Case Study I

Randy Kim, Kati Schuerger, Will Sherman

September 8, 2022

1 Introduction

Objective

The objective of this case study is to build a linear regression model using regularization methods, to predict the *critical temperature* of different materials as potential superconductors. In addition, our analysis will seek to identify which variables carry the most importance in predicting the *critical temperature* (the temperature at which materials may become new superconductors).

Dataset

The dataset that we are using was brought to us from a group of scientists that are looking at superconductors. Superconductors are materials that give little or no resistance to electrical current.

Data points include material composition (chemical elements), critical temperature (the temperature at which each will superconduct), atomic mass, thermal conductivity, and many others. We confirmed that the datasets do not have any missing values.

2 Methods

Normalization

We used SKLEARN StandardScaler to transform the data to measure features on the same scale. Standard Scaler is a method of standardizing the data, transforming it so that each feature has 0 mean and a standard deviation of 1 (standard normal distribution) (1).

$$Z = \frac{(x - \overline{x})}{s} \tag{1}$$

This pre-processing step is important: normalizing the distributions of the features. Not doing so may lead to incorrect conclusions regarding feature importance— true feature importance might be hidden behind the scale of the data—and, thereby, the model's fit. Passing equally weighted and distributed features into the model generally creates a model that more accurately reflects the relationship between each of the explanatory (feature) variables and the target variable (*critical temperature*).

Overfitting

One of the challenges of statistical models is overfitting, also known as statistical bias. An overfit model would be over-tuned to make predictions based solely on the feature values that exist in the dataset used to train the model. An overfit model may perform well on the training dataset but will likely not do as well when making predictions of the target variable using new data (*i.e.*, the model will not generalize well). Because the task is to predict *new* superconductors, a generalizable model is essential.

Train/Test/Validate split
Before we began modeling, we randomized the arrangement of the data samples and separated the data into 2 groups: a training dataset (to fit the model), and a validation dataset (holdout group of 10%) that was used to measure model performance on unseen data—evaluating whether the model generalizes well.

Scoring metric
The task was to create a regression model for critical temperature prediction, so we used mean squared error (MSE) in combination with 5-fold cross-validation to get model scores and compare models. Cross-validation is a way to assess different portions of the data using resampling to identify the model parameters. This was used with regularized linear regression, which may utilize a loss function as a method of avoiding overfitting. We iterated through the value of alpha (the penalty term) to generate the optimal loss function prior to fitting the regularized linear model.

Regularization methods to combat overfitting
To help us in guarding against overfitting our model, we used regularization tools, namely, LASSO (L1) regularization, and Ridge (L2) regularization.

Regularization penalty (L1) for feature selection
LASSO is an acronym that stands for Least Absolute Shrinkage and Selection Operator (*i.e.*, L1) and is useful for feature selection. We will use L1 regularization to identify the features that have the most importance in the model. The coefficients represent the slopes of the difference planes or lines that we are trying to fit. L1 introduces sparsity, meaning that it will drive the coefficients of some of the variables (less important features, features with weaker interactions with the target variable) to 0.
One key goal of L1 and L2 regularization was to introduce the concept of a penalty as the value of those slopes increase. And while we say the value increases, we do not care if that value is getting larger in the positive scale or the negative scale *(i.e.*, the magnitude).
As this penalty increases, we will have two competing terms. One in which as the slope grows larger, our loss will decrease. And another, as the slope grows larger, the loss will increase. That second term with the increasing loss is our penalty and will be known as regularization. The L1 penalty term equation is below (2).

$$\lambda \sum_{j=0}^{k} |m_j| \qquad (2)$$

L2 (Ridge) Regression
Ridge regression provides a more general method to prevent overfitting. Ridge regression does not induce sparsity, meaning that coefficients are not driven to 0 independently of each other. Instead, Ridge regression penalizes feature coefficients in a more uniform manner, meaning that weak interactions are not suppressed before strong interactions.

3 Results

Penalty (alpha)

We manipulated the value of alpha to identify the penalty value that gives us the optimal loss function (Figure 1).

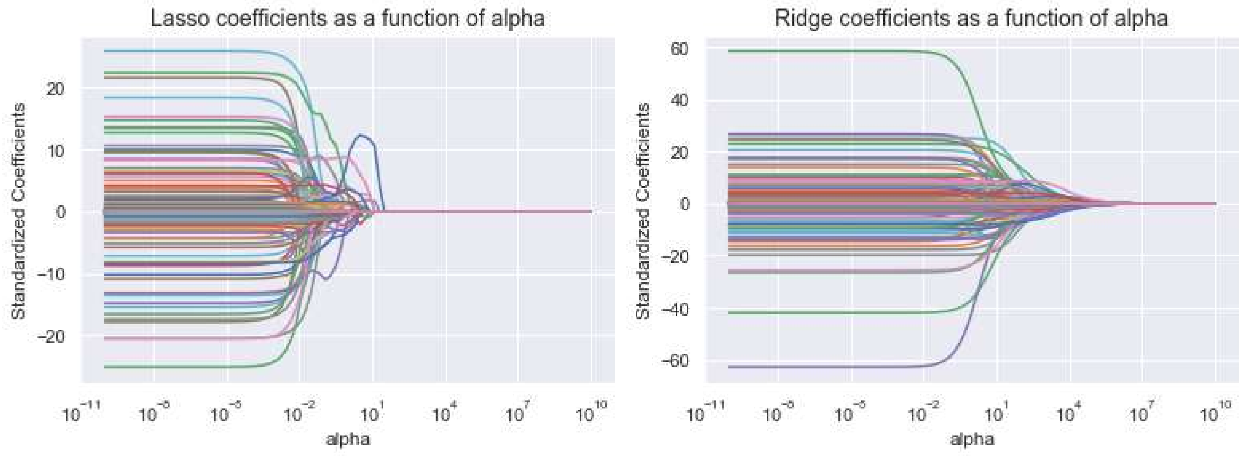

Figure 1. Depiction of the optimization of the regularization coefficient, alpha, for (a) Lasso regularization and (b) Ridge regularization.

The L1 alpha optimized the MSE value when alpha was 0.1963, which can be seen to take our feature space from 170 features and collapses it to just a few (Figure 1a). The same is true of the L2 optimization, though the alpha optimized to 1353.0478 and included a larger number of features. The total number of remaining features for the L1 model was 78 features while the total number of features used in the L2 model was 158 features.

Feature importance

Based on the comprehensive list of features that were provided, we opted to leave in all possible features for first pass regularization. Using L1, we identified the top five features for the general linear model (Table 1).

weights	features
11.69839	wtd_mean_ThermalConductivity
10.33756	wtd_gmean_ThermalConductivity
8.47517	Ba
6.86237	range_atomic_mass
4.60242	wtd_std_Valence

Table 1. Top five features for the L1 regularized linear model and their coefficients (weights).

There was some overlap between the top five features between L1 and L2—metrics regarding thermal conductivity being important for accurately predicting critical temperature (Table 2).

weights	features
7.68874	Ba
5.47554	wtd_mean_ThermalConductivity
4.98043	wtd_std_Valence
4.73911	wtd_std_ThermalConductivity
4.23346	wtd_gmean_ThermalConductivity

Table 2.Top five features for the L2 regularized linear model and their coefficients (weights).

The feature space is still quite large for both models (78 and 158 features); therefore, we took the opportunity to adjust the required minimum threshold for the value of the coefficients. Adjusting the threshold to be 1.000 or larger creates a more explainable model with 28 features and 50 features for L1 and L2, respectively.

Model Comparison

Based on the unadjusted models with full feature space, we were able to generate outputs of the models against the unseen 10% holdout data. As can be seen (Table 3), including almost the full feature space, the MSE is lower than reducing features. This is further exhibited when selecting only those features with coefficients greater than 1.000 to include in the model (Table 4). However, this produces a feature space that is more manageable.

regularization model	MSE	number of features
Lasso	314.47711	78
Ridge	310.26005	158

Table 3. Linear regression model comparison based on number of included features; mean squared error (MSE) evaluated against the validation set (size 10% of total data).

regularization model	MSE	number of features
Lasso	324.24592	28
Ridge	322.90568	50

Table 4. Linear regression model comparison based on number of included features with minimum coefficient threshold (>1.0); mean squared error (MSE) evaluated against the validation set (size 10% of total data).

Excluding coefficients which fall below the specified weight worsened the MSE. However, there is a significant decrease in the remaining features to be evaluated. Additionally, the total increase in error on the holdout data represents only a 3.11% increase (comparing L1 regularization MSEs).

4 Conclusion

The general linear model proposed here is based on feature selection with minimum weight 1.000. The remaining features of interest include 19 chemical/atomic properties and 9 elements of interest (Appendix) which contributed to the chemical formula (*material* column from original dataset).

Note, these are hypothetical models only, practical application may not be relevant, given new information, lab capabilities, material properties, *etc*. Future modeling should include a collaborative effort for feature selection in tandem with individuals from the original scientific team or materials engineers; certain features are correlated (*e.g.*, *wtd_mean_ThermalConductivity* & *wtd_gmean_ThermalConductivity*) and would benefit from preclusion prior to utilizing modeling techniques.

Appendix

Features from L1 regularized linear model with weights greater than 1:

- wtd_entropy_atomic_mass
- range_atomic_mass
- wtd_range_atomic_mass
- wtd_std_atomic_mass
- gmean_fie
- range_atomic_radius
- gmean_Density
- entropy_Density
- wtd_range_Density
- wtd_gmean_ElectronAffinity
- wtd_entropy_ElectronAffinity
- std_ElectronAffinity
- wtd_entropy_FusionHeat
- wtd_std_FusionHeat
- wtd_mean_ThermalConductivity
- wtd_gmean_ThermalConductivity
- wtd_entropy_ThermalConductivity
- wtd_std_ThermalConductivity
- wtd_std_Valence
- Si
- S
- Cl
- Ca
- Ag
- Ba
- Hg
- Tl
- Bi

Code

```
# %%
# read in the data -- blah
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression, Ridge, Lasso
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score

# %%
### Will - read data ###
data_1 = pd.read_csv(r'C:\Users\sherm\Documents\Grad School - Classes\MSDS -
7333 - Quantifying the World\Case Study 1\superconduct\train.csv')
### Randy - read data ###
```

```
# data_1 = pd.read_csv("train.csv")
### Kati - read data ###
# data_1 =

print(data_1.shape)
print(data_1.columns)
data_1.head()

# %%
### Will - read data ###
data_2 = pd.read_csv(r'C:\Users\sherm\Documents\Grad School - Classes\MSDS -
7333 - Quantifying the World\Case Study 1\superconduct\unique_m.csv')
### Randy - read data ###
# data_2 = pd.read_csv("unique_m.csv")
### Kati - read data ###
# data_2 =

print(data_2.shape)
print(data_2.columns)
data_2.head()

# %%
print(data_1.isnull().values.any())
print(data_1.isna().values.any())

# %%
print(data_2.isnull().values.any())
print(data_2.isna().values.any())

# %%
data_raw = pd.merge(data_1, data_2, left_index=True, right_index=True)
print(data_raw.shape)
print(data_raw.columns)
data_raw.head()

# %%
data_raw = data_raw.drop(['material'], axis=1)

# %%
y_data = pd.DataFrame(data_raw.loc[:,'critical_temp_y'])
y_data

# %%
x_data = data_raw.loc[:,data_raw.columns != 'critical_temp_y']
x_data = pd.DataFrame(x_data.loc[:,x_data.columns != 'critical_temp_x'])
x_data.head()

# %%
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# %%
x_scaled = scaler.fit_transform(x_data)
x_scaled = pd.DataFrame(x_scaled, columns=x_data.columns)
```

```
x_scaled.head()

# %%
x_scaled.describe()
# mean should be close to 0
# std should be close to 1

# %% [markdown]
# ### train test split (x_scaled)

# %%
xs_train, xs_test, ys_train, ys_test = train_test_split(x_scaled, y_data,
test_size=0.1, random_state=10)

# %% [markdown]
# ### train test split (before scale)

# %%
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data,
test_size=0.1, random_state=10)

# %%
from IPython.core.display import display, HTML
display(HTML("<style>div.output_scroll { height: 14em; }</style>"))
import warnings
warnings.filterwarnings('ignore')

for i in x_data.columns:
    sns.distplot(x_data.loc[:,i]).set(title=i)
    plt.show()

# %%
warnings.filterwarnings('ignore')

for i in x_scaled.columns:
    sns.distplot(x_scaled.loc[:,i]).set(title=i)
#     sns.title(i)
    plt.show()

# %%
features = [x for x in list(x_scaled.columns) if x in list(data_1.columns)]
# features

sns.set_theme(style="white")
# Compute the correlation matrix
corr = x_scaled.loc[:,features].corr()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)
```

```
# Draw the heatmap with the mask and correct aspect ratio
sns.set(rc={'figure.figsize':(20,20)})
sns.heatmap(corr, mask=mask, cmap=cmap, center=0,
            square=True, linewidths=1, cbar_kws={"shrink": 1})

# %% [markdown]
# #### Scaled

# %%
from sklearn.linear_model import Lasso
from sklearn.model_selection import KFold

l1_reg = Lasso()
cv = KFold(shuffle=True, n_splits=5, random_state = 10)

# %% [markdown]
# ### L1 & alpha

# %%
from sklearn.model_selection import cross_val_score

alphas = np.logspace(-10,10,100)
coefs1 = []
best1 = -10000000000
best_alpha1 = 'error'

for a in alphas:
    l1_reg.alpha = a
    out = cross_val_score(l1_reg, xs_train, ys_train,
scoring='neg_mean_squared_error', cv=cv, n_jobs=10).mean()
    l1_reg.fit(xs_train, ys_train)
    coefs1.append(l1_reg.coef_)

    if(out > best1):
        best1 = out
        best_alpha1 = a

print('\tbest MSE\t\tbestalpha\n', best1, '\t', best_alpha1)

# %%
# L1 alphas convergence
ax = plt.gca()

ax.plot(alphas, coefs1)
ax.set_xscale('log')

plt.rcParams["figure.figsize"] = 8,4
plt.axis('tight')
plt.xlabel('alpha')
plt.ylabel('Standardized Coefficients')
plt.title('Lasso coefficients as a function of alpha', fontdict={'fontsize':
14})

# %% [markdown]
# ### L2 & alpha

# %%
l2_reg = Ridge()
coefs2 = []
best2 = -10000000000
best_alpha2 = 'error'

for a in alphas:
    l2_reg.alpha = a
    out = cross_val_score(l2_reg, xs_train, ys_train,
scoring='neg_mean_squared_error', cv=cv, n_jobs=10).mean()
    l2_reg.fit(xs_train, ys_train)
    coefs2.append(l2_reg.coef_.flatten())

    if(out > best2):
        best2 = out
        best_alpha2 = a

print('\tbest MSE\t\tbestalpha\n', best2, '\t', best_alpha2)

# %%
# L2 alpha convergence
ax = plt.gca()

ax.plot(alphas, coefs2)
ax.set_xscale('log')

plt.rcParams["figure.figsize"] = 8,5
plt.axis('tight')
plt.xlabel('alpha')
plt.ylabel('Standardized Coefficients')
plt.title('Ridge coefficients as a function of alpha', fontdict={'fontsize':
14})

# %%
### MODEL FITTING w/ BESTALPHA ###
lasso_model = Lasso(alpha=best_alpha1).fit(xs_train, ys_train)
ridge_model = Ridge(alpha=best_alpha2).fit(xs_train, ys_train)

# %%
print(abs(lasso_model.coef_))
# print("{:.6}".format(lasso_model.intercept_))

# %%
print(abs(ridge_model.coef_))

# %%
imp_Xs_L1 = abs(lasso_model.coef_)
imp_Xs_L1 = (imp_Xs_L1 > 1).astype(int)
imp_Xs_L1 = imp_Xs_L1.astype(np.bool)
# print(imp_Xs)
imp_features = np.array(xs_train.columns)
print(np.count_nonzero(imp_features[imp_Xs_L1]))
print(imp_features[imp_Xs_L1])
```

```
# %%
imp_Xs_L2 = abs(ridge_model.coef_.flatten())
imp_Xs_L2 = (imp_Xs_L2 > 1).astype(int)
imp_Xs_L2 = imp_Xs_L2.astype(np.bool)
# print(imp_Xs)
imp_features = np.array(xs_train.columns)
print(np.count_nonzero(imp_features[imp_Xs_L2]))
print(imp_features[imp_Xs_L2])

# %%
import six

feature_df = pd.DataFrame({'weights':abs(lasso_model.coef_),
                        'features':xs_train.columns})
top5 = feature_df.sort_values('weights', ascending=False).head(5)
top5 = top5.round({'weights':5})

def render_mpl_table(data, col_width=3.0, row_height=0.625, font_size=14,
                     header_color='#40466e', row_colors=['#f1f1f2', 'w'],
edge_color='w',
                     bbox=[0, 0, 1, 1], header_columns=0,
                     ax=None, **kwargs):
    if ax is None:
        size = (np.array(data.shape[::-1]) + np.array([0, 1])) *
np.array([col_width, row_height])
        fig, ax = plt.subplots(figsize=size)
        ax.axis('off')
    mpl_table = ax.table(cellText=data.values, bbox=bbox,
colLabels=data.columns, **kwargs)
    mpl_table.auto_set_font_size(False)
    mpl_table.set_fontsize(font_size)

    for k, cell in mpl_table._cells.items():
        cell.set_edgecolor(edge_color)
        if k[0] == 0 or k[1] < header_columns:
            cell.set_text_props(weight='bold', color='w')
            cell.set_facecolor(header_color)
        else:
            cell.set_facecolor(row_colors[k[0]%len(row_colors) ])
    return ax.get_figure(), ax

fig,ax = render_mpl_table(top5, header_columns=0, col_width=5.0)
fig.savefig("table_mpl.png")

# credit to volodymyr: https://stackoverflow.com/questions/19726663/how-to-
save-the-pandas-dataframe-series-data-as-a-figure

# %%
feature_df = pd.DataFrame({'weights':abs(ridge_model.coef_.flatten()),
                           'features':xs_train.columns})
top5 = feature_df.sort_values('weights', ascending=False).head(5)
top5 = top5.round({'weights':5})

fig,ax = render_mpl_table(top5, header_columns=0, col_width=5.0)
```

```
fig.show()

# %%
### MODEL FITTING w/ BESTALPHA ###
lasso_model = Lasso(alpha=best_alpha1).fit(xs_train, ys_train)
ridge_model = Ridge(alpha=best_alpha2).fit(xs_train, ys_train)

# %%
from sklearn.metrics import mean_squared_error

lm_stat = mean_squared_error(ys_test, lm.predict(xs_test))
lasso_stat = mean_squared_error(ys_test, lasso_model.predict(xs_test))
ridge_stat = mean_squared_error(ys_test, ridge_model.predict(xs_test))

# %%
L1_features = abs(lasso_model.coef_)
L1_features = (L1_features > 0.000001).astype(int)
L1_features = L1_features.astype(np.bool)
np.count_nonzero(imp_features[imp_Xs_L1])

L2_features = abs(ridge_model.coef_.flatten())
L2_features = (L2_features > 0.000001).astype(int)
L2_features = L2_features.astype(np.bool)

model_metrics = pd.DataFrame({'regularization model':['Lasso','Ridge'],
                              'MSE':[lasso_stat,ridge_stat],
                              'number of features':
[np.count_nonzero(imp_features[L1_features]),

np.count_nonzero(imp_features[L2_features])]})
# model_metrics
model_metrics = model_metrics.round({'MSE':5})
fig,ax = render_mpl_table(model_metrics, header_columns=0, col_width=3.0)
fig.savefig("metrics_tbl.png")

# %%
### MODEL FITTING w/ BESTALPHA ###
lasso_model = Lasso(alpha=best_alpha1).fit(xs_train[imp_features[imp_Xs_L1]],
ys_train)
ridge_model = Ridge(alpha=best_alpha2).fit(xs_train[imp_features[imp_Xs_L2]],
ys_train)

lasso_stat = mean_squared_error(ys_test,
lasso_model.predict(xs_test[imp_features[imp_Xs_L1]]))
ridge_stat = mean_squared_error(ys_test,
ridge_model.predict(xs_test[imp_features[imp_Xs_L2]]))

# %%
model_metrics = pd.DataFrame({'regularization model':['Lasso','Ridge'],
                              'MSE':[lasso_stat,ridge_stat],
                              'number of
features':[np.count_nonzero(imp_features[imp_Xs_L1]),

np.count_nonzero(imp_features[imp_Xs_L2])]})
# model_metrics
```

```
model_metrics = model_metrics.round({'MSE':5})
fig,ax = render_mpl_table(model_metrics, header_columns=0, col_width=3.0)
fig.savefig("metrics_tbl_reduced.png")

# %%
### Correlation Check
corr['wtd_mean_ThermalConductivity']['wtd_gmean_ThermalConductivity']

# %%
### Increase in MSE
print((324.24592-314.47711)/314.47711)
```