Data Analysis and Machine-Learning

Chapter 5.5:

Regularization (L2, L1)



1. Introduction

Throughout the previous subchapters, we have considered resolving multicollinearities (correlations and causalities, variance inflation factors), feature selection methodologies (forward selection, backward elimination, stepwise selection, recursive feature elimination), standard scaling, and variable weight evaluation algorithms (eli5, permutation importance, shap, coefficients and p-value) as the main algorithms and methods to avoid overfitting problems in terms of modelling a statistically significant model. In other words, we have focused on the **features (variables) per se**, by selecting the most explanatory variables and eliminating unnecessary ones to come up with a concise model with less variables and high accuracy.

There is yet one more powerful way to avoid overfitting problems, which this time focuses on controlling (or, regularizing) parameters (theta) instead of controlling (or eliminating) the features at firsthand, namely, by utilizing regularization method. In order to understand the concept of regularization, consider the following equation:

For a model with multiple coefficients (parameters) to :

The first part of the equation is not at all different from the OLS (LSE) estimation that we have learned previously. This would refer to the estimation of regular training accuracy. The essence of regularization comes from the second part of the equation with lambda parameter, which adds the concept of generalization accuracy to the model by assigning penalties to the beta coefficient. This concept of reducing the parameter estimates via adding the concept of generalization accuracy is referred to as the regularization method. There are representatively two types of regularization, namely, Ridge regression (L2 method) and Lasso regression (L1 method).

2. Ridge Regression (L2 Regularization)

Let us begin by considering the equation for L2 to understand the mechanism:

Similarly to the equation before, ridge regression is composed of two parts: residual sum of squares (RSS) and the penalty term (Beta values). More specifically, the penalty term of the ridge regression is the sum of parameters squared, which thus allows for a differentiation; and this enables optimization via gradient descent (consider the concepts that I have illustrated in chapter 2 and 3, differentiation and gradient descent). Thus as in the case for the optimization process in gradient descent algorithms, the product of lambda decreases the parameters (with faster decreasing rates for larger parameters). In other words, the larger the value of lambda become, the closer the regression coefficient estimates get to 0. And when the value of lambda is 0, the equation would be exactly the same with the OLS (LSE) estimation, which means there is no penalty term assigned to the regression equation.

For better understanding, consider the following set of samples.

3. Application of L2 Regularization

# Generate overfitted sample model

import numpy as np

np.random.seed(1234)

x = np.linspace(0, 1, 1000)

y = np.sin(2 \* np.pi \* x) + (np.random.randn(1000)/5)

x\_train = np.linspace(0, 1, 11)

y\_train = np.sin(2 \* np.pi \* x\_train) + (np.random.randn(11)/5)

x\_test = np.linspace(0, 1, 50)

y\_test = np.sin(2 \* np.pi \* x\_test) + (np.random.randn(50)/5)

x\_poly = np.c\_[

 x, x \*\* 2, x \*\* 3, x \*\* 4, x \*\* 5, x \*\* 6, x \*\* 7,

 x \*\* 8, x \*\* 9, x\*\*10

]

x\_train\_poly = np.c\_[

  x\_train, x\_train \*\* 2, x\_train \*\* 3, x\_train \*\* 4, x\_train \*\* 5,

  x\_train \*\* 6, x\_train \*\* 7, x\_train \*\* 8, x\_train \*\* 9, x\_train\*\*10

]

x\_test\_poly = np.c\_[

  x\_test, x\_test \*\* 2, x\_test \*\* 3, x\_test \*\* 4, x\_test \*\* 5,

  x\_test \*\* 6, x\_test \*\* 7, x\_test \*\* 8, x\_test \*\* 9, x\_test\*\*10

]

#Check coefficients for the original model

def coefficients(model, x):

    list\_ = []

    for coef in model.coef\_:

        list\_.append('{:0.02f}'.format(coef))

    if isinstance(x, np.ndarray):

        x = pd.DataFrame(x)

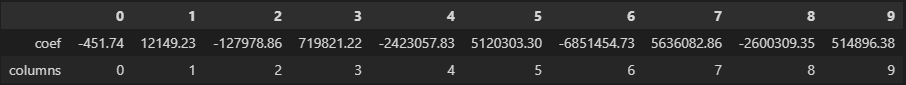
    coef\_ = pd.DataFrame( list\_, columns=['coef'] )

    col = pd.DataFrame( x.columns, columns=['columns'] )

    return pd.concat( [coef\_, col], axis=1)

coefficients(polyModel, x\_train\_poly).T

output:



#Visualization

import matplotlib.pyplot as plt

plt.plot(x\_train, y\_train, 'o', label='train')

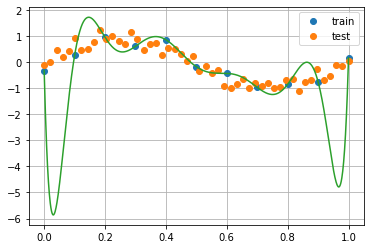
plt.plot(x\_test, y\_test, 'o', label='test')

plt.plot(x, polyModel.predict(x\_poly))

plt.legend()

plt.grid()

plt.show()



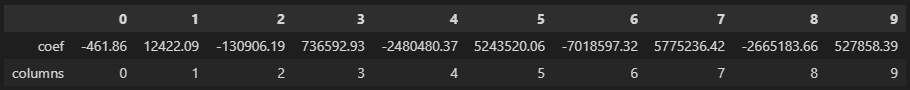
The shape of the estimated line and the overly large values of the coefficients imply that the model is obviously overfitted. Now, let us assign penalties to the parameters via L2 regularization process, by customizing the value of lambda.

#when lambda value is 0, estimation is exactly the same with ordinary OLS estimation (no penalty assigned to beta values)

from sklearn.linear\_model import Ridge, Lasso, ElasticNet

l2model = Ridge(alpha=0).fit(x\_train\_poly, y\_train)

coefficients(l2model, x\_train\_poly).T



import matplotlib.pyplot as plt

plt.plot(x\_train, y\_train, 'o', label='train')

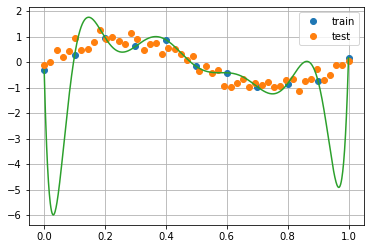
plt.plot(x\_test, y\_test, 'o', label='test')

plt.plot(x, l2model.predict(x\_poly))

plt.legend()

plt.grid()

plt.show()



#the larger the value of lambda become, the closer the regression coefficient estimates get to 0.

l2model = Ridge(alpha=500).fit(x\_train\_poly, y\_train)

coefficients(l2model, x\_train\_poly).T



import matplotlib.pyplot as plt

plt.plot(x\_train, y\_train, 'o', label='train')

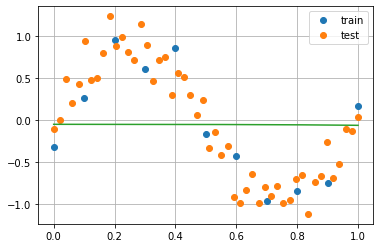
plt.plot(x\_test, y\_test, 'o', label='test')

plt.plot(x, l2model.predict(x\_poly))

plt.legend()

plt.grid()

plt.show()



#Best fit model with appropriate value of lambda

l2model = Ridge(alpha=0.0001).fit(x\_train\_poly, y\_train)

coefficients(l2model, x\_train\_poly).T



import matplotlib.pyplot as plt

plt.plot(x\_train, y\_train, 'o', label='train')

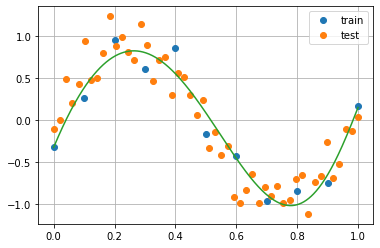
plt.plot(x\_test, y\_test, 'o', label='test')

plt.plot(x, l2model.predict(x\_poly))

plt.legend()

plt.grid()

plt.show()



As the fact that the coefficients for all variables remain with approximate values assigned, it is possible to generate a model with best-fit without **eliminating** variables, which is contrary to the case of feature selections.

4. Lasso Regression (L1 Regularization)

The main difference between Ridge regression and Lasso regression is that the penalty term of the Lasso is composed of the sum of absolute values of the parameters. Similarly to the case for Ridge, Lasso increases the value of residual sum of squares in LSE estimation. The concept is equivalent to the difference between LAD (Least Absolute Deviation) and LSE that we have covered in chapter 2 and 3. In fact, LAD and LSE utilizes the exact concept of L1 loss and L2 loss, respectively, in their estimation of errors. Since Lasso applies the same level of regularization regardless of the size of the parameters, it eventually derives the parameters with small values to become 0, which is essentially the same with eliminating the corresponding variable (i.e., 0\*X=0); and this simplifies the model with better interpretability.

5. Application of L1 Regularization

l1model = Lasso(alpha=1).fit(x\_train\_poly, y\_train)

coefficients(l1model, x\_train\_poly).T

plt.plot(x\_train, y\_train, 'o', label='train')

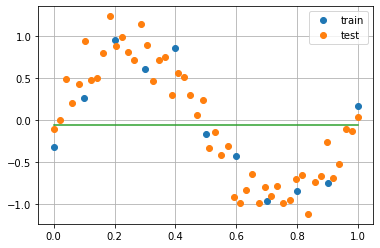
plt.plot(x\_test, y\_test, 'o', label='test')

plt.plot(x, l1model.predict(x\_poly))

plt.legend()

plt.grid()

plt.show()



#Best fit model with appropriate value of lambda

l1model = Lasso(alpha=0.0001).fit(x\_train\_poly, y\_train)

coefficients(l1model, x\_train\_poly).T

plt.plot(x\_train, y\_train, 'o', label='train')

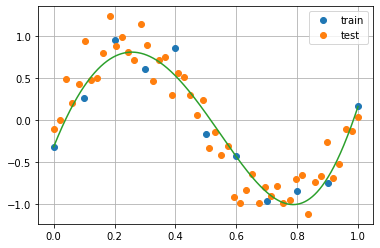
plt.plot(x\_test, y\_test, 'o', label='test')

plt.plot(x, l1model.predict(x\_poly))

plt.legend()

plt.grid()

plt.show()



6. Elastic Net

Elastic Net incorporates both algorithms of L1 and L2 regularization, and thus it is possible to customize the ratio of L1/L2 penalty parameters. This is done by setting the L1 ratio (from 0 to 1); for example, the L1 ratio of 0.5 would mean the combination of parameters (by half each) L1 and L2 penalties. In actual applications for modelling, therefore, elastic net is usually applied for various customizations of penalty terms.

7. Applications of Elastic Net

#If L1-ratio is equal to 0, the penalty is L2 penalty (Ridge Regression)

model\_elastic = ElasticNet(l1\_ratio=0, alpha=0.0001).fit(x\_train\_poly, y\_train)

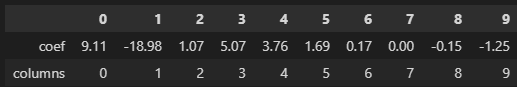
coefficients(model\_elastic, x\_train\_poly).T



#If L1-ratio is equal to 1, the penalty is L1 penalty (Lasso Regression)

model\_elastic = ElasticNet(l1\_ratio=1, alpha=0.0001).fit(x\_train\_poly, y\_train)

coefficients(model\_elastic, x\_train\_poly).T



#If L1-ratio is 0.5, the penalty is the combination of L2 and L1 penalty (by half each)

model\_elastic = ElasticNet(l1\_ratio=0.5, alpha=0.0001).fit(x\_train\_poly, y\_train)

coefficients(model\_elastic, x\_train\_poly).T

