Ahmed Awadallah

Using L1 and L2 Regularization with Superconductivity Dataset

Introduction

Superconductivity is a state transition where conducting materials go from transmitting current with some resistance(dependent on material) to 0. This occurs at a critical temperature threshold similar to how water will freeze at 0 degrees celsius. Metallic crystals become superconducting generally at low temperatures and it is an active area of research to find more metallic crystals that could be superconductive as well as discover ways to create a superconducting material stable at standard earth temperatures.

Methods

Our data comes from the DS 7333 Case Study 1. Our objective was to use a linear model with regularization in order to predict the critical temperature as accurately as possible as well as provide the features that proved most helpful to the model.

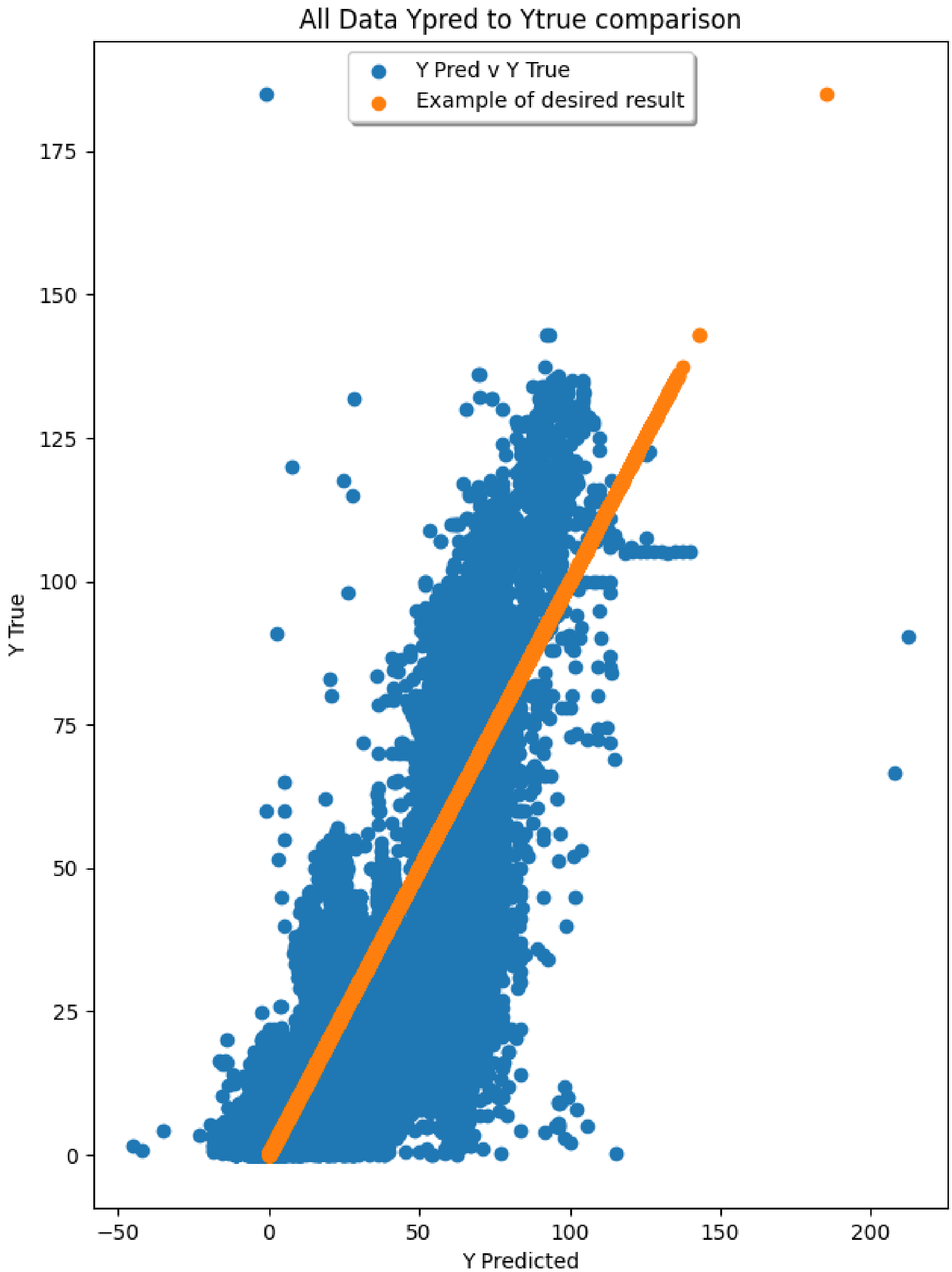
Personally I added a small twist in the spirit of learning where I produce two models. One that utilized Lasso regularization for feature selection to first shrink the dataset then it was put into a model with Ridge regularization to produce the desired result. The second model simply fed the entire dataset into the model with Ridge regularization.

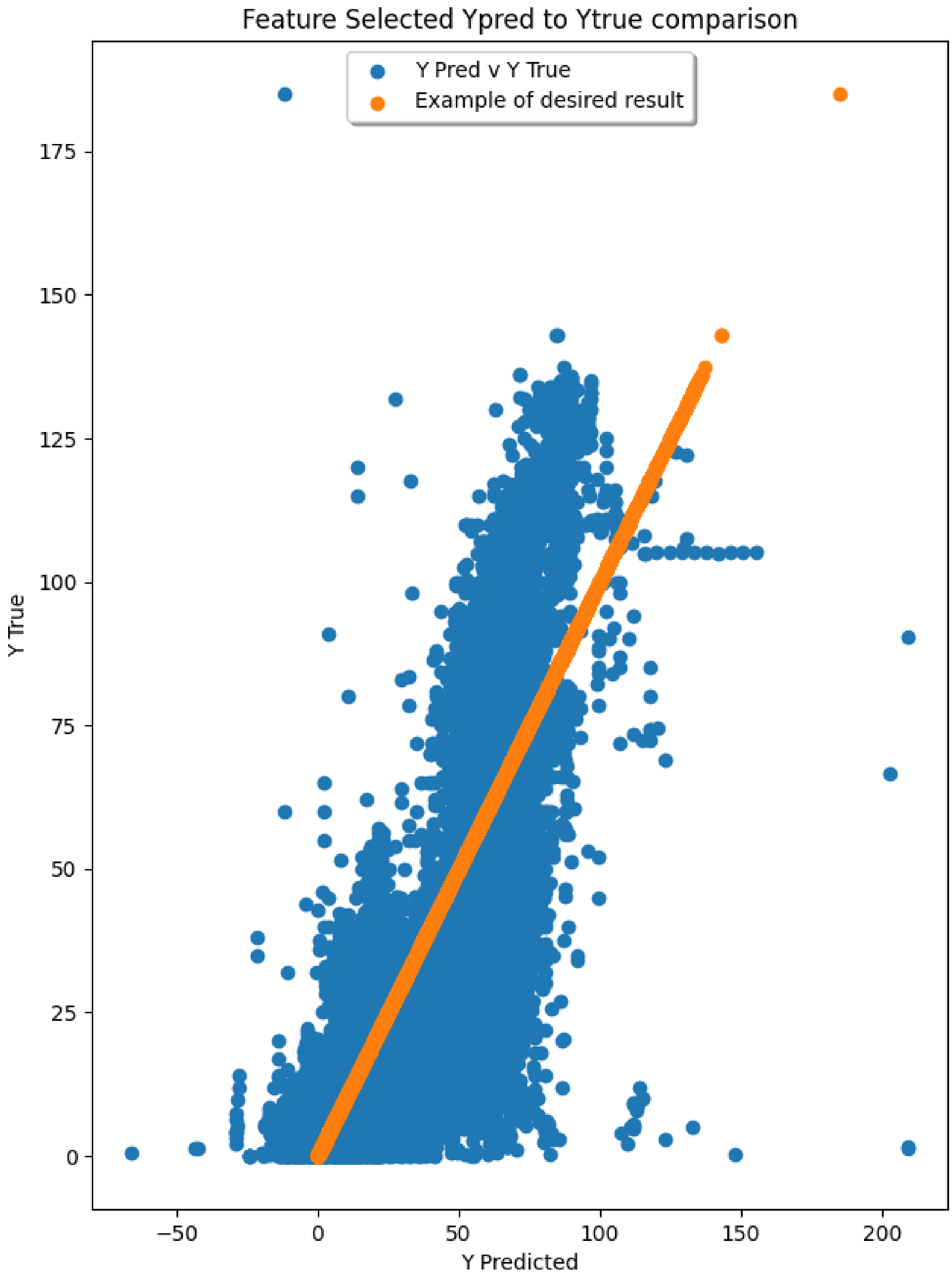
All models use 5 fold cross validation and our optimizing metric is the root mean squared error.

Dataset contains 168 features, 1 target feature(critical temperature), and 21263 observations. Dataset contains no NAs and contains no glaring issues. The “material” feature was removed as it contained data that was already one hot encoded by the rest of the dataset.

Results

Overall the rmse of both models are not terrible but not great, upon inspection of the graphs and that we have 21263 observations its likely our model is biased and thus is incapable of improving any further. Possible future courses of action would likely include exploring if log and polynomial feature transformations may help performance.





Feature Selected Ypred to Ytrue comparison
Y Pred v Y True
Example of desired result
Y True
Y Predicted
175
150
125
100
75
50
25
0
−50
0
50
100
150
200

The all data model had an rmse of 19.15, used an alpha of 1232.85, and took around 22.9 seconds to find the best model.

```
---All Data---
Best ridge rmse=19.1543, alpha=1232.8467394420684, time=22.9383s
Top 5 Features and Coefficients
Ba: 8.5059
wtd_mean_ThermalConductivity: 5.6796
wtd_std_Valence: -5.1740
wtd_std_ThermalConductivity: 4.8497
Bi: 4.6012
```

The feature selected model had an rmse of 20.38, used an alpha of 932.6, and took around 4.47 seconds to find the best model.

```
---Feature Selected Data---
Best ridge rmse=20.3756, alpha=932.6033468832219, time=4.4681s
wtd_std_ThermalConductivity: 9.4389
Ba: 8.2699
mean_ElectronAffinity: -4.9017
wtd_entropy_atomic_mass: 4.8534
range_atomic_radius: 4.8400
```

The feature selected model performs worse than the full data model. This is to be expected since feature selection removes data from the model that are deemed inconsequential but they still help slightly. In situations where you would want to have optimal performance it may be beneficial to consider leaving everything in

The time it took to train the feature selected was around 5x faster than the full data model as it has less data to train with. In situations where compute power is a limiting factor it may be beneficial to consider taking a small hit to performance in order to reduce the computational load.

Finally it’s interesting to note that both models achieved optimal performance at different values of alpha. The all data model opted for a higher alpha than the feature selected model. A higher alpha translates to more regularization so the all data model performed better with stronger penalties to the weights than the model with feature selection in place. This is likely due to the fact that part of the regularization work was carried out by L1 regularization through feature selection so the L2 regularization did not need to be punished as much.

Both models did not give the exact same top 5 important features. Out of the 5 only 2 were shared so this leaves 8 unique features that were deemed important by both models.

- Barium
    - Positive Relationship
    - Present in both models
- wtd_mean_ThermalConductivity
    - Positive Relationship
- wtd_std_Valence
    - Negative Relationship
- wtd_std_ThermalConductivity
    - Positive Relationship
    - Present in both models
- Bismuth
    - Positive Relationship
- mean_ElectronAffinity
    - Negative Relationship
- Wtd_entropy_atomic_mass
    - Positive Relationship
- range_atomic_radius
    - Positive Relationship

All of these features have a strong relationship in at least one of the models so the are worth looking into for future studies. With that said, Barium and wtd_std_ThermalConductivity were present in both models which grants them slightly higher priority for investigation than the rest.

Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sys
import time
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.model_selection import cross_val_score

# Your case study is to build a linear regression model using L1 or L2
# regularization (or both) the task to predict the Critical Temperature as
# closely as possible. In addition, include in your write-up which variable
# carries the most importance.

INVALID_INPUT = 1


def try_to_read_input_csvs(inputs):
    try:
        read_input_csvs(inputs)
    except Exception:
        print("This script expects the two csv files associated with Case "
              "Study1. You either input nothing, 1 file path, or one or more "
              "of your inputs contains an invalid path.")
        sys.exit(INVALID_INPUT)


def read_input_csvs(inputs):
    global df1, df2
    df1_path = inputs[1]
    df2_path = inputs[2]
    df1 = pd.read_csv(df1_path)
    df2 = pd.read_csv(df2_path)
```

```python
def linear_cv_feature_selection(x, y, features, print_progress=False):
    best_alpha_lasso = 0
    best_rmse_lasso = 0
    first = True
    report_freq = 1
    iter_count = 5
    param_alpha = 0.01
    for i in range(0, iter_count):
        param_alpha = param_alpha*10

        lasso_model = Lasso(alpha=param_alpha)

        rmse_lasso_raw = cross_val_score(lasso_model, x, y, cv=5,
                                         scoring='neg_root_mean_squared_error')

        rmse_lasso = -1*sum(rmse_lasso_raw)/len(rmse_lasso_raw)

        if first:
            first = False
            best_alpha_lasso = param_alpha
            best_rmse_lasso = rmse_lasso
        else:
            if (rmse_lasso < best_rmse_lasso):
                best_alpha_lasso = param_alpha
                best_rmse_lasso = rmse_lasso

        if ((i+1) % report_freq == 0) and print_progress:
            print(f"Iteration {i+1}. "
                  f"Percent done = {(i+1)/iter_count*100:.4f}%")

    # print(f"Best lasso rmse={best_rmse_lasso:.4f} alpha={best_alpha_lasso}")
    lasso_model = Lasso(alpha=best_alpha_lasso)
    lasso_model.fit(x, y)
    nonzero_features = list(lasso_model.coef_ > 0)
    selected_features = []

    for i in range(len(nonzero_features)):
        if nonzero_features[i]:
            selected_features.append(features[i])
    return selected_features
```

```python
def linear_cv_compare(x, y, print_progress=False):
    best_alpha_ridge = 0
    best_rmse_ridge = 0
    first = True
    report_freq = 10
    iter_count = 100
    i = 0
    t1 = time.time()
    for param_alpha in np.logspace(-6, 6, iter_count):
        i += 1

        ridge_model = Ridge(alpha=param_alpha)

        rmse_ridge_raw = cross_val_score(ridge_model, x, y, cv=5,
                                         scoring='neg_root_mean_squared_error')

        rmse_ridge = -1*sum(rmse_ridge_raw)/len(rmse_ridge_raw)

        if first:
            first = False
            best_alpha_ridge = param_alpha
            best_rmse_ridge = rmse_ridge
        else:
            if (rmse_ridge < best_rmse_ridge):
                best_alpha_ridge = param_alpha
                best_rmse_ridge = rmse_ridge

        if ((i+1) % report_freq == 0) and print_progress:
            print(f"Iteration {i+1}. "
                  f"Percent done = {(i+1)/iter_count*100:.4f}%")
    t2 = time.time()
    elapsed_time = t2-t1

    print(f"Best ridge rmse={best_rmse_ridge:.4f}, "
          f"alpha={best_alpha_ridge}, time={elapsed_time:.4f}s")

    return best_alpha_ridge

```

```python
def get_top_n_coef(feature_list, coef_list, n=5):
    feature_scores_abs = {}
    feature_scores = {}

    for i in range(len(feature_list)):
        feature = feature_list[i]
        coef = coef_list[i]
        feature_scores_abs[feature] = abs(coef)
        feature_scores[feature] = coef

    feature_scores_abs = dict(sorted(feature_scores_abs.items(),
                                     key=lambda item: item[1], reverse=True))
    top_n_features = list(feature_scores_abs.keys())[0:n]

    top_feature_scores = {}
    for feature in top_n_features:
        top_feature_scores[feature] = feature_scores[feature]

    return top_feature_scores


if __name__ == '__main__':
    inputs = sys.argv

    try_to_read_input_csvs(inputs)

    # Combine dataframes
    y = df1["critical_temp"]
    df1 = df1.drop(["critical_temp"], axis=1)
    df2 = df2.drop(["critical_temp", "material"], axis=1)

    # Find which dataframe contains the cts variables that need to be scaled
    df1_features = list(df1.columns)
    df2_features = list(df2.columns)
    if "number_of_elements" in df1_features:
        cts_features = df1_features
    else:
        cts_features = df2_features

    # Scale cts data
    scaler = StandardScaler()
    train = pd.concat([df1, df2], axis=1)
    train[cts_features] = scaler.fit_transform(train[cts_features])
```

```python
    # Lasso Feature Selection
    print("Lasso Feature Selection Starting. "
          "This step takes a little bit of time...")
    selected_features_main = linear_cv_feature_selection(train,
                                                         y,
                                                         list(train.columns))
    train_fs = train[selected_features_main]
    print("33% Done")

    # Ridge Results
    print("---All Data---")
    best_alpha_all_data = linear_cv_compare(train, y)
    all_data_model = Ridge(alpha=best_alpha_all_data)
    all_data_model.fit(train, y)
    all_data_top5 = get_top_n_coef(train.columns, all_data_model.coef_)
    print("Top 5 Features and Coefficients")
    for feature in all_data_top5.keys():
        coef = all_data_top5[feature]
        print(f"{feature}: {coef:.4f}")

    print("66% Done")

    print("---Feature Selected Data---")
    best_alpha_fs = linear_cv_compare(train_fs, y)
    feature_selected_model = Ridge(alpha=best_alpha_fs)
    feature_selected_model.fit(train_fs, y)
    feature_selected_top5 = get_top_n_coef(train_fs.columns,
                                           feature_selected_model.coef_)
    for feature in feature_selected_top5.keys():
        coef = feature_selected_top5[feature]
        print(f"{feature}: {coef:.4f}")

    print("100% Done")
```

```python
    # Y_pred to True plots
    figure, axis = plt.subplots(1, 2, figsize=(16, 10))

    y_pred_all_data = all_data_model.predict(train)
    axis[0].scatter(y_pred_all_data, y, label="Y Pred v Y True")
    axis[0].scatter(y, y, label="Example of desired result")
    axis[0].set_title("All Data Ypred to Ytrue comparison")
    axis[0].set_xlabel("Y Predicted")
    axis[0].set_ylabel("Y True")
    axis[0].legend(loc='upper center', shadow=True)

    y_pred_fs = feature_selected_model.predict(train_fs)
    axis[1].scatter(y_pred_fs, y, label="Y Pred v Y True")
    axis[1].scatter(y, y, label="Example of desired result")
    axis[1].set_title("Feature Selected Ypred to Ytrue comparison")
    axis[1].set_xlabel("Y Predicted")
    axis[1].set_ylabel("Y True")
    axis[1].legend(loc='upper center', shadow=True)

    plt.show()

    print("Program Finished")
```