Critical Temperature Linear Regression Model of Superconductor Materials

Introduction

Superconductors are materials that give little to no resistance to electrical current, making it a perfect conductor of electricity. In order to exhibit this 100% efficiency of carrying current, the superconductor needs to be cooled below a specific temperature, known as its critical temperature. This critical temperature is different based on the material used to create the superconductor. Both categories of superconductors, high temperature superconductors and low temperature superconductors, need to be cooled to extremely low temperatures.

Currently the conditions required for superconductivity to occur are quite extreme, which do not allow practical uses like electrical transmission. In addition, some materials are hard to produce and chemically unstable. Research into superconductors can lead to a better understanding of how superconducting materials work and allows us to gain the ability to design better materials and possibly achieve a superconductor at room-temperature. Utilizing these high-temperature superconductors could contribute to a solution towards solving the energy crisis by minimizing heat energy loss with suitable superconductors.

The objective of this case study is to explore Linear Regression with L1 and L2 regularization, and its impact on predicting the critical temperature of a superconductor. Additionally, feature importance is also investigated with the best model.

Dataset

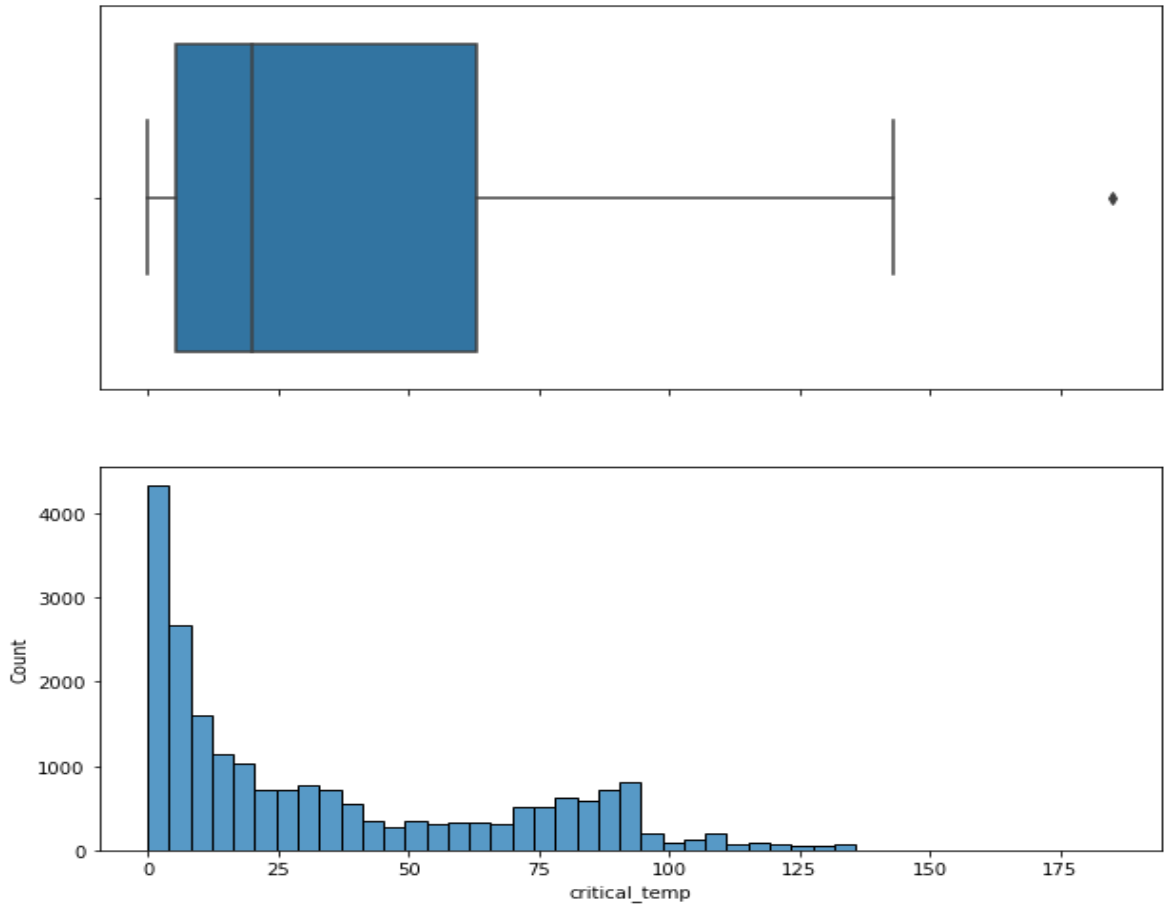
There are two data files: (1) train.csv contains 81 features that describe 21263 superconductors along with the critical temperature in the 82nd column, (2) unique_m.csv contains the chemical formula broken down for all 21263 superconductors from the train.csv file. The last two columns have the critical temperature and chemical formula. The 2 files have a 1-to-1 relationship, so we were able to append the unique_m dataset to the train dataset.

Some initial observations are:

- There are no features missing data that would require imputation
- The chemical formula from the unique_m file is a string object so it was removed since it will not help with building a regression model
- Critical temperature was in both files so it was removed from one of the files

The data represents different materials with their mix of elements used to create them. The data also includes the critical temperature required for the material to exhibit their superconductive characteristics. Using the provided data our work will build a linear regression model that can be used to predict the critical temperature of any new material based on the elements used to make the new material.

Below is an analysis of the critical temperature distribution within the dataset.



Number of samples with Tc in range 0 K- 49 K: 14855
Number of samples with Tc in range 50 K- 100 K: 5611
Number of samples with Tc in range 101 K- 140 K: 760
Number of samples with Tc in range 141 K- 150 K: 2
Number of samples with Tc greater than 150K: 1

The boxplot above indicates a critical temp outlier at around 180 but a lack of domain knowledge prevents us from excluding any of the data points.

Linear Regression Model

Methods

In a dataset with many features, there is a strong tendency for a linear regression model to overfit. We will be using three different models: Lasso(L1), Ridge(L2) and Elastic Net to overcome the issue of overfitting and to optimize this linear regression problem. We will also apply GridSearchCV to tune the hyper-parameters for these models, over 10 folds for each of the three models.

Evaluation Metrics

- For each model we'll use MSE (Mean Squared Error) as a cross validation metric, which can explain the variance of the dependent variable explained by the independent variables of the model. It measures the strength of the relationship between your model and the dependent variable. However, R^2 alone cannot be used for comparing the models as the value of R^2 increases with the increase in the number of predictors. This is important to note since we will have a different number of predictors in our models due to LASSO performing feature reduction by shrinking coefficients to absolute zero.
- We will also investigate the top coefficients that contribute the most to the chosen model and utilize them to interpret the model.

Model Building and Evaluation

Below we create three models for Lasso, Ridge and Elastic Net. We'll create a pipeline for each model that will scale the data using StandardScaler and then use Grid Search with cross validation. Our data is split between an 80% training and a 20% test set. We'll then use the test set to evaluate each of the models at the end.

The Pipeline approach in SKlearn will allow us to define a sequence of steps to be performed on the data. First, we'll scale the data using StandardScaler. Because of the skewness and outliers of some features, we decided to use the StandardScaler which scales features using statistics that are robust to outliers. Next, we'll define the model to be used (Lasso vs Ridge vs ElasticNet). Finally, we'll use Grid Search to find the optimal hyperparameters. By using a Pipeline with GridSearchCV we only scale the data in the training set. The test sets in each fold are then scaled using the same scaler. We are trying to eliminate data leakage. Please see references below for more information.

For Lasso and Ridge, we'll vary the hyperparameter alpha. For Elastic Net, we'll vary the hyperparameters alpha, which is a constant that multiplies the penalty terms. Additionally, we'll vary the hyperparameter l1_ratio which controls the amount of mixing between L1 and L2. If l1_ratio is 1, then we'll only use L1 regularization. If l1_ratio is 0, then we'll only use L2 regularization.

We tried using the general tune range below for models Ridge and ElasticNet, but Ridge will keep selecting the highest alpha that we gave it, meaning that Ridge should require a much higher alpha than 10,000 to produce the lowest MSE. We decided to also stop fine-tuning the alpha for ElasticNet because LASSO should always be the better model.

Since the lowest MSE for LASSO appeared at alpha = 1 for the general tune range below, we were able to fine-tune it further to produce an MSE of -430.625 at alpha = 1.9 over 10 folds.

Hyperparameters Used

tune_range = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 10, 100, 1000, 10000]

lasso_tune_range = [1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3]

param_l1_ratio = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

Results

Regularization Method	Ridge (L1)	LASSO (L2)	ElasticNet

MSE	-546.535 @ alpha = 10,000	-464.626 @ alpha = 1	-472.926 @ alpha = 1, L1 ratio = 0.9

Regularization Method	LASSO (L2)
MSE	-430.625 @ alpha = 1.9

Conclusion

We decided to conclude with the LASSO model because of the performance as explained above, and the reduction of collinearity.

The most important variable in the lasso model were the following:

- 'wtd_std_ThermalConductivity': 11.16443191217042
- 'Ba': 7.711433130954184

These 2 variables are the only ones that have a reasonably high weight, compared to the one below it with the third having a coefficient of 3.266, and are the most important variables to predict critical superconductor temperature.

Because we had the necessity to normalize our data due to different distribution of the values in the variables. In order to provide appropriate variables to compare in the model, we interpret the variables accounting for our normalized data. We used the standard scalar for normalization. This Scaler removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range). The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile). This can be further inferenced in our Jupyter Notebook.

Therefore, the interpretation of the above variables are the following:

- For every one increase in wtd_std_ThermalConductivity, we can predict a mean increase in the critical superconductor temperature of 11.164, when keeping other variables constant.
- For every one increase in Ba, we can predict a mean increase in the critical superconductor temperature of 7.711, when keeping other variables constant.

Moving forward, future prospective studies should concentrate on these 2 variables to predict the critical high temperature in superconductors.

Appendix

```
def find_score_and_best(X, y):
    # seed = 12
    lasso = make_pipeline(StandardScaler(), Lasso(random_state=12))
```

```
    # ridge = make_pipeline(StandardScaler(), Ridge(random_state=12))
    # elastic = make_pipeline(StandardScaler(), ElasticNet(random_state=12))

    # tune_range = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 0.2, 0.3, 0.4, 0.5,
    #               0.6, 0.7, 0.8, 0.9, 1, 10, 100, 1000, 10000]
    tune_range = [1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2,
                  2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3]
    # l1_ratio_range = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

    lasso_grid_range = [{'lasso__alpha': tune_range}]
    # ridge_grid_range = [{'ridge__alpha': tune_range}]
    # elastic_grid_range = [{'elasticnet__alpha': tune_range,
    #                        'elasticnet__l1_ratio': l1_ratio_range}]

    gs_lasso = GridSearchCV(estimator=lasso, param_grid=lasso_grid_range,
                            scoring='neg_mean_squared_error', cv=10, n_jobs=-1)
    gs_lasso.fit(X, y)
    # gs_ridge = GridSearchCV(estimator=ridge, param_grid=ridge_grid_range,
    #                         scoring='neg_mean_squared_error',
    #                         cv=10, n_jobs=-1)
    # gs_ridge.fit(X, y)
    # gs_elastic = GridSearchCV(estimator=elastic,
    #                           param_grid=elastic_grid_range,
    #                           scoring='neg_mean_squared_error',
    #                           cv=10, n_jobs=-1)
    # gs_elastic.fit(X, y)

    print("\nLasso: Final average loss was ", gs_lasso.best_score_,
          "at", gs_lasso.best_params_, "over", gs_lasso.n_splits_, "folds.\n")
    # print("\nRidge: Final average loss was ", gs_ridge.best_score_,
```

```
    #       "at", gs_ridge.best_params_, "over",
    #       gs_ridge.n_splits_, "folds.\n")
    # print("\nElastic: Final average loss was ", gs_elastic.best_score_,
    #       "at", gs_elastic.best_params_, "over",
    #       gs_elastic.n_splits_, "folds.\n")

    lasso_weights = {X.columns[k]: abs(v)
                     for k, v in
                     enumerate(gs_lasso.best_estimator_['lasso'].coef_)}

    most_important = dict(sorted(lasso_weights.items(),
                                 key=lambda item: item[1], reverse=True)[:10])
    print(most_important)

if __name__ == "__main__":

    do_my_study()
```