**Predicting the Critical Temperature of Superconductors using Machine Learning**

ME8813

Patrick Gardocki

Spring 2023

**Introduction**

Superconductors are a material that under a certain critical temperature, the material is able to conduct current with zero resistance. This makes superconductors very favorable in certain applications. The most common application is the use of them in fMRI machines used by the healthcare industry, as well as various research facilities, such as the Large Hadron Collider.

Since the discovery of superconductors in 1911 by Heike Kamerlingh Onnes, there have been two main limitations that prevent more abundant use of superconductors [1]. One being that superconductors favorable properties emerge only at very low temperatures making it challenging to keep the material under its critical temperature. The other issue is that there is no current model to predict the critical temperature of superconductors.

The goal of this project is to attempt to apply machine learning methods in order to predict the critical temperature of a superconductor given its chemical formula.

The data will be accessed from the Superconducting Material Database maintained by Japan’s National Institute for Materials Science (NIMS) [2]. There are 21,263 superconductor entries with 81 variables per entry. These variables will be features of chemical properties of the material such as atomic mass, thermal conductivity, or number of valance electrons.

Initially, dimensionality reduction techniques will be used to determine if all 81 inputs are required to accurately predict the critical temperature, and if not how many will be sufficient. Neural Networks and Regression will be utilized to attempt to predict the critical temperatures of superconductors, and the two model will be compared in accuracy, efficiency, and reliability.

**Dataset**

The dataset used for this project was borrowed from 2018 paper titled, “A Data-Driven Statistical Model for Predicting the Critical Temperature of a Superconductor” [3]. It contains 21,263 rows and 81 input columns. The 81 input columns are made up of various feature extractions of each semiconductor material. Eight element properties were used to create features. The eight properties are atomic mass, first ionization energy, atomic radius, density, electron affinity, fusion heat, thermal conductivity, and number of valence. For each of the eight properties, ten methods were applied to extract features.

The following are the ten ways that the element data is manipulated.

*Mean:* (1)

*Weighted Mean:* (2)

*Geometric Mean:* (3)

*Weighted Geometric Mean:* (4)

*Entropy:* (5)

*Weighted Entropy:* (6)

*Range:* (7)

*Weighted Range:* (8)

*Standard Deviation:* (9)

*Weighted Standard Deviation:* (10)

Where,

t­i is an elements given property,

pi is the ratio of an element in a material,

is the fraction of a given property of a material,

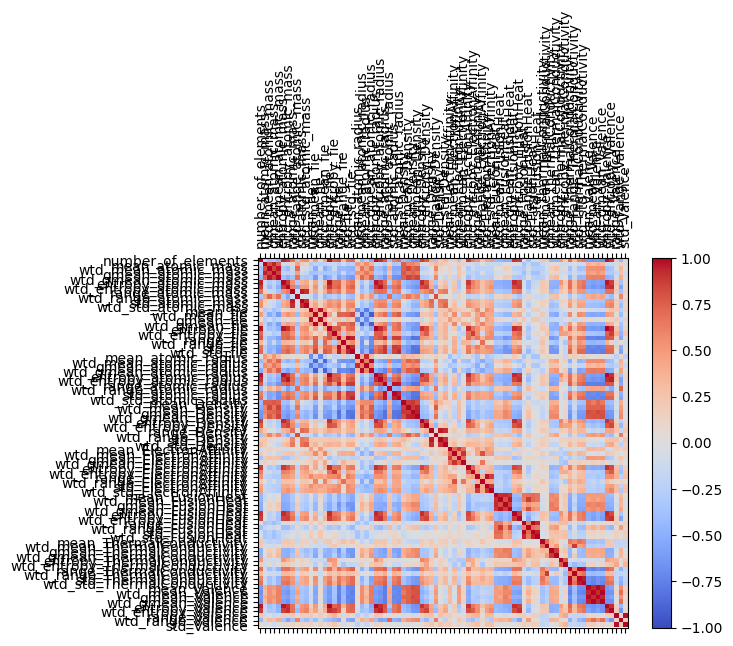
A

B =

This is how 80 input columns were generated with the last one being the number of elements in the material. The element data was gathered using ElementData from Mathematica, and the superconductors material information was obtained from the Superconducting Material Database supported by NIMS.

**Dimensionality**

Knowing that the 80 input dimensions were extracted from only 8 original inputs, some analysis was done to better understand how the data is related to each other. First, a Pearson Correlation was done to see if any two variables were linearly related with each other. The full correlation matrix is shown below in Figure 1 with a subsection of the atomic mass property in Figure 2. It is seen that there are groups of variables that show strong relationship as well as groups that show no relationship at all. For example, the standard deviation and range of atomic mass have a Pearson Correlation of 0.971. This shows a very strong linear relationship and can lead to dimensionality reduction.



*Figure 1 - Complete Pearson Correlation Matrix for 80 dimensions*

A picture containing treemap chart

Description automatically generated

*Figure 2 - Sample Pearson Correlation matrix for single element property (atomic mass)*

Next, Principal Component Analysis was done to see if the provided dataset can be represented in less dimensions. A Scree Plot is shown below in Figure 3. The first two components make up 50% of the variance of the model. Expanding to 20 principal components captures 97% of the variance.

Chart, line chart

Description automatically generated

*Figure 3 - Scree Plot*

**Model Implementation**

ANN

First, a neural network was modeled using the original dataset containing all 80 dimensions. The dataset was standardized to a mean of 0 and a standard deviation of 1 for each feature. This is to prevent certain features from dominating due to different magnitudes but still maintain the differences in the range of the values. This is very important in neural networks as they put weights on features during training.

The initial network architecture followed the following guidelines to avoid over fitting. The first equation (11) suggests a number of neurons and the next (12) suggests the total number of parameters to use [4]. Alpha is a scaling factor usually between 2 and 10. IT can be adjusted to modify how general a model will be [5]. This parameter is similar to a branching factor. Given 21,263 samples and 80 input neurons with 1 output neuron, the suggested total number of neurons is 131 with an alpha of 2 and the number of parameters to be 1,722,303. Therefore, the initial architecture was set to have four hidden layers with 32 neurons each. This leads to a total parameter count of 1,048,576.

Number of Neurons: (11)

Number of Parameters: *Ns*∗(*Ni* + *No*) (12)

*Where,*

*Ni = number of input neurons.  
No = number of output neurons.  
Ns = number of samples in training data set.  
α = an arbitrary scaling factor usually 2-10.*

Using the standard Adam optimizer with a learning rate of 0.01 and ‘relu’ activations for each layer, the following training loss function in Figure 4 is observed over 200 epochs.

Chart, histogram

Description automatically generated

*Figure 4 - Initial Network Architecture*

Model accuracy was measured using R^2 scoring. This model had a training accuracy of 91.07% and a test accuracy of 87.95%. This is already a good model with similar results for training and testing, implying that overfitting has not occurred. The mean-square-error was 0.5861 for this model.

Attempting other activation functions such as sigmoid, logistic, and tanh functions, all resulted in subpar results compared to ‘relu’. Therefore, ‘relu’ activation functions were used for the remainder of the modeling process.

Using a constant learning rate for the Adam optimizer, is one option that can be used for a learning rate schedule. Other forms include time-based decay, step decay and exponential decay. Each of these schedules were ran, with the time-based decay resulting in the best results. The time-based decay is modeled by the following equation (13).

(13)

*Where,*

*LR = Learning Rate*

*LRp = Previous Learning Rate*

*decay = initial learning rate / total number of epochs*

*epoch = iteration number*

This was implemented with an initial learning rate of 0.1 for 100 epochs. The time-based decay model got a training accuracy of 92.92% and a test accuracy of 89.46%. This is about a 2% improvement in both the test and training accuracy. The mean-square-error was 0.570, a slight improvement from the initial model.

The number of neurons and hidden layers was altered to see if the model’s performance could be improved. First the hidden layers were reduced to two while maintaining the total number of neurons. Two hidden layers can sufficiently model any functions with any shape. [cite]. Also, reducing the number of hidden layers further aids in preventing overfit as well as reducing training time. The two layered model with 64 neurons per layer returned the same accuracies and mean-square-error as the 4 layered model, but reducing to one layer negatively impacted the performance. Increasing the number of neurons in the two hidden layer model did not change the performance or accuracy of the model. The only change is a greater possibility of overfitting since more parameters were used to create a model of similar performance. However, if the initial four layered model had an increased number of neurons to 96 per layer, the model had an increased training accuracy of 97.8% and testing accuracy of 90.8%. The mean-square-error was also reduced to 0.54. This model has a greater discrepancy between the test and training error, which can be a symptom of overfitting. The optimal neural network model contained 2 hidden layers with 64 neurons each, used a time-decay learning schedule, and utilized k-fold cross-validation. Figure 5 below shows the true versus predicted critical temperature values as well as the prediction error distribution.

Chart, scatter chart

Description automatically generated Chart, histogram

Description automatically generated

*Figure 5 - True vs Predicted Tc and Prediction Error Distribution*

ANN with PCA

After applying principal component analysis, it was observed that 20 principle components can maintain over 97% of the datasets variance. A neural network will be modeled using this data to see if reducing the dimensions from 80 to 20 will still maintain the same accuracy.

Using the same guidelines as before, the number of neurons was determined initially to 515, and 446,523 total parameters. 515 neurons are a very large network and can lead to overfitting, so 208 neurons, with 96 in three hidden layers were tested at first to observe the outcome. The model also used the learning rate schedule as the previous model. Already, the model performed just as well as the original dimensioned model, with a training accuracy of 93.8% and a test accuracy of 90.1%. The mean-square-error was also comparable at 0.55. This is promising, showing that not all dimensions were needed to create an accurate model.

However, if only 12 principal components are used, covering over 90% of the original variance, the neural networks performance suffers, and drops to a training accuracy of 85.85 and a test error of only 83.0%.

XGBoost

Extreme Gradient Boosting, XGBoost, is another type of machine learning model that is a tree learning model. It adds new models iteratively to correct mistakes of previous models. XGBoost’s framework contains custom loss functions and regularization methods to prevent overfitting. XGBoost is used due to its efficiency in speed and memory use. It utilizes parallel processing and can handle very large datasets very well [7].

Using an XGBoost model, and the standardized dataset with all 80 dimensions, it attains a training accuracy of 97.01% and a testing accuracy of 91.15%. This is an improvement from the neural network models. The mean-square-error is also reduced to 0.54. If the 20-dimensional dataset is used, the model performs slightly worse, with a training accuracy of 95.4% and test accuracy of 88.46%. This is slightly worse than the neural network that used the same 20 principal components.

The real benefits of using XGBoost is its speed and ability to adapt to the dataset without requiring much hyperparameter optimization. While there are hyperparameter optimization that can be done to adjust the number of trees to fit, or the depth of each tree, the impact was not as severe as when hyperparameter optimization was done with neural networks. After tuning some hyperparameters, the model did not perform significantly better. Also, the execution time is much faster for XGBoost compared to neural networks. XGBoost took only 33.6 seconds to run, while the neural network model took over 112 seconds.

**Conclusion**

Using data from the Superconducting Material Database, which contains 21,263 entries on superconductors, machine learning methods were applied to determine the accuracy in regard to predicting the critical temperature. From the chemical properties of the superconductors, 81 features were extracted. Firstly, dimensionality reduction techniques were applied in hopes of being able to utilize a smaller dimensioned dataset for machine learning. Using Principal Component Analysis, 20 principal components were settled on as they represented over 97% of the data’s variance. Next, neural network models were introduced and their hyperparameters were optimized to reach a final training accuracy of 93.8% and a test accuracy of 90.1%. Using the 20 principal components, resulted in a neural network model with extremely similar performance, supporting that 20 dimensions were sufficient to predict the critical temperature. A XGBoost model was also implemented and performed slightly better than the neural networks and with less reliance on hyperparameter tunning and greater execution times.

Using machine learning models, it is possible to predict superconductor critical temperatures with about 90% accuracy. Even though there is a good basic framework for researchers to use for predictions, there are not many high temperature superconductors, which means that the machine learning models are much less accurate in those less populated regions.

More research should be done to help increase the accuracy of predicting critical temperatures to better direct the search for superconductors with higher critical temperatures.

**References**

[1] ‘Superconductivity’ CERN. Accessed on: <https://home.cern/science/engineering/superconductivity>

[2] MDR SuperCon Datasheet. National Institute for Materials Science Available: <https://mdr.nims.go.jp/collections/5712mb227>

[3] Hamidieh, Kim. (2018). ‘A Data-Driven Statistical Model for Predicting the Critical Temperature of a Superconductor’.Available*:* <https://arxiv.org/abs/1803.10260>

[4] ‘Training an artificial Network – Intro’. Frontline Solvers. Accessed on: <https://www.solver.com/training-artificial-neural-network-intro>

[5] Hagan, M., Demuth, H., Beale, M., Jesus, O. ‘Network Design’ pgs., 13-2.

[6] Brightwell, G., Paugam-Moisy, C., ‘Multilayer neural networks: one or two hidden layers?’

[7] ‘XGBoost Documentation’ Accessed on: <https://xgboost.readthedocs.io/en/stable/>