CHAPTER 7

Ensemble Learning and Random Forests

Chapter 7 presents **ensemble learning** methods, where multiple models are combined to get better performance than any single model, with a focus on **Random Forests**, bagging, boosting, and stacking.

Voting ensembles

The chapter starts with **voting classifiers**, where several diverse classifiers (e.g., Logistic Regression, SVM, Random Forest, KNN) are trained independently and their predictions combined. In **hard voting**, the final prediction is the class with the most votes; even if each model is only slightly better than random (weak learners), the majority vote can become a strong learner via the law of large numbers, provided errors are not highly correlated. In **soft voting**, the ensemble averages predicted class probabilities and picks the class with highest average probability, generally performing better than hard voting if all base models support predict_proba.

Bagging, pasting, and Random Forests

Bagging (bootstrap aggregating) trains many instances of the same base model (often trees) on different bootstrap samples of the training data (sampling **with** replacement), while **pasting** samples without replacement. Each base model has higher bias than one trained on all data but lower variance when their predictions (mode for classification, mean for regression) are aggregated, yielding an ensemble with similar bias but reduced variance. Scikit-Learn’s BaggingClassifier/BaggingRegressor implement both approaches via bootstrap=True/False, with n_estimators and max_samples controlling ensemble size and sample size. Bagging is parallelizable across cores or machines, and setting oob_score=True performs **out-of-bag evaluation** by using each model’s unused training instances as a built-in validation set, avoiding a separate hold-out set.

A **Random Forest** is a bagging ensemble of Decision Trees where, at each split, only a random subset of features is considered, increasing tree diversity. RandomForestClassifier/RandomForestRegressor expose most tree hyperparameters (e.g., max_depth, max_leaf_nodes) plus ensemble hyperparameters (n_estimators, max_features, bootstrap); forests typically offer strong accuracy, robustness, and good default behavior. The related **Extra-Trees** ensembles (ExtraTreesClassifier/ExtraTreesRegressor) add more randomness by using random split thresholds as well as random feature subsets, trading slightly higher bias for lower variance and significantly faster training, so they are often competitive with or better than Random Forests. Forests also provide **feature importances** via feature_importances_, computed from the impurity reduction contributed by each feature across all trees, weighted by node sample counts.

Boosting (AdaBoost and Gradient Boosting)

Boosting builds an ensemble sequentially, where each new model focuses on correcting the predecessors' errors. The chapter describes two main boosting methods:

- **AdaBoost**: trains a sequence of weak learners (often shallow trees), re-weighting training instances after each iteration so that misclassified points receive more weight. Each learner's contribution is scaled by a weight that depends on its error rate, and final predictions are a weighted vote over all learners; AdaBoostClassifier implements this with base estimator (typically DecisionTreeClassifier(max_depth=1)), n_estimators, and learning_rate controlling the trade-off between number of trees and step size.
- **Gradient Boosting (GBRT)**: fits each new tree to the **residual errors** of the current ensemble, effectively performing gradient descent in function space. GradientBoostingRegressor grows trees one by one, adding each scaled by learning_rate; smaller learning rates require more trees but usually improve generalization (shrinkage). Overfitting is controlled by limiting tree depth and using **early stopping**: monitor validation error versus number of trees and keep the ensemble at the tree count that minimizes it, which can be implemented via staged_predict or by using XGBoost's native early stopping. The chapter briefly highlights **XGBoost** as an optimized, widely used gradient boosting library with a Scikit-Learn–like API and built-in early stopping.

Stacking

Stacking (stacked generalization) trains a second-level model (a **blender** or meta-learner) to combine the predictions of several base learners instead of using a fixed rule like voting. Typically, the training set is split: base models are trained on one part, and their predictions on a held-out part form a new feature matrix for training the blender, which learns how to best weight or combine these predictions. This idea can be extended to multiple layers of models, but increases complexity and risk of overfitting, so careful validation is required.