Timothy Cabaza

May 2024

Case Study1

**Problem Statement**:

The aim of this case study is to build a linear regression model using L1 (Lasso), L2 (Ridge), and ElasticNet. The task is to predict the Critical Temperature as closely as possible using the Super-Conductor dataset. In addition, the top 5 features that are most important and the best regularization parameter for each model should be found.

**Exploratory Data Analysis:**

The super-conductor dataset was loaded. There were two files that the client has asked that we combine. While rare, in this case the data is clean with no missing data and thus no imputations to consider.

The traincsv was loaded and inspected. The column dytpes are numeric, there are no null values and there 82 columns with 21263 rows of data (Figure 1). The describe function was also used but given the 82 columns it would be an inefficient use of space to include in our table below.

|  |
| --- |
|  |
| **Figure 1: train dataset loaded and inspected**  Utilized pandas functions such as info(), shape, and describe to inspect the first of two datasets for Super Conductors. |

The second dataset was loaded, unique\_m csv, and inspected in a similar fashion.

|  |
| --- |
|  |
| **Figure 2: unique\_m dataset loaded and inspected.**  Utilized pandas functions such as info(), shape, and describe to inspect the second of two datasets for Super Conductors. |

Once the data was loaded and inspected for any missing values, appropriate data type, and shape, the columns for each file were identified. Before combining and splitting the super conductor datasets into target variable and features, duplicate columns in both datasets (‘critical\_temp’) were found and one was dropped before joining. The 'material' column was dropped from the unique csv as it is a composite of all the other features in the data and it would be redundant to include as a feature in the models. To perform drops

|  |
| --- |
|  |
| **Figure 3: Identifying duplicative and redundant columns**  Before joining the train csv and the unique csv the drop and columns functions were utilized to identify duplicative and redundant columns. |

The client has asked that both datasets be joined so that we have one joined dataset to train and evaluate our models on. The pandas concat function was called to join the two dataframes. The columns and head functions were then used to once again verify the data was joined appropriately. The shape function was utilized to check the number of rows and columns in the dataframe (21263, 168).

|  |
| --- |
|  |
| **Figure 4: Joining datasets and checking columns and shape** |

The joined dataframe (joined\_df) still had the target column (‘critical\_temp) in the dataframe, so the target variable was created and the target column 'critical\_temp' isolated and dropped using the drop function.

|  |
| --- |
|  |
| **Figure 5: Target Variable dropped from the joined dataframe.**  The target variable we use double brackets on the line *target = joined\_df[['critical\_temp']]* so that the target is a pandas dataframe rather than pandas series, which allows us to use the columns function to print the column name 'critical\_temp'. |

The target and the features (joined\_features) are now in their own respective dataframes. Before moving forward, the data is then visualized to view the distribution (normal, skewed, bimodal, central tendancy, spread, outliers) and extract insights into the feature relationships.

|  |
| --- |
|  |
| **Figure 6: Distribution of target variable**  The distribution is right skewed. Utilized matplotlib histogram to create visualization. |

The histogram function was called with 50 bins and the histogram for the target was generated. The histogram indicates that the target 'critical\_temp' is right skewed with most of the temperatures clustered at the lower end with a long tail. The takeaway here is that that the critical temperature for most of the materials are usually on the lower end with a only a few higher critical temperature values. The temperatures above ~90 Kelvin on the right end of the tail of the distribution could possibly impact the performance of the regression models. However, we will proceed with no transformation on the target (dependent) variable to preserve interpretability, any transformation would make it difficult to understand the predictions of the model in the context of the problem statement.

Given that our joined features dataframe has a shape of (21263, 167), histograms or any other type of visual would be inefficient in the report, instead the correlation of the features versus the target variable and the description of the data is utilized to assess our features and the next steps.

|  |
| --- |
|  |
| **Figure 7: Top 20 feature correlations with the target variable and description of features 10 out of the 167 features**  Tabulate package was imported to create organized tables. |

The resulting table (Figure 7) illustrates that the features need to be normalized as we have a wide range between min and max values as well as high standard deviations. To address the spread in values and variation in averages the standard scaler package was utilized so that the models will not give undue importance to features with larger values. The scaling will transform the features into a comparable range with a mean of zero and a standard deviation of one. By scaling the data our models should theoretically perform better as the models assumes that the features are centered around zero and have a similar scale.

|  |
| --- |
|  |
| **Figure 8: Features after scaling using Standard Scaler.** |

**Modeling:**

|  |
| --- |
| **L1-Lasso**  **First attempt:**    alphas = np.logspace(-6, 1, 50) # going from to with 50 values  **Second attempt:**    alphas = np.logspace(-10, 1, 20) # going from to with 50 values  **Final Model:**    alphas = np.logspace(-10, 1, 20) # going from to with 20 values |
| **Figure 9: The alpha parameter is grid searched over a log space to find the optimal regularization parameter.**  The top 5 features are listed and the Best Model are captured in the tables. The logspace was adjusted on the second attempt and third attempt and the MSE slightly improved while the top 5 most important features remained the same. |

|  |
| --- |
| **L2-Ridge**  **First attempt:**    alphas = np.logspace(-6, 1, 50) # going from to with 50 values  **Second attempt:**    alphas = np.logspace(-6, 2, 50) # going from to with 50 values  **Third attempt:**    alphas = np.logspace(-6, 12, 20) # going from to with 20 values  **Final Model:**    alphas = np.logspace(1, 30, 20) # going from to with 20 values |
| **Figure 10: The alpha parameter is grid searched over a log space to find the optimal regularization parameter.**  The top 5 features are listed and the Best Model are captured in the tables. The number of iterations was reduced from 2000 to 1000, and the alpha logspace was adjusted through four attempts as the best alpha returned for the first 3 were nearly at the end of the spectrum. Interestingly as the alpha logspace was adjusted the top 5 features changed slightly. |

|  |
| --- |
| **Elastic Net**  **First Attempt:**  *Analysis timed out, too large a space to search.*  **Second Attempt:**    alphas = np.logspace(-2, 1, 20) # going from to with 20 values  l1\_ratios = np.linspace(0, 1, 20) # going from 0 to 1 with 20 values  **Final Attempt:**    alphas = np.logspace(-4, 2, 20) # going from to with 20 values  l1\_ratios = np.linspace(0, 1, 20) # going from 0 to 1 with 20 values |
| **Figure 11: The alpha parameter is grid searched over a log space to find the optimal regularization parameter.**  The top 5 features are listed and the Best Model are captured in the tables. The first attempt timed out because the grid search was too large. The second and third attempt reduced the grid search for the alpha parameter. The top 5 most important features remained the same despite the change in the alpha logspace. The l1\_ratio that was also grid-searched and settled on 1, which means this ElasticNet model was essentially an l1(lasso) model. |

**Summary:**

In conclusion, the best performing model was the l2 (Ridge) model with a regularization parameter of 11288.4 and MSE of 405.986. The model ranked the following as the top 5 most important variables: Ba, wtd\_std\_Valence, wtd\_mean\_ThermalConductivity, Ca, wtd\_std\_ThermalConductivity. Each models grid search range (logspace) was adjusted and tested to find the most optimal regularization value. The best l1 (Lasso) model and l2 (Ridge) model had drastically different regularlization parameters with the l1 best alpha being ~0.2. This is to be expected due to the fact that l1 regularilization has the ability to drive some coefficients to zero and thus the low alpha implies that many of the features were found to be weak contributors. L2 on the other hand had a high regularlization parameter (11288.4) as this method does not drive coefficients to zero and instead penalizes for the number of features in the model. L2’s high alpha value also indicates multicollinearity between features and captures these complex relationships in the dataset. ElasticNet performed as well as the best l1 (lasso) model but this is due to the fact that the gridsearched l1\_ratio

|  |
| --- |
|  |
| **Figure 12: Best Model l2 Ridge** |

**Appendix:**

|  |  |
| --- | --- |
| Code: | # To begin, we first load our imports necessary to run our models, normalize the data, perform cross validation, and a grid search for the regularization strength parameter alpha (aka C).  import pandas as pd  from sklearn.preprocessing import StandardScaler  from sklearn.linear\_model import LinearRegression  from sklearn.linear\_model import Lasso  from sklearn.linear\_model import Ridge  from sklearn.model\_selection import cross\_val\_score  import matplotlib.pyplot as plt  import seaborn as sns  from sklearn.model\_selection import GridSearchCV  from tabulate import tabulate  from sklearn.metrics import mean\_squared\_error, accuracy\_score  import pandas as pd  pd.set\_option('display.max\_rows', None) # Display all rows  pd.set\_option('display.max\_columns', None) # Display all columns  # The data was downloaded from the SMU \_\_\_ website and then the file paths for both files are a assigned a variable (filepath and filepath2).  filepath = "/Users/tmc/Desktop/MS\_SMU\_Admin/05\_2024Summer/QUANTIFIYING\_TW/02\_module/Case\_study1/train.csv" # one dot current directory, two dots means the parent directory one level up from the current directory  filepath2 = "/Users/tmc/Desktop/MS\_SMU\_Admin/05\_2024Summer/QUANTIFIYING\_TW/02\_module/Case\_study1/unique\_m.csv"  # Now that the the files are assigned a variable name, the data is then converted to a pandas dataframe using the pd.read\_csv function.  # excel 1  traincsv = pd.read\_csv(filepath)  traincsv.info()  traincsv.dtypes  traincsv.shape  traincsv.describe()  # excel 2  uniquemcsv = pd.read\_csv(filepath2)  uniquemcsv.info()  uniquemcsv.dtypes  uniquemcsv.shape  # Once the data was loaded, the columns for each file were identified. The data is clean and there are no missing values so no imputation is needed. To verify there are no missing values we use the isnull() function. Missing values can create errors when passing the data training a model.    traincsv.columns  uniquemcsv.columns  if traincsv.isnull().values.any() or uniquemcsv.isnull().values.any() :  print("There are missing values in the data.")  else:  print("The data is clean and there are no missing values.")  # The columns 'critical\_temp' and 'material' are dropped from the second dataset (uniquecsv) as the 'critical\_temp' column is already located in the first the dataset (traincsv). The 'material' column is a composite of all the other features in the data so it would be redundant to include as a feature in the models.  df2\_unique = uniquemcsv.drop(columns=['critical\_temp', 'material'])  # The columns function was then used to verify that the appropriate columns were in fact dropped from the dataframe.  df2\_unique.columns  # The client has asked that both datasets be joined so that we have one joined dataset to train and evaluate our models on. The pandas pd.concat function is called to join the data frames. The columns and head functions is then used to once again verify the data was joined appropriately. The shape function was utlized to check the number of rows and columns in the dataframe (21263, 168).  joined\_df = pd.concat([traincsv, df2\_unique], axis=1)  print(joined\_df)  # pd.set\_option('display.max\_columns', None)  print(joined\_df.head(10))  print(joined\_df.shape)  print(joined\_df.columns)  #############  #look at your data  joined\_df.describe()  joined\_df.info()  joined\_df.dtypes  joined\_df.shape  # look at correlation  ###############  # The joined dataframe (joined\_df) is now ready for use, however we still had the target column in the dataframe, so the target variable was created and the target column 'critical\_temp' isolated and dropped using the drop function. Note for the target variable we use double brackets on the line target = joined\_df[['critical\_temp']] so that the target is a pandas dataframe rather than pandas series, which allows us to use the columns function to print the column name 'critical\_temp'.  target = joined\_df[['critical\_temp']] # our target variable  joined\_features = joined\_df.drop(columns=['critical\_temp'])  print(joined\_features.columns)  print(target.columns)  # The target and the features (joined\_features) are now in their in own respective dataframes. Before moving forward, the data is then visualized to view the distribution (normal, skewed, bimodal, central tendancy, spread, outliers) and extract insights into the feature relationships.  # The histogram function was called with 50 bins and the histogram for the target was generated. The histogram indicates that the target 'critical\_temp' is right skewed with a majority of the temperatures clustered at the lower end with a long tail. The takeaway here is that that the critical temperature for most of the materials are usually on the lower end with a only a few higher critical temperature values. The temperatures above 90 \_\_ on right end of the tail of the distribution could possibly impact the performance of the regression models. However, we will proceed with no transformation on the target.  plt.hist(target, bins=50)  plt.xlabel("Critical Temperature")  plt.ylabel("Frequency")  plt.title("Distribution of Critical Temperature")  plt.show()  # We have 167 columns and visualizing them via a histogram individually is not the not the most efficient way to gain insights into the data. Instead the describe function is utilized which provides summary statistics for each feature in the dataframe. The output is transposed to provide easier viewing but it still did not look quite right for a report, so after some investigating a package called tabulate was utilized to create a table that was more appropriate for the report and the number of decimal points were reduced to two.  # plt.hist(joined\_features, bins=10)  # plt.show()  # Calculate correlations for your DataFrame  # lets review the correlation structure of the variables to the target  correlations = joined\_df.corr()  # Filter for correlations with the target variable above 0.8  target\_correlations = correlations['critical\_temp']  target\_correlations = target\_correlations[target\_correlations.abs() > 0.5]  top\_correlations = target\_correlations.sort\_values(ascending=False)[:20]  # Format the result for better display (with only two columns)  top\_correlations\_df = pd.DataFrame(top\_correlations).reset\_index()  top\_correlations\_df.columns = ['Feature', 'Correlation']  # Print the table with top correlations  print(tabulate(top\_correlations\_df, headers='keys', tablefmt='psql', floatfmt=".2f"))  joined\_features.describe().T  summary\_stats = joined\_features.describe().T.applymap(lambda x: f"{x:.2f}") # Format data  print(tabulate(summary\_stats, headers='keys', tablefmt='psql', floatfmt=".2f"))  summary\_stats\_top10 = summary\_stats.iloc[:10]  print(tabulate(summary\_stats\_top10, headers='keys', tablefmt='psql', floatfmt=".2f"))  # The resulting table illustrates that the features need to be normalized as we have a wide range between min and max values as well as high standard deviations. To address the wide spread in values and variation in averages for the features the standard scaler package was utilized so that the models will not give undue importance to features with larger values. The scaling will transform the features into a comparable range with a mean of zero and a standard deviation of one. By scaling the data our models should theoretically perform better as the models assumes that the features are centered around zero and hae a similar scale.  scale = StandardScaler()  X\_scaled = pd.DataFrame(scale.fit\_transform(joined\_features))  # plt.hist(X\_scaled, bins=10)  # plt.show()  summary\_stats\_scaled = X\_scaled.describe().T.applymap(lambda x: f"{x:.2f}") # Format data  print(tabulate(summary\_stats\_scaled, headers='keys', tablefmt='psql', floatfmt=".2f"))  summary\_stats\_scaled\_top10 = summary\_stats\_scaled.iloc[:10]  print(tabulate(summary\_stats\_scaled\_top10, headers='keys', tablefmt='psql', floatfmt=".2f"))  # In creating a linear model using lasso (l1 regularization), the alpha hyperparameter is the most critical parameter to explore. The alpha parameter controls the regularlization strength with higher values reducing the least important coefficients to zero, hence feature selection. The max-iter is another parameter that can be used to optimize the algorithm and was adjusted so that model convereged.  # To perform a grid search on the regularlization strength, the GridSearchCV class was utilized with the scoring metric negative mean squared error to minimize the MSE. The score generated is then converted back to MSE by taking the absolute value to make it more interpratble.  #############################################################  # 01 Lasso  #############################################################  import numpy as np  from sklearn.model\_selection import GridSearchCV, KFold  from sklearn.linear\_model import Lasso  # alphas = np.logspace(-6, 1, 50) # going from 10^6 to 10^1 with 50 samples in logspace  alphas = np.logspace(-10, 2.5, 20)  l1\_model = Lasso(alpha=1, max\_iter=2000, random\_state=1)  # Create the parameter grid for GridSearchCV  param\_grid = {'alpha': alphas}  grid\_search = GridSearchCV(l1\_model, param\_grid, scoring='neg\_mean\_squared\_error', cv=5, n\_jobs=-1)  grid\_search.fit(X\_scaled, target)  # Get the best alpha and its corresponding model  best\_alpha = grid\_search.best\_params\_['alpha']  best\_model = grid\_search.best\_estimator\_  best\_score = abs(grid\_search.best\_score\_)  # Determine top 5 features  feature\_importances = pd.DataFrame({'Feature': joined\_features.columns, 'Importance': np.abs(best\_model.coef\_)})  top\_5\_features = feature\_importances.nlargest(5, 'Importance')  results\_data = [  ["Best Alpha", best\_alpha],  ["Best Model Coefficients", best\_model.coef\_],  ["Best Model MSE", best\_score],  ]  headers = ["Metric: abs(MSE)", "Score"]  print(tabulate(results\_data, headers=headers, tablefmt="fancy\_grid"))  table\_data = top\_5\_features.values.tolist()  headers = top\_5\_features.columns.tolist() # Get column names as headers  print(f"Best alpha: {best\_alpha:.6f}")  print(f"Best model MSE: {best\_score:.4f}")  print("\nTop 5 Features:")  print(tabulate(table\_data, headers=headers, tablefmt="fancy\_grid"))  table\_data = [  ["Best Alpha", best\_alpha],  ["Best Model MSE", best\_score]  ]  # Add top 5 features to the table  for \_, row in top\_5\_features.iterrows():  table\_data.append([row['Feature'], row['Importance']])  headers = ["Metric/Best Alpha/Feature", "Value/Importance"]  print(tabulate(table\_data, headers=headers, tablefmt="fancy\_grid"))  #############################################################  # L2 Ridge -- start here Monday to complete  #############################################################  # Define the Ridge model  ridge\_model = Ridge(max\_iter=2000, random\_state=1)  # Create the parameter grid for GridSearchCV  # alphas = np.logspace(-6, 1, 50) # going from 10^6 to 10^1 with 50  # alphas = np.logspace(-6, 2, 50) # going from 10^(-6) to 10^2 with 50  # alphas = np.logspace(-6, 12, 20) # going from 10^(-6) to 10^(2.5) with 10 samples in logspace  alphas = np.logspace(1, 30, 20)  param\_grid = {'alpha': alphas}  # Create the GridSearchCV object  grid\_search = GridSearchCV(ridge\_model, param\_grid, scoring='neg\_mean\_squared\_error', cv=5, n\_jobs=-1)  # Perform the grid search  grid\_search.fit(X\_scaled, target)  # Get the best alpha and its corresponding model  best\_alpha\_ridge = grid\_search.best\_params\_['alpha']  best\_model\_ridge = grid\_search.best\_estimator\_  best\_score\_ridge = abs(grid\_search.best\_score\_)  ########  # Determine top 5 features  feature\_importances\_ridge = pd.DataFrame({'Feature': joined\_features.columns, 'Importance': np.abs(best\_model\_ridge.coef\_.ravel())})  top\_5\_features\_ridge = feature\_importances\_ridge.nlargest(5, 'Importance')  results\_data\_ridge = [  ["Best Alpha", best\_alpha\_ridge],  ["Best Model Coefficients", best\_model\_ridge.coef\_],  ["Best Model MSE", best\_score\_ridge],  ]  headers = ["Metric: abs(MSE)", "Score"]  print(tabulate(results\_data\_ridge, headers=headers, tablefmt="fancy\_grid"))  table\_data\_ridge = top\_5\_features\_ridge.values.tolist()  headers= top\_5\_features\_ridge.columns.tolist() # Get column names as headers  print(f"Best alpha: {best\_alpha\_ridge:.6f}")  print(f"Best model MSE: {best\_score\_ridge:.4f}")  print("\nTop 5 Features:")  print(tabulate(table\_data\_ridge, headers=headers, tablefmt="fancy\_grid"))  table\_data\_ridge = [  ["Best Alpha", best\_alpha\_ridge],  ["Best Model MSE", best\_score\_ridge]  ]  # Add top 5 features to the table  for \_, row in top\_5\_features\_ridge.iterrows():  table\_data\_ridge.append([row['Feature'], row['Importance']])  headers = ["Metric/Best Alpha/Feature", "Value/Importance"]  print(tabulate(table\_data\_ridge, headers=headers, tablefmt="fancy\_grid"))  #############################################################  # ElasticNet  #############################################################  # Define the Elastic Net model  from sklearn.linear\_model import ElasticNet  # elastic\_net\_model = ElasticNet(max\_iter=2000, random\_state=1)  elastic\_net\_model = ElasticNet(max\_iter=100, random\_state=1)  # Create the parameter grid for GridSearchCV  # alphas = np.logspace(-6, 1, 10) # going from 10^6 to 10^1 with 10 samples in logspace  alphas = np.logspace(-2, 1, 10)  alphas = np.logspace(-4, 2, 15)  # l1\_ratios = np.linspace(0, 1, 10)  l1\_ratios = np.linspace(0, 1, 20)# going from 0 to 1 with 10 samples in logspace  param\_grid = {'alpha': alphas, 'l1\_ratio': l1\_ratios}  # Create the GridSearchCV object  grid\_search = GridSearchCV(elastic\_net\_model, param\_grid, scoring='neg\_mean\_squared\_error', cv=5, n\_jobs=-1)  # Perform the grid search  grid\_search.fit(X\_scaled, target)  # Get the best alpha and its corresponding model  best\_alpha = grid\_search.best\_params\_['alpha']  best\_l1\_ratio = grid\_search.best\_params\_['l1\_ratio']  best\_model = grid\_search.best\_estimator\_  best\_score = abs(grid\_search.best\_score\_)  ########  results\_data = [  ["Best Alpha", best\_alpha],  ["Best Model Coefficients", best\_model.coef\_],  ["Best Model MSE", best\_score],  ]  headers = ["Metric: abs(MSE)", "Score"]  print(tabulate(results\_data, headers=headers, tablefmt="fancy\_grid"))  table\_data = top\_5\_features.values.tolist()  headers = top\_5\_features.columns.tolist() # Get column names as headers  print(f"Best alpha: {best\_alpha:.6f}")  print(f"Best model MSE: {best\_score:.4f}")  print("\nTop 5 Features:")  print(tabulate(table\_data, headers=headers, tablefmt="fancy\_grid"))  table\_data = [  ["Best Alpha", best\_alpha],  ["Best l1\_ratio", best\_l1\_ratio],  ["Best Model MSE", best\_score]  ]  # Add top 5 features to the table  for \_, row in top\_5\_features.iterrows():  table\_data.append([row['Feature'], row['Importance']])  headers = ["Metric/Best Alpha/Feature", "Value/Importance"]  print(tabulate(table\_data, headers=headers, tablefmt="fancy\_grid")) |