Chapter 4 Summary: Training Models

Chapter 4 moves beyond treating Machine Learning models as black boxes and explains how training algorithms actually work. Understanding these internal mechanisms helps practitioners choose appropriate models, tune hyperparameters effectively, debug issues, and prepare for more advanced topics such as neural networks. The chapter focuses on Linear Regression and its training methods, Gradient Descent variants, Polynomial Regression, model complexity, regularization techniques, and concludes with Logistic and Softmax Regression for classification.

Linear Regression

Linear Regression predicts a target value by computing a weighted sum of input features plus a bias term. The model is defined by a parameter vector θ, and training consists of finding the parameter values that minimize a cost function. For regression tasks, the **Mean Squared Error (MSE)** is typically used because it is mathematically convenient and leads to the same optimum as RMSE.

The Normal Equation

One way to train Linear Regression is through a **closed-form solution** known as the Normal Equation. This approach computes the optimal parameters directly using matrix operations, without iteration. Scikit-Learn commonly relies on the **SVD/pseudoinverse** approach for numerical stability, especially when the matrix in the Normal Equation is not invertible.

Computational Complexity

Closed-form solutions scale poorly with the number of features due to matrix inversion costs, while making predictions after training remains fast and scales linearly with both instances and features. This motivates iterative approaches such as Gradient Descent for large-scale problems.

Gradient Descent

Gradient Descent (GD) is an iterative optimization algorithm that updates parameters step by step to minimize a cost function. The learning rate determines step size: too small leads to slow convergence, while too large can cause divergence. For Linear Regression, the MSE cost function is convex, ensuring convergence to a global minimum if the learning rate is appropriate. Feature scaling is important because unscaled features can create elongated cost surfaces that slow convergence.

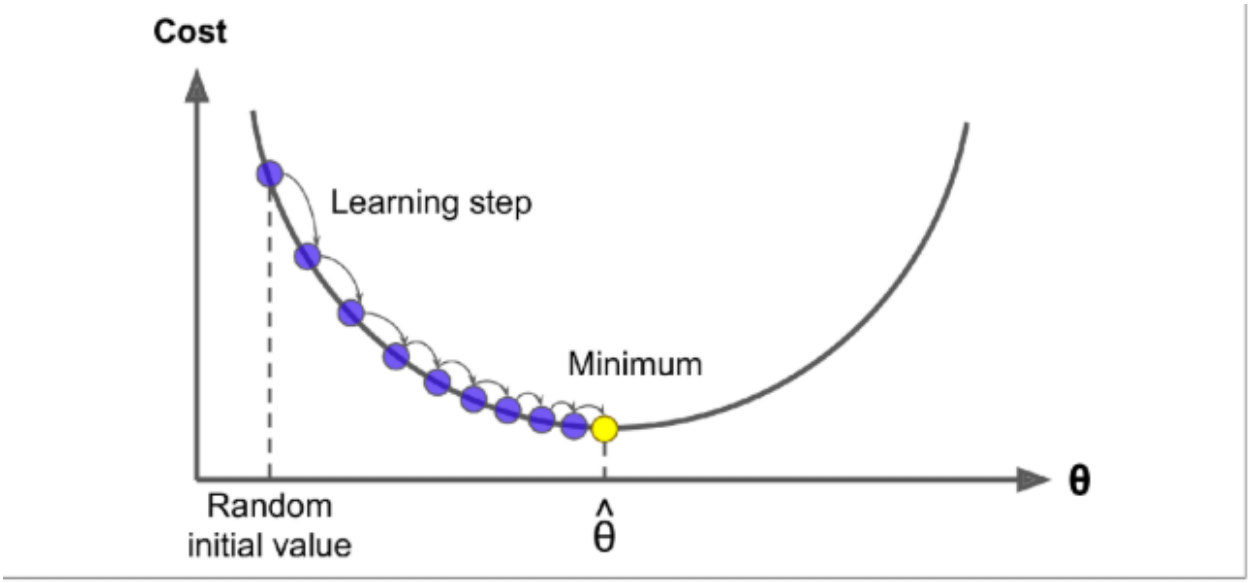

Figure 4-3. In this depiction of Gradient Descent, the model parameters are initialized randomly and get tweaked repeatedly to minimize the cost function

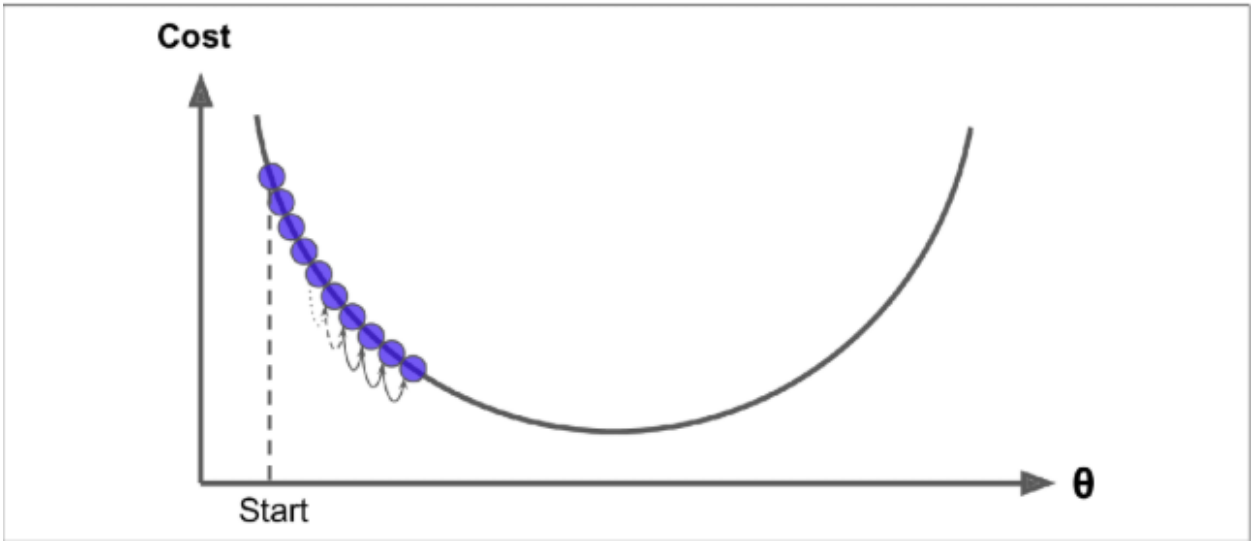

Figure 4-4. The learning rate is too small

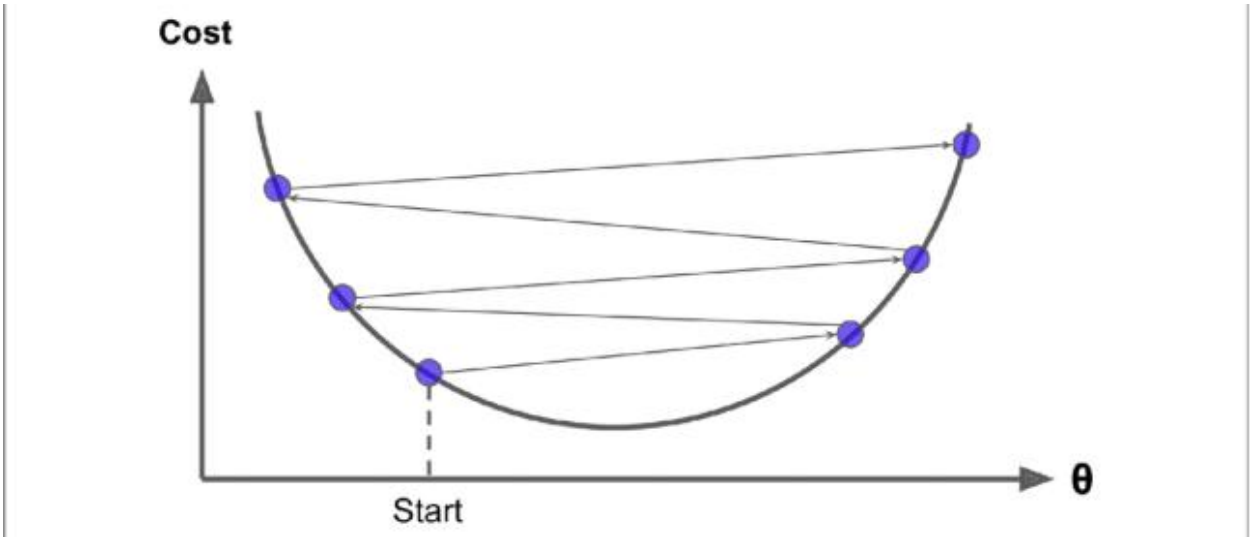

Figure 4-5. The learning rate is too large

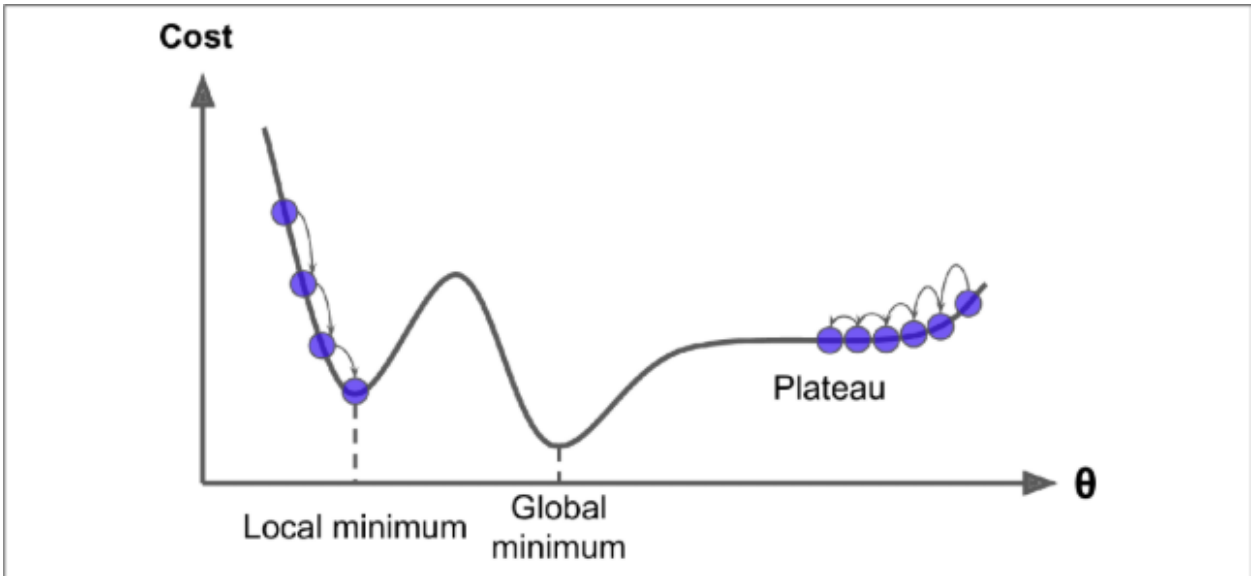

Figure 4-6. Gradient Descent pitfalls

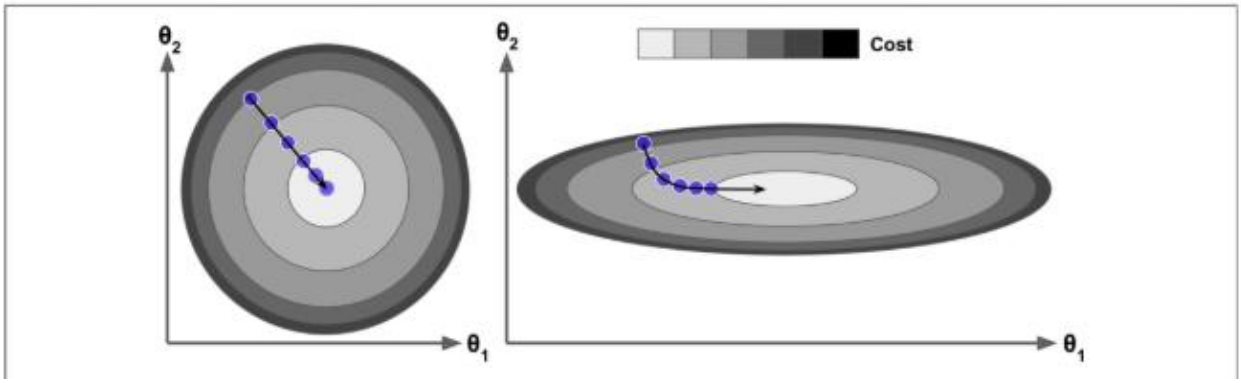

Figure 4-7. Gradient Descent with (left) and without (right) feature scaling

---

Batch Gradient Descent

Batch Gradient Descent computes gradients using the entire training set for each update. This produces stable convergence but becomes slow on large datasets. It is reliable for convex problems, but training time grows with dataset size.

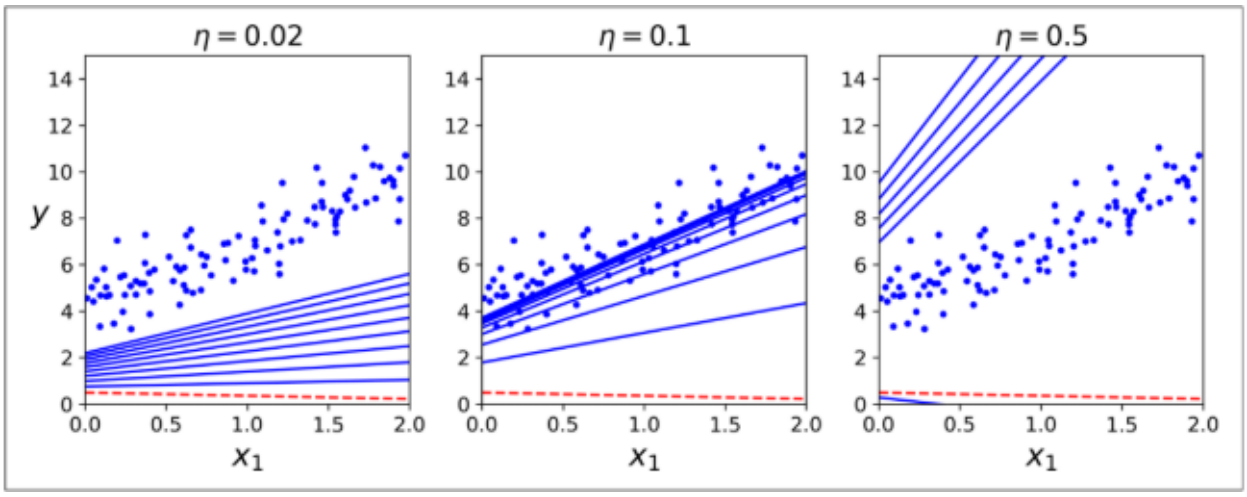

Figure 4-8. Gradient Descent with various learning rates

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) updates parameters using one randomly chosen instance at a time, making each step fast and suitable for large datasets or out-of-core learning. However, updates are noisy and the algorithm fluctuates around the minimum. A learning schedule that gradually reduces the learning rate helps the model settle closer to the optimum.

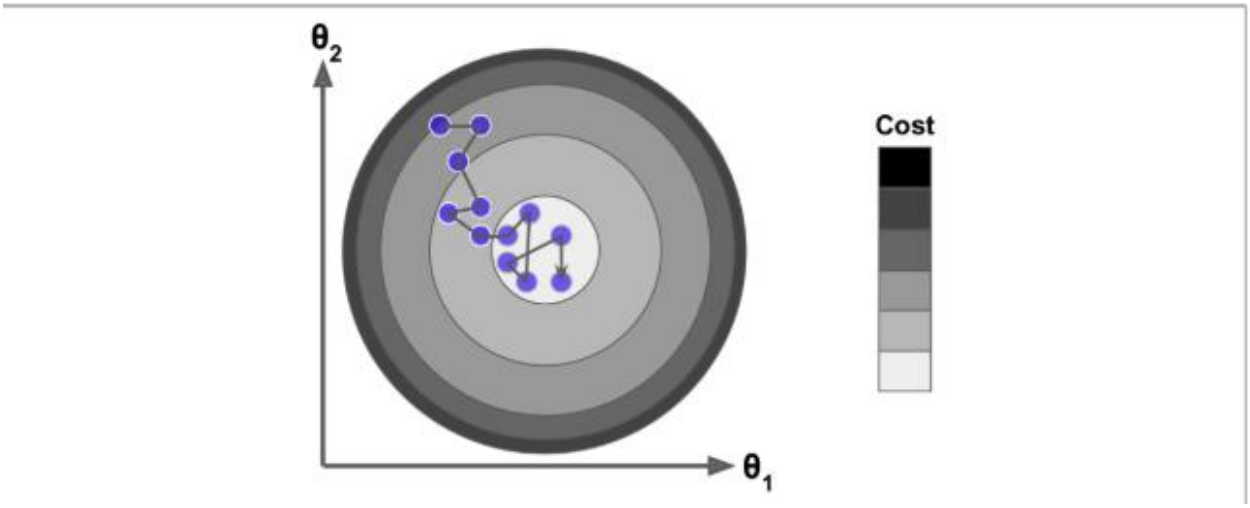

Figure 4-9. With Stochastic Gradient Descent, each training step is much faster but also much more stochastic than when using Batch Gradient Descent

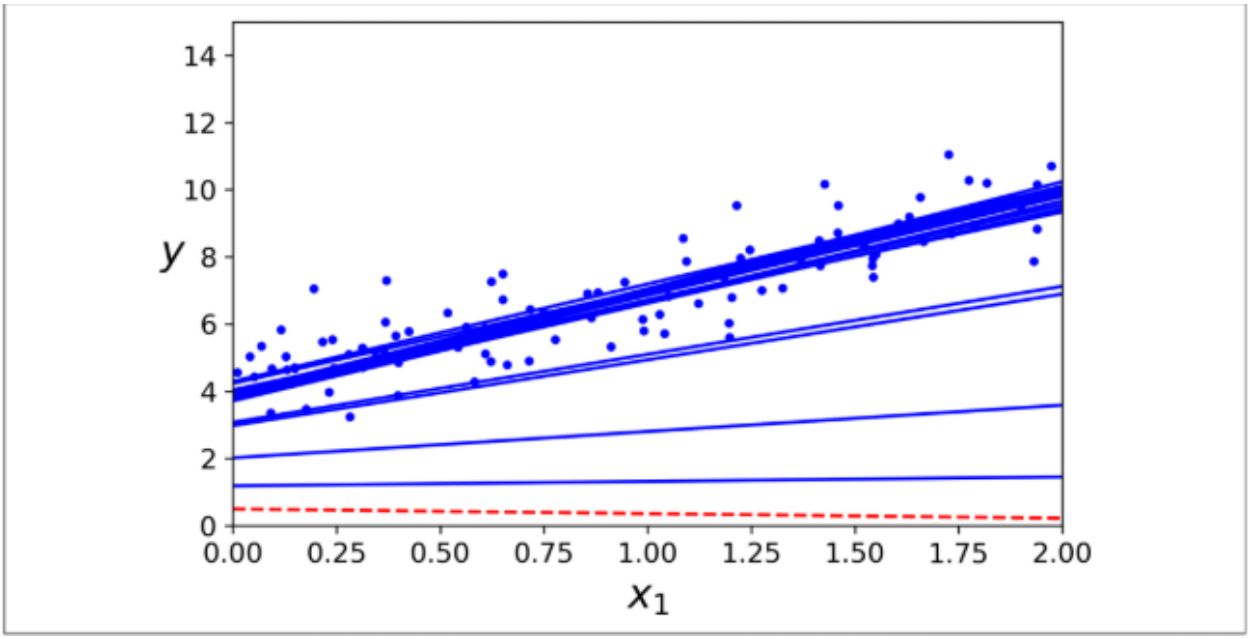

Figure 4-10. The first 20 steps of Stochastic Gradient Descent

---

Mini-batch Gradient Descent

Mini-batch Gradient Descent uses small batches of data per step. It reduces the noise of SGD while remaining significantly faster than Batch GD. It also benefits from hardware acceleration via vectorized operations and GPUs.

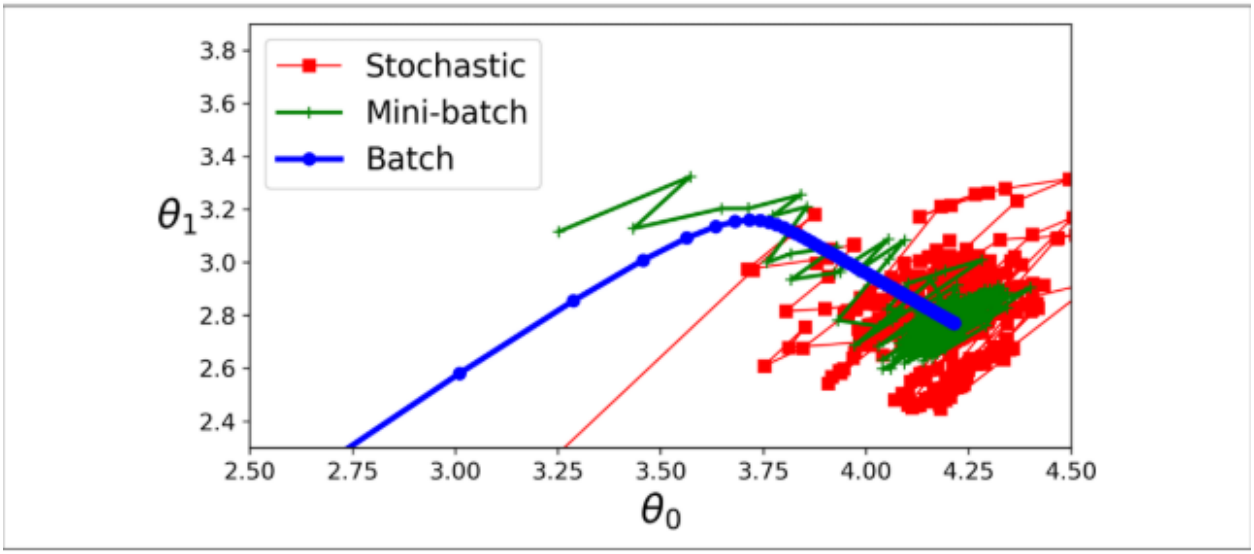

Figure 4-11. Gradient Descent paths in parameter space

Algorithm	Large *m*	Out-of-core support	Large *n*	Hyperparams	Scaling required	Scikit-Learn
Normal Equation	Fast	No	Slow	0	No	N/A
SVD	Fast	No	Slow	0	No	`LinearRegression`
Batch GD	Slow	No	Fast	2	Yes	`SGDRegressor`
Stochastic GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`
Mini-batch GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`

Table 4-1. Comparison of algorithms for Linear Regression

Polynomial Regression

Polynomial Regression enables linear models to fit nonlinear data by adding polynomial features. Increasing polynomial degree improves flexibility but also increases risk of overfitting, and may cause feature explosion when multiple inputs exist.

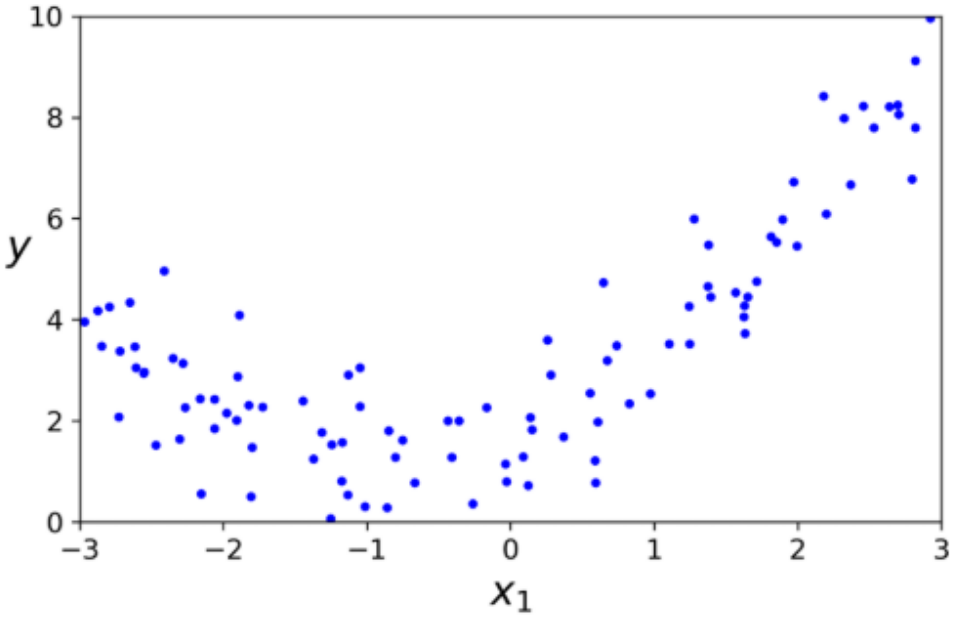

Figure 4-12. Generated nonlinear and noisy dataset

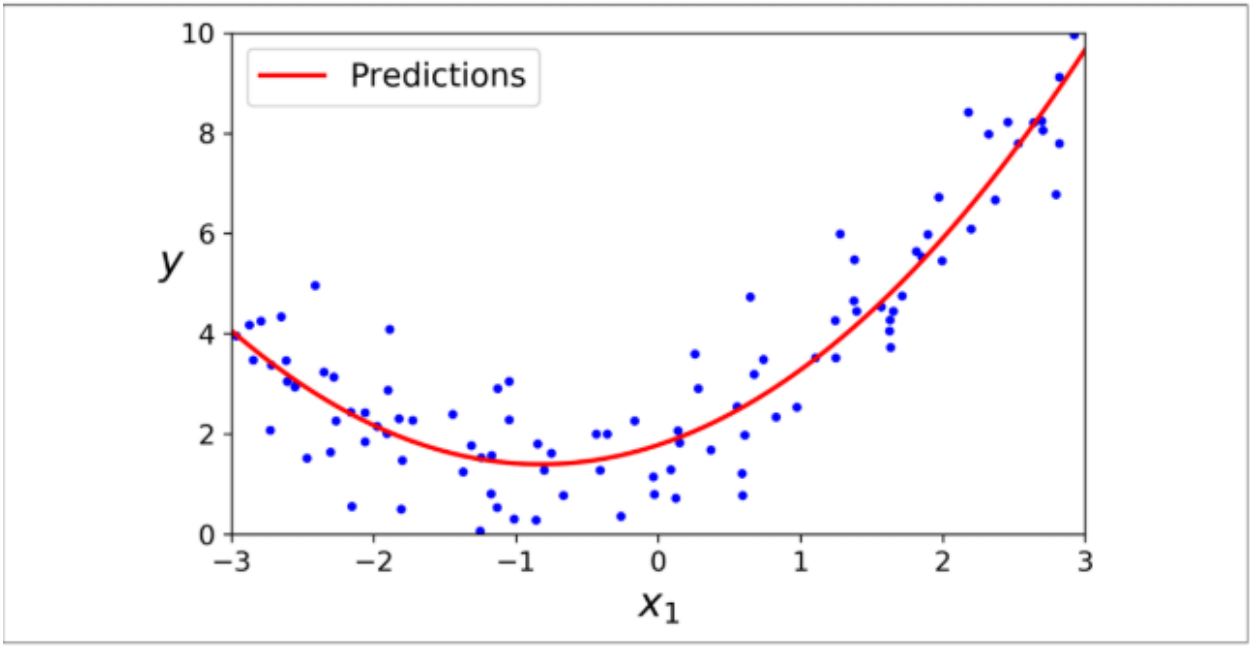

Figure 4-13. Polynomial Regression model predictions

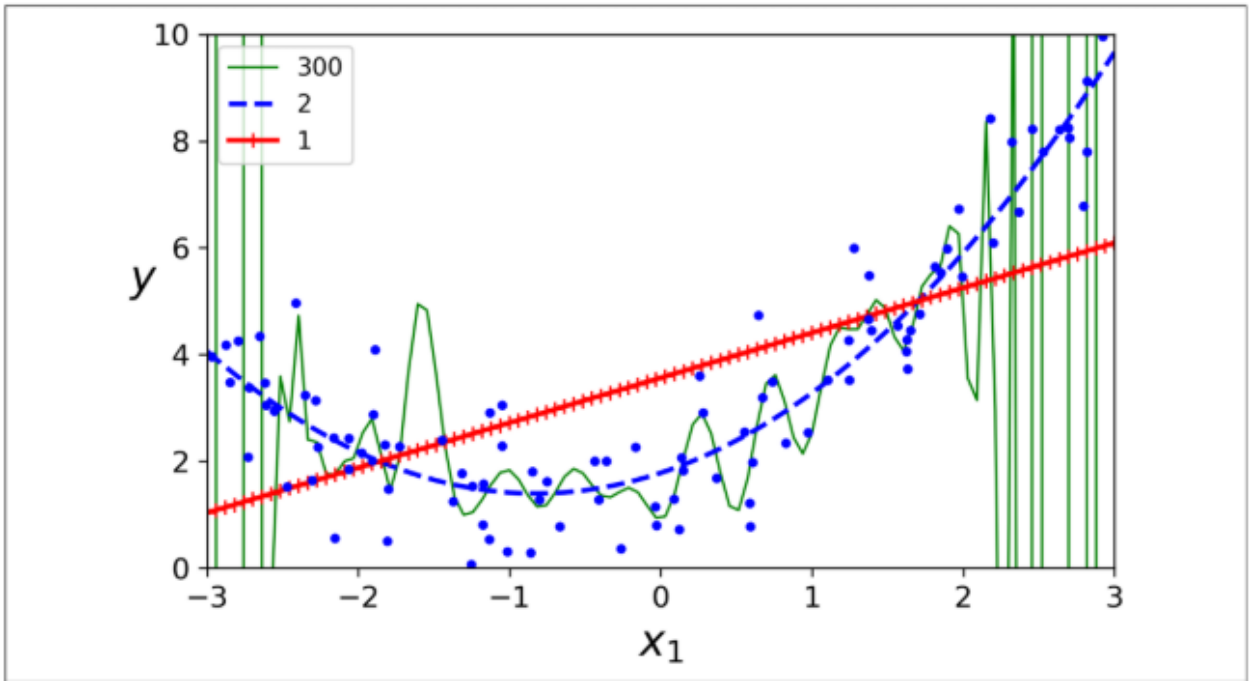

Figure 4-14. High-degree Polynomial Regression

Learning Curves

Learning curves evaluate training and validation performance as training size increases. Underfitting appears when both curves plateau at high error. Overfitting appears when training error is low but validation error remains significantly higher, though additional data can reduce the gap.

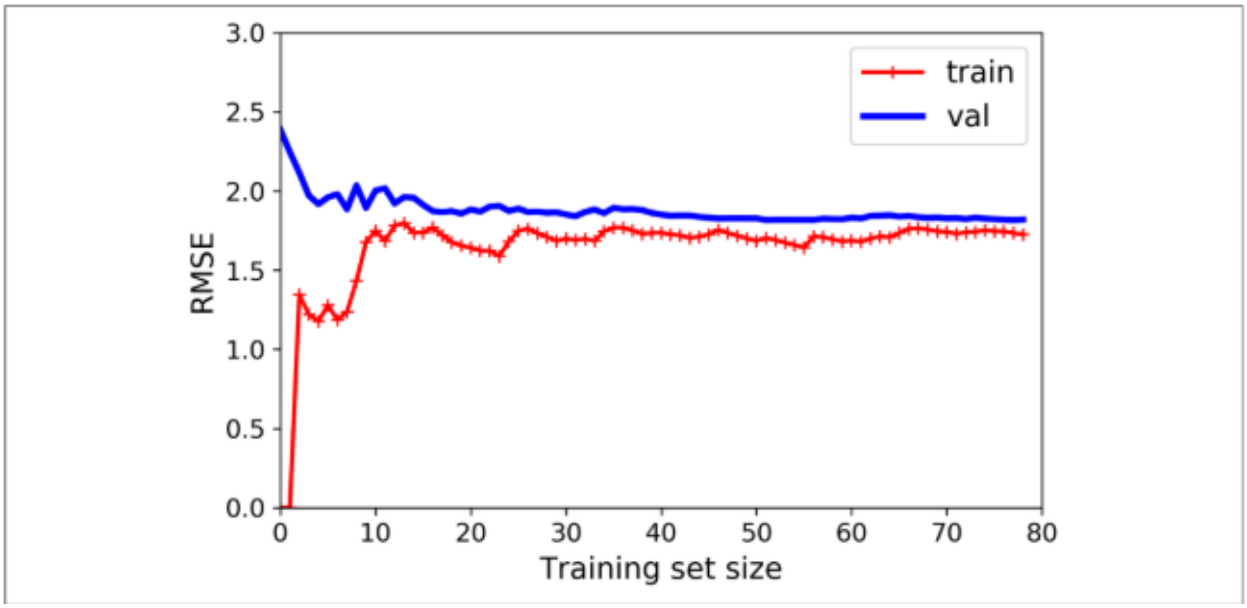

Figure 4-15. Learning curves

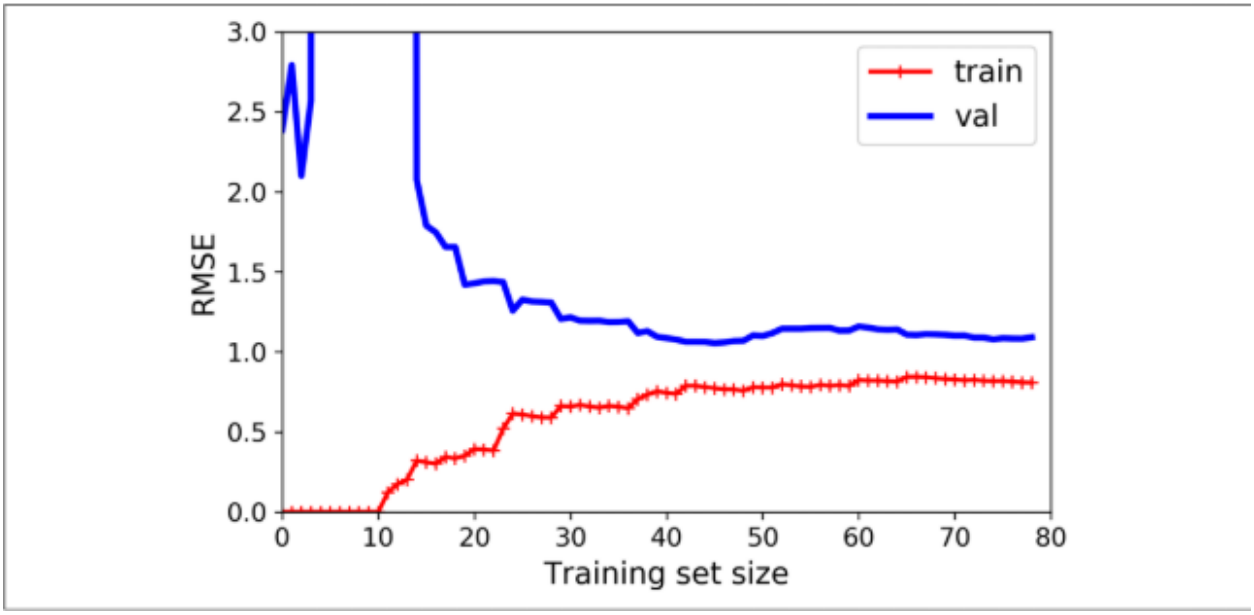

Figure 4-16. Learning curves for the 10th-degree polynomial model

---

Bias–Variance Trade-off

Generalization error can be decomposed into bias (wrong assumptions leading to underfitting), variance (high sensitivity leading to overfitting), and irreducible error (data noise). Increasing complexity typically reduces bias but increases variance, requiring trade-off management.

Regularized Linear Models

Regularization constrains model parameters to reduce overfitting. Ridge Regression uses L2 penalties to shrink weights smoothly. Lasso Regression uses L1 penalties that can push some weights to exactly zero, performing feature selection. Elastic Net mixes L1 and L2 penalties, balancing sparsity and stability, and is often preferred when features are correlated. Feature scaling is critical for these methods.

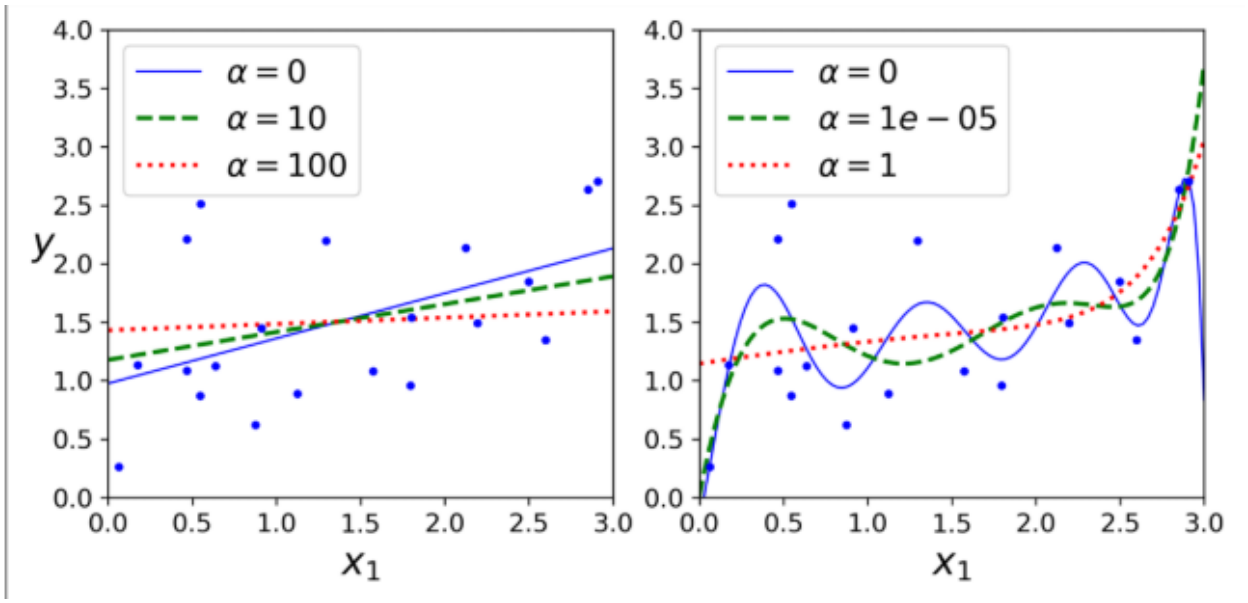

Figure 4-17. A linear model (left) and a polynomial model (right), both with various levels of Ridge regularization

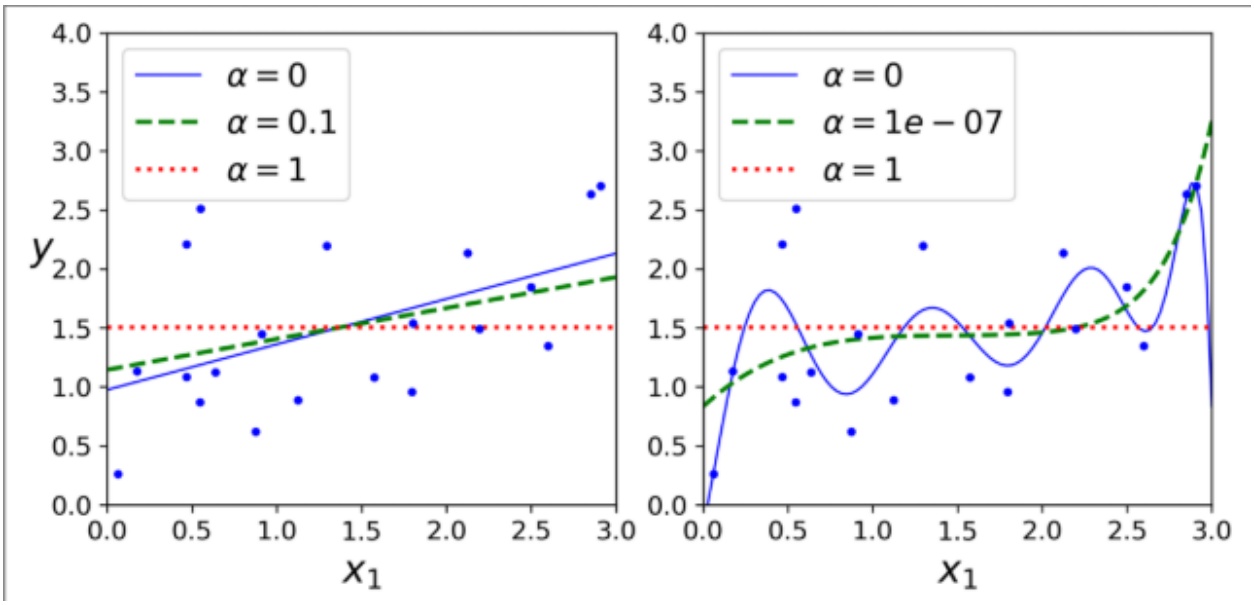

Figure 4-18. A linear model (left) and a polynomial model (right), both using various levels of Lasso regularization

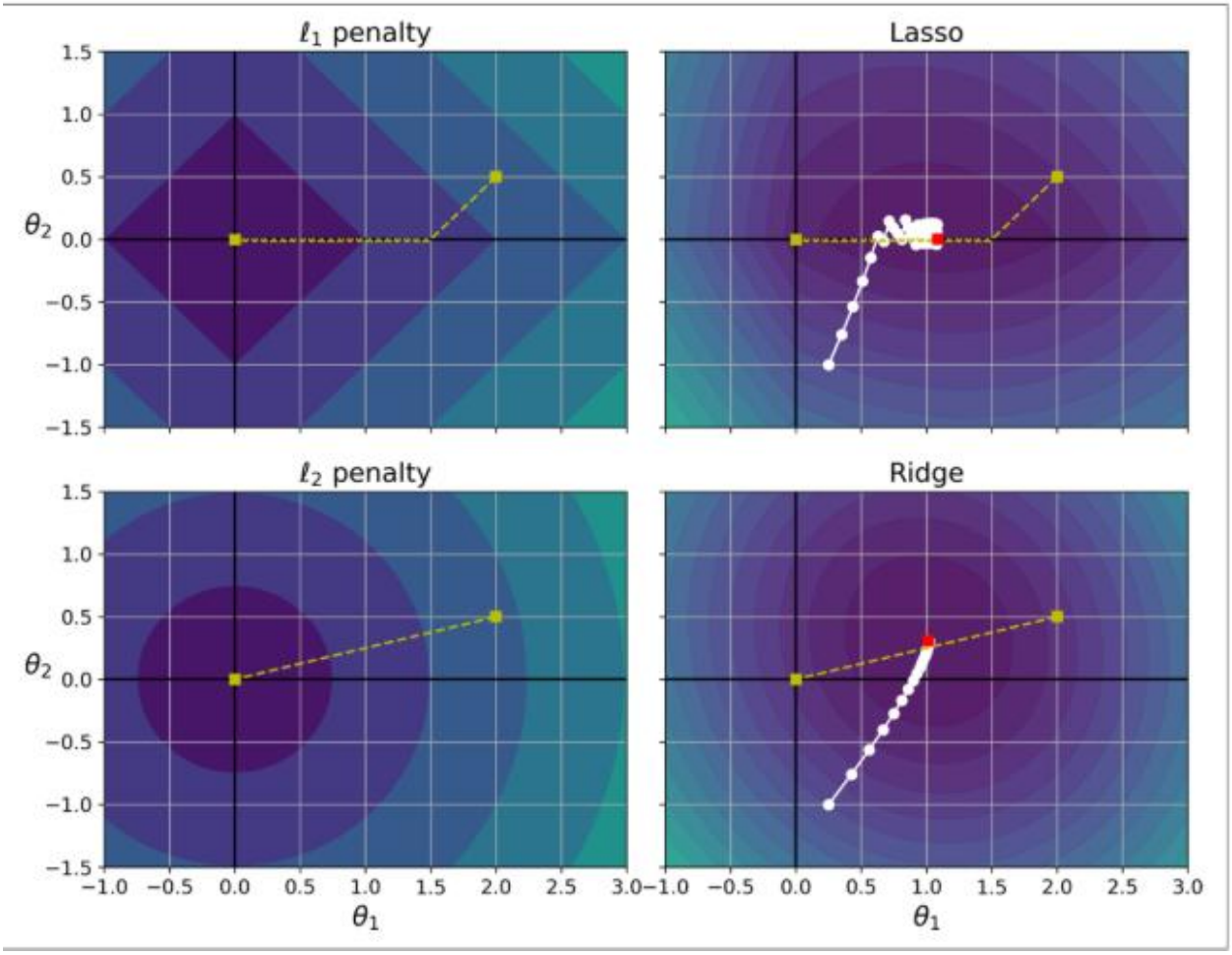

Figure 4-19. Lasso versus Ridge regularization

Early Stopping

Early stopping regularizes iterative training by stopping when validation error stops improving, preventing the model from overfitting. It is efficient and often performs as well as explicit penalties.

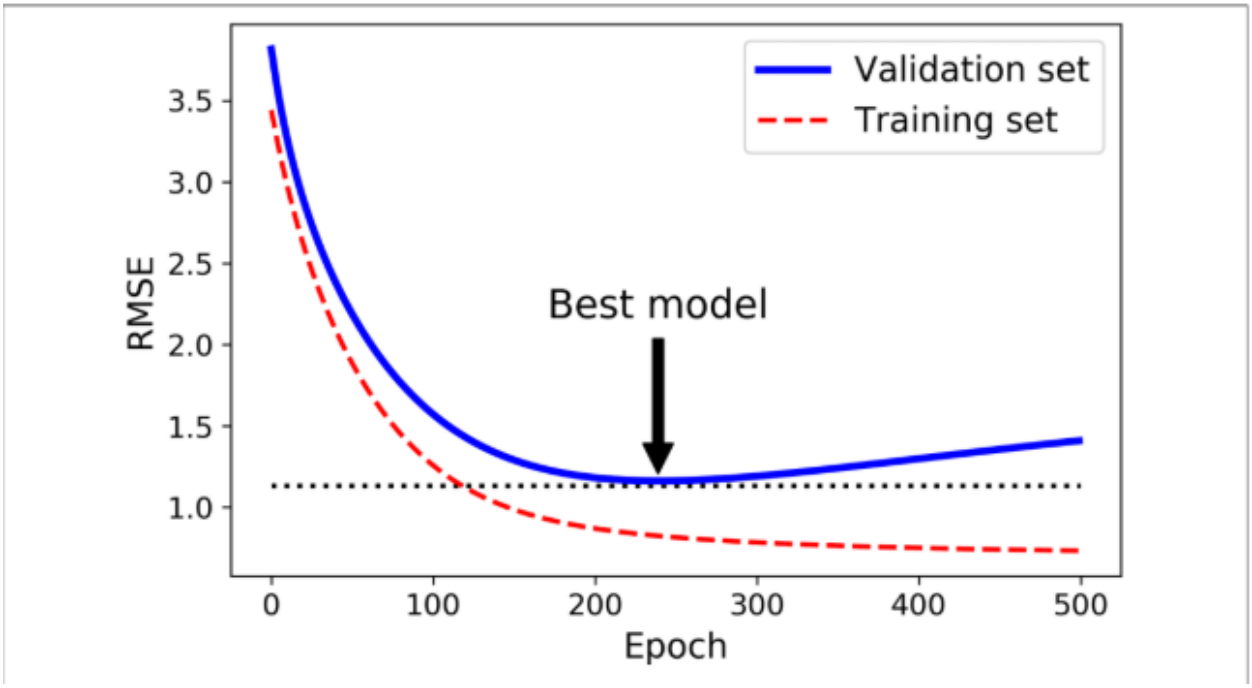

Figure 4-20. Early stopping regularization

Logistic Regression

Logistic Regression is a binary classifier that estimates probabilities using the sigmoid (logistic) function, then predicts class 1 if probability ≥ 0.5. Training minimizes log loss, a convex function, ensuring convergence to a global minimum using Gradient Descent-style optimization. The model's decision boundary is linear, and regularization is controlled by parameter C (inverse of regularization strength).

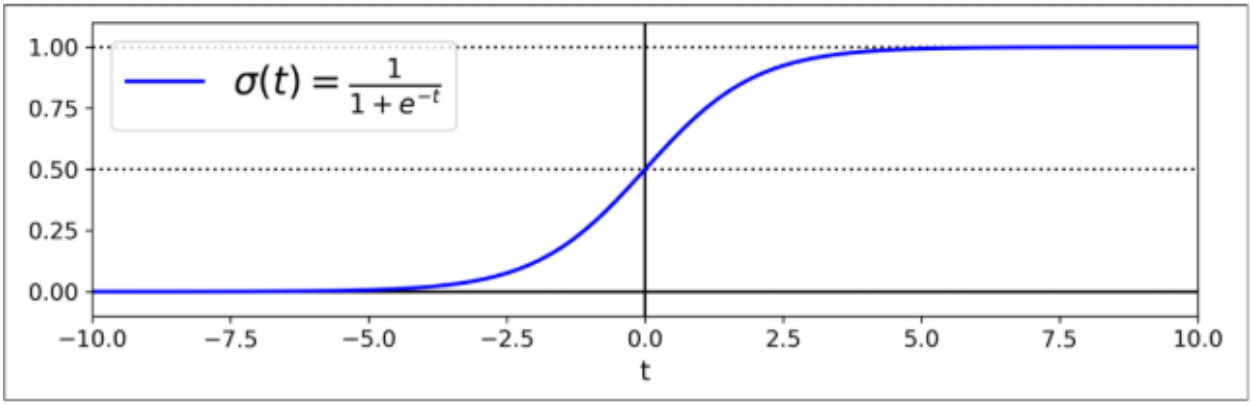

Figure 4-21. Logistic function

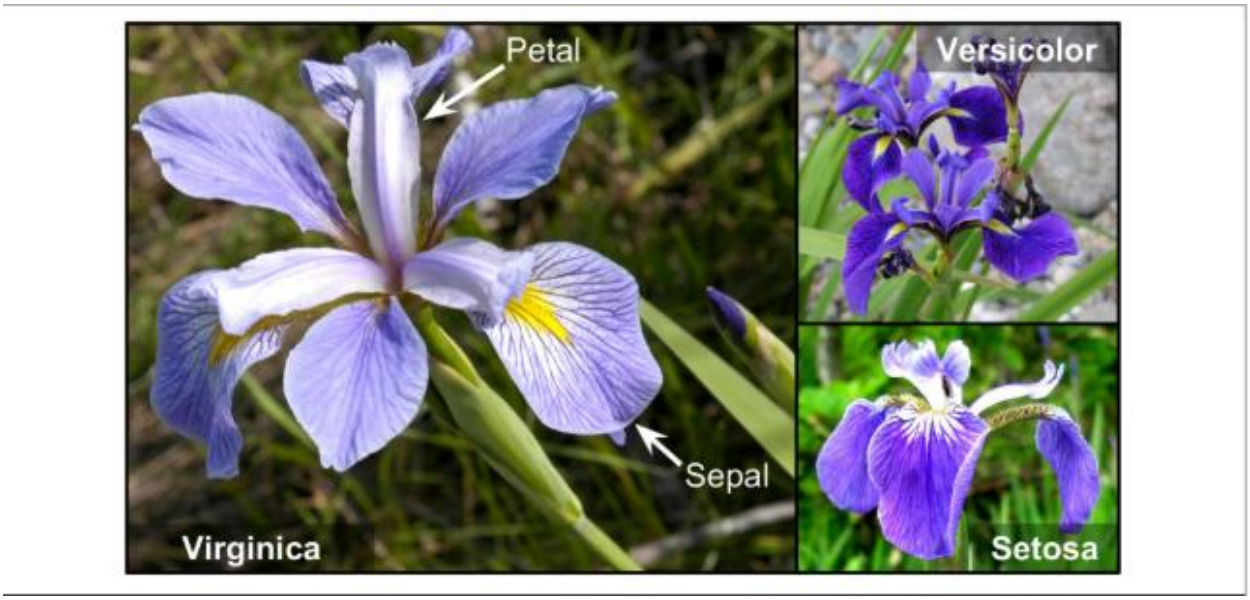

Figure 4-22. Flowers of three iris plant species

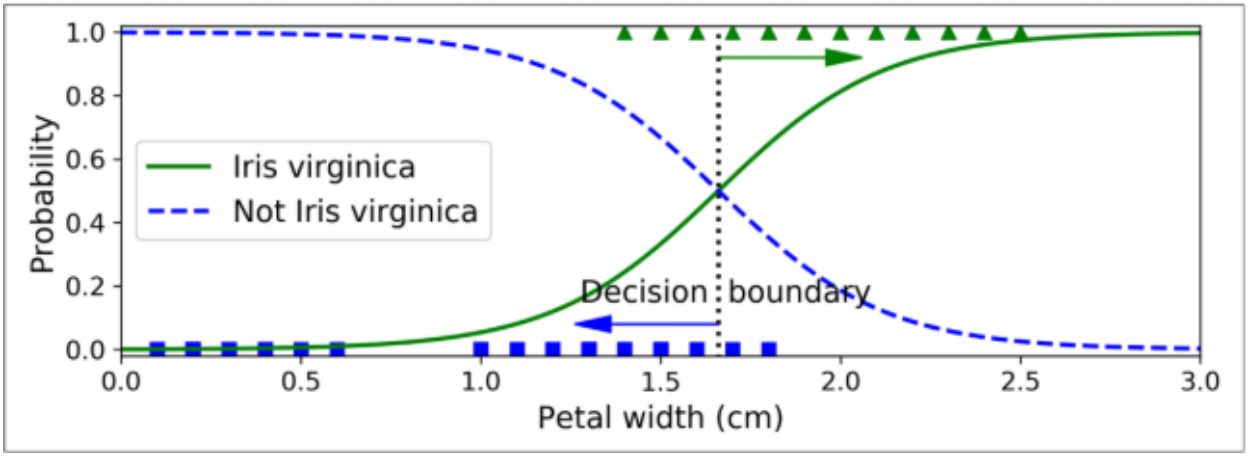

Figure 4-23. Estimated probabilities and decision boundary

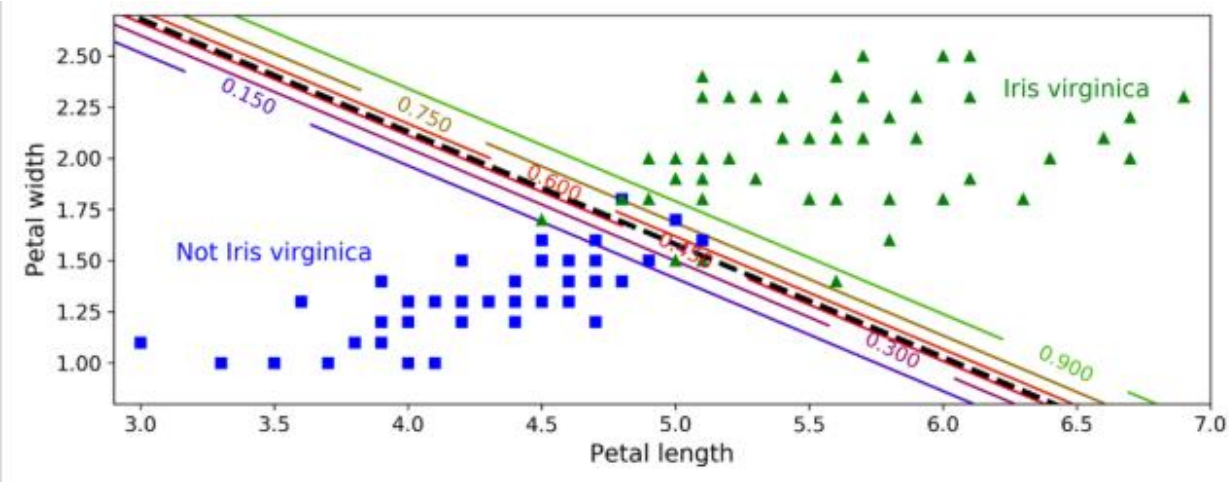

Figure 4-24. Linear decision boundary

Softmax Regression

Softmax Regression generalizes Logistic Regression to multiclass classification by computing class scores and converting them into probabilities using the softmax function. Training minimizes cross-entropy loss, and the classifier predicts the class with the highest probability. Decision boundaries between classes remain linear.

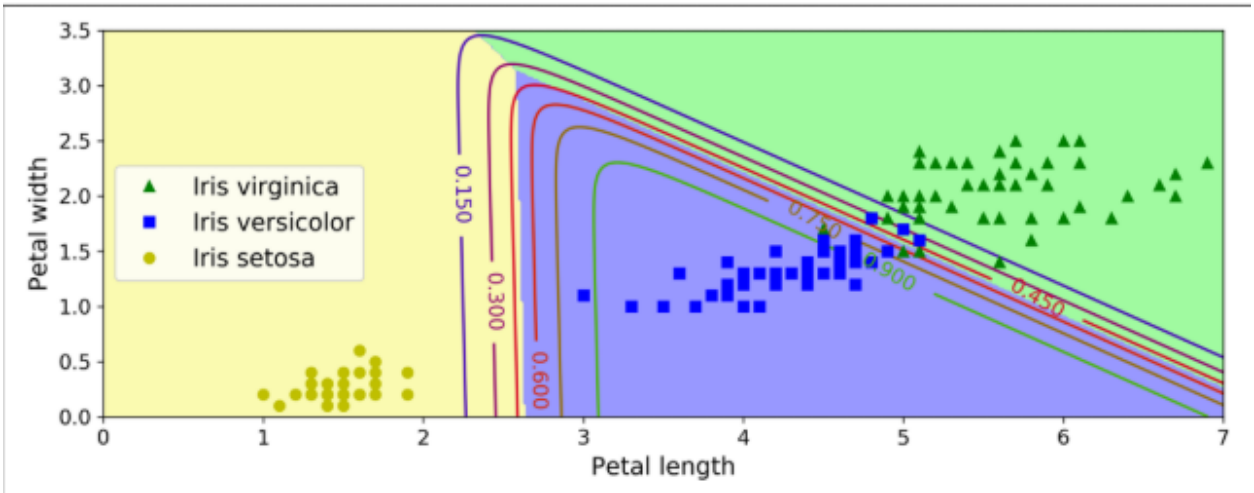

Figure 4-25. Softmax Regression decision boundaries

Conclusion

Chapter 4 explains how models are trained through direct mathematical solutions and iterative optimization. It introduces Gradient Descent variants, demonstrates how model complexity affects generalization, explains regularization and early stopping, and extends linear modeling concepts into classification via Logistic and Softmax Regression. These topics form the foundation for training neural networks in later chapters.