Lecture 3: Regularization Methods

DATA 602 @ UMBC

Masoud Soroush

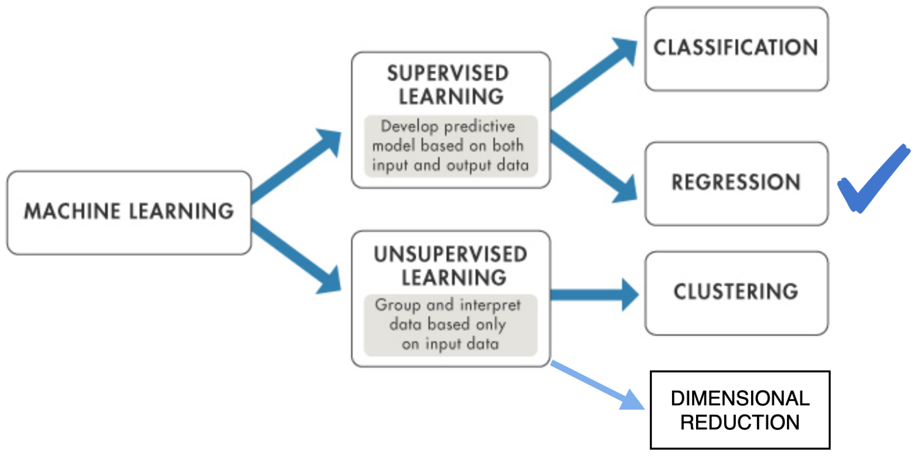
September 12, 2022

1 Regularization Methods

In today's lecture[1], we introduce regularization methods to reduce overfitting. From previous lecture, we recall that one common source of error in statistical analysis is due to variance. Regularization methods are introduced to reduce the effect of errors that are caused by variance. We introduce regularization in the context of regression analysis. However, as we will see later on, regularization methods are employed to reduce the variance in the context of classification algorithms as well.

1.1 Background

Just as the previous lecture, our discussion is in the realm of *supervised learning* with a *continuous target variable*. The ultimate goal of the model is to predict the continuous target variable.



As observed last time, there are different sources to account for errors in statistical analyses. Errors that are due to *high bias* often lead to an *underfit* of the training data. In contrast, errors caused by

[1] This product is a property of UMBC, and no distribution is allowed. These notes are solely for your own use, as a registered student in this class.

high variance often lead to an *overfit* of the training data. There is a third type of error in statistical analyses, called *irreducible* error. An irreducible error is caused by the noise in the data itself. The only way to reduce this type of error is to clean up the data. Today, we focus on errors due to high variance.

1.2 What Is Regularization?

To set the stage, let us review some basic facts we derived last time. Our starting point is a dataset in which the features (d of them) are denoted by $\vec{x} \in \mathbb{R}^d$, and y denotes the target variable. The dataset consists of n observations, and both $\vec{x}$ and y are continuous variables.

Observation	$\vec{x} \in \mathbb{R}^d$	y
1	$\vec{x}^{(1)}$	y_1
2	$\vec{x}^{(2)}$	y_2
3	$\vec{x}^{(3)}$	y_3
$\vdots$	$\vdots$	$\vdots$
n	$\vec{x}^{(n)}$	y_n

Associated with the above dataset, we constructed an $n \times (d+1)$ matrix, $\mathbb{X}$, that captures the observations in the above dataset:

$$\mathbb{X} = (\mathbb{1}, x_1, x_2, \cdots, x_d) = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_d^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_d^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & x_2^{(n)} & \cdots & x_d^{(n)} \end{pmatrix} . \tag{1}$$

The linear regression model assumes a linear relation between the target and the features. We represented the linear relation in the following compact matrix form

$$\hat{\mathbb{Y}} = \mathbb{X} \cdot \omega^{\mathsf{T}} , \tag{2}$$

where matrix $\omega = (\omega_0, \omega_1, \cdots, \omega_d)$. As usual, the hatted quantities are the predictions of the model. We then defined the cost function $J(\omega)$

$$J(\omega) = \frac{1}{2}(\mathbb{Y} - \hat{\mathbb{Y}})^{\mathsf{T}} \cdot (\mathbb{Y} - \hat{\mathbb{Y}}) , \tag{3}$$

and showed that *minimizing the cost function* $J(\omega)$ in equation (3) determines the coefficients ω of the linear model as follows:

$$\omega^{\mathsf{T}} = \left(\mathbb{X}^{\mathsf{T}}\mathbb{X}\right)^{-1} \mathbb{X}^{\mathsf{T}}\mathbb{Y} . \tag{4}$$

The above linear model has $d + 1$ **degrees of freedom**. If one uses polynomial regression, then one would have even more degrees of freedom. For instance, if one uses quadratic regression, one would find $\frac{d(d+1)}{2}$ more degrees of freedom (in addition to $d + 1$ linear degrees of freedom). *The greater degrees of freedom a model possesses, the easier it will overfit the data.* Equivalently, the fewer degrees of freedom a model has, the harder it will be for it to overfit the data. Therefore, in order to avoid overfitting, one has to reduce the degrees of freedom of the model. **Regularization is a method to reduce degrees of freedom of a model to avoid overfitting the data**.

It is important to note that in regularizing a model, we *do not eliminate any of the features by hand*. We rather *constrain the model*. The model will then learn through the process of training to identify irrelevant (or less relevant) features.

1.3 Different Types of Regularization

There are several types of regularization methods. These methods have similarities and differences with one another. In here, we consider three common types of regularizations, namely the *Ridge*, *Lasso*, and *Elastic Net* regressions. In these regularization methods, a constraint on the weights ω (*i.e.* parameters of the model) is imposed. The cost function is then minimized in the presence of the introduced constraint.

Before we continue with the details of the constraints used in regularization methods, we need to distinguish between *parameters* and *hyperparameters* of machine learning models.

Parameter: In machine learning, a parameter is a factor that is introduced by the model, and its optimal value is determined through the training process once the training is complete. In other words, *parameters of a model are learned* by through the training process. For instance, coefficients of linear regression ω are parameters of the linear model, and their optimal values are determined by minimizing the cost function through the training process.

Hyperparameter: A hyperparameter is a factor whose value is set by a user before the training process starts. Unlike parameters, hyperparameters are not learned by the model, and their values are are fixed before training starts. Without specifying fixed values for hyperparameters, the training process cannot start. An important example of a hyperparameter is the *train-test split ratio*.

The constraint that is introduced to the model should possess the following important properties:

- The constraint is **independent of the training data**. This implies that the feature matrix $\mathbb{X}$ and the target matrix $\mathbb{Y}$ do not show up in the constraint. This is important as we desire a general constraint that does not depend on the details and specifics of the model.
- The constraint involves a **positive definite term** that only depends on the magnitude of the weights ω.
- The constraint puts an **upper bound on the magnitude of the parameters** of the model (in this case the coefficients of linear model ω).
- To incorporate and control the positive definite constraint in the minimization of the cost function, a hyperparameter λ is introduced to the model. The hyperparameter λ is positive semi-definite (*i.e.* $\lambda \geq 0$). This hyperparameter λ is called the **regularization factor** or the **penalty factor**.

- The case $\lambda = 0$ corresponds to the unconstrained linear model we studied last time. The greater λ is, the stronger the penalty term will be. A stronger penalty term forces the coefficients ω to be smaller in magnitude.

We now introduce three regularization methods and check that the above conditions are fulfilled by each regularization scheme.

1.3.1 Ridge Regularization

The first regularization scheme is the Ridge regularization (sometimes called Tikhonov regularization). This regularization scheme takes the advantage of the L_2-norm. Let us denote the weight vector $\tilde{\omega}$ to be $\tilde{\omega} = (\omega_1, \cdots, \omega_d)$. In other words, the weight vector $\tilde{\omega}$ does not include the intercept of regression ω_0 (*i.e.* the bias term). Then, in Ridge regression, we minimize the cost function $J(\omega)$ in the presence of a constraint. The constraint is that the sum of squares of components of $\tilde{\omega}$ (*i.e.* the L_2-norm of $\tilde{\omega}$) must be smaller than a positive quantity (s^2). Note that in Ridge regression, we minimize the same function (*i.e.* cost function) as in the case of ordinary regression. The difference is that in the Ridge regression, we have to satisfy a constraint (*i.e.* L_2-norm of the weight vector $\tilde{\omega}$ must be smaller than a chosen upper bound). In terms of equations, the Ridge regression is summarized as follows:

$$\boxed{\begin{aligned} &\underset{\omega}{\operatorname{argmin}}\Big(J(\omega) = \frac{1}{2}(\mathbb{Y} - \hat{\mathbb{Y}})^\intercal \cdot (\mathbb{Y} - \hat{\mathbb{Y}})\Big) \\ &\text{subject to: } (\|\tilde{\omega}\|_2)^2 = \sum_{i=1}^{d} \omega_i^2 \le s^2 \,. \end{aligned}} \tag{5}$$

Important Note: Note that only weights associated with features (*i.e.* $\omega_1, \omega_2, \cdots, \omega_d$) are regularized. In other words, the *bias term ω_0 is never regularized*. The sum in (5) starts from 1, not 0!

To incorporate the constraint in the minimization process of the cost function, we can take the advantage of a known calculus trick (known as Lagrange Multiplier Method). The Lagrange multiplier trick allows us to get rid of the constraint at the cost of adding a term to the function that is being optimized. Instead of minimizing the cost function $J(\omega)$ in (5), we minimize $\tilde{J}(\omega)$ in the following equation

$$\tilde{J}(\omega) = \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2 + \frac{\lambda}{2}\sum_{i=1}^{d}\omega_i^2 = \frac{1}{2}(\mathbb{Y} - \hat{\mathbb{Y}})^\intercal \cdot (\mathbb{Y} - \hat{\mathbb{Y}}) + \frac{\lambda}{2}\,\tilde{\omega}\,\tilde{\omega}^\intercal \,. \tag{6}$$

However, the benefit is that for minimizing $\tilde{J}(\omega)$ in (6), we do not have any constraints, and hence, we follow the usual optimization process. The coefficient λ in (6) is called the penalty factor. If we set $\lambda = 0$, $\tilde{J}(\omega)$ reduces to the cost function $J(\omega)$, and we will have the ordinary regression. The greater the penalty factor λ is, the greater the effect of regularization will be. The extreme case $\lambda \to \infty$ corresponds to the most severe penalty in which case all weights $\omega_i \to 0$ for all $i = 1, 2, \cdots, d$ (because that is the only way to keep $\tilde{J}(\omega)$ finite). In summary, the coefficient λ *controls the severity of the imposed penalty* on the cost function.

Question: How does regularization treat weakly correlated features?

Answer: Weakly correlated features to the target are typically variables whose regression coefficients are very small. However, regression algorithm uses these small coefficients to minimize the cost function to its least residual standard error. Ordinary regression has no way to distinguish weak features from noise. But once you consider a penalty term for the cost function, the regression algorithm sets the coefficients of weakly correlated features to zero. That is a benefit that is obtained by regularizing the regression.

Side Question: We saw that it was possible to solve the regression problem exactly. Is it still possible to obtain an exact solution for the Ridge regression when a penalty term is present?

Answer: The answer is yes! We can derive an exact formula analogous to the ordinary regression. We do not include the derivation in here and simply state the result. *Interested students can come to me to show how the exact formula is derived.* To find the exact result, we need to do two things. First, we need to place the origin of the coordinate system at the mean of the features (*i.e.* $\frac{1}{n}\sum_{i=1}^{n}\vec{x}^{(i)}$). After translating the origin of the coordinate system to the mean of the features, we define matrix $\tilde{\mathbb{X}}$ as follows:

$$\tilde{\mathbb{X}} = (x_1 - \bar{x}_1, \cdots, x_d - \bar{x}_d) = \begin{pmatrix} x_1^{(1)} - \bar{x}_1 & x_2^{(1)} - \bar{x}_2 & \cdots & x_d^{(1)} - \bar{x}_d \\ x_1^{(2)} - \bar{x}_1 & x_2^{(2)} - \bar{x}_2 & \cdots & x_d^{(2)} - \bar{x}_d \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(n)} - \bar{x}_1 & x_2^{(n)} - \bar{x}_2 & \cdots & x_d^{(n)} - \bar{x}_d \end{pmatrix} . \tag{7}$$

The exact result for the weights $\tilde{\omega}$ and the bias term ω_0 is then given by

$$\boxed{\omega_0 = \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y^{(i)} , \quad \tilde{\omega}^{\mathsf{T}} = \left(\tilde{\mathbb{X}}^{\mathsf{T}}\tilde{\mathbb{X}} + \lambda \mathbb{1}_{d\times d}\right)^{-1}\tilde{\mathbb{X}}^{\mathsf{T}}\mathbb{Y} ,} \tag{8}$$

where $\mathbb{1}_{d\times d}$ is the $d \times d$ identity matrix. Notice that equation (8) states that bias term ω_0 is nothing but the *mean of the target variable* y. Moreover, note the presence of the penalty factor λ in the exact result for $\tilde{\omega}$. The exact formula (8) shows that the greater λ is, the smaller $\tilde{\omega}$ will be. In other words, equation (8) shows that $\tilde{\omega} \to \vec{0}$ as $\lambda \to \infty$.

Question: Does scikit-learn use the exact formula (8) to calculate the weights of the Ridge regression?

Answer: No! For the same reason as the ordinary regression, scikit-learn library does not use the exact formula. It rather uses a gradient descent approach which is much faster for big datasets!

1.3.2 Lasso Regularization

The second regularization method that we will discuss here is LASSO (Least Absolute Shrinkage and Selection Operator) regression. In many ways, Lasso regression is similar to the Ridge regression. However, the key difference between the two approaches is that Lasso considers an L_1-norm constraint (rather than L_2-norm constraint in the case of Ridge). In terms of equations, the Lasso regression is defined by

$$\boxed{\begin{aligned}&\underset{\omega}{\operatorname{argmin}}\Big(J(\omega) = \frac{1}{2}(\mathbb{Y}-\hat{\mathbb{Y}})^{\mathsf{T}}\cdot(\mathbb{Y}-\hat{\mathbb{Y}})\Big)\\ &\text{subject to: } \|\tilde{\omega}\|_1 = \sum_{i=1}^{d}|\omega_i| \leq s\,.\end{aligned}} \tag{9}$$

Note that the L_1-norm concerns the absolute value of the weights ω_i. As in the Ridge regression, the bias term ω_0 is never regularized. As is clear from equations (5) and (9), the same cost function is shared by Ridge and Lasso regressions. The difference between the two concerns the imposed constraints. As in the Ridge case, we can employ the Lagrange multiplier method to facilitate the minimization of the cost function by adding an appropriate penalty term.

$$\tilde{J}(\omega) = \frac{1}{2}\sum_{i=1}^{n}(y^{(i)}-\hat{y}^{(i)})^2 + \frac{\lambda}{2}\sum_{i=1}^{d}|\omega_i|\,. \tag{10}$$

Again, setting $\lambda = 0$ reduces the problem to the ordinary regression. As in the Ridge regression, the greater the penalty factor λ is, the smaller the weights $\tilde{\omega}$ will be.

1.3.3 Elastic Net Regularization

The third regularization method is the Elastic Net regularization which is a mixture of both the Ridge and the Lasso regressions. In Elastic Net regularization, two separate constraints are imposed on the minimization of the cost function $J(\omega)$. One constraint considers the L_1-norm and the other considers the L_2-norm.

$$\boxed{\begin{aligned}&\underset{\omega}{\operatorname{argmin}}\Big(J(\omega) = \frac{1}{2}(\mathbb{Y}-\hat{\mathbb{Y}})^{\mathsf{T}}\cdot(\mathbb{Y}-\hat{\mathbb{Y}})\Big)\\ &\text{subject to: } \begin{cases}\|\tilde{\omega}\|_1 = \sum_{i=1}^{d}|\omega_i| \leq s_1\\ (\|\tilde{\omega}\|_2)^2 = \sum_{i=1}^{d}\omega_i^2 \leq s_2\end{cases}\end{aligned}} \tag{11}$$

Employing the same Lagrange multiplier technique, we get rid of the constraints at the cost of adding two penalty terms to the cost function (one for each constraint)

$$\tilde{J}(\omega) = \frac{1}{2}\sum_{i=1}^{n}(y^{(i)}-\hat{y}^{(i)})^2 + \frac{\lambda_1}{2}\sum_{i=1}^{d}\omega_i^2 + \frac{\lambda_2}{2}\sum_{i=1}^{d}|\omega_i|\,. \tag{12}$$

The Elastic Net regression possesses two independent penalty factors, λ_1 and λ_2. Setting $\lambda_1 = 0$ reduces the Elastic Net regression to the Lasso regression, and setting $\lambda_2 = 0$, reduces the Elastic Net regression to the Ridge case. Moreover, setting $\lambda_1 = 0$ and $\lambda_2 = 0$ reduces the problem to the ordinary regression. As in previous regularization methods, the greater λ_1 and λ_2 are, the smaller the weights $\tilde{\omega}$ will be.

1.4 Regularization Methods Comparison

Now that we have introduced three different regularization methods, it would be appropriate to compare them with one another and identify their differences.

First, let us compare the Ridge and Lasso regressions. As mentioned above, introducing a penalty term (Ridge or Lasso) puts an upper bound on the magnitude of the weights $\tilde{\omega}$, and the greater the penalty factor λ is, the smaller the weights $\tilde{\omega}$ will be. Despite this similarity between the Ridge and Lasso regressions, the two regularization methods have an important difference:

Difference between Ridge and Lasso Regularizations: Although the Ridge regression shrinks all the weights toward smaller values (*i.e.* close to zero) as the penalty factor λ increases, it will not set any of the weights ω_i $\underline{\text{exactly}}$ to zero (unless you set $\lambda \to \infty$). This may not be an issue for the accuracy of the model, but it can pose a challenge in interpreting the model in which the number of features is quite large. In contrary, the Lasso regularization has the effect of forcing some of the weights ω_i to be $\underline{\text{exactly}}$ zero when the penalty factor λ is sufficiently large. In other words, *Lasso regularization can be used as a feature selection criterion*, and hence, it offers a simpler model to interpret!

Question: How does the Lasso regularization set some of the weights *exactly* to zero? And why can't the Ridge regularization do the same thing?

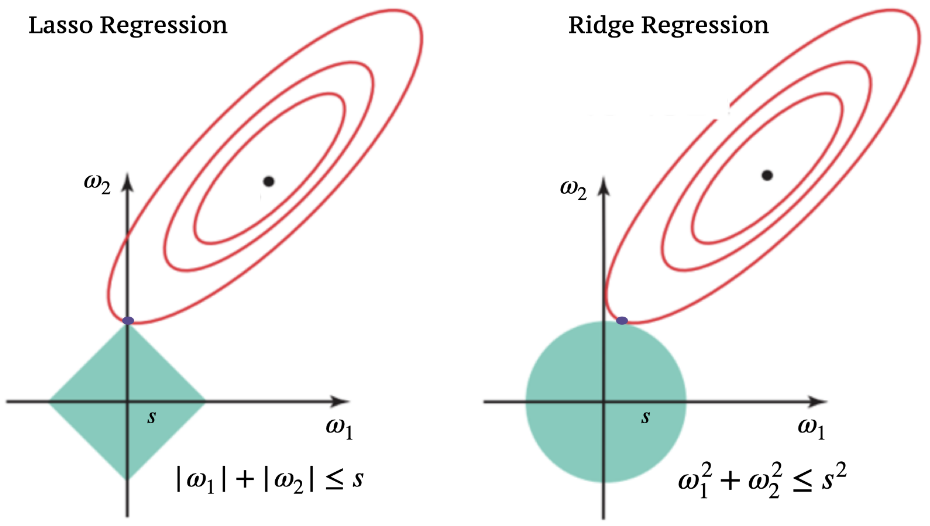
To answer this question, and in order to keep the analysis simple, we consider a special case. Suppose we have a model that possesses only two features x_1 and x_2, and a target variable y. In this case, $\tilde{\omega} = (\omega_1, \omega_2)$, and the cost function $J(\omega)$ is given by

$$\begin{aligned} J(\omega_0, \omega_1, \omega_2) &= \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2 = \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - \omega_1 x_1^{(i)} - \omega_2 x_2^{(i)} - \omega_0)^2 \\ &= A\,\omega_1^2 + B\,\omega_2^2 + C\,\omega_1\omega_2 + D\,\omega_1 + E\,\omega_2 + F\,, \end{aligned} \tag{13}$$

where the constant coefficients A, B, C, D, E, and F can be easily expressed in terms of the values of the tabular dataset and ω_0 (*e.g.* $A = \frac{1}{2}\sum_{i=1}^{n}(x_1^{(i)})^2$). Now, consider a two-dimensional space ($\mathbb{R}^2$) with coordinate system (ω_1, ω_2). Setting the cost function (13) to a constant, we obtain the equation of an ellipse in (ω_1, ω_2)-space. Changing this constant value, we find a family of **(cocentric) ellipses**, one ellipse associated with each constant (In calculus, the curves obtained in this manner are called the *contour plots*). Now, in order to apply the Ridge and Lasso regularizations, we need to apply an appropriate constraint for each case in the (ω_1, ω_2)-space. For the Ridge case, we have to apply $\omega_1^2 + \omega_2^2 \leq s^2$, and for the Lasso case, $|\omega_1| + |\omega_2| \leq s$. These two regions correspond to a disc and the interior of a rhombus, respectively.

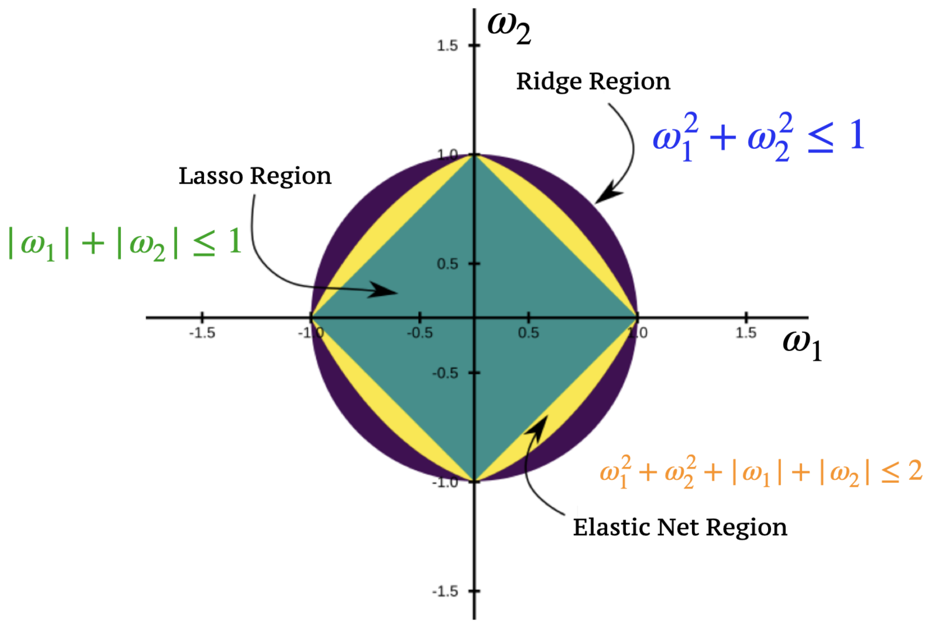
To minimize the cost function in (13), we need to intersect the contour plots (*i.e.* the red ellipses) with the regions corresponding to the constraints. It is evident from the above picture that the ellipses intersect the disc (in the Ridge case) at a point for which ω_1 is very small, but not exactly zero. On the other hand, the ellipses in the Lasso case intersect one of the vertices of the rhombus for which ω_1 is exactly zero!

Moral of the Story: In applying a regularization method, if the corresponding constraint region possesses **sharp corners on the coordinate axes** (like the rhombus in Lasso case), the regularization method is more likely to set some of the weights to *exactly* zero! In other words, the sharper the corners of the constraint region are, the greater likelihood for setting features to exactly zero



value will be. This conclusion generalizes to more number of features (*i.e.* $d > 2$) straightforwardly.

Now, you may wonder what can be said about Elastic Net regularization. In order to see what Elastic Net does, we plot the its constraint region in the context of the above setup (*i.e.* with only two features). The below picture depicts the constraint regions of all three regularizations together with their equations.


ω_2
Ridge Region
$\omega_1^2 + \omega_2^2 \leq 1$
Lasso Region
$|\omega_1| + |\omega_2| \leq 1$
ω_1
$\omega_1^2 + \omega_2^2 + |\omega_1| + |\omega_2| \leq 2$
Elastic Net Region
1.5
1.0
0.5
-0.5
-1.0
-1.5
-1.5
-1.0
-0.5
0.5
1.0
1.5

As is clear from the above picture, the constraint region associated with the Elastic Net is a region in between of the Ridge and the Lasso. Elastic Net region does possess four sharp corners, but the corners are not as sharp as the corners of the Lasso region. Therefore, we expect the Elastic Net regularization to operate similar to the Lasso when the penalty factor of the L_1-term is sufficiently large.

Note: Although we studied regularization techniques in the context of regression analysis, they are applied to classification problems as well. The philosophy for adding regularization terms in classification problems is completely analogous to what we saw in this lecture. In future lectures, we will use regularization techniques in the context of classification problems as well.

2 Coding for Regularized Regression Analysis

2.1 A Toy Model

As our first example, we construct a toy dataset. This dataset has only one feature x, and one target variable y. In the next step, we apply polynomial regression (without regularization) to this dataset to predict the target y. Finally, we add Ridge and Lasso regularizations to the regression analysis and compare the results.

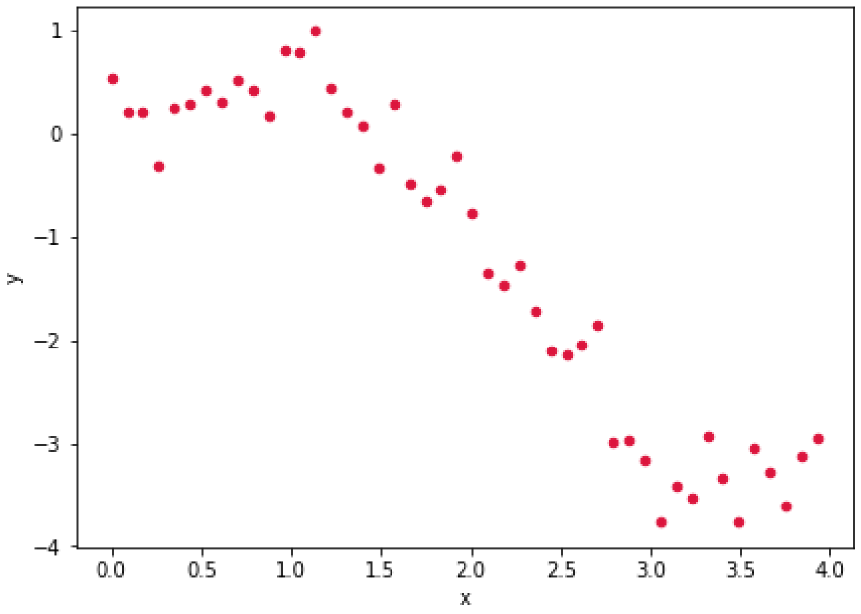
The toy dataset is generated based on the model $\boxed{y = x\cos x}$ for $0 \le x \le 4$.

Note: Since this example is a toy example and is solely used for educational purposes, we do not split the data into train and test, and simply apply regression to the whole dataset.

[1]:
```
import numpy as np      # Import numpy
import pandas as pd     # Import pandas

np.random.seed(3)       # Fix a seed for reproducibility purposes

x = np.array([j*np.pi/180 for j in range(0, 230, 5)])    # Create a numpy array␣
 ↪as the feature 'x'

# Create the target 'y' and add some noise chosen from a normal distribution␣
 ↪with mean 0 and std 0.3
y = x*np.cos(x) + np.random.normal(0, 0.3, x.shape[0])

# Construct a dataframe with two columns 'x' and 'y'
df = pd.DataFrame(np.column_stack([x, y]), columns=['x', 'y'])

df.head()               # Display the first few rows of 'df'
```

[1]:
```
          x         y
0  0.000000  0.536589
1  0.087266  0.217887
2  0.174533  0.200831
3  0.261799 -0.306169
4  0.349066  0.244798
```

```
[2]: import matplotlib.pyplot as plt     # Import matplotlib

     df.plot.scatter(x='x', y='y', color='crimson', figsize=(7,5))    # Scatter plot␣
      ↪of '(x, y)'
     plt.show()
```



```
[3]: # We now need to construct powers of 'x' to be able to perform polynomial␣
      ↪regression

     power_col = ['x^%d' %(i) for i in range(2, 11)]     # Create name for the power␣
      ↪columns

     for col in power_col:
         power = int(col.split('^')[1])    # Extract 'power'
         df[col] = df['x']**power          # Construct the column 'x^power'

     df = df[['x'] + power_col + ['y']]    # Change the order of columns (put 'y' at␣
      ↪the end)

     df.head()     # Display dataframe
```

```
[3]:           x       x^2       x^3       x^4       x^5           x^6  \
     0  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000e+00
     1  0.087266  0.007615  0.000665  0.000058  0.000005  4.416561e-07
     2  0.174533  0.030462  0.005317  0.000928  0.000162  2.826599e-05
     3  0.261799  0.068539  0.017943  0.004698  0.001230  3.219673e-04
```

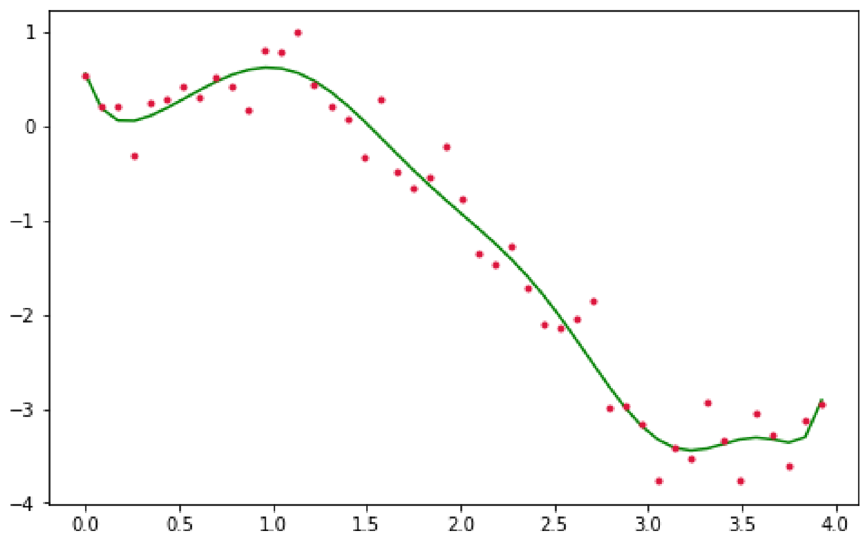
```
4  0.349066  0.121847  0.042533  0.014847  0.005182  1.809023e-03

            x^7           x^8           x^9          x^10         y
0  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.536589
1  3.854177e-08  3.363404e-09  2.935123e-10  2.561378e-11  0.217887
2  4.933346e-06  8.610313e-07  1.502783e-07  2.622851e-08  0.200831
3  8.429084e-05  2.206729e-05  5.777203e-06  1.512468e-06 -0.306169
4  6.314683e-04  2.204240e-04  7.694250e-05  2.685800e-05  0.244798
```

2.1.1 Linear Regression without Regularization

We define a function do_linear_reg that performs regression of above dataset df at a specified degree power.

[4]:
```
from sklearn.linear_model import LinearRegression  # Import 'LinearRegression'␣
 ↪from sklearn

def do_linear_reg(df, power):     # Defining a function to do polynomial␣
 ↪regression on 'df'
    cols = ['x']
    if power > 1:
        cols += ['x^%d' %(j) for j in range(2, power+1)]  # Considering colmuns␣
 ↪up to 'power'+1

    X = df[cols].values      # Features of regression
    y = df['y'].values       # Target of regression

    lin_reg = LinearRegression()  # Instantiate 'LinearRegression'
    lin_reg.fit(X, y)             # Fit the data

    r2_score = lin_reg.score(X, y)   # Calculate R^2 score
    coefs = lin_reg.coef_            # Calculate the coefficients/weights of␣
 ↪regression
    intercept = lin_reg.intercept_   # Calculate the intercept/bias term

    y_pred = lin_reg.predict(X)      # Predict the target
    df['yhat'] = y_pred              # Add prediction to 'df'
    plt.plot(df['x'], df['yhat'], color='green')       # Plot the prediction
    plt.plot(df['x'], df['y'], '.', color='crimson')   # Plot the truth target
    plt.gcf().set_size_inches(8, 5)
    plt.show()

    return r2_score, coefs, intercept    # Spit out R^2-score, weights, bias term
```

[5]:
```
do_linear_reg(df, 10)  # Perform regression up to power 11
```



```
[5]: (0.9730499031907367,
      array([-6.29571185e+00,  2.89348931e+01, -6.71520801e+01,  1.01133427e+02,
             -9.94047993e+01,  6.19916574e+01, -2.40840194e+01,  5.62586282e+00,
             -7.21606449e-01,  3.90059952e-02]),
      0.5551844911454311)
```

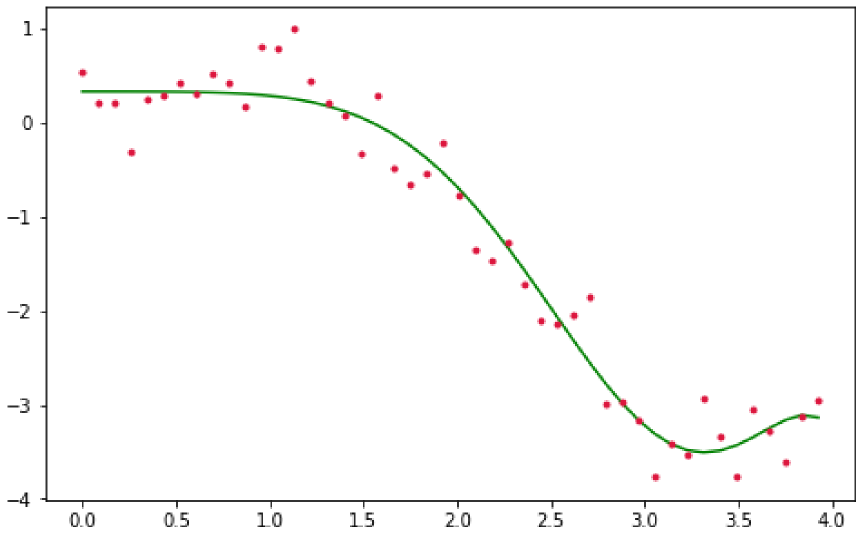
2.1.2 Ridge Regression

We define a function do_ridge_reg that performs Ridge regression of above dataset df at a specified degree power.

```
[6]: from sklearn.linear_model import Ridge    # Import 'Ridge' from sklearn

     # Defining a function to do Ridge polynomial regression on 'df' with penalty␣
      ↪factor 'a'
     def do_ridge_reg(df, power, a):
         cols = ['x']
         if power > 1:
             cols += ['x^%d' %(j) for j in range(2, power+1)]   # Considering colmuns␣
      ↪up to 'power'+1

         X = df[cols].values     # Features of regression
         y = df['y'].values      # Target of regression

         ridge = Ridge(alpha=a, max_iter=10e5)  # Instantiate Ridge regression with␣
      ↪penalty strength 'a'
         ridge.fit(X, y)                        # Fit the data
```

```
    r2_score = ridge.score(X, y)            # Calculate R^2 score
    coefs = ridge.coef_                     # Calculate the coefficients/weights␣
↪of regression
    intercept = ridge.intercept_            # Calculate the intercept/bias term

    y_pred = ridge.predict(X)               # Predict the target
    df['yhat'] = y_pred                     # Add prediction to 'df'
    plt.plot(df['x'], df['yhat'], color='green')       # Plot the prediction
    plt.plot(df['x'], df['y'], '.', color='crimson')   # Plot the truth target
    plt.gcf().set_size_inches(8, 5)
    plt.show()

    return r2_score, coefs, intercept       # Spit out R^2-score, weights, bias␣
↪term
```

[7]:
```
# Perform Ridge regression up to power 11 with penalty 100

do_ridge_reg(df, 10, 10e2)
```



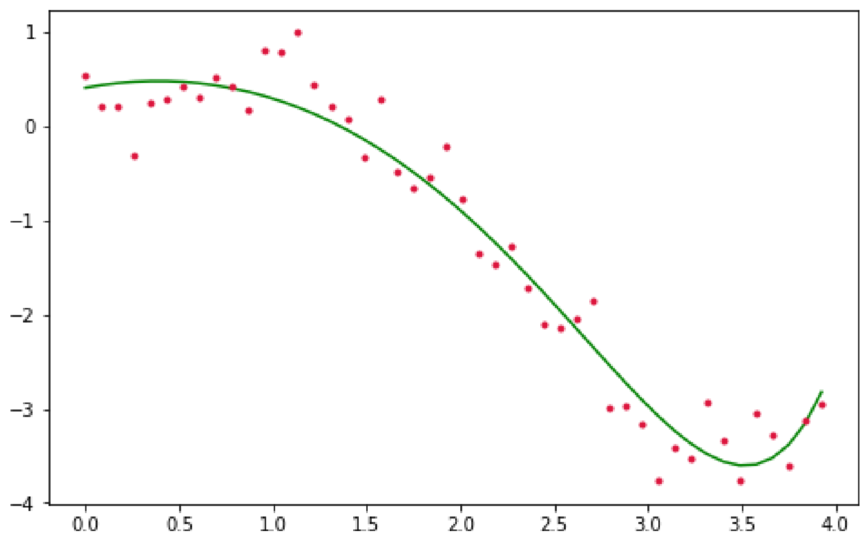
[7]:
```
(0.9608158063924134,
 array([-0.00055512, -0.00217727, -0.0049782 , -0.0089425 , -0.01309923,
        -0.01415432, -0.00644623,  0.00890727, -0.00214406,  0.00014723]),
 0.327215068908727)
```

2.1.3 Lasso Regression

We define a function do_lasso_reg that performs Lasso regression of above dataset df at a specified degree power.

```
[8]: from sklearn.linear_model import Lasso    # Import 'Lasso' from sklearn

     # Defining a function to do Lasso polynomial regression on 'df' with penalty␣
      ↪factor 'a'
     def do_lasso_reg(df, power, a):
         cols = ['x']
         if power > 1:
             cols += ['x^%d' %(j) for j in range(2, power+1)]    # Considering colmuns␣
      ↪up to 'power'+1

         X = df[cols].values        # Features of regression
         y = df['y'].values         # Target of regression

         # Instantiate Lasso regression with penalty strength 'a'
         lasso = Lasso(alpha=a, max_iter=10e5, normalize=True, tol=1e-4)
         lasso.fit(X, y)                    # Fit the data

         r2_score = lasso.score(X, y)       # Calculate R^2 score
         coefs = lasso.coef_                # Calculate the coefficients/weights of␣
      ↪regression
         intercept = lasso.intercept_       # Calculate the intercept/bias term

         y_pred = lasso.predict(X)          # Predict the target
         df['yhat'] = y_pred                # Add prediction to 'df'
         plt.plot(df['x'], df['yhat'], color='green')          # Plot the prediction
         plt.plot(df['x'], df['y'], '.', color='crimson')      # Plot the truth target
         plt.gcf().set_size_inches(8, 5)
         plt.show()

         return r2_score, coefs, intercept     # Spit out R^2-score, weights, bias␣
      ↪term
```

```
[9]: # Perform Lasso regression up to power 11 with penalty 0.001

     do_lasso_reg(df, 10, 0.001)
```



```
[9]: (0.954828796681126,
      array([ 3.53613496e-01, -4.32855847e-01, -3.66426461e-02, -0.00000000e+00,
              0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  4.40289735e-05,
              8.03485467e-06,  0.00000000e+00]),
      0.40846829050248434)
```

Note that many of the weights (regression coefficients) are **exactly 0**! So, Lasso regression can be employed as a **feature selection tool**.

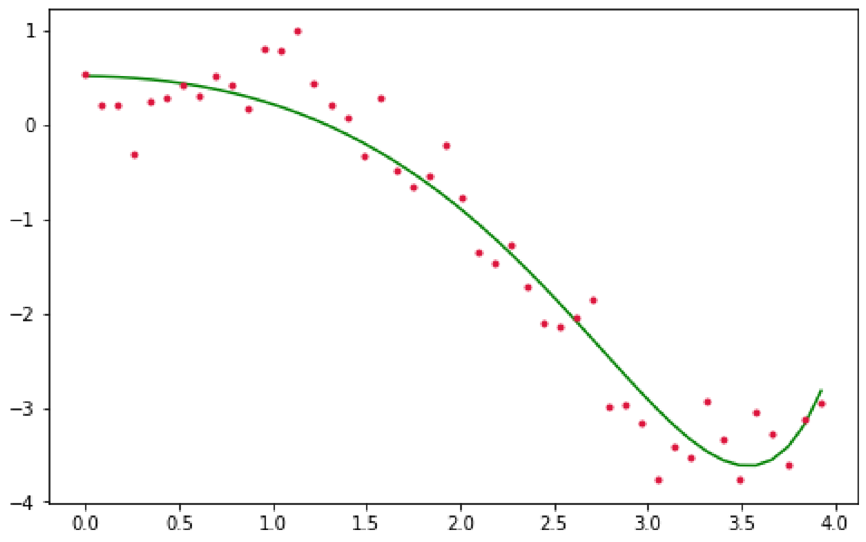
2.1.4 Elastic Net Regression

We define a function do_elastic_reg that performs Elastic Net regression of above dataset df at a specified degree power.

```
[10]: from sklearn.linear_model import ElasticNet     # Import 'ElasticNet' from sklearn

      # Defining a function to do elastic net polynomial regression on 'df' with␣
       ↪penalty factor 'a' and l1_ratio 'b'
      def do_elastic_reg(df, power, a, b):
          cols = ['x']
          if power > 1:
              cols += ['x^%d' %(j) for j in range(2, power+1)]    # Considering colmuns␣
       ↪up to 'power'+1

          X = df[cols].values        # Features of regression
          y = df['y'].values         # Target of regression
```

```
    # Instantiate Elastic Net regression with penalty strength 'a' and␣
↪'l1_ratio=b'
    elastic = ElasticNet(alpha=a, l1_ratio=b, normalize=True)
    elastic.fit(X, y)                  # Fit the data

    r2_score = elastic.score(X, y)     # Calculate R^2 score
    coefs = elastic.coef_              # Calculate the coefficients/weights of␣
↪regression
    intercept = elastic.intercept_     # Calculate the intercept/bias term

    y_pred = elastic.predict(X)        # Predict the target
    df['yhat'] = y_pred                # Add prediction to 'df'
    plt.plot(df['x'], df['yhat'], color='green')         # Plot the prediction
    plt.plot(df['x'], df['y'], '.', color='crimson')     # Plot the truth target
    plt.gcf().set_size_inches(8, 5)
    plt.show()

    return r2_score, coefs, intercept     # Spit out R^2-score, weights, bias␣
↪term
```

[11]:
```
# Perform Lasso regression up to power 11 with penalty 0.001 and l1_ratio 0.7

do_elastic_reg(df, 10, 0.001, 0.7)
```



[11]:
```
(0.9481961137453978,
 array([-1.53575642e-02, -2.28044309e-01, -4.87006646e-02, -5.21972761e-03,
        -0.00000000e+00,  0.00000000e+00,  2.52324987e-05,  1.83077009e-05,
         6.46806773e-06,  1.82872042e-06]),
 0.5156384926555218)
```

As observed above, fewer number of weights are exactly zero when compared with Lasso case. The general sentiment is that the Elastic Net regression is in between of Ridge and Lasso regressions.

2.2 California Housing Dataset

In here, we consider a more realistic example where we apply regularization. The following dataset contains the housing data of the state of California.

[12]:
```
# Loading California housing dataset

from sklearn.datasets import fetch_california_housing

california_housing = fetch_california_housing(as_frame=True)
```

[13]:
```
# Finding the description of the involved variables

print(california_housing.DESCR)
```

```
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

    :Number of Instances: 20640

    :Number of Attributes: 8 numeric, predictive attributes and the target

    :Attribute Information:
        - MedInc        median income in block
        - HouseAge      median house age in block
        - AveRooms      average number of rooms
        - AveBedrms     average number of bedrooms
        - Population    block population
        - AveOccup      average house occupancy
        - Latitude      house block latitude
        - Longitude     house block longitude

    :Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
http://lib.stat.cmu.edu/datasets/
```

```
The target variable is the median house value for California districts.

This dataset was derived from the 1990 U.S. census, using one row per census
block group. A block group is the smallest geographical unit for which the U.S.
Census Bureau publishes sample data (a block group typically has a population
of 600 to 3,000 people).

It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.

.. topic:: References

    - Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
      Statistics and Probability Letters, 33 (1997) 291-297
```

[14]:
```
# Defining a dataframe for the California housing dataset

import numpy as np
import pandas as pd

ca_df = pd.DataFrame(california_housing.data)        # Defining dataframe ca_df
ca_df.columns = california_housing.feature_names     # Defining the headers of␣
 ↪the dataframe
ca_df['Price'] = california_housing.target           # Adding Price column
```

[15]:
```
ca_df.shape   # Shape of 'ca_df'
```

[15]:
```
(20640, 9)
```

[16]:
```
ca_df.head()
```

[16]:
```
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85
3  5.6431      52.0  5.817352   1.073059       558.0  2.547945     37.85
4  3.8462      52.0  6.281853   1.081081       565.0  2.181467     37.85

   Longitude  Price
0    -122.23  4.526
1    -122.22  3.585
2    -122.24  3.521
3    -122.25  3.413
4    -122.25  3.422
```

This time, we do not perform a correlation analysis. Instead, we use regularized regression. In particular, we use Lasso regression as a feature selection method.

```
[17]: # Defining the features and the target of the model

      features = ca_df.columns[:-1]       # Feature names
      target = ca_df.columns[-1]          # Target name

      X = ca_df[features].values          # Features
      y = ca_df[target].values            # Target
```

```
[18]: # Breaking the data into train and test subsets

      from sklearn.model_selection import train_test_split

      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
       ↪random_state=3)
```

2.2.1 Linear Regression without Regularization

```
[19]: # Importing 'LinearRegression' through linear_model module

      from sklearn.linear_model import LinearRegression

      reg = LinearRegression()          # Instantiate linear regression
      reg.fit(X_train, y_train)         # Fit the train data

      r2_train_score = reg.score(X_train, y_train)    # Calculating R^2 score for train␣
       ↪data

      print('R^2 score for train dataset = ', round(r2_train_score, 4), '\n')
      print('Coefficients of Linear Model:', reg.coef_, '\n')
      print('Intercept:', reg.intercept_)
```

```
R^2 score for train dataset =  0.6097

Coefficients of Linear Model: [ 4.53053771e-01  1.01139824e-02 -1.30682277e-01
8.47640845e-01
 -3.48804147e-06 -3.35207458e-03 -4.23235312e-01 -4.37132608e-01]

Intercept: -37.3622661064085
```

```
[20]: # Finding the predictions of the model for test dataset

      y_pred = reg.predict(X_test)

      # Evaluating the performance of the model on the test dataset

      r2_test_score = reg.score(X_test, y_test)    # Calculating R^2 score for train
```

```
print('R^2 score for test dataset = ', round(r2_test_score, 4), '\n')
```

```
R^2 score for test dataset =  0.591
```

2.2.2 Lasso Regression as a Feature Selection Method

[21]:
```
# Importing 'LinearRegression' through linear_model module

from sklearn.linear_model import Lasso

penalty_factor = [0.0001, 0.0002, 0.0004, 0.0005, 0.001, 0.01]

for a in penalty_factor:
    lasso = Lasso(alpha=a, max_iter=10e5, normalize=True, tol=1e-4)    #␣
 ↪Instantiate lasso regression
    lasso.fit(X_train, y_train)        # Fit the train data

    r2_train_score = lasso.score(X_train, y_train)    # Calculating R^2 score for␣
 ↪train dataset
    y_pred = reg.predict(X_test)       # Finding predictions of the model for␣
 ↪test dataset
    r2_test_score = reg.score(X_test, y_test)        # Calculating R^2 score for␣
 ↪test dataset

    print('Penalty Factor:', a)
    print('R^2 score for train dataset = ', round(r2_train_score, 4), '\n')
    print('R^2 score for test dataset = ', round(r2_test_score, 4), '\n')
    print('Coefficients of Linear Model:', lasso.coef_, '\n')
    print('Intercept:', lasso.intercept_, '\n******************************')
```

```
Penalty Factor: 0.0001
R^2 score for train dataset =  0.6038

R^2 score for test dataset =  0.591

Coefficients of Linear Model: [ 0.41899438  0.01062602 -0.0643584   0.47283896
-0.          -0.00237304
 -0.36287832 -0.36864814]

Intercept: -31.16400615808835
******************************
Penalty Factor: 0.0002
R^2 score for train dataset =  0.5866
```

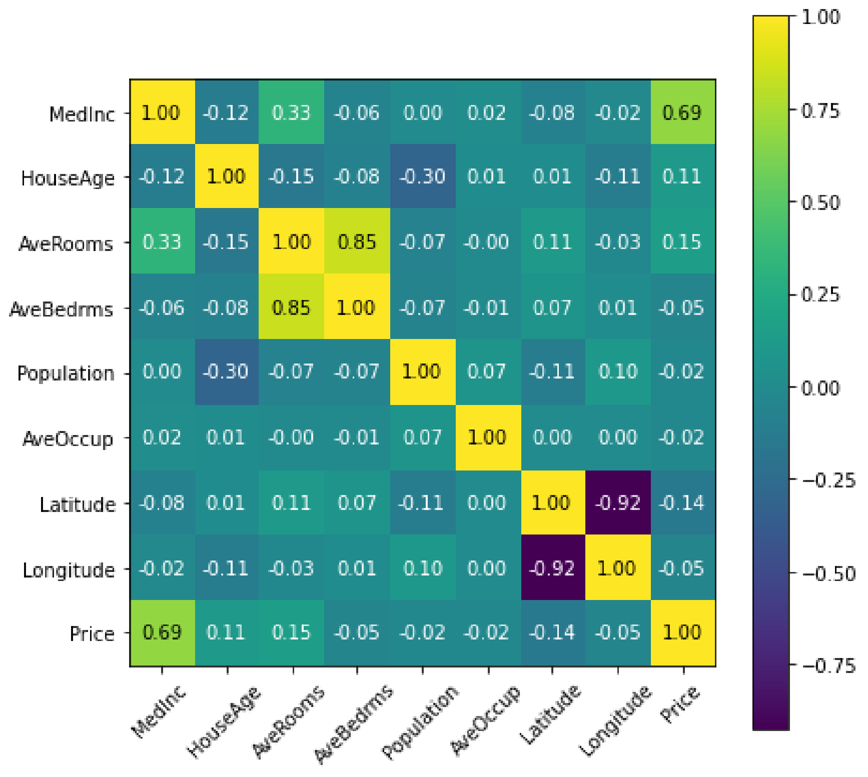
```
R^2 score for test dataset =  0.591

Coefficients of Linear Model: [ 0.38581058  0.01102402 -0.          0.10660339
-0.         -0.00137014
 -0.30254781 -0.30014893]

Intercept: -24.95696456533762
******************************
Penalty Factor: 0.0004
R^2 score for train dataset =  0.5464

R^2 score for test dataset =  0.591

Coefficients of Linear Model: [ 0.3890422   0.01152146 -0.          0.
-0.         -0.
 -0.13680877 -0.12416842]

Intercept: -9.734187700554518
******************************
Penalty Factor: 0.0005
R^2 score for train dataset =  0.5206

R^2 score for test dataset =  0.591

Coefficients of Linear Model: [ 0.39110984  0.01185677 -0.          0.
0.         -0.
 -0.05968579 -0.04198419]

Intercept: -2.67275447720176
******************************
Penalty Factor: 0.001
R^2 score for train dataset =  0.4829

R^2 score for test dataset =  0.591

Coefficients of Linear Model: [ 0.3616773   0.00713597 -0.          0.
-0.         -0.
 -0.         -0.        ]

Intercept: 0.47017136998675
******************************
Penalty Factor: 0.01
R^2 score for train dataset =  0.0

R^2 score for test dataset =  0.591

Coefficients of Linear Model: [ 0.  0.  0. -0. -0. -0. -0. -0.]
```

```
Intercept: 2.0751977311738647
******************************
```

2.2.3 Comparing the Regularization Method and Correlation Results

In here, it would be appropriate to compare the results of the Lasso regression on feature selection with those of the Pearson correlation.

[22]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import mlxtend
from mlxtend.plotting import heatmap

cols = ca_df.columns      # List of colmuns of dataframe ca_df

cm = np.corrcoef(ca_df[cols].values.T)      # Calculate Pearson correlation
hm = heatmap(cm, figsize=(7,7), row_names=cols, column_names=cols)   # Represent␣
 ↪correlation by a heat map
plt.show()
```



	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price
MedInc	1.00	-0.12	0.33	-0.06	0.00	0.02	-0.08	-0.02	0.69
HouseAge	-0.12	1.00	-0.15	-0.08	-0.30	0.01	0.01	-0.11	0.11
AveRooms	0.33	-0.15	1.00	0.85	-0.07	-0.00	0.11	-0.03	0.15
AveBedrms	-0.06	-0.08	0.85	1.00	-0.07	-0.01	0.07	0.01	-0.05
Population	0.00	-0.30	-0.07	-0.07	1.00	0.07	-0.11	0.10	-0.02
AveOccup	0.02	0.01	-0.00	-0.01	0.07	1.00	0.00	0.00	-0.02
Latitude	-0.08	0.01	0.11	0.07	-0.11	0.00	1.00	-0.92	-0.14
Longitude	-0.02	-0.11	-0.03	0.01	0.10	0.00	-0.92	1.00	-0.05
Price	0.69	0.11	0.15	-0.05	-0.02	-0.02	-0.14	-0.05	1.00

There are a couple of comments in order:

- MedInc is by far the most significant feature for the target Price.

- The rest of the feature have little correlation with Price. The next most significant feature is AveRooms. Note that AveRooms and AveBedrms are *highly correlated*! Therefore, one should not consider both features independently. The same holds for Latitude and Longitude features.
- It should be noted that although Lasso regression works with a different mechanism, it gives MedInc the greatest weight.
- Lasso regularization (with appropriate penalty factor) sets the weights of 4 features *exactly zero*. The rest of the nonzero weights (other than MedInc) are small weights which shows that these features are relatively insignificant.

As is observed from above the findings of Lasso regularization are in agreement with the results of Pearson correlation analysis.

```
[ ]:
```