Machine Learning

|Lasso & Elastic Net Regularization

Prof. Dr. Mostafa Elhosseini

https://youtube.com/@ProfElhosseiniTechVerse

Agenda

- Lasso Regularization
- Measuring how complex a model is? - L1 norm
- Example
- Modifying the Error Function
- Regularization parameter λ
- Linear Regression Algorithm
- Elastic Net
- Use cases

Lasso Regularization

- L1 Regularization
- Adds a penalty equivalent to the absolute value of the magnitude of the coefficients.
- Encourages sparsity, which can set some coefficients to zero.

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

- $\boldsymbol{J(\theta) = MSE(\theta) + \alpha \sum_{i=1}^{n} |\theta_i|}$

Measuring how complex a model is? L1 norm

- L1 norm: The sum of the absolute values of the coefficients
- We use **absolute values** 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 L1 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

L1 norm

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

L1 norm:

- Model 1: |2| + |1.4| + |–0.5| = 3.9
- **Model 2**: |22| + |–103| + |–14| + |109| + |–93| + |203| + |87| + |–55| + |378| + |–25| = 1,089

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

L1 norm:

- Model 1: |2| = 2
- Model 2: |–1| + |6| = 7
- Model 3: |1| + |4| + |–9| + |3| + |–14| + |–2| + |–9| + |1| + |6| = 49

Modifying the Error Function

- We have two measures for our model: a **measure of performance** (the error function) and a **measure of complexity** (the L1 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 L1 norm of the model.
- Error = Regression error + Regularization term

Modifying the Error Function

- Error = Regression error + Regularization term
- If we train our regression model using the **L1 norm**, the model is called **lasso** regression.
- Lasso stands for "**l**east **a**bsolute **s**hrinkage and **s**election **o**perator." The error function follows:

Lasso regression error = Regression error + L1 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 L1

L1 Regularization (Lasso Regression)

- Encourages sparsity in the model by turning some coefficients to zero.
- Ends up with fewer non-zero coefficients, simplifying the model and potentially improving interpretability.

Linear Regression Algorithm

Inputs: A dataset of points

Outputs: A linear regression model that fits that dataset

- Procedure:
  - Pick a model with random weights and a random bias.
  - Repeat many times:
    - Pick a random data point.
    - Slightly adjust the weights and bias to improve the prediction for that data point.

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

- Subtract λ from each of the positive parameters and add λ to each of the negative parameters. If they are zero, leave them alone.
- we get the numbers 2.99, 9.99, and 17.99.

- we are training the model with **L1 regularization**, or **lasso regression**
- Let's say that our coefficient is 2, **2** → 1.99 → 1.98 → … → 0.02 → 0.01 → 0

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
- $\boldsymbol{J(\theta) = MSE(\theta) + \alpha \sum_{i=1}^{n} |\theta_i|}$

Lasso Regression

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

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

- Gradient of the MSE Term: $\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(X^{(i)}) - y^{(i)}\right)x_j^{(i)}$
- Gradient of the Regularization Term: $\frac{\lambda}{m}\operatorname{sgn}(\theta_j)$

$$\operatorname{sgn}(\theta_j) = \begin{cases} 1 & \text{if } \theta_j > 0 \\ -1 & \text{if } \theta_j < 0 \\ 0 & \text{if } \theta_j = 0 \end{cases}$$

Lasso Regression

- Gradient of the MSE Term: $\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(X^{(i)}) - y^{(i)}\right)x_j^{(i)}$
- Gradient of the Regularization Term: $\frac{\lambda}{m}\operatorname{sgn}(\theta_j)$
- Gradient Descent Update Rule: $\theta_j = \theta_j - \alpha\frac{\partial J(\theta)}{\partial \theta_j}$
- $\theta_j = \theta_j - \alpha\left(\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(X^{(i)}) - y^{(i)}\right)x_j^{(i)} + \frac{\lambda}{m}\operatorname{sgn}(\theta_j)\right)$
- $\theta_j = \theta_j - \alpha\frac{\lambda}{m}\operatorname{sgn}(\theta_j) - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(X^{(i)}) - y^{(i)}\right)x_j^{(i)}$

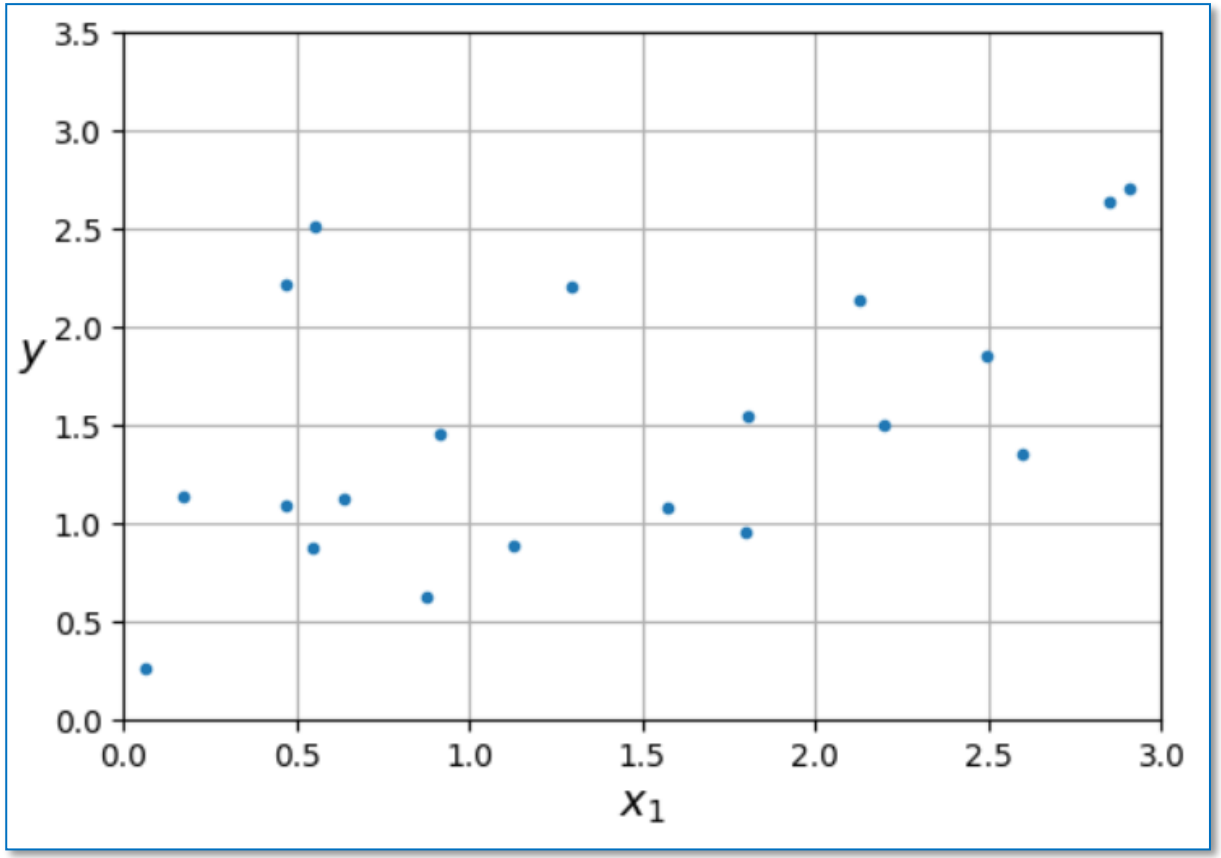
Lasso Regression

$$\theta_j = \theta_j - \alpha \frac{\boldsymbol{\lambda}}{\boldsymbol{m}} \operatorname{sgn}(\boldsymbol{\theta_j}) - \alpha \frac{\mathbf{1}}{\boldsymbol{m}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{m}} \left(\boldsymbol{h_\theta}(\boldsymbol{X^{(i)}}) - \boldsymbol{y^{(i)}} \right) \boldsymbol{x_j^{(i)}}$$

- Decrease the parameter θ_j by subtracting a constant value $\alpha \frac{\boldsymbol{\lambda}}{\boldsymbol{m}} \operatorname{sgn}(\boldsymbol{\theta_j})$, rather than multiplying by a factor as in Ridge regression.
- This subtraction can reduce some coefficients to zero, promoting sparsity in the model.

Lasso Regression - Python

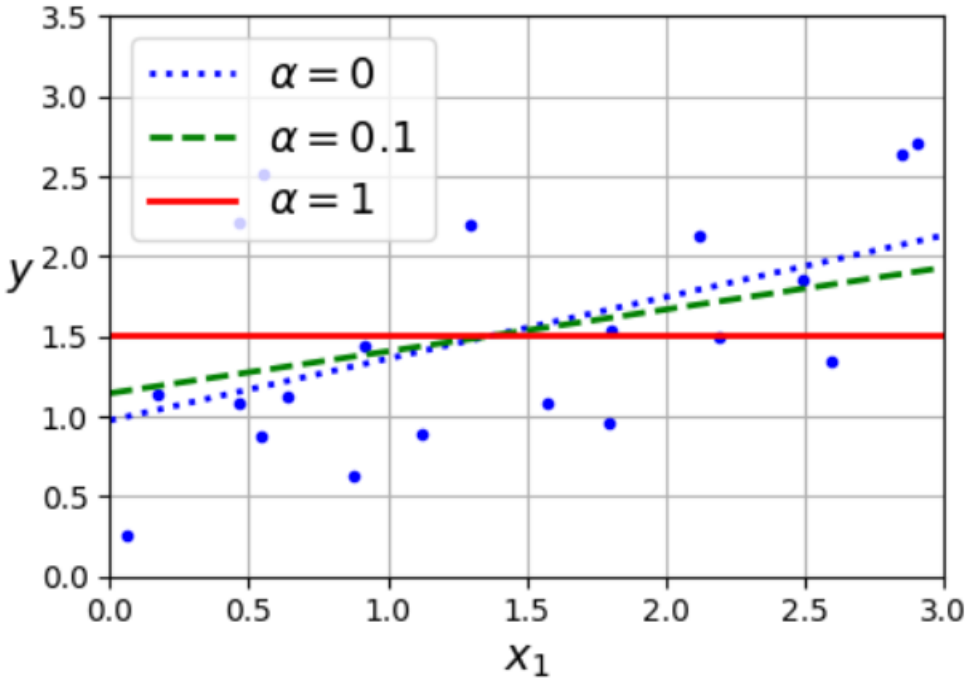
```
np.random.seed(42)
m = 20
X = 3 * np.random.rand(m, 1)
y = 1 + 0.5 * X + np.random.randn(m, 1) / 1.5
X_new = np.linspace(0, 3, 100).reshape(100, 1)
```



Lasso Regression

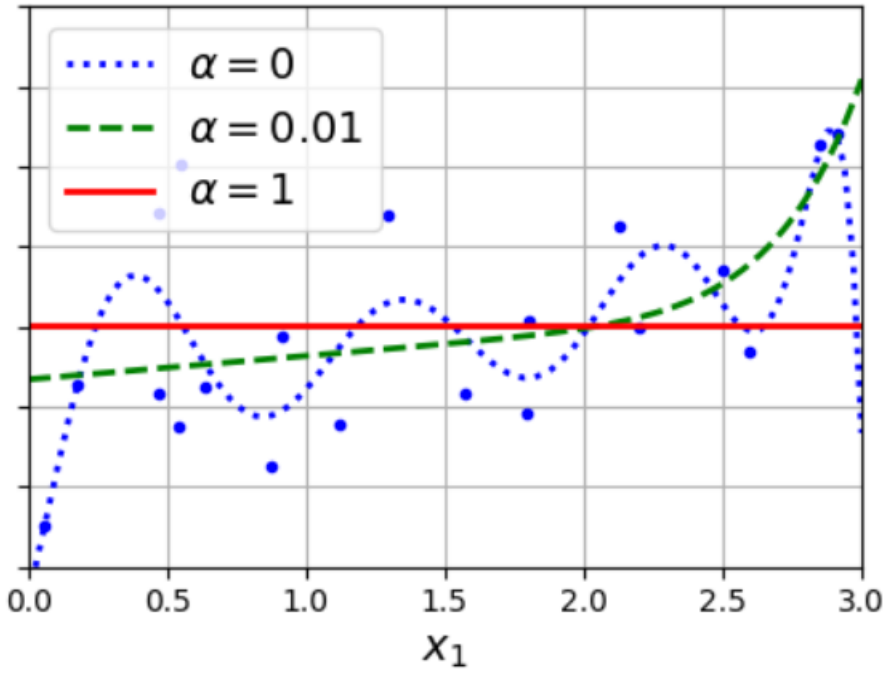
```
from sklearn.linear_model import Lasso

lasso_reg = Lasso(alpha=0.1)
lasso_reg.fit(X, y)
lasso_reg.predict([[1.5]])
```

```
array([1.53788174])
```



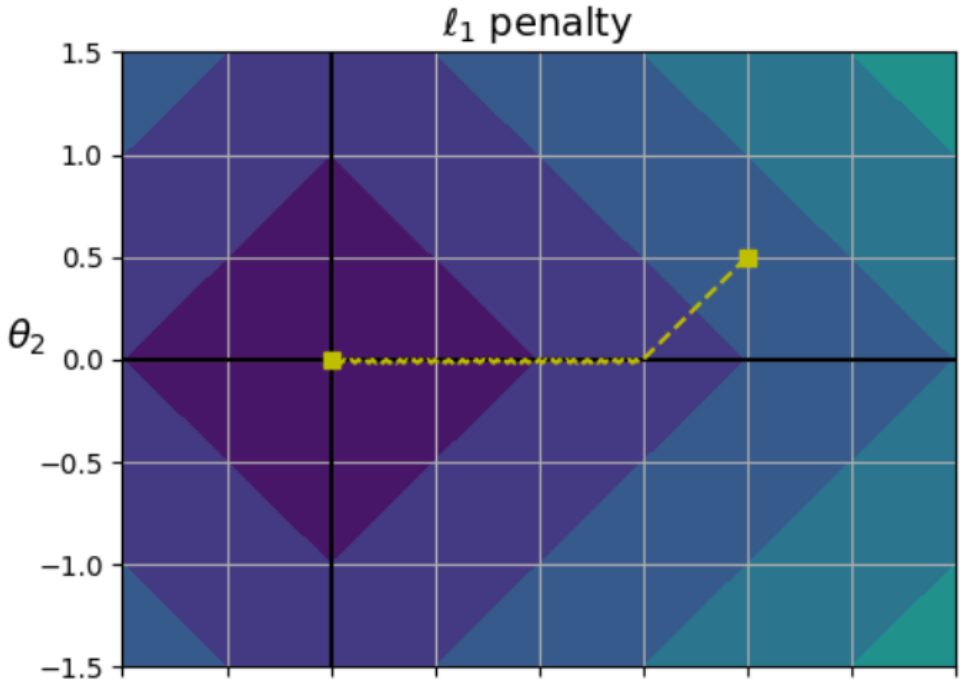
Ridge Regression

- The dashed line (with α = 0.01) looks roughly cubic: all the weights for the high-degree polynomial features are equal to zero
- lasso regression automatically performs feature selection and outputs a sparse model with few nonzero feature weights



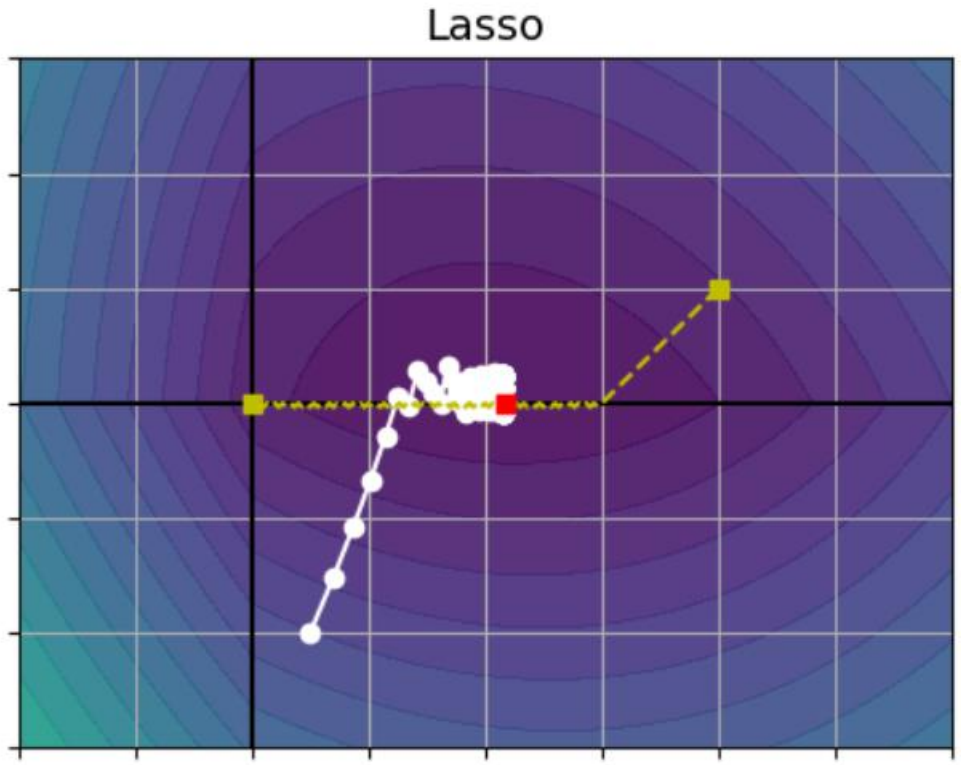
Lasso Regression

- The contours represent the ℓ_1 **loss** ($|\theta_1| + |\theta_2|$), which drops linearly as you get closer to any axis
- initialize the model parameters to $\theta_1 = 2$ and $\theta_2 = 0.5$
- Running gradient descent will decrement both parameters equally; therefore θ_2 will reach 0 first



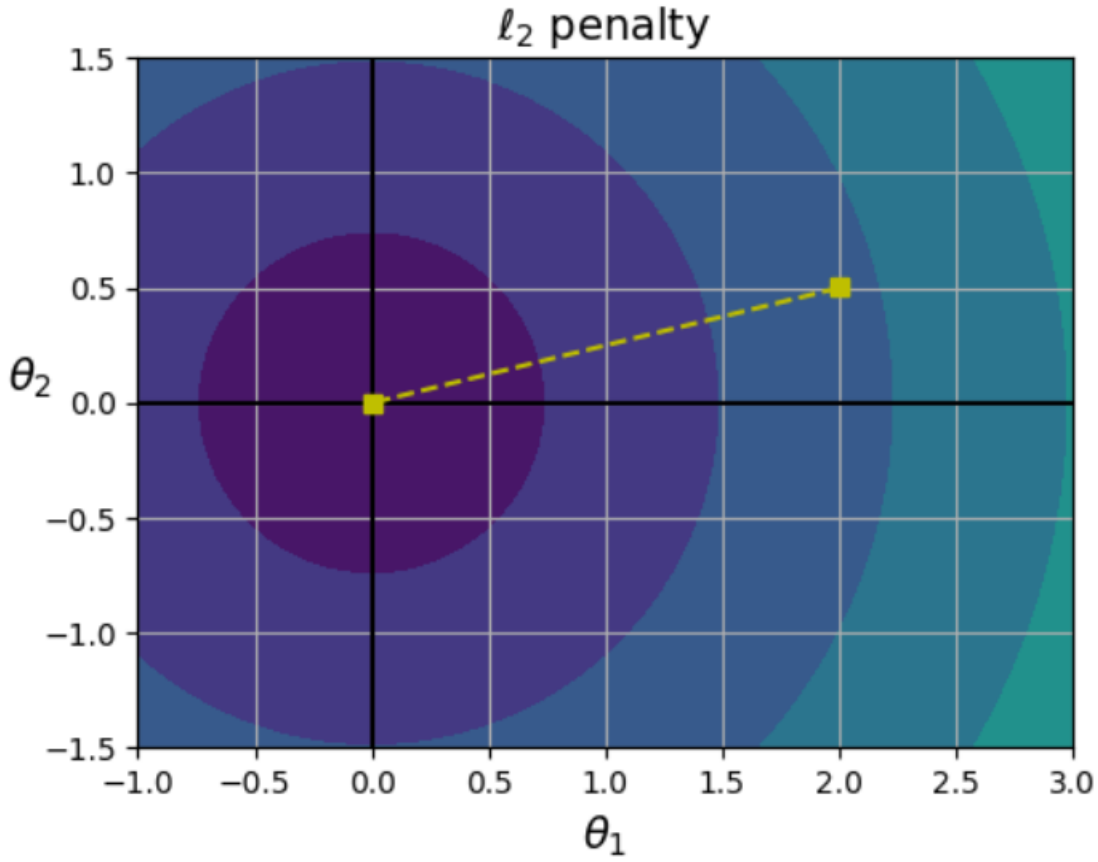
Lasso Regression

- Contours represent lasso regression's cost function (i.e., an MSE cost function plus an ℓ_1 $\boldsymbol{loss}$).
- The small white circles show the path that gradient descent takes to optimize some model parameters that were initialized around $\theta_1 = 0.25$ and $\theta_2 = -1$
- Notice once again how the path quickly reaches $\theta_2 = 0$



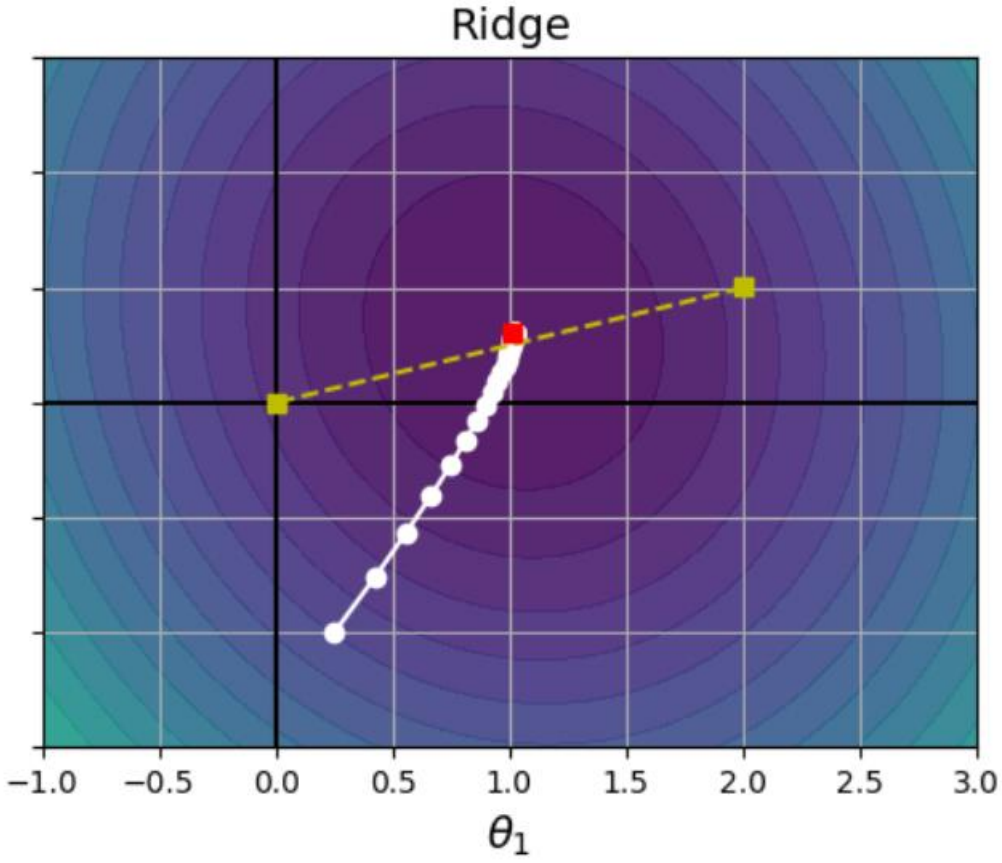
Ridge Regression

- ℓ_2 loss decreases as we get closer to the origin, so gradient descent just takes a straight path toward that point



Ridge Regression

- Contours represent ridge regression's cost function (i.e., an MSE cost function plus an ℓ_2 loss)
- **Gradients get smaller** as the parameters approach the global optimum, so gradient descent naturally slows down.
- This **limits the bouncing around**, which helps **ridge converge faster** than lasso regression



Elastic Net

- Elastic Net is a middle ground between Ridge Regression and Lasso Regression.
- The regularization term is a simple mix of both Ridge and Lasso's regularization terms, and you can control the mix ratio r.
- When $r = 0$, Elastic Net is equivalent to Ridge Regression, and when $r = 1$, it is equivalent to Lasso Regression
- $\boldsymbol{J(\theta) = MSE(\theta) + r\alpha \sum_{i=1}^{n} |\theta_i| + \frac{1-r}{2} \alpha \sum_{j=1}^{n} \theta_i^2}$

Use cases

when should you use Linear Regression, Ridge, Lasso, or Elastic Net?

- It is almost always preferable to have at least a little bit of regularization, so generally you should avoid plain Linear Regression.
- Ridge is a good default, but if you suspect that only a few features are actually useful, you should prefer Lasso or Elastic Net since they tend to reduce the useless features' weights down to zero as we have discussed.

Use cases

- In general, Elastic Net is preferred over Lasso since Lasso may behave erratically when the number of features is greater than the number of training instances or when several features are strongly correlated.

```
from sklearn.linear_model import ElasticNet

elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5)
elastic_net.fit(X, y)
elastic_net.predict([[1.5]])
```

```
array([1.54333232])
```

Early stopping

- **Regularization** and **early stopping** are both techniques used to prevent overfitting in machine learning models, but they approach the problem in fundamentally different ways

- Regularization directly modifies the learning algorithm to reduce the complexity of the model. It does this by adding a penalty term to the loss function, which the training process aims to minimize

Early stopping

- Early stopping, on the other hand, **doesn’t alter** the model itself or its loss function. Instead, it **monitors the model's performance** on a **validation dataset** during the training process.

- **Training is halted** when the model’s performance on the validation dataset starts deteriorating, despite potential improvements on the training dataset