P1 - PREDICTING SUPERCONDUCTOR CRITICAL TEMPERATURE

11th March 2020

190020774
University of St. Andrews
CS5014 Machine Learning

Contents

1 Introduction

This practical aims to design and create a supervised learning regression model to predict the critical temperature of a superconductor based on various physical properties.

Superconductors were first discovered in 1911 and are very unique materials in that at or below its critical temperature, the material is able to conduct current with zero electrical resistance. Another unique property is that these critical temperatures are often extremely low, either close to or lower than the boiling point of Nitrogen at 77K. The applications of superconductors are wide ranging, for example in MRI (Magnetic Resonance) imaging and use in the Large Hadron Collider at CERN. Furthermore, they have the potential to revolutionise electrical power systems as they can deliver lossless energy.

Supercooling materials to experimentally find their critical temperatures can be extremely costly and time consuming due to these low temperature requirements, and so a novel method of predicting the critical temperature of a superconductor from its physical features was presented in the paper *"A data-driven statistical model for predicting the critical temperature of a superconductor"* (Hamidieh, 2018[1]). This paper details how numeric physical features were extracted from a materials' chemical composition, and this data was provided to us for use in the regression model. The paper also details two regression models which were used to predict these critical temperatures, a multiple regression model, and another known as XGBoost. This project is not concerned with the XGBoost model, however I have created a multiple linear regression model with similar results to those published in the paper. The following report details the steps followed to create this linear regression model, as well as results and evaluation.

2 Design, Implementation and Evaluation

2.1 Loading and cleaning

The data used for this practical is contained in the csv file `train.csv`, this was loaded into the program into a Pandas dataframe using the function `pd.read_csv('data/train.csv')`.

As there is only one output for this problem, namely the critical temperature, the data was then split into an input dataframe X, and and output dataframe y. There are a large

number of features in dataframe X, the use of which will be discussed later.

Once the data is loaded, it is important to split it into a training and testing set. This is to avoid what is known as *data snooping bias* which arises from visualising the entire data set and overfitting based on observations. This splitting was performed using the `sklearn` function `train_test_split`, setting aside 80% of the data for testing and the remaining 20% for training. A random seed was also used so that each time the data is loaded, it is split in the same way. If this was not performed in this way, then over time we would end up viewing the entire data set.
Once the data has been loaded and split, we need to ensure that there is no garbage data such as missing data or corrupted values that have been recognised as NaN. A quick glance at the data in the training set is inconclusive, as there are so many values it is infeasible to check them all manually for garbage values. Therefore, we use the pandas function `dropna()` which removes all values from the dataset which are null or NaN. Fortunately, there appears to be none of these values in the data set, which we can see by comparing the sizes of the training set before and after running `dropna()`.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
   test_size=0.2, random_state=42)
X_train.shape >> (17010, 81)
X_train = X_train.dropna(axis = 0, how = 'any')
X_train.shape >> (17010, 81)
```

2.2 Analysing and visualising the data

Once the data has been sufficiently split into training and testing, and cleaned accordingly, it is necessary to analyse and visualise the data to see if we can gain any insights into choosing a particular regression model. Seeing as there are 81 input features, plotting each of these against the critical temperature proved to be exhaustive, with many plots looking similar to each other. For reference, these plots are provided in the appendix. Looking at these plots, we can see that there is some linearity between the input and output features, whilst some features also appear to display polynomial behaviour and gaussian features. Therefore, we suggest starting with a linear regression model and potentially expending to polynomial regression after analysing the linear model.

2.3 Preparing the inputs

2.3.1 Selecting Features

Since we have a very large number of input features, we may want to select a subset of these features as not all of them may be relevant to the critical temperature. It is quite difficult to see the correlation between various features and the outputs from just plotting the data so we do not want to disregard any features from just these basic plots.

To get a better understanding of the correlation between features and the critical temperature, the standard correlation coefficient between features and the output was calculated. With the correlation coefficient, a value close to +1 suggests a strong positive correlation, whereas a value close to -1 suggests a strong negative correlation. Values close to 0 suggest little to no correlation at all. These correlations are for a linear case. Using the pandas function corr() on the training data we can visualise the correlation coefficients for each feature, which are shown below. For the sake of simplicity, only the 10 largest correlations are displayed (5 positive and 5 negative).

```
data_in = pd.read_csv('data/train.csv')
train, test = train_test_split(data_in, test_size=0.2, random_state=42)
corr_matrix = train.corr
print(corr_matrix["critical_temp"].sort_values(ascending=False))
```

```
critical_temp                1.000000
wtd_std_ThermalConductivity  0.720134
range_ThermalConductivity    0.686189
std_thermalConductivity      0.652264
range_atomic_radius          0.652252
wtd_entropy_atomic_mass      0.62036
...
gmean_Density                -0.540816
gmean_Valence                -0.572613
mean_Valence                 -0.598686
wtd_gmean_Valence            -0.615452
wtd_mean_valence             -0.631542
```

The full output of this correlation can be found by running `correlation.py`. This output shows a wide range of correlation coefficients across all the different input features. As a starting point and for simplicity by reducing computing time we have chosen to use the above 10 features as the sole features for the model. this will be evaluated later through the addition of more features to see if this is a reasonable choice to make.

2.3.2 Scaling the data

Many machine learning algorithms do not perform well on data with large ranges, therefore before running our regression model the data needs to be scaled in a suitable manner. There are two ways to approach this scaling - normalisation and standardisation.

Normalisation shifts the data so that it is in the range [0, 1]. This is performed by the following equation for an input feature X:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Standardisation scales the data such that it is centered around 0, and as such is much less affected by outliers. Standardisation is computed in the following manner:

$$X' = \frac{X - \mu}{\sigma}$$

Where μ, σ are the mean and standard deviation of the original data set, respectively. From the feature plots provided in the appendix, we can see that some features appear to be symmetrical whereas others do not, making it difficult to choose a particular scaling choice. However, standardisation is much less affected by outliers in the data, and we can see that in a number of plots there are a number of outliers[2], as shown in the plots below.

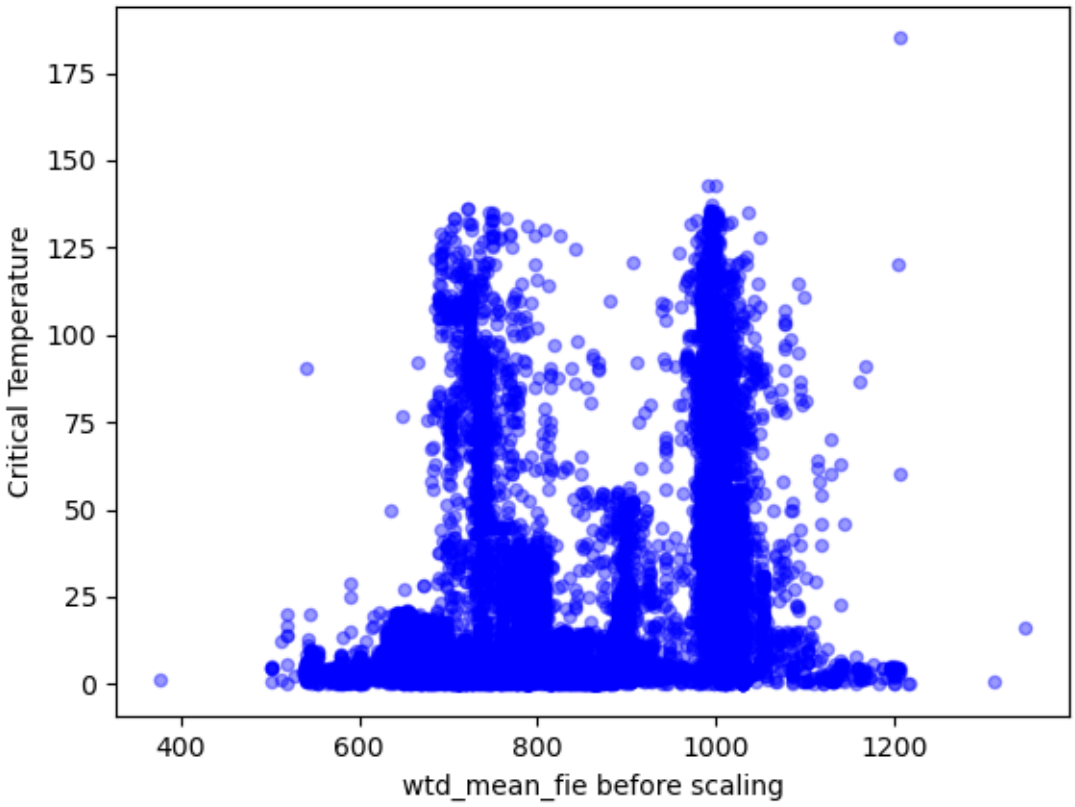

Figure 1: Data before scaling - note various outliers

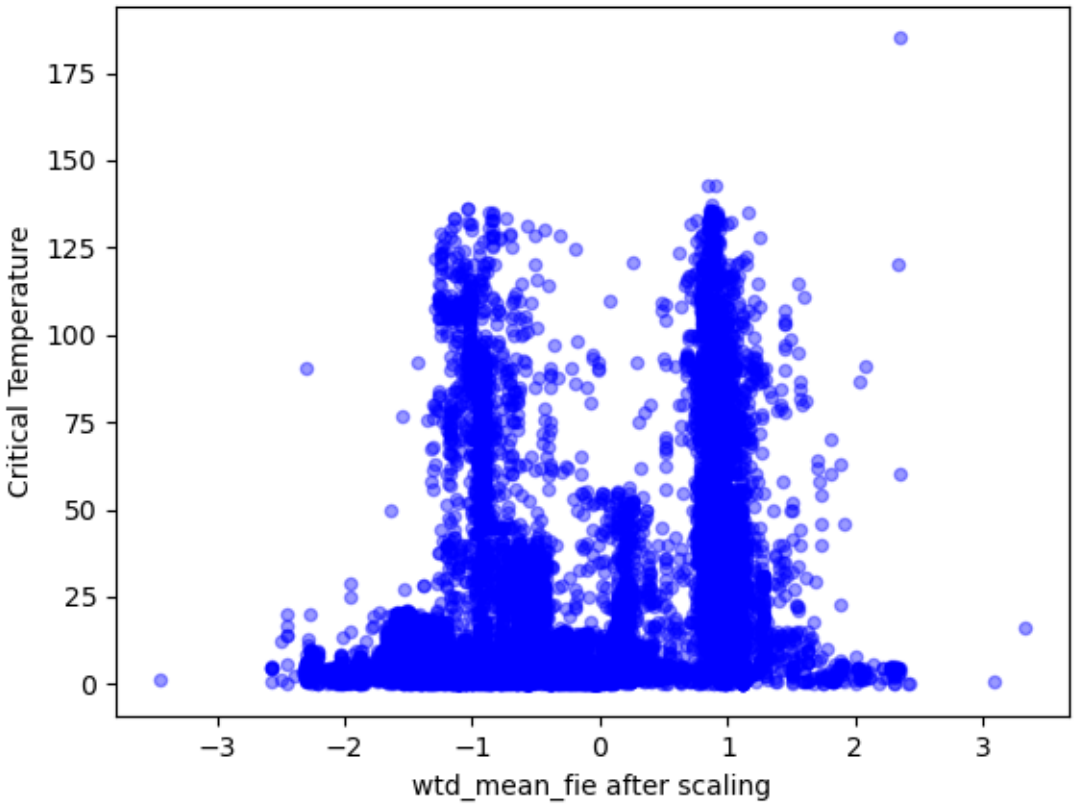

Figure 2: Data after scaling - outliers scaled down

2.4 Selecting and training a regression model

Now that we have a suitable set of features and scaled the data appropriately, we are ready to begin to select and train a regression model. There are a wide range of linear regression models each with their own benefits drawbacks, and so we have chosen a number of "quick and dirty" models with standard parameters which we will evaluate before moving on to evaluation on the test set.

Linear regression attempts to fit the input data to the output by assigning weights to each of the input parameters and optimising these parameters such that the error between the model and the actual values are minimised according to a specified loss function.
For a model with a number of input features the predicted value is given by the following equation:

$$\hat{Y} = f(X, \theta)$$

Where:

$$f(X, \theta) = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + ... + \theta_n X_n$$

Where θ is an array of weights and n is the number of input features.
A typical loss function which we will use for evaluating the different models is the mean squared error between the actual and predicted values, given by:

$$L(\theta) = \frac{1}{2N} \sum_{i=1}^{n} (Y - f(X, \theta)^2$$

For comparison purposes we use the root-mean-squared-error, which is simply the square root of the above equation

2.4.1 Ordinary Least-Squares

Ordinary least-squares regression attempts to fit the data by minimising the residual sum of squares between the outputs of the dataset and the actual values.

2.4.2 Elastic Net

Elastic net regression is a combination of lasso and ridge regression which attempts to minimise the linear combination of the penalty terms from both ridge and lasso.[3]

2.4.3 Stochastic Gradient Descent

Stochastic gradient descent is an iterative method to minimise the loss function. Initially, weights are randomised and the loss function is calculated as well as its derivative. If the loss function derivative is close to zero, then the algorithm stops and returns the values of the weights. If not, then the weights are updated through the use of the partial derivative of the loss function and the learning rate α as follows:

$$\hat{\theta_n} = \hat{\theta_n} - \alpha \frac{\partial}{\partial \theta_n} L(\theta_n)$$

This method continues until the solution converges or the maximum number of iterations has been reached.[4]

2.4.4 Support Vector Machines

Support vector machine regression minimises a Lagrangian function based on a linear combination of the input parameters to the model[5].

2.4.5 Random Forests

A random forest regressor runs many separate decision trees in parallel and takes the average of these average of these predictions to produce the final regression prediction.[6]

2.4.6 Comparison of Models

The different models are compared against each other using k-fold cross validation. K-fold cross validation randomly splits the training data into subsections known as folds, and takes k-1 folds for training and the remaining fold for evaluation. It performs this k times, picking a different fold for evaluation on each iteration. This was performed using sklearn's `cross_val_score`, with 10 folds and using the root-mean-squared-error as the scoring metric. The results of the cross-validation scores across the different models are shown in the table below.

Model	RMSE	Mean	Standard Deviation
Least Squares	17.6130	17.6825	0.4473
Elastic Net	21.0871	21.0977	0.4431
SGD	17.9785	17.9106	0.4731
Support Vector Machines	16.3488	16.4911	0.3962
Random Forest	19.9017	19.9336	0.3334

These values were obtained by using all of the features in the data set, as it was found that increasing the number of features reduced the error across all the different models. This makes sense as whilst some of the correlation coefficients were quite small, they were all non-zero and so therefore contribute in some way to the output temperatures.

2.5 Evaluating model performance

Now that we have selected a model, we can run our model on the testing data and observe the output. For the model we have chosen - Support Vector Machines - the results of this model on the testing data are as follows: rmse = 15.8023, which is an improvement on the training data. In fact, the linear model produced by Hamidieh in the corresponding paper returned an rmse value of 17.6.

2.6 Discussion of results

Whilst our model predicts the results with some degree of accuracy, there are ways in which this could be improved. For example, we have only considered linear regression when choosing a suitable model. As many of the data plots suggest a nonlinear model we may want to look at polynomial regression models which could improve the accuracy of our model. Furthermore, choosing all of the features may prove to be costly if the size of the dataset were to increase. for example, running the cross-validation scores on the support vector machines took a couple of minutes to compute, and this time would only increase as the size of the dataset increases.

3 Appendix

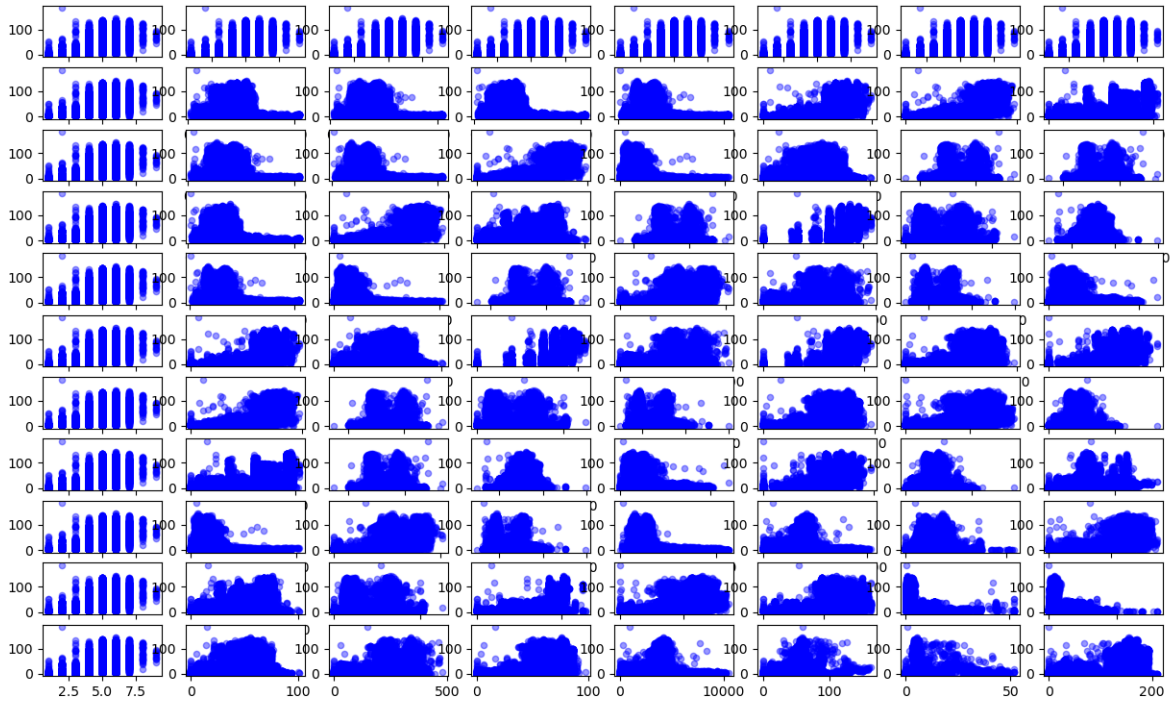

Figure 3: Individual Features Plotted Against Critical Temperature

References

[1] Hamidieh, Kam *A data-driven statistical model for predicting the critical temperature of a superconductor* Computational Materials Science, 2018.

[2] Géron, Aurélien *Hands-on machine learning with scikit-learn and tensorflow p. 65* O'Reilly Media, 2017.

[3] Wikipedia, the free encyclopedia 10th February 2020. `https://en.wikipedia.org/wiki/Elastic_net_regularization`

[4] Harris-Birtill, David and Terzic, Kasim *CS5014 Machine Learning - Lecture 3: Gradient Descent p. 13-14* University of St. Andrews 3rd February 2020.

[5] Mathworks *Understanding Support Vector Machine Regression* `https://uk.mathworks.com/help/stats/understanding-support-vector-machine-regression.html` 2019.

[6] Chakure, Afroz *Random Forest Regression* `https://towardsdatascience.com/random-forest-and-its-implementation-71824ced454f` June 29, 2019.