CHAPTER 4

Training Models

Chapter 4 explains in detail how common models are trained, focusing on linear and logistic models and on optimization methods like Gradient Descent. The goal is to open the "black box" so you understand cost functions, closed-form solutions, and iterative optimization, which later connect directly to neural networks.

The chapter starts with **Linear Regression** as a base case. A linear model predicts using a weighted sum of input features plus a bias,

$$\hat{y} = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n = \boldsymbol{\theta}^\top \mathbf{x}$$

and is trained by minimizing the Mean Squared Error (MSE) over the training set. The MSE cost is

$$\text{MSE}(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} - y^{(i)})^2$$

where m is the number of training instances. There is a **closed-form solution** called the Normal Equation,

$$\boldsymbol{\theta} = (X^\top X)^{-1} X^\top \mathbf{y}$$

which directly gives the parameters that minimize MSE but scales poorly when the number of features is very large. Scikit-Learn's LinearRegression uses an SVD-based approach related to the pseudoinverse, which is numerically more stable than the raw Normal Equation.

Because closed-form methods become slow or infeasible when features are numerous or data is streamed, the chapter then introduces **Gradient Descent (GD)**. GD iteratively updates parameters in the direction of the negative gradient of the cost,

$$\boldsymbol{\theta}_{\text{next}} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \text{MSE}$$

with learning rate η. For Linear Regression, the gradient is

$$\nabla_{\boldsymbol{\theta}}\text{MSE} = \frac{2}{m} X^{\top}(X\boldsymbol{\theta} - \mathbf{y})$$

The chapter emphasizes how the **learning rate** affects convergence (too small: slow; too large: divergence) and why **feature scaling** (e.g., StandardScaler) is crucial to avoid extremely elongated "bowls" in parameter space. It compares three GD variants:

- **Batch GD**: uses all training instances for every gradient step; stable but slow for large m.
- **Stochastic GD (SGD)**: uses one randomly chosen instance per step; very fast and works with out-of-core data but noisy and only converges "on average," typically with a decreasing learning schedule.
- **Mini-batch GD**: uses small batches; benefits from vectorized hardware acceleration and is less noisy than pure SGD; standard in deep learning.

The chapter then tackles **Polynomial Regression**, where you augment the original features with polynomial combinations (e.g., x, x^2, x^3) and then fit a linear model on the expanded feature space using PolynomialFeatures plus LinearRegression or SGDRegressor. While this allows fitting nonlinear relationships, it increases the risk of **overfitting**. To diagnose underfitting vs overfitting, the chapter introduces **learning curves**, which plot training and validation error against training set size. A high, closely aligned pair of curves indicates underfitting; a low training error but much higher validation error indicates overfitting. This naturally leads to the **bias–variance trade-off**, where model complexity increases variance and decreases bias, and vice versa.

To control overfitting in linear models, the chapter presents **regularized linear models**. In **Ridge Regression** (L2 regularization), the cost adds a penalty on the squared weights,

$$J(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\theta}) + \alpha \sum_{j=1}^{n} \theta_j^2$$

encouraging small weights and reducing variance. **Lasso Regression** (L1 regularization) uses

$$J(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\theta}) + \alpha \sum_{j=1}^{n} |\, \theta_j \,|$$

which can drive some weights exactly to zero, performing automatic feature selection. **Elastic Net** combines both L1 and L2 penalties with a mixing ratio r,

$$J(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\theta}) + r\alpha \sum_j | \theta_j | + (1-r)\alpha \sum_j \theta_j^2$$

providing a flexible middle ground between Ridge and Lasso. The chapter also introduces **early stopping** as a dynamic regularization technique: when training with GD, monitor validation error and stop training when it starts to increase, rolling back to the best epoch.

Finally, the chapter covers **Logistic Regression** for binary classification and **Softmax Regression** for multiclass problems. Logistic Regression estimates the probability of the positive class using the logistic (sigmoid) function,

$$\hat{p} = \sigma(\boldsymbol{\theta}^\top \mathbf{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^\top \mathbf{x}}}$$

and predicts class 1 if $\hat{p} \geq 0.5$. Training minimizes the **log loss** (negative log-likelihood), a convex cost function. Softmax Regression generalizes this by computing a score $s_k(\mathbf{x}) = \boldsymbol{\theta}_k^\top \mathbf{x}$ for each class k, and then applying the softmax function,

$$\hat{p}_k = \frac{e^{s_k(\mathbf{x})}}{\sum_{j=1}^{K} e^{s_j(\mathbf{x})}}$$

and predicting the class with maximum $\hat{p}_k$. It is trained by minimizing the **cross-entropy** over all classes, which encourages high probability for the true class and low for others.