Intuition

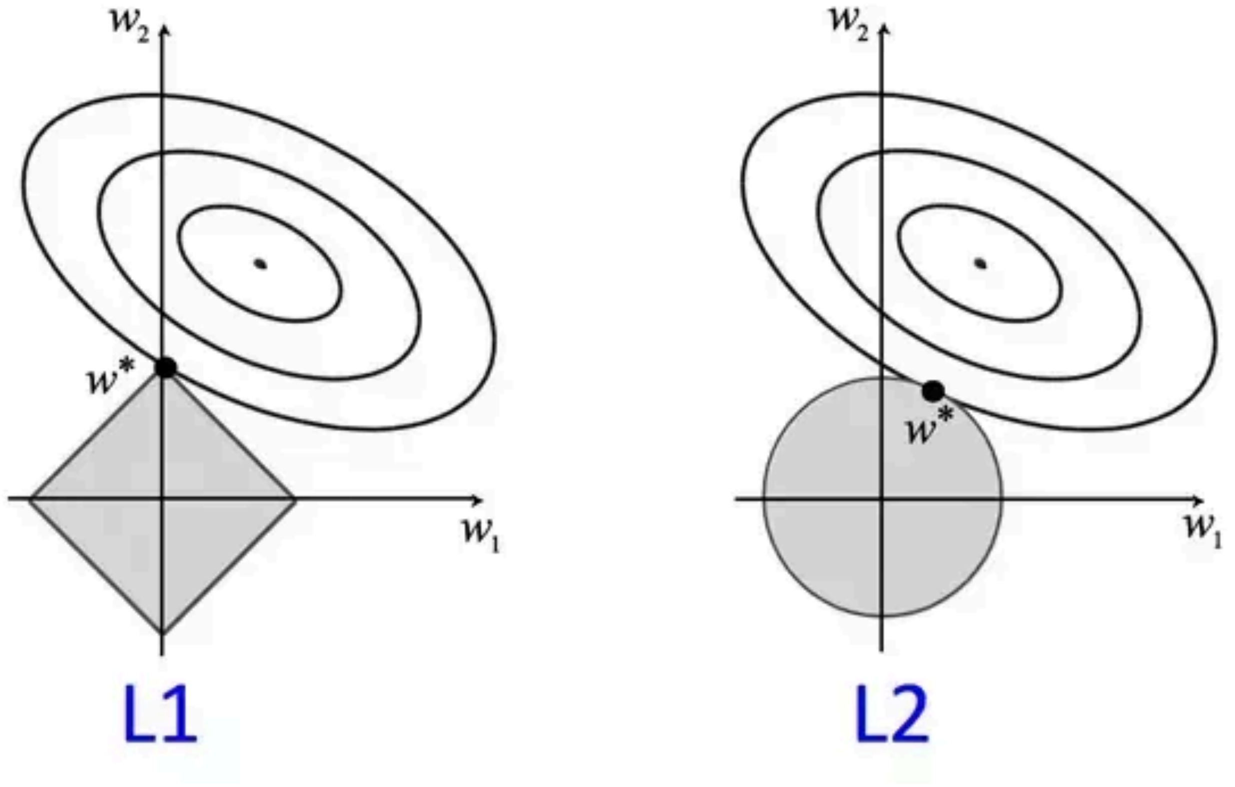
MACHINE LEARNING AND MATHEMATICS

Understanding L1 and L2 regularization with analytical and probabilistic views

Derive L1 and L2 regularization via analytical and probabilistic solution

Yuki Shizuya Follow 11 min read · May 25, 2024

584 4



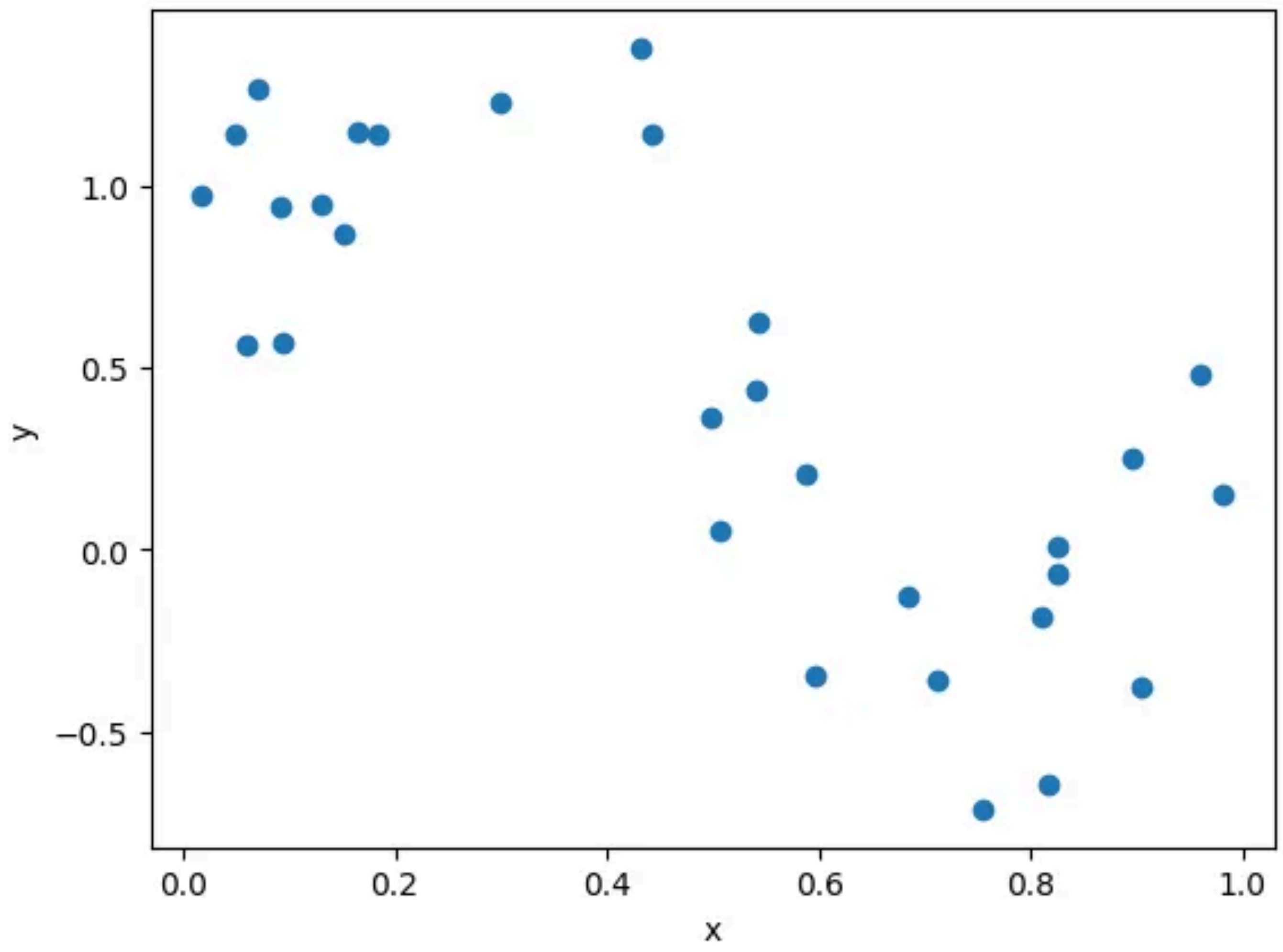
L1 and L2 regularization intuition referenced from here

When you learn about machine learning, you're bound to come across L1 and L2 regularization. I think many awesome blogs explain these concepts intuitively with visualizations. However, only a few blogs explain L1 and L2 regularization with analytic and probabilistic views in detail. So, I decided to write about both regularizations with both perspectives. In this blog, I will introduce L1 and L2 regularizations with detailed mathematical derivations and visualization to help you understand these concepts well.

Table of Contents

1. The overview of regularization

Firstly, let's recap the concept of regularization. To understand concretely, let's assume we have small data with two dimension below [1].

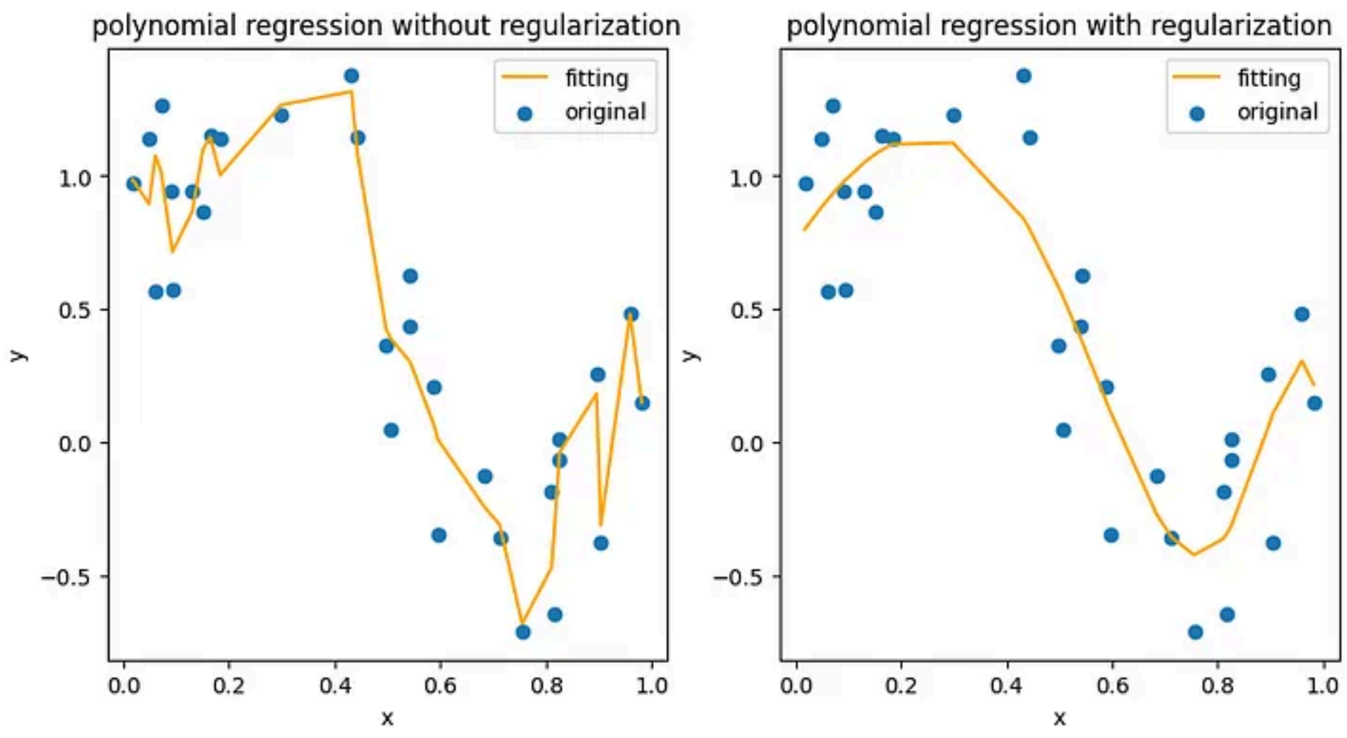


The visualization of the sample data

As you can see, this data is non-linear. Since the data is non-linear, we can easily guess that the simple linear regression doesn't fit this data well. In this case, let's consider polynomial regression to express non-linear data. To understand the importance of regularization, we use 15 polynomial regression, meaning we use an overly complex function to predict data.

$$\textbf{Polynomial regression:}$$
$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_{15} x^{15}$$
$$= \sum_{p=0}^{15} \beta_p x^p$$

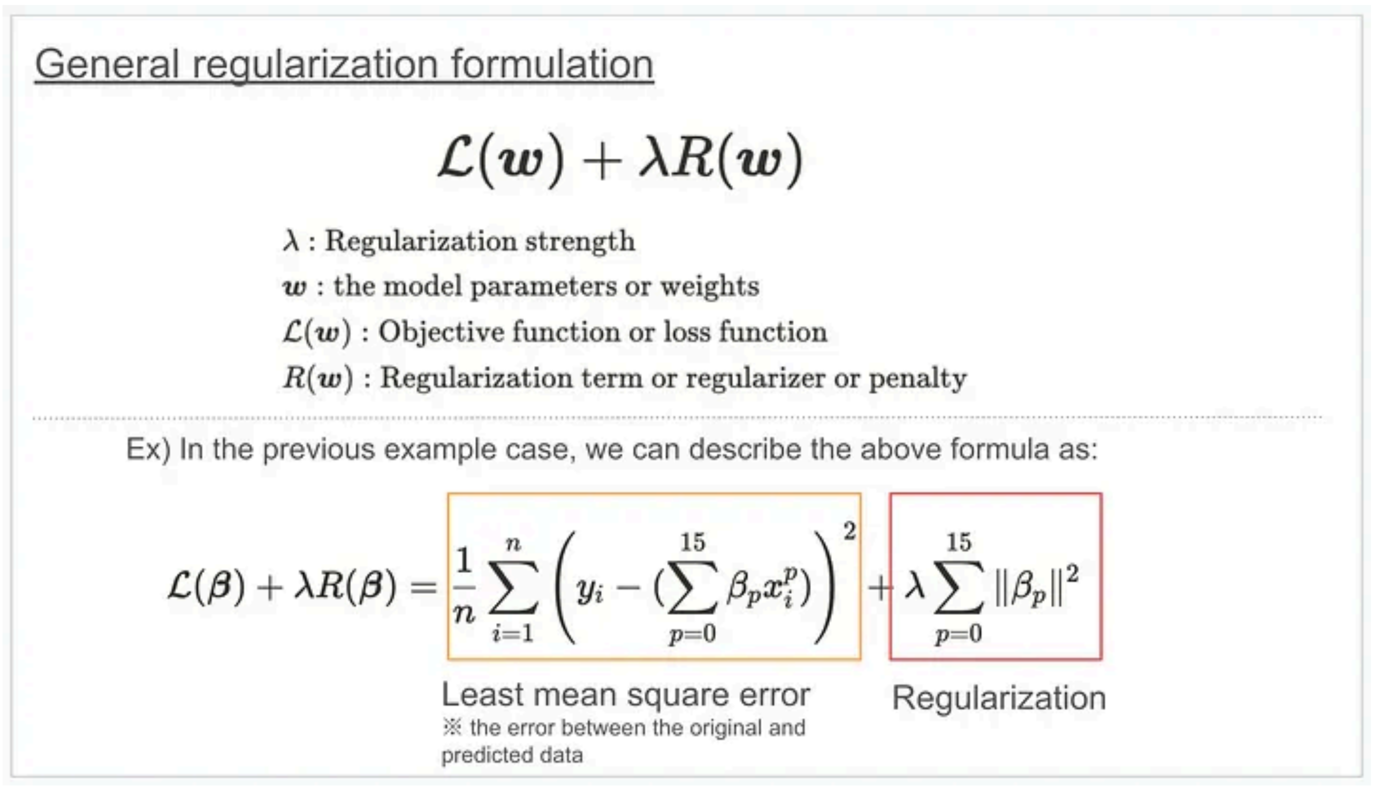
I use the PolynomialFeatures and Ridge classes in the scikit-learn library to fit the data. The result is shown below.



(left) The predicted result without regularization (right) The predicted result with regularization from the author

The left figure shows the polynomial regression without regularization, and the right one shows the polynomial regression with regularization. The polynomial regression without regularization overfits the data because we used an overly complex function compared to the original data. On the other hand, the polynomial regression with regularization can fit better than the one without regularization and reduce the complexity of the model shape. Generally, we use regularization like the example above to prevent the model from overfitting.

How can we do regularization? Theoretically, we add the regularization term to the objective function and optimize the parameters based on it, as shown below.



General regularization formulation

$$\mathcal{L}(\boldsymbol{w}) + \lambda R(\boldsymbol{w})$$

λ : Regularization strength
$\boldsymbol{w}$: the model parameters or weights
$\mathcal{L}(\boldsymbol{w})$: Objective function or loss function
$R(\boldsymbol{w})$: Regularization term or regularizer or penalty

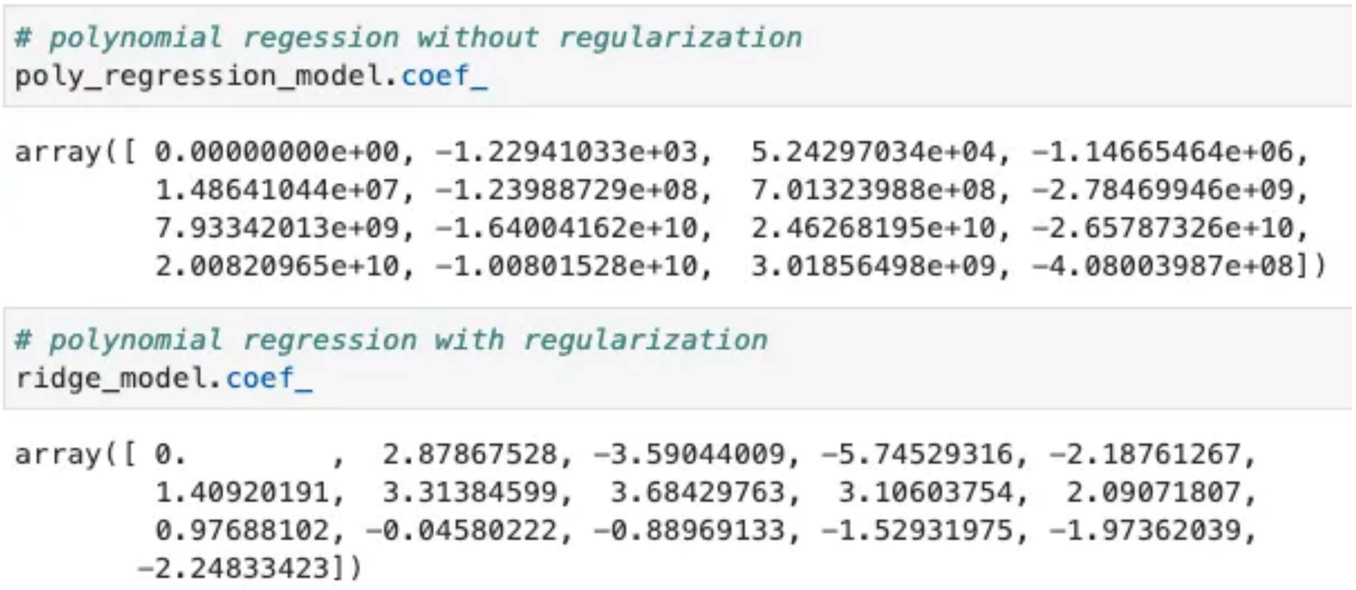
Ex) In the previous example case, we can describe the above formula as:

$$\mathcal{L}(\boldsymbol{\beta}) + \lambda R(\boldsymbol{\beta}) = \underbrace{\boxed{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - (\sum_{p=0}^{15}\beta_p x_i^p)\right)^2}}_{\text{Least mean square error}} + \underbrace{\boxed{\lambda\sum_{p=0}^{15}\|\beta_p\|^2}}_{\text{Regularization}}$$

※ the error between the original and predicted data

General regularization formulation

The regularization term imposes a penalty for the coefficient values not increasing. Why do we impose a penalty for the coefficients? To understand it intuitively, let's look at the coefficients of the previous example.



```
# polynomial regession without regularization
poly_regression_model.coef_

array([ 0.00000000e+00, -1.22941033e+03,  5.24297034e+04, -1.14665464e+06,
        1.48641044e+07, -1.23988729e+08,  7.01323988e+08, -2.78469946e+09,
        7.93342013e+09, -1.64004162e+10,  2.46268195e+10, -2.65787326e+10,
        2.00820965e+10, -1.00801528e+10,  3.01856498e+09, -4.08003987e+08])

# polynomial regression with regularization
ridge_model.coef_

array([ 0.        ,  2.87867528, -3.59044009, -5.74529316, -2.18761267,
        1.40920191,  3.31384599,  3.68429763,  3.10603754,  2.09071807,
        0.97688102, -0.04580222, -0.88969133, -1.52931975, -1.97362039,
       -2.24833423])
```

The coefficient values of each model

The above line shows the coefficients of polynomial regression without regularization, and the below line shows the coefficients of polynomial regression with regularization(L2). The polynomial regression without regularization has larger coefficient values. Intuitively, if the model has larger coefficient values, it means that the model can change its shape drastically. Thus, the model without regularization fits the given data more accurately but not generally. Meanwhile, the model with regularization can search parameters to fit the given data more generally because the coefficient values are relatively small.

So far, you understand the concept of the regularization and its power. Now, let’s dive into the theoretical background behind regularization.

2. L1 regularization

L1 regularization [2] adds the absolute value of the magnitude of the coefficient, or the l1-norm of the coefficients, as the regularization term. L1 regularization helps the feature selection of the coefficients, which means **it can reduce the number of irrelevant independent variables**. Specifically, regression model with L1 regularization is called Least Absolute Shrinkage and Selection Operator (Lasso) regression. The formula of L1 regularization is below.

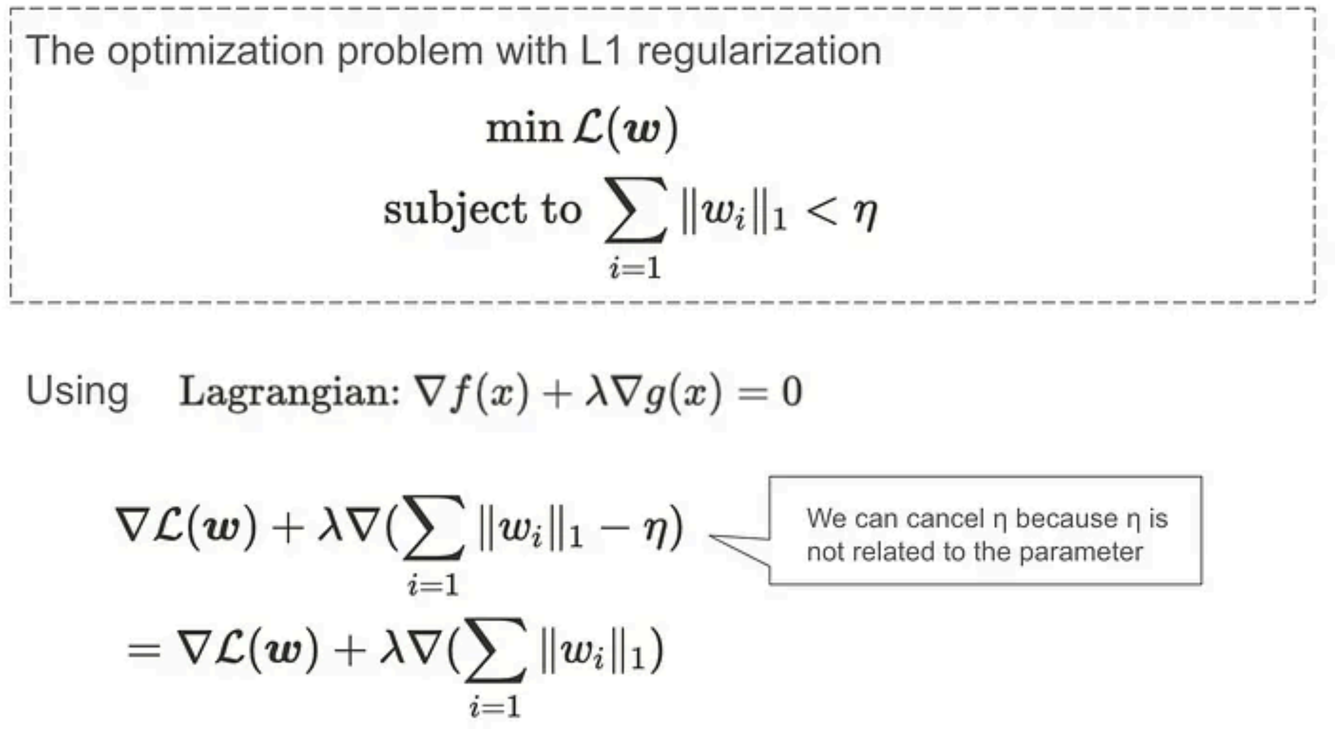
$$\min_{\boldsymbol{w}} \left(\mathcal{L}(\boldsymbol{w}) + \lambda \sum_{i} \|w_i\|_1 \right)$$

The L1 regularization formula

where w is the parameter. From now on, we will learn how to solve this problem.

2.1 Analytical derivation of L1 regularization

How can we optimize the L1 regularization formula? To solve it analytically, this formula can be seen as constraint optimization with Lagrange multipliers.



The optimization problem with L1 regularization

$$\min \mathcal{L}(\boldsymbol{w})$$
$$\text{subject to } \sum_{i=1} \|w_i\|_1 < \eta$$

Using $\textbf{Lagrangian: } \nabla f(x) + \lambda \nabla g(x) = 0$

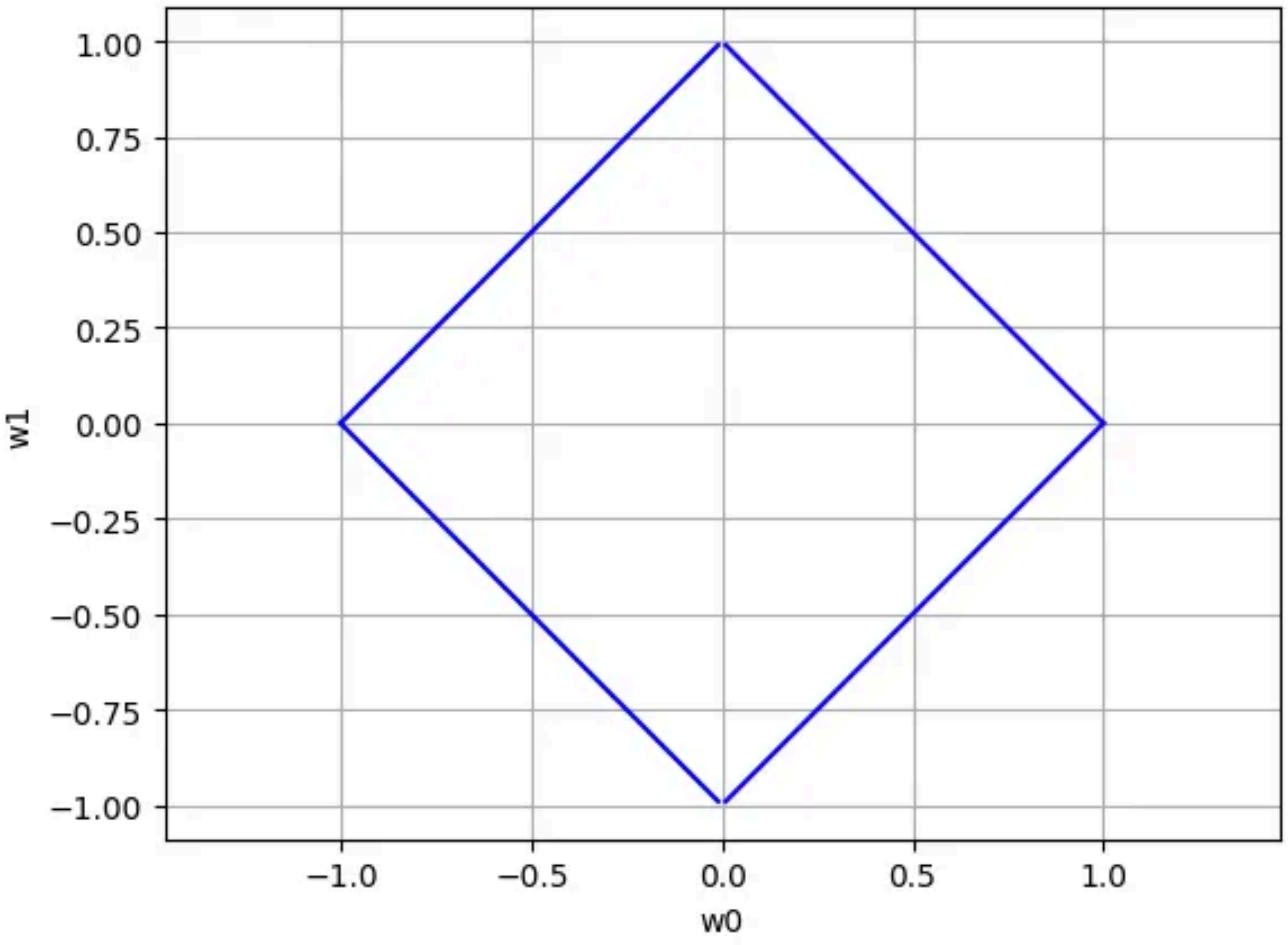
$$\nabla \mathcal{L}(\boldsymbol{w}) + \lambda \nabla (\sum_{i=1} \|w_i\|_1 - \eta)$$

We can cancel η because η is not related to the parameter

$$= \nabla \mathcal{L}(\boldsymbol{w}) + \lambda \nabla (\sum_{i=1} \|w_i\|_1)$$

The optimization problem with L1 regularization

As you can see, the last equation is the same as the L1 regularization formula. Next, how do we optimize parameters analytically based on the derived Lagrangian? Unfortunately, we generally cannot have a closed solution for L1 regularization because we cannot differentiate the regularization term. You can check this fact in the figure below. Assume that we have two parameters function, and the L1 regularization term can be depicted as:



The visualization of L1 norm function

When we calculate the derivative on the edge, it has two different values from right and left side, so we cannot differentiate on the edge (You can check more mathematical detail in [7]). Although there are a few exceptions to finding closed-form solutions, we can find a closed-form solution when the matrix X is orthonormal [3], and the number of parameters is one. However, such a situation rarely happens in practical analysis.

Then, how do we find parameters? The most common methods for solving the lasso problem are subgradient and coordinate descent. In the scikit-learn implementation, they use coordinate descent to optimize the Lasso problem, so let's learn the coordinate descent [4].

The coordinate descent idea is straightforward. Assuming we have an n-dimensional function *f*, we minimize *f* by successively minimizing each parameter dimension of f repeatedly. Let's see the mathematical definition.

$$
\begin{aligned}
&\text{Assume } f(\boldsymbol{x}), \quad f : \mathbb{R}^n \to \mathbb{R}, \quad \boldsymbol{x} \in \mathbb{R}^n \\
&\text{Set initial parameter } x_i^{(0)} \text{ , and repeat} \\
&\text{for } k = 1, 2, 3, \cdots \\
&x_1^{(k)} \in \arg\min_{x_1} f(x_1, x_2^{(k-1)}, x_3^{(k-1)}, \cdots, x_n^{(k-1)}) \\
&x_2^{(k)} \in \arg\min_{x_2} f(x_1^{(k)}, x_2, x_3^{(k-1)}, \cdots, x_n^{(k-1)}) \\
&x_3^{(k)} \in \arg\min_{x_3} f(x_1^{(k)}, x_2^{(k)}, x_3, \cdots, x_n^{(k-1)}) \\
&\vdots \\
&x_n^{(k)} \in \arg\min_{x_n} f(x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, \cdots, x_n)
\end{aligned}
$$

The formula of the coordinate descent

As its name suggests, we calculate parameters individually and update them based on past values. This process continues until convergence or the maximum repetitive count we set is achieved. It is very simple, isn’t it? Now, let’s look at a concrete example of the Lasso formulation. We consider the Lasso problem with mean-squared error(MSE). So, the formula will be as follows:

$$\text{Let } \boldsymbol{y} \in \mathbb{R}^n, \boldsymbol{X} \in \mathbb{R}^{n\times p}, \boldsymbol{X}_i \in \mathbb{R}^n, \boldsymbol{\beta} \in \mathbb{R}^p$$
$$\min_{\beta \in \mathbb{R}^p} \left(\frac{1}{2} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1 \right)$$
$$※ \|\boldsymbol{\beta}\|_1 = \sum_{i=1}^{p} |\beta_i|$$

The formula for Lasso problem

Since we proceed with the coordinate descent, we must minimize this formula over each parameter with other non-target parameters fixed.

For the MSE term:
$$\mathbf{0} = \nabla f(\beta) = \nabla \frac{1}{2} \|\boldsymbol{y} - \boldsymbol{X}\beta\|_2^2 = (\boldsymbol{X}^T\boldsymbol{X})\beta - \boldsymbol{X}^T\boldsymbol{y} = \boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y})$$
For each i :
$$\mathbf{0} = \nabla_i f(\beta) = \underbrace{\underset{(p \times n)}{X_i^T}\;(\underbrace{\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y}}_{(n \times p) \times (p,) = (n,)})}_{(p,)}$$

The coordinate descent update formula for MSE term

The last formulation is a bit tricky (at least for me). When you consider each dimension of the term, you can understand that only the i-th row of the transpose matrix of X is relevant to the i-th gradient. To formulate the i-th parameter gradient, we need to reformulate the above formula.

$$\mathbf{0} = \nabla_i f(\boldsymbol{\beta}) = X_i^T(X_i\beta_i + X_{j\neq i}\beta_{j\neq i} - \boldsymbol{y})$$

$$\beta_i = \frac{X_i^T(\boldsymbol{y} - X_{j\neq i}\beta_{j\neq i})}{X_i^T X_i}$$

The gradient of the coordinate descent update formula for MSE term

To obtain the parameter update equation, we need to decompose the XB term. We divide it into the i-th column of the XB and the other columns. As you can see, we can derive the parameter-update formula. How about the L1 regularization term? We will introduce soft-thresholding to solve it. Since we cannot differentiate the L1 term, we utilize the subdifferentials and subgradients to approximate it. For the concept of the subgradients, you can check the theorem in the [5]. Using subgradients, we can derive as:

$$\partial_{\beta_i}\lambda|\beta_i| = \begin{cases} -\lambda & \text{if } \beta_i < 0 \\ [-\lambda, \lambda] & \text{if } \beta_i = 0 \\ \lambda & \text{if } \beta_i > 0 \end{cases}$$

The subgradients for the parameters

We substitute the L1 regularization term for the Lasso problem and get:

$$0 = \begin{cases} X_i^T(X_i\beta_i + X_{j\neq i}\beta_{j\neq i} - \boldsymbol{y}) - \lambda & \text{if } \beta_i < 0 \\ [X_i^T(X_{j\neq i}\beta_{j\neq i} - \boldsymbol{y}) - \lambda, X_i^T(X_{j\neq i}\beta_{j\neq i} - \boldsymbol{y}) + \lambda] & \text{if } \beta_i = 0 \\ X_i^T(X_i\beta_i + X_{j\neq i}\beta_{j\neq i} - \boldsymbol{y}) + \lambda & \text{if } \beta_i > 0 \end{cases}$$

The gradient for the Lasso problem

We can reformulate the last equations to solve for β as:

$$\text{Assume:} \quad X_i^T(\boldsymbol{y} - X_{j\neq i}\beta_{j\neq i}) = a_i$$

$$\beta_i = S_\lambda(a_i) = \begin{cases} \dfrac{a_i + \lambda}{X_i^T X_i} & \text{if } a_i < -\lambda \\ 0 & \text{if } -\lambda \leq a_i \leq \lambda \\ \dfrac{a_i - \lambda}{X_i^T X_i} & \text{if } a_i > \lambda \end{cases}$$

$$S_\lambda(a_i) = \operatorname{sgn}(a_i)\max\left(|a_i| - \lambda, 0\right)$$

The parameter update formula

We can organize the conditional branch using the sign and max function. Now, we update our parameters based on this final formula repeatedly until convergence. Let's solve a concrete example. For the visualization, we assume to optimize the function with two parameters and no bias term.

$$y = \beta_0 x_0 + \beta_1 x_1 + \epsilon$$

The sample function

We use $\lambda = 1$ as the regularization strength and update the parameter values based on the equation we derived.

As you can see, the convergence is fast in this case. The coefficients' values are almost the same as the scikit-learn implementation.

```
# coordinate descent from scratch
b0 = 1.00, b1 = 2.00

# scikit-learn
b0 = 1.11, b1 = 2.04
```

For the practical case, you should use the scikit-learn implementation because they use cython to compute the coordinate descent; their implementation is much faster than the native Python code. Later, I will share the code I used with you.

2.2 Probabilistic derivation of L1 regularization

Before diving into the L1 regularization with the probabilistic aspect, we should know the MAP estimation. Let's learn the MAP estimation and its difference between maximum likelihood estimation. If you are already familiar with it, you can skip the next section.

Prerequisite — Maximum likelihood estimation and MAP estimation

We assume that we have the multiple linear function, and the error between the observed data point and the predicted value follows the Gaussian distribution with the mean 0 and the standard deviation σ. The likelihood, or probability density that the error takes under the assumed distribution can be derived as follows:

Assume :

$$\begin{aligned}&(1)y_i = \beta_0 x_{i0} + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \epsilon\\&(2)\hat{y}_i = \beta_0 x_{i0} + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}\\&(3)\epsilon_i = y_i - \hat{y}_i\end{aligned}$$

Likelihood:

$$\begin{aligned}P(\boldsymbol{\epsilon}|\beta) &= \Pi_{i=1}^{n} P_Y(\epsilon_i|\mu = 0, \sigma^2)\\&= \Pi_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{(y_i-(\beta_0x_{i0}+\beta_1x_{i1}+\cdots+\beta_px_{ip})^2}{2\sigma^2}}\end{aligned}$$

The assumed setting

where the number of the data points is n, and the number of parameters is p. You want to find the parameters that maximize this probability density, which is called maximum likelihood estimation (MLE). We can write the MLE formulation as follows:

$$\hat{\beta}_{\text{MLE}} = \arg\max_{\beta} \log P(\boldsymbol{\epsilon}|\boldsymbol{\beta})$$

Maximum likelihood estimation formula

In general, we take a logarithm to change the multiplying to the summation. In frequentist statistics, we consider the parameters as constant values but unknown, so we find the parameters to maximize the likelihood using differentiation.

On the other hand, we can take the parameters as random variables in Bayes' theorem. Applying Bayes' theorem, we can view the likelihood as follows:

$$\begin{aligned}
\text{posterior} &= \frac{\text{likelihood} \cdot \text{prior}}{\text{evidence}} \\
p(\theta|y) &= \frac{p(y|\theta) \cdot p(\theta)}{p(y)} \\
&\propto p(y|\theta) \cdot p(\theta)
\end{aligned}$$

Bayes' theorem

The posterior probability is proportional to the likelihood and prior probability. In this setting, we should maximize the posterior probability rather than the likelihood like MLE. Maximizing the posterior probability is called maximum a posteriori probability estimate, or MAP estimation. We can formulate as follows:

$$\begin{aligned}\hat{\theta}_{\text{MAP}} &= \arg\max_{\theta} P(\theta|y) \\ &= \arg\max_{\theta} P(y|\theta) \cdot P(\theta) \\ &\approx \arg\max_{\theta} \log P(y|\theta) P(\theta) \\ &= \arg\max_{\theta} (\log P(y|\theta) + \log P(\theta))\end{aligned}$$

The derivation of MAP estimation

When we apply it to our previous example in contrast to the MLE:

$$\begin{aligned}\hat{\boldsymbol{\beta}}_{\text{MAP}} &= \arg\max_{\boldsymbol{\beta}} (\log P(\boldsymbol{\epsilon}|\boldsymbol{\beta}) + \log P(\boldsymbol{\beta})) \\ \hat{\boldsymbol{\beta}}_{\text{MLE}} &= \arg\max_{\boldsymbol{\beta}} (\log P(\boldsymbol{\epsilon}|\boldsymbol{\beta}))\end{aligned}$$

MAP estimation and MLE

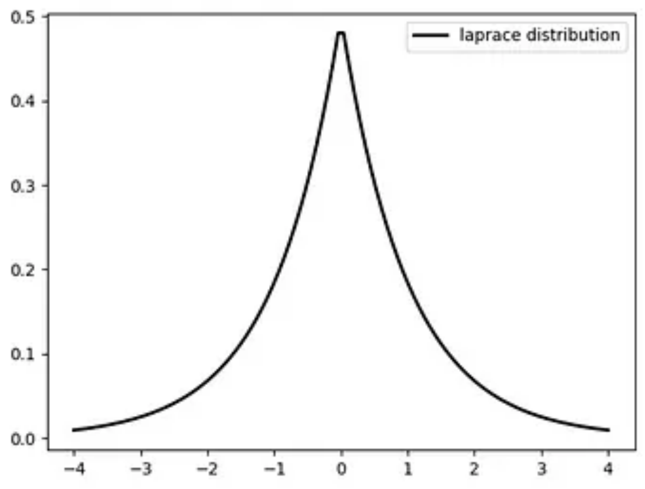
You realize that MAP estimation needs the prior probability. We can use any probability distribution for it. Now, back to the L1 regularization.

Probabilistic derivation of L1 regularization using Laplace prior

When we choose Laplace distribution for the prior, the MAP estimation becomes the L1 regularization formula. Let's derive it! Laplacian priors is one of the probability distribution that has the shape below.

Laplacian Priors:

$$\text{Laplace}(\mu, b) = \frac{1}{2b} \exp^{-\frac{|x-\mu|}{b}}$$



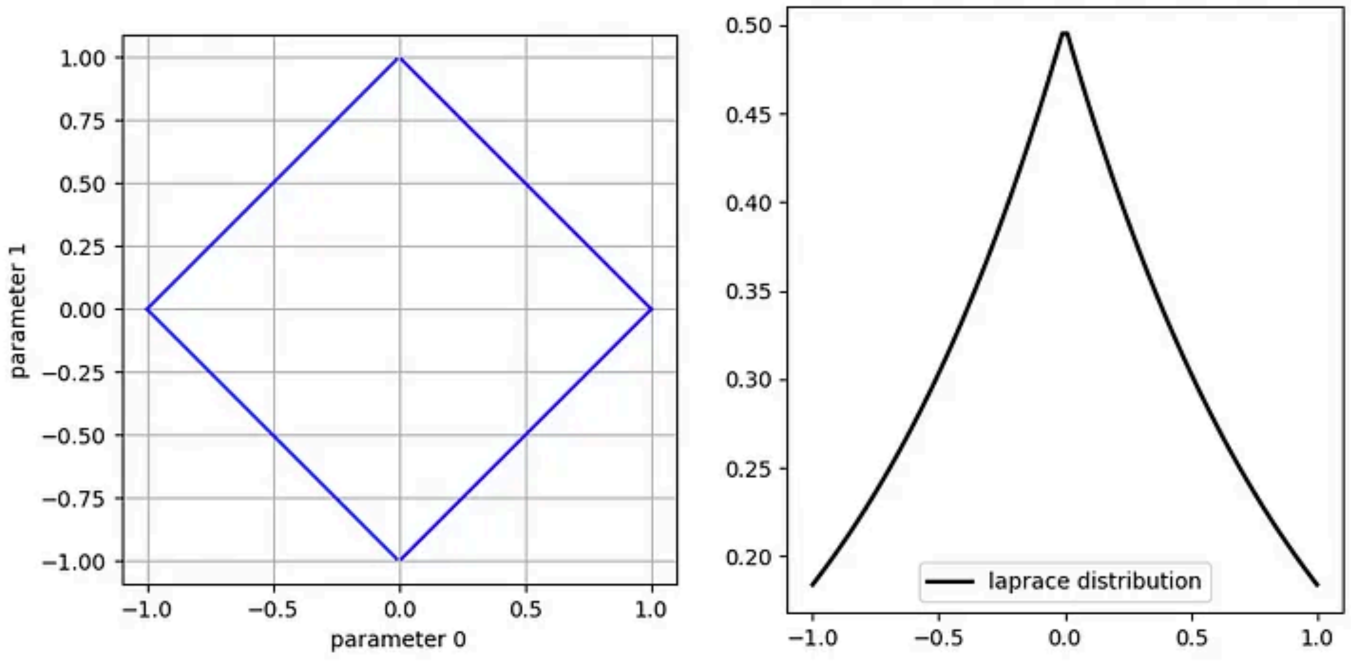
Laplace distribution

You can notice the exponent term of the exponential function is similar to the L1 regularization term. Now, we substitute the Laplace prior with mean 0 for the prior probability in the MAP estimation.

$$
\begin{aligned}
&= \arg\max_{\beta}\left(\log \Pi_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{(y_i-(\beta_0 x_{i0}+\beta_1 x_{i1}+\cdots+\beta_p x_{ip}))^2}{2\sigma^2}} + \log \Pi_{j=0}^{p} \frac{1}{2b} \exp^{-\frac{|\beta_j|}{b}}\right) \\
&= \arg\max_{\beta}(-\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-(\beta_0 x_{i0}+\beta_1 x_{i1}+\cdots+\beta_p x_{ip}))^2 + \\
&\qquad p\log\frac{1}{2b} - \sum_{j=0}^{p}\frac{|\beta_j|}{b}) \\
&= \arg\min_{\beta}\frac{1}{2\sigma^2}(\sum_{i=1}^{n}(y_i-(\beta_0 x_{i0}+\beta_1 x_{i1}+\cdots+\beta_p x_{ip}))^2 + \frac{2\sigma^2}{b}\sum_{j=0}^{p}|\beta_j|) \\
&= \arg\min_{\beta}(\|y-\boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1)
\end{aligned}
$$

MAP estimation using Laplace prior

As you can see, the last formula is the same as the L1 regularization! Intuitively, the shape of the Laplacian distribution is much sharper than the Gaussian distribution. It is similar to the L1 regularization term in the analytical derivation section as shown below.



(Left) L1 regularization term (right) Laplace distribution

So far, we understand L1 regularization derivation using both analytical and probabilistic perspectives. Next, Let's learn L2 regularization.

3. L2 regularization

L2 regularization adds the squared values of coefficients, or the l2-norm of the coefficients, as the regularization term. **L2 regularization helps to promote smaller coefficients**. A regression model with L2 regularization is called Ridge regression. The formula of L2 regularization is below.

$$\min_{\boldsymbol{w}} \left(\mathcal{L}(\boldsymbol{w}) + \lambda \sum_i \|w_i\|_2^2 \right)$$

L2 regularization formula

3.1 Analytical derivation of L2 regularization

Like the L1 regularization, we see the L2 regularization problem as constraint optimization with Lagrange multipliers.

The optimization problem with L2 regularization

$$\min \mathcal{L}(\boldsymbol{w})$$
$$\text{subject to} \sum_{i=1} \|w_i\|_2 < \eta$$

Using $\textbf{Lagrangian: } \nabla f(x) + \lambda \nabla g(x) = 0$

$$\nabla \mathcal{L}(\boldsymbol{w}) + \lambda \nabla (\sum_{i=1} \|w_i\|_2 - \eta)$$
$$= \nabla \mathcal{L}(\boldsymbol{w}) + \lambda \nabla (\sum_{i=1} \|w_i\|_2)$$

The oprimization problem with L2 regularization

The last equation is the same as the L2 regularization formula. In contrast to the L1 regularization, this formula can be differentiated. So, we don't need to introduce new concepts; we just differentiate them!

$$
\begin{aligned}
0 &= \nabla \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \nabla \frac{\lambda}{2}\|\boldsymbol{\beta}\|_2^2 \\
&= \boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y}) + \lambda\boldsymbol{\beta} \\
\boldsymbol{\beta} &= (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}
\end{aligned}
$$

The derivation of the parameters with differentiation

Now, we obtain a closed-form solution for L2 regularization. Let's implement it and compare the scikit-learn ridge results.

```
# sample data
X = np.random.randn(100, 2)
beta = np.array([2, 3]).reshape(1, 2)
Y = X @ beta.T + np.random.normal(beta.shape[0])

lam = 1.0
inv_mat = np.linalg.inv(X.T @ X + np.eye((X.T @ X).shape[0]))

ridge_coef = inv_mat @ X.T @ Y
print(ridge_coef.reshape(-1)
# [1.998, 2.937]

ridge = Ridge(alpha=1.0)
ridge.fit(X, Y)
```

```
    print(ridge.coef_.reshape(-1)
    # [1.979, 2.973]
```

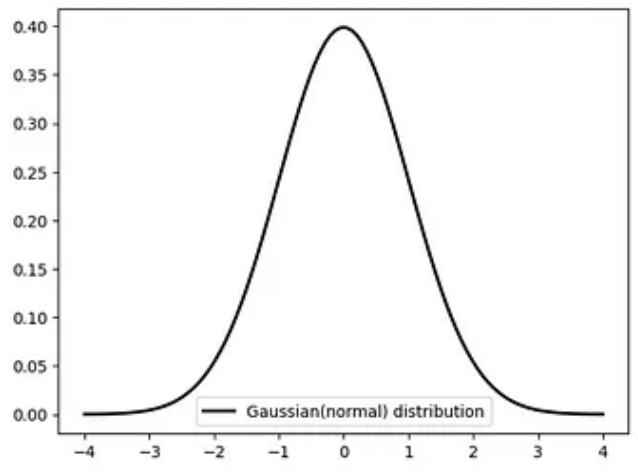
You can see we obtain almost the same results.

3.2 Probabilistic derivation of L2 regularization

We consider MAP estimation again. When we derive L1 regularization, we use the Laplace distribution as a prior. In the L2 regularization case, we utilize the Gaussian distribution with 0 mean as a prior.

Gaussian distribution:

$$\mathcal{N}(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



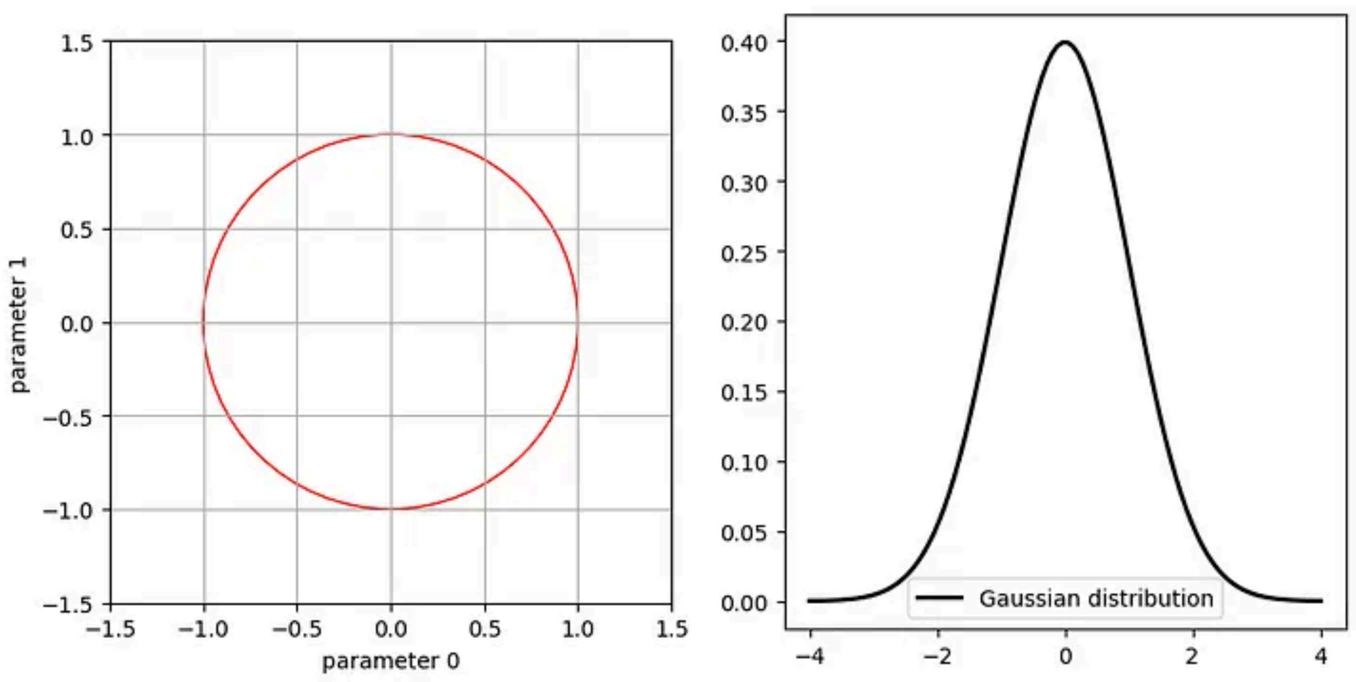
Gaussian distribution

You notice the exponent term of the exponential function is similar to the L2 regularization term. Now, we substitute the Gaussian prior with mean 0 for the prior probability in the MAP estimation.

$$\begin{aligned}
\hat{\boldsymbol{\beta}}_{\mathrm{MAP}} &= \arg\max_{\boldsymbol{\beta}} (\log P(\boldsymbol{\epsilon}|\boldsymbol{\beta}) + \log P(\boldsymbol{\beta})) \\
&= \arg\max_{\boldsymbol{\beta}} \left(\log \Pi_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{(y_i-(\beta_0 x_{i0}+\beta_1 x_{i1}+\cdots+\beta_p x_{ip}))^2}{2\sigma^2}} + \log \Pi_{j=0}^{p} \frac{1}{\sqrt{2\pi\sigma'^2}} \exp^{-\frac{\beta_j^2}{2\sigma'^2}} \right. \\
&= \arg\max_{\boldsymbol{\beta}} (-\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-(\beta_0 x_{i0}+\beta_1 x_{i1}+\cdots+\beta_p x_{ip}))^2 + \\
&\qquad -\frac{n}{2}\log(2\pi\sigma'^2) - \sum_{j=0}^{p}\frac{\beta_j^2}{2\sigma'^2}) \\
&= \arg\min_{\boldsymbol{\beta}} \frac{1}{2\sigma^2}(\sum_{i=1}^{n}(y_i-(\beta_0 x_{i0}+\beta_1 x_{i1}+\cdots+\beta_p x_{ip}))^2 + \frac{\sigma^2}{\sigma'^2}\sum_{j=0}^{p}\beta_j^2) \\
&= \arg\min_{\boldsymbol{\beta}} (\|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2)
\end{aligned}$$

MAP estimation using Gaussian prior

As you can see, the last formula is the same as the L2 regularization. Intuitively, the shape of the Gaussian distribution has a gentler curve than that of Laplace prior. So, it is also similar to the L2 regularization term.



(Left) L2 regularization term (right) Gaussian distribution

Finally, I will share you the code used in this blog.

In this blog, I introduced the detailed derivation of L1 and L2 regularization via analytical and probabilistic views. I hope this blog helps you to understand the mathematical background of regularizations. Thank you for reading.

References

[1] Underfitting vs.Overfitting, scikit-learn document

[2] Manfredi, V., Lecture 12: Regularization, Wesleyan university lecture

[3] https://stats.stackexchange.com/questions/17781/derivation-of-closed-form-lasso-solution

[4] Tibshirani, R., Coordinate Descent, Camegie Mellon University lecture

[5] Giba, L., Subgradient Descent Explained, Step by Step, MLC

[6] Kang, B., A probabilistic interpretation of regularization, Bounded Rationality

[7] https://math.dartmouth.edu/opencalc2/cole/lecture21.pdf