Chapter 3

Classification

Overview

This chapter introduces **classification**, the second major family of supervised learning tasks. Unlike regression, which predicts continuous values, classification predicts discrete classes. The chapter uses the well-known **MNIST dataset**—70,000 labeled handwritten digit images—as a practical foundation for exploring binary, multiclass, multilabel, and multioutput classification, as well as essential evaluation metrics such as precision, recall, F1, ROC curves, and confusion matrices.

MNIST

MNIST consists of 28×28 grayscale digit images represented as 784-dimensional feature vectors. Scikit-Learn's fetch_openml loads the dataset, providing a dictionary structure with data, target, and metadata. Visualizing an instance by reshaping its feature vector produces an easily recognizable digit. Labels are cast to integers, and the dataset is divided into 60,000 training images and 10,000 test images. The training set is already shuffled to ensure statistically consistent cross-validation folds. **Figure 3-1** displays sample digits, illustrating variation in handwriting styles and the challenge of classification.

Figure 3-1. Digits from the MNIST dataset

Training a Binary Classifier

The chapter first simplifies MNIST into a **binary classification** problem: detecting whether an image is a **5** or **not-5**. Boolean target vectors are created, and an **SGDClassifier** is trained due to its scalability and suitability for large datasets and online learning. Initial predictions appear correct on sample digits, but proper evaluation is required.

Performance Measures

Accuracy alone is misleading on imbalanced datasets. For example, a trivial classifier that always predicts “not-5” achieves over **90% accuracy** because only about 10% of the images are 5s. This exposes accuracy’s limitations and motivates richer evaluation metrics.

Cross-validation using cross_val_score yields realistic performance estimates, and a manual version using StratifiedKFold demonstrates how stratified sampling preserves class proportions across folds.

Confusion Matrix

A more informative metric is the **confusion matrix**, which counts how predictions align with actual classes. Using cross_val_predict, predictions are produced in a "clean" manner, then compared with true labels. The matrix reveals true positives, false positives, true negatives, and false negatives. A perfect classifier has nonzero values only on the main diagonal.

From the confusion matrix, **precision**, **recall**, and the combined **F1 score** are computed:

- Precision answers: *Of the images labeled as 5, how many were actually 5?*
- Recall answers: *Of all real 5s, how many did we correctly detect?*

The classifier achieves roughly **73% precision** and **76% recall**, much less impressive than its raw accuracy implies, illustrating why confusion-based metrics are crucial.

Precision/Recall Trade-off

Classifiers often compute a **decision score** for each instance, then apply a threshold to determine class membership. Raising the threshold increases precision but reduces recall; lowering it does the opposite. **Figure 3-3** visualizes this ranking effect.

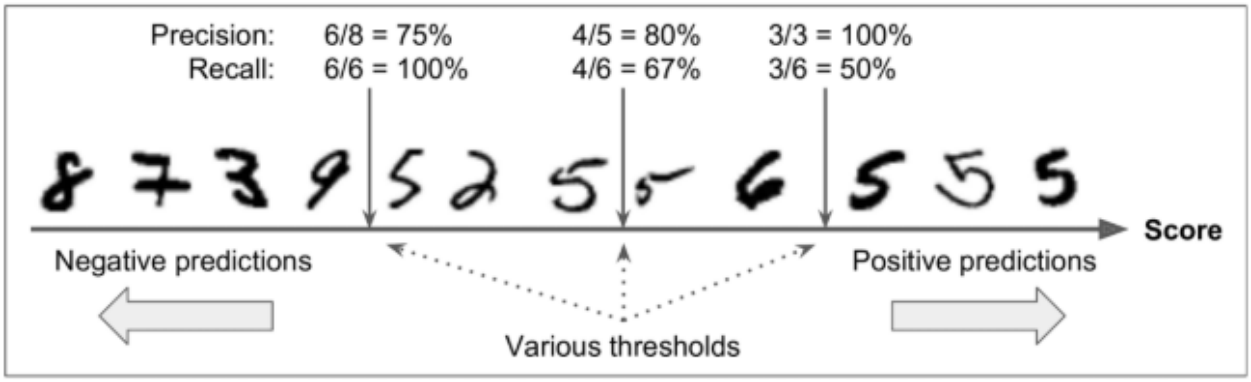

Figure 3-3. In this precision/recall trade-off, images are ranked by their classifier score,and those above the chosen decision threshold are considered positive; the higher the threshold, the lower the recall, but (in general) the higher the precision

Using decision scores from cross_val_predict, the chapter plots precision and recall curves over all thresholds (**Figure 3-4**) and precision-versus-recall directly (**Figure 3-5**).

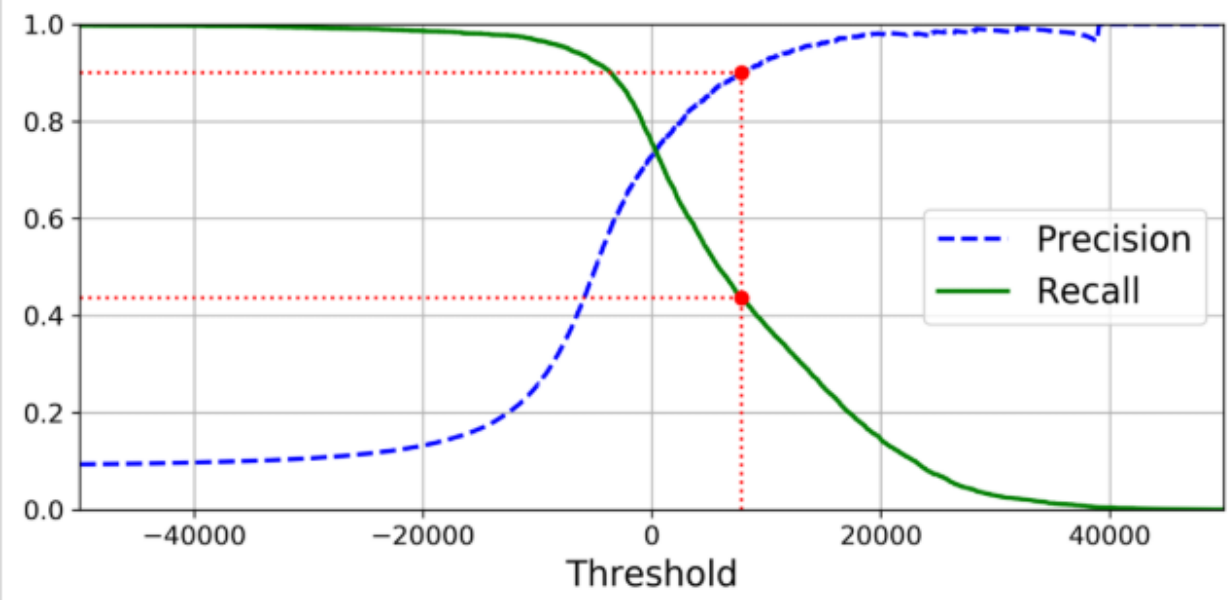

Figure 3-4. Precision and recall versus the decision threshold

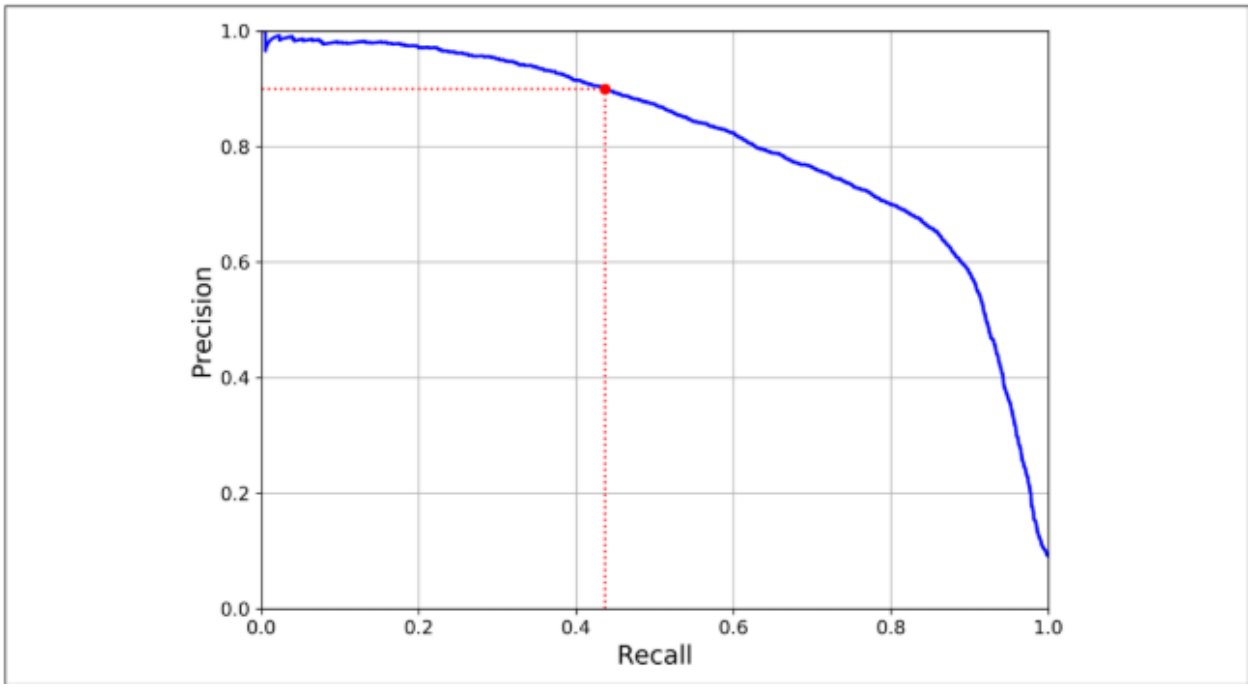

Figure 3-5. Precision versus recall

These reveal performance trade-offs and guide threshold selection. For example, targeting 90% precision dramatically reduces recall (~44%). The principle is clear: **you can increase precision to any level you want—if you are willing to sacrifice recall.**

ROC Curve

The **ROC curve** plots the true positive rate (recall) against the false positive rate for various thresholds. A classifier close to the top-left corner performs best; the diagonal line is random guessing. The **AUC** quantifies curve quality. The SGD classifier achieves an ROC AUC of about **0.96**, but a **RandomForestClassifier** performs significantly better, with an ROC AUC near **0.998** (**Figure 3-7**). The PR curve, however, remains more informative when the positive class is rare.

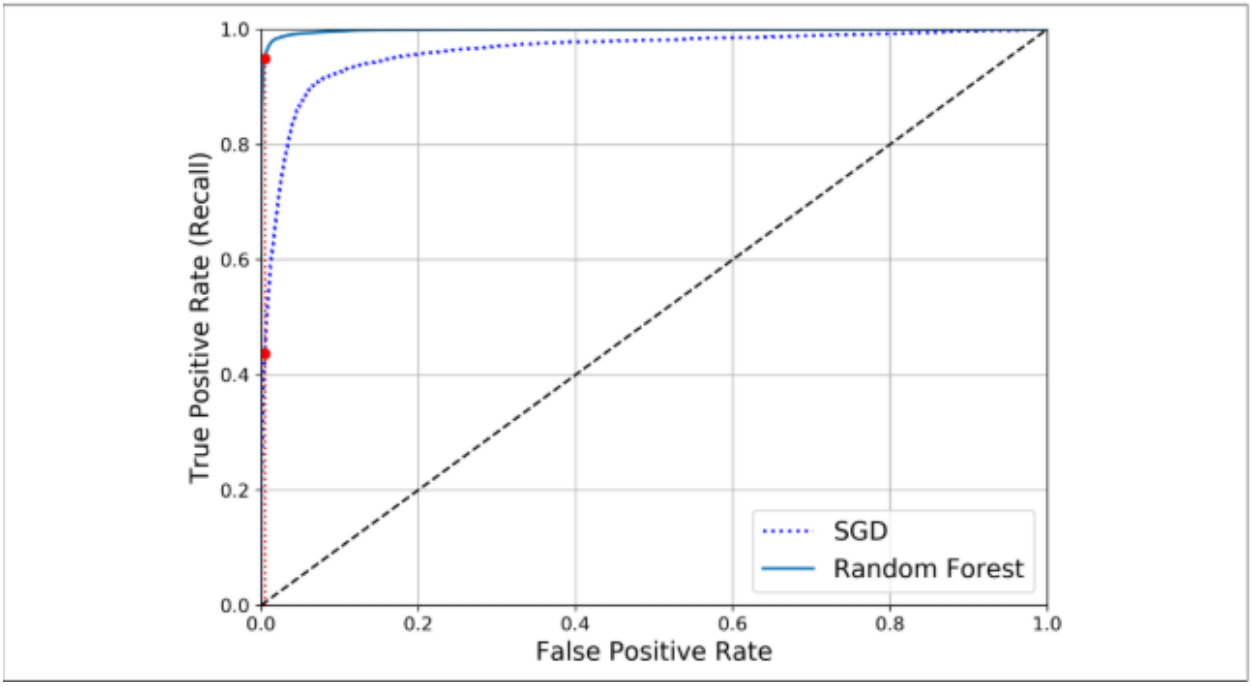

Figure 3-7. Comparing ROC curves: the Random Forest classifier is superior to the SG classifier because its ROC curve is much closer to the top-left corner, and it has a greater AUC

Multiclass Classification

Moving beyond binary classification, the chapter demonstrates **multiclass classification** (digits 0–9). Some algorithms (SGD, Random Forest, Naive Bayes) handle multiclass directly, while others (SVM, Logistic Regression) require strategies such as:

- **One-vs-Rest (OvR)**: one binary classifier per class.
- **One-vs-One (OvO)**: one classifier for every pair of classes.

Scikit-Learn automatically chooses the strategy depending on the algorithm. Training an SVC uses OvO internally, producing 10 decision scores. Accuracy improves after scaling inputs, reaching about **90%**.

Error Analysis

To improve models, the chapter emphasizes **examining error patterns**. A confusion matrix for multiclass predictions reveals frequent confusions, especially between visually similar digits such as 3s and 5s. Normalizing the matrix (dividing each row by total instances in that class) highlights relative error rates rather than raw counts.

Visualizing misclassified instances shows the limits of linear models like SGD: since classification is based on weighted pixel intensities, small shifts or rotations can cause misclassification. Better preprocessing (centering, deskewing images) or more powerful models may reduce these errors.

Multilabel Classification

Some tasks require predicting multiple labels per instance. The chapter creates a toy example where each digit image receives two labels: whether the digit is **large** (≥ 7) and whether it is **odd**. A **KNeighborsClassifier** predicts both labels simultaneously. Multilabel evaluation often averages metrics across labels, though weighting by label frequency may be preferable.

Multioutput Classification

A generalized case of multilabel classification is **multioutput-multiclass** classification, where each output can take multiple values. The chapter demonstrates this by training a k-NN model to **denoise images**: the input is a noisy digit, and the output is a clean 28×28 pixel array. Though pixel values are continuous, the method is framed as classification over 256 possible intensity levels. The k-NN model successfully reconstructs cleaner images, illustrating the flexibility of multioutput systems.

Conclusion

Chapter 3 provides a comprehensive foundation in classification techniques:

- Understanding the limits of accuracy
- Using confusion matrices, precision, recall, F1, and ROC curves

- Adjusting thresholds to control precision/recall trade-offs
- Comparing classifiers with AUC
- Handling binary, multiclass, multilabel, and multioutput tasks
- Performing error analysis to guide model improvement

These tools are essential for building robust classification systems across a wide range of real-world applications.