Machine Learning

| Ridge Regularization

Prof. Dr. Mostafa Elhosseini

https://youtube.com/@ProfElhosseiniTechVerse

Agenda

- Ridge Regularization
- L2 norm
- Measuring how complex a model is?
- Example
- Modifying the Error Function
- Regularization parameter λ
- Multiply all of them by $1 - \lambda$

Ridge Regularization

- L2 Regularization - Also called **Tikhonov** regularization
- Adds a penalty equivalent to the **square of the magnitude** of the coefficients $\sum_{j=1}^{n} \boldsymbol{\theta}_j^2$.
- Note that the regularization term should only be added to the cost function during training
- This discourages large coefficients but **does not** set them to zero.

Measuring how complex a model is? L2 norm

- L2 norm: The sum of the squares of the coefficients
- We use **squares** to **get rid** of the negative coefficients; otherwise, large negative numbers will cancel out with large positive numbers, and we may end up with a small value for a very complex model.
- The **bias in the models is not included** in the L2 norm. Why? Well, the bias in the model is precisely the number of seconds that we expect a user to watch movie 11 if they haven't watched any of the previous 10 movies. This number **is not associated with the complexity of the model**; therefore, we leave it alone

L2 norm

- Model 1: $\hat{y} = 2x_3 + 1.4x_7 - 0.5x_9 + 4$
- Model 2: $\hat{y} = 22x_1 - 103x_2 - 14x_3 + 109x_4 - 93x_5 + 203x_6 + 87x_7 - 55x_8 + 378x_9 - 25x_{10} + 8$

L2 norm:

- Model 1: $2^2 + 1.4^2 + (-0.5)^2 = 6.21$
- **Model 2**: $22^2 + (-103)^2 + (-14)^2 + 109^2 + (-93)^2 + 203^2 + 87^2 + (-55)^2 + 378^2 + (-25)^2 = 227{,}131$

Example

- Model 1: $\hat{y} = 2x + 3$
- Model 2: $\hat{y} = -x^2 + 6x - 2$
- Model3: $\hat{y} = x^9 + 4x^8 - 9x^7 + 3x^6 - 14x^5 - 2x^4 - 9x^3 + x^2 + 6x + 10$

Example

- Model 1: $\hat{y} = 2x + 3$
- Model 2: $\hat{y} = -x^2 + 6x - 2$
- Model3: $\hat{y} = x^9 + 4x^8 - 9x^7 + 3x^6 - 14x^5 - 2x^4 - 9x^3 + x^2 + 6x + 10$

L2 norm:

- Model 1: $2^2 = 2$
- Model 2: $(-1)^2 + 6^2 = 37$
- Model 3: $1^2 + 4^2 + (-9)^2 + 3^2 + (-14)^2 + (-2)^2 + (-9)^2 + 1^2 + 6^2 = 425$

Modifying the Error Function

- We have two measures for our model: a **measure of performance** (the error function) and a **measure of complexity** (the L2 norm).

- **Regression error:** A measure of the quality of the model. In this case, it can be the absolute or square errors.

- **Regularization term**: A measure of the complexity of the model. It can be the L2 norm of the model.
- Error = Regression error + Regularization term

Modifying the Error Function

- Error = Regression error + Regularization term
- we train the model using the **L2 norm**, it is called **ridge** regression.
- The **name ridge** comes from the **shape of the error function**, because adding the L2 norm term to the regression error function turns a sharp corner into a smooth valley when we plot it.

- The error function follows:

Ridge regression error = Regression error + L2 norm

Regularization parameter

- Trying to make the model perform better may make it more complex, whereas trying to reduce the complexity of the model may make it perform worse.
- Fortunately, most machine learning techniques come with **knobs** (**hyperparameters**) for the data scientist to turn and build the best possible models, and regularization is not an exception.
- This hyperparameter is called the **regularization parameter**, and its goal is to determine if the model-training process should emphasize **performance or simplicity**

Regularization parameter

- Error = Regression error + λ Regularization term
- Picking a value of 0 for λ cancels out the regularization term
- Picking a large value for λ results in a simple model, perhaps of low degree, which may not fit our dataset very well
- It is crucial to pick a good value for λ, and for this, validation is a useful technique. It is typical to choose powers of 10, such as 10, 1, 0.1, 0.01, but this choice is somewhat arbitrary

Effects of L2

- Discourages large coefficients but does not typically reduce them to zero.
- Produces a model where all features contribute, albeit with smaller, more regulated impact, enhancing model stability and reducing overfitting.

Main Goal

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(X^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

- Minimize the cost function and restrict the parameters not to become too large – thanks to the term of **regularization**
- λ is a constant the control the weight and importance of regularization term
  - Control the trade off between the two goals
  - Fit the training model very well
  - Keep parameters small
- n is the numbers of features

Main Goal

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(X^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

- By convention you do not penalize θ_0- minimization is from θ_1 onwards
- Minimizing the cost function consists of reducing MSE term and regularization term
- When **parameter** is updated to minimize MSE, and if it is becoming **large**, it will **increase** the value of **cost function** by regularization term, and as a result it will be **penalized** and **updated** to a **small value**

Regularization

- If λ is very large we end up penalizing all the parameters, so all the parameters end up being close to zero "Except θ_0"
  - Like we get rid of all the terms – underfitting
  - Too biased $\boldsymbol{h(\theta) = \theta_0}$
- λ should be chosen carefully – not too big

Regularization

- $J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_\theta(X^{(i)}) - y^{(i)}\right)^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$
- θ_0 is not regularized
  - $\theta_0 = \theta_0 - \frac{\alpha}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)$
- $\theta_j = \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j{}^{(i)} + \frac{\lambda}{m}\theta_j\right] \quad j = 1,2,3,\dots,n$
- $\theta_j = \theta_j\left(1 - \alpha\frac{\lambda}{m}\right) - \frac{\alpha}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j{}^{(i)} \quad j = 1,2,3,\dots,n$

Regularization

- $\theta_j = \boldsymbol{\theta_j}\left(\mathbf{1} - \boldsymbol{\alpha}\frac{\boldsymbol{\lambda}}{\mathbf{m}}\right) - \frac{\boldsymbol{\alpha}}{\mathbf{m}}\sum_{\mathbf{i=1}}^{\mathbf{m}}\left(\mathbf{h_\theta}(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)}\right)\mathbf{x_j}^{(i)} \quad j = 1,2,3,\ldots,n$
- Regularization Term: $\boldsymbol{\theta_j}\left(\mathbf{1} - \boldsymbol{\alpha}\frac{\boldsymbol{\lambda}}{\mathbf{m}}\right)$
  - Usually learning rate is small and m is large
  - Less than 1
  - This term is often around 0.99 to 0.95
  - Reduces the magnitude of θ_j to prevent overfitting
- **The term** on right is the same as before "Gradient descent update"

Linear Regression Algorithm

- To simplify things, let's say that we are in the middle of our training, and we want to **make the model simpler**. We can do this by reducing the coefficients. For simplicity, let's say that our model has three coefficients: **3, 10, and 18**.

- Can we take a small step to decrease these three by a small amount?

Linear Regression Algorithm

- Multiply all of them by $1 - \lambda$. Notice that this number is close to 1, because λ is small.
- we get the numbers 2.97, 9.9, and 17.82.
- Now let's see what would happen, again 200 times and with the same learning rate of $\lambda = 0.01$. We get the following sequence of values:
- $2 \rightarrow 1.98 \rightarrow 1.9602 \rightarrow \ldots \rightarrow 0.2734 \rightarrow 0.2707 \rightarrow 0.2680$
- Notice that the coefficient decreased dramatically, but it didn't become zero

Ridge Regression

- The hyperparameter α controls how much you want to regularize the model.

$$J(\theta) = MSE(\theta) + \frac{\boldsymbol{\alpha}}{2m}\sum_{i=1}^{m}\theta_i^2$$

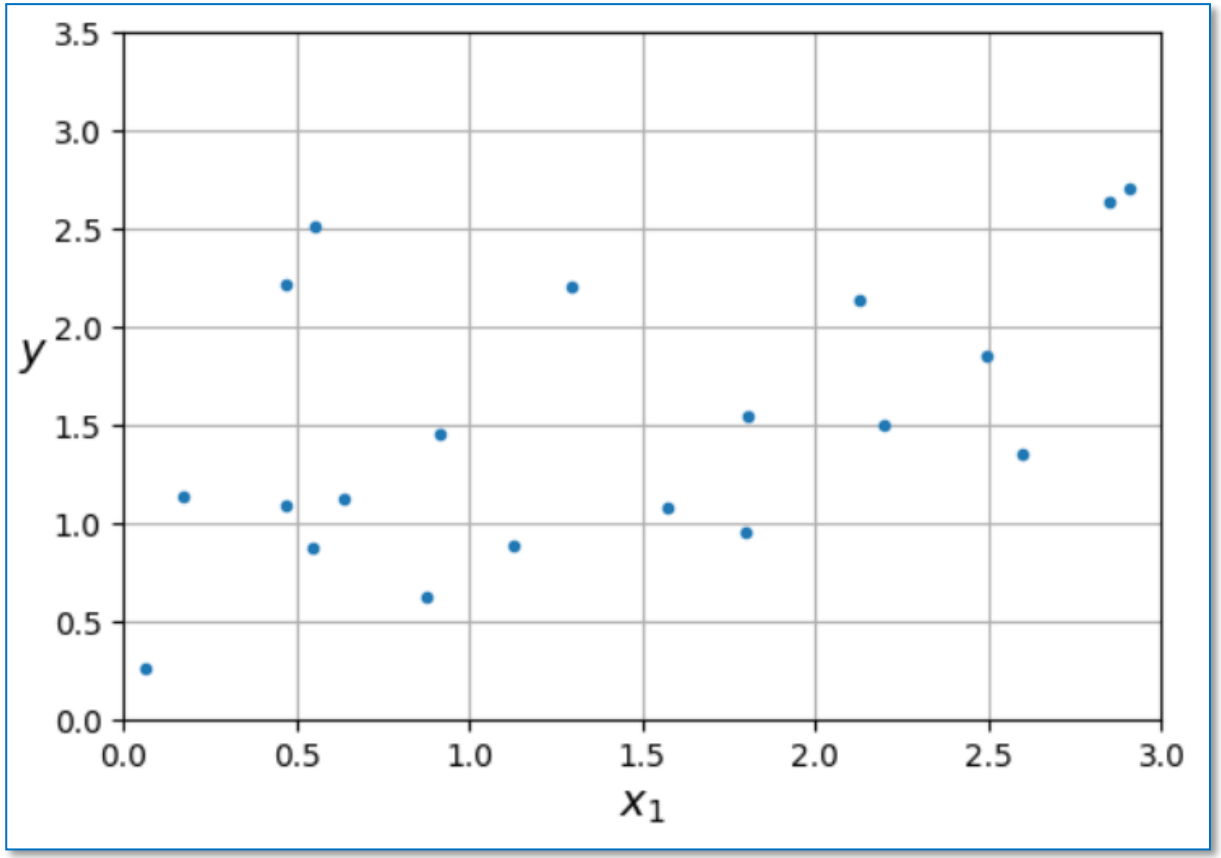
- If α = 0, then ridge regression is just linear regression.
- If α is very large, then all weights end up very close to zero and the result is a flat line going through the data's mean

Ridge Regression

- It is important to scale the data (e.g., using a StandardScaler) before performing Ridge Regression, as it is sensitive to the scale of the input features. This is true of most regularized models.

Ridge Regression - Python

```
np.random.seed(42)
m = 20
X = 3 * np.random.rand(m, 1)
y = 1 + 0.5 * X + np.random.randn(m, 1) / 1.5
X_new = np.linspace(0, 3, 100).reshape(100, 1)
```

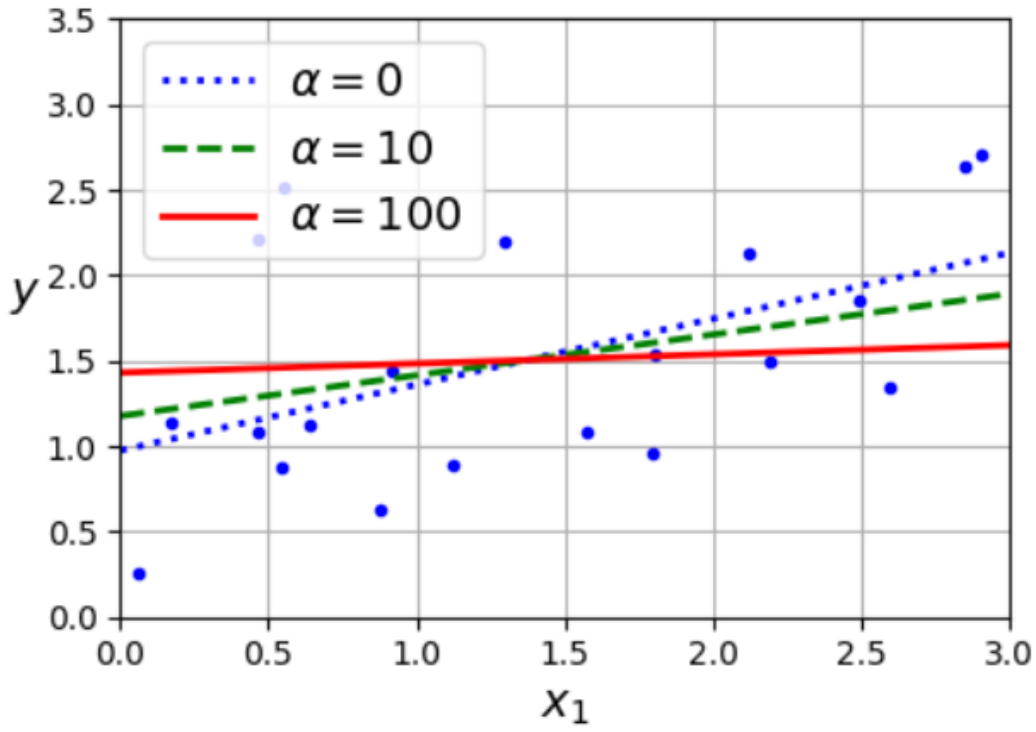


Ridge Regression

- As with Linear Regression, we can perform Ridge Regression either by computing a **closed-form** equation or by performing **Gradient Descent**.
- The pros and cons are the same

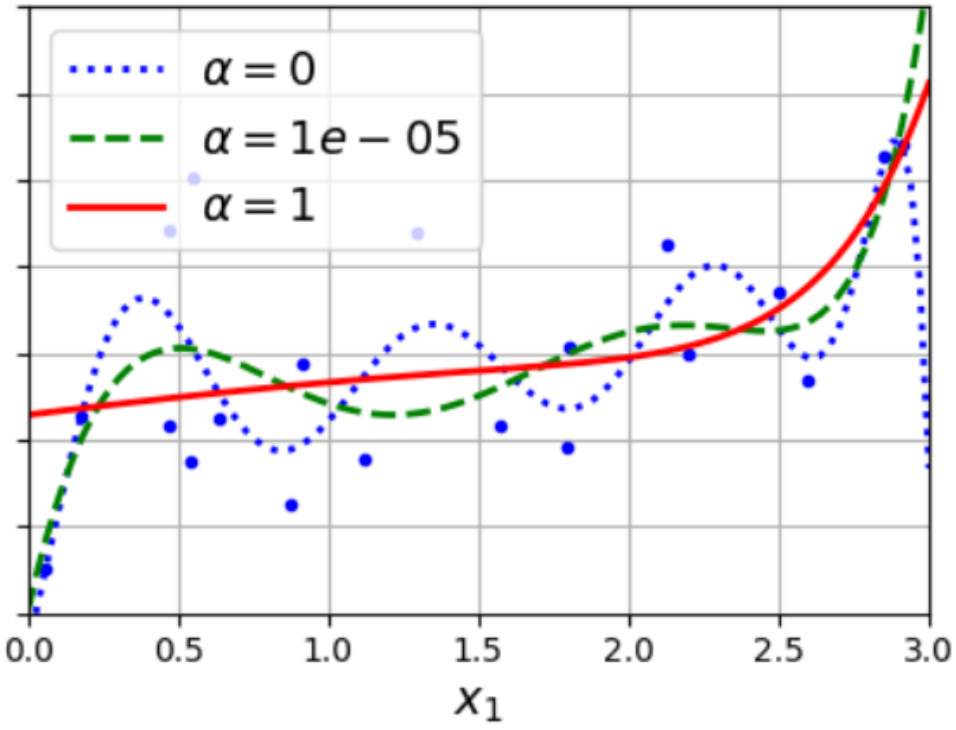
```
from sklearn.linear_model import Ridge

ridge_reg = Ridge(alpha=0.1, solver="cholesky")
ridge_reg.fit(X, y)
ridge_reg.predict([[1.5]])
```

```
array([[1.55325833]])
```



Ridge Regression

- The data is first expanded using PolynomialFeatures(degree=10), then it is scaled using a StandardScaler, and finally the ridge models are applied to the resulting features.
- Note how increasing α leads to flatter (i.e., less extreme, more reasonable) predictions, thus **reducing the model's variance** but increasing its bias



Ridge Regression

- Using stochastic gradient descent

```
sgd_reg = SGDRegressor(penalty="l2", alpha=0.1 / m, tol=None,
                       max_iter=1000, eta0=0.01, random_state=42)
sgd_reg.fit(X, y.ravel())  # y.ravel() because fit() expects 1D targets
sgd_reg.predict([[1.5]])
```

```
array([1.55302613])
```

- There's no division by m in this case; that's why we passed $\alpha = 0.1 \ / \ m$, to get the same result as Ridge($\alpha = 0.1$).

Lasso Regression

- Least Absolute Shrinkage and Selection Operator Regression (simply called Lasso Regression)
- like Ridge Regression, it adds a regularization term to the cost function, but it uses the **ℓ1 norm** of the weight vector instead of half the square of the ℓ2 norm
- $\boldsymbol{J(\theta) = MSE(\theta) + \alpha \sum_{i=1}^{n} |\theta_i|}$
- Next Lecture