CLASSE compute farm. We also made several bash aliases and functions to simplify the compute farm workflow, which are also available on the github repository.

Since we used multiple files, we chose to create shared dependencies files where we defined functions to import data, train models, and generate our graphs. These files are then imported in all the relevant scripts to reduce redundancy.

II.2. Running the Code

First, the featurizer script imports the dataset, extracts features from the material compositions, and exports the csv data. This script is one of the most computationally expensive and takes several hours to run on the CLASSE compute farm with 64 dedicated cores.

After the features are exported, our analysis jupyter notebook imports the data with the shared import function and exports histograms and a correlation matrix.

Next, the training_single jupyter notebook or script can train individual models with the shared evaluation functions. This is used to get a landscape of initial performance before optimization. After training, the function plots the actual T_C versus the model prediction, using a heatmap to visualize the difference from the ideal prediction.

The optimizer script then uses a grid of manually defined hyperparameters to optimize models based on R2 score. This allowed significant improvements to baseline models. After optimization, the optimized models can be plotted in our single training notebook. After confirmation that the model is better than the baseline, the models can then be plotted together in a single graph using our bulk training notebook.

We evaluated our models using several metrics - R2 scores for regression evaluation, Mean Squared Error (MSE) and Mean Absolute Error (MAE) for error evaluation, and prediction intervals for uncertainty evaluation.

II.3. Uncertainty

Our evaluation functions can produce uncertainty calculations using forestci, mapie, or lolopy [9–11].

Forestci is python implementation of an algorithm from [12] that predicts confidence intervals for random forest models. It is the fastest of the uncertainty methods listed.

Mapie is

III. RESULTS

III.1. Fonts

It is preferable to avoid the older TEX and LATEX 2.09 macros for controlling fonts such as `\rm`, `\it`, *etc.* Rather, it is better to use the macros introduced in LATEX 2_ε. If the older font commands are used (they really should be avoided!), be sure to use curly braces to properly limit the extent of the font change. `{\bf ...}` is the correct method. Commands for controlling text and math font changes are summarized in Table **??**.

[1] J. W. Rohlf, Superconductivity, in *Modern Physics: From Alpha to Z* (Wiley, 1994) Chap. 15.
[2] L. Ward, A. Dunn, A. Faghaninia, N. E. Zimmermann, S. Bajaj, Q. Wang, J. Montoya, J. Chen, K. Bystrom, M. Dylla, K. Chard, M. Asta, K. A. Persson, G. J. Snyder, I. Foster, and A. Jain, Computational Materials Science **152**, 60 (2018).
[3] 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, Journal of Machine Learning Research **12**, 2825 (2011).
[4] S. Flennerhag, Ml-ensemble (2017).
[5] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, Nature **585**, 357 (2020).
[6] T. pandas development team, pandas-dev/pandas: Pandas (2020).
[7] J. D. Hunter, Computing in Science & Engineering **9**, 90 (2007).
[8] M. L. Waskom, Journal of Open Source Software **6**, 3021 (2021).
[9] K. Polimis, A. Rokem, and B. Hazelton, Journal of Open Source Software **2** (2017).
[10] V. Taquet, G. Martinon, N. Brunel, I. Ibnouhsein, F. Deheeger, R. Adon, A. Papp, A. A. Goumbala, A. Borgohain, T. Morzadec, and et al., Mapie - model agnostic prediction interval estimator (2022).
[11] M. Hutchinson, Citrineinformatics/lolo: A random forest library (2022).
[12] S. Wager, T. Hastie, and B. Efron, Journal of machine learning research : JMLR **15**, 1625 (2014).

[2] Overfitting occurs when a model is trained to be too specific to a particular dataset and is not generalizable.