CHAPTER 3

Classification

Chapter 3 focuses on the practical and conceptual basics of **classification**, using the MNIST handwritten digits dataset as the main example. The chapter starts by loading the 70,000 grayscale digit images (28×28 pixels flattened to 784 features) and splitting them into a training set (60k) and test set (10k), then visualizing a few samples to build intuition about the task. MNIST is treated as the "hello world" of machine learning classification, and the dataset structure in Scikit-Learn (with data, target, and DESCR keys) is explained so you can reuse the same pattern for other datasets.

The first concrete model is a **binary classifier** that detects the digit "5" versus "not-5", using SGDClassifier because it scales well to large datasets and supports online learning. The chapter constructs binary labels (True for 5, False otherwise), fits the classifier, and then evaluates it; this exposes the limitations of using **accuracy** alone on skewed datasets, since a "never-5" dummy classifier already achieves over 90% accuracy by always predicting "not-5". This motivates the use of richer metrics: **confusion matrix**, **precision**, **recall**, and the **F1-score**. Precision and recall are defined via true positives (TP), false positives (FP), and false negatives (FN), with formulas

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

and the F1-score as their harmonic mean,

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Using cross_val_predict, the chapter shows how to get per-instance predictions, compute the confusion matrix, and then derive precision, recall, and F1 for the "5" detector, which turn out to be much less impressive than the raw accuracy.

A key idea is the **precision–recall trade-off**: the classifier computes a decision score for each instance, and comparing this score to a threshold determines the prediction. Raising the threshold generally increases precision but reduces recall, while lowering it does the opposite. The chapter uses decision_function to obtain scores, then precision_recall_curve to plot

precision and recall versus threshold, and to select a threshold that achieves a desired precision (e.g., 90%) at the cost of lower recall. It also introduces the **ROC curve** (true positive rate vs false positive rate), the **ROC AUC** metric, and shows how to compare classifiers (e.g., SGD vs RandomForest) using ROC and PR curves, especially when the positive class is rare.

After binary classification, the chapter moves to **multiclass classification** (digits 0–9). Some algorithms like SGDClassifier, RandomForestClassifier, and naive Bayes support multiclass natively, while strictly binary algorithms (e.g., SVMs, Logistic Regression) can be extended via **one-vs-rest (OvR)** or **one-vs-one (OvO)** strategies. With Scikit-Learn these strategies are usually automatic: fitting SGDClassifier directly on digit labels yields a 10-class model, and cross_val_score is then used to evaluate accuracy, which improves when the input features are scaled. The chapter emphasizes **error analysis**: computing a full confusion matrix for the 10 classes, visualizing it as an image (often normalized row-wise), and inspecting where the model confuses certain pairs of digits (notably 3 vs 5). This leads to concrete ideas like centering digits and engineering features (e.g., detecting loops) to reduce specific error types.

The final part introduces **multilabel** and **multioutput** classification. In multilabel classification, each instance can belong to multiple binary labels simultaneously—for example, labeling each digit as "large" (7–9 or not) and "odd" or "even", and training a KNeighborsClassifier that outputs pairs like [False, True] for a digit 5. Evaluation can use averaged F1 across labels, with different averaging strategies (e.g., macro, weighted) depending on label importance. Multioutput classification generalizes this further: the output is a vector of labels where each component can take multiple values. The chapter's example adds random noise to MNIST images and trains a k-NN model that maps noisy images to clean ones, effectively performing **image denoising** as a multioutput classification/regression task over all pixel intensities