

Study & Revision Guide: Applied Machine Learning & Data Processing

Module 1: Ensemble-Based Machine Learning (Notebook 1)

Theoretical Foundations

- **Strong vs. Weak Learnability:** A concept is strongly learnable if a polynomial-time algorithm produces a model with high accuracy; it is weakly learnable if predictions are only slightly better than random chance. Schapire (1990) proved weak and strong learnability are equivalent via boosting.
- **Benefits of Ensembles:** Combines diverse inductive biases, reduces risk of selecting an unsuited model for a specific domain, decomposes non-linear boundaries using divide-and-conquer, enables parallelized training on data partitions, and handles class imbalance well.

Mechanisms for Ensemble Diversity

- **Data Manipulation:**
 - *Boosting:* Dynamically reweights hard-to-classify samples.
 - *Bagging:* Resamples training data with replacement via bootstrap samples.
 - *Partitioning:* Divides data into random disjoint sets.
- **Inducer & Feature Manipulation:**
 - *Parameter Variation:* Changes tree confidence levels, node counts, or neural network topologies.
 - *Random Subspace Method:* Trains models on random subsets of input features.
 - *Heterogeneous Ensembles:* Mixes fundamentally different algorithms (e.g., SVMs, Decision Trees, Neural Networks).
- **Diversity Metrics:**
 - *Pairwise:* Evaluates 2×2 agreement matrices across $\frac{L(L-1)}{2}$ classifier pairs (Q-Statistics, Correlation, Double Fault measure).
 - *Non-Pairwise:* Single global metric across the ensemble (Entropy, Kohavi-Wolpert variance).
 - *No Free Lunch Theorem:* No single diversity metric consistently correlates with accuracy across all datasets.

Output Combination & Selection Strategies

- **Ensemble Fusion (Global):**

- *Majority/Plurality Voting*: Unweighted decision. Optimal under 3 conditions: (1) odd number of models, (2) individual accuracy $p > 0.5$, and (3) independent errors.
- *Weighted Majority Voting*: Weights model votes (α_t) based on accuracy:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$$

- **Ensemble Selection (Local)**:
 - *Dynamic Classifier Selection*: Chooses local experts dynamically based on target instance region (e.g., k -NN neighbourhood).
 - *Cascades & Gating*: Routes samples hierarchically or uses gating networks (Mixture of Experts).
- **Sizing & Pruning**: Fixed iterations, dynamic halting criteria (Kappa-error curves), or post-hoc pruning to remove redundant models (Double Fault clustering, ROC search).

The AdaBoost Algorithm

- **Core Concept**: Fits sequential weak learners (e.g., decision stumps) on non-uniform sample distributions.

Step	Mechanics & Formulas
1. Initialize	Equal sample weights: $D_1(i) = \frac{1}{N}$
2. Fit Weak Learner	Train $h_t(x) \in \{-1, +1\}$ using current weights D_t
3. Compute Error	Weighted error rate: $\varepsilon_t = \sum_{i: h_t(x_i) \neq y_i} D_t(i)$
4. Calculate Weight (α_t)	Classifier strength: $\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$
5. Update & Normalize	$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$ where Z_t normalizes weights
6. Final Output	$H(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right)$

- **Margin Distribution**: Defined as $y_i \sum \alpha_t h_t(x_i) / \sum \alpha_t$. AdaBoost maximizes the minimum margin. Larger positive margins indicate higher confidence, stronger discriminative power, and better generalization error bounds.
- **Convergence Behavior**: Training error can reach 0% while generalization error continues to drop because boosting continues enlarging prediction margins. However, complex base learners or noisy labels will cause divergence (overfitting).

Module 2: Time-Series Forecasting & Feature Engineering (Notebook 2)

Forecasting Fundamentals

- **Univariate vs. Multivariate:**
 - *Univariate*: Models a single time-dependent variable based strictly on its own historical values.
 - *Multivariate*: Predicts multiple interdependent time-series variables simultaneously.
 - *Covariates / Exogenous Variables*: External features (weather, calendar events, promotions) that influence the target variable but are not predicted by the model.
- **Forecast Horizon (h):**
 - *Short-Term*: High accuracy, focuses on immediate steps ahead.
 - *Long-Term*: Error compounds; requires direct or recursive multi-step forecasting strategies.

Feature Engineering for Time Series

- **Stationarity Transformations**: Removing trend and seasonality via differencing ($\Delta y_t = y_t - y_{t-1}$) or log transformations to stabilize variance.
- **Lag Features**: Converting time series into tabular format by using past values ($y_{t-1}, y_{t-2}, \dots, y_{t-k}$) as predictors.
- **Rolling Statistics**: Summary metrics (rolling mean, rolling standard deviation, min/max) computed over fixed sliding historical windows.

Experimental Design & Validation

- **Data Leakage Prevention**: Standard random k -fold cross-validation causes leakage by using future data to predict the past.
 - **Validation Strategy**: Use temporal splits such as **TimeSeriesSplit** (expanding or rolling window cross-validation).
-

Module 3: Gradient Boosting & Boosting Frameworks (Notebook 3)

Disclaimer: *This section provides a high-level theoretical overview of Notebook 3. Students must review the original source material, code implementations, hyperparameter setups, and practical exercises in Notebook 3 for full details and exam preparation.*

Theoretical Foundations

- **Functional Gradient Descent**: Unlike AdaBoost (which rescales sample weights based on exponential loss), Gradient Boosting builds sequential

ensembles by optimizing arbitrary differentiable loss functions via gradient descent in function space.

- **Pseudo-Residuals:** At each iteration, a weak learner (typically a shallow decision tree) is trained to fit the negative gradient (pseudo-residuals) of the loss function with respect to the current ensemble's predictions.

Algorithm Mechanics & Mathematical Steps

Step	Mechanics & Formulas
1. Initialize Model	Constant value prediction: $F_0(x) = \arg \min_{\gamma} \sum_{i=1}^N L(y_i, \gamma)$
2. Compute Pseudo-Residuals	Calculate negative gradient for $i = 1, \dots, N$: $r_{it} = - \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{t-1}(x)}$
3. Fit Weak Learner	Train base tree $h_t(x)$ to predict terminal region pseudo-residuals r_{it}
4. Compute Leaf Values	Solve line search for optimal leaf output values $\gamma_{jt} =$ $\arg \min_{\gamma} \sum_{x_i \in R_{jt}} L(y_i, F_{t-1}(x_i) + \gamma)$
5. Update Model	$F_t(x) = F_{t-1}(x) + \nu \cdot \sum_j \gamma_{jt} \mathbb{I}(x \in R_{jt})$ where ν is the learning rate

Key Differences: AdaBoost vs. Gradient Boosting

- **Loss Functions:** AdaBoost relies on Exponential Loss (sensitive to outliers); Gradient Boosting supports any differentiable loss function (e.g., Mean Squared Error for regression, Log Loss / Binary Cross-Entropy for classification, Huber loss for outlier robustness).
- **Optimization Mechanics:** AdaBoost adjusts sample weights $D_t(i)$; Gradient Boosting fits residual error vectors directly.

Regularization & Modern Implementations

- **Shrinkage (Learning Rate ν):** Scales tree predictions ($0 < \nu \leq 1$). Lower values reduce individual tree impact, requiring more trees but improving test generalization.
- **Stochastic Gradient Boosting:** Subsamples training rows and/or feature columns per iteration to inject randomness, improve speed, and combat overfitting.
- **Modern Scalable Frameworks:**
 - *XGBoost*: Second-order Taylor expansion (gradients and Hessians), L1/L2 regularization on leaf weights, and exact/approximate quantile split algorithms.

- *LightGBM*: Leaf-wise (best-first) tree growth, Gradient-based One-Side Sampling (GOSS), and Exclusive Feature Bundling (EFB) for memory efficiency.
 - *CatBoost*: Ordered boosting and target encoding designed to handle high-cardinality categorical variables without target leakage.
-

Module 4: Explainable AI (XAI) using SHAP (Notebook 4)

Foundations of XAI

- **Core Goal**: Interpreting black-box machine learning models to ensure trust, transparency, safety, and regulatory compliance.
- **SHAP (SHapley Additive exPlanations)**: Based on game theory. Calculates the marginal contribution of each feature across all possible feature combinations to guarantee fair allocation.
- **Additive Feature Attribution**:

$$f(x) = E[f(X)] + \sum_{i=1}^M \phi_i$$

Where $E[f(X)]$ is the baseline expected value, and ϕ_i is the SHAP value for feature i .

SHAP Plot Analysis

Summary Plot

- **Feature Importance**: Features are ordered vertically by global importance (mean absolute SHAP value).
- **Impact & Directionality**:
 - *Point Position*: Position along the x-axis indicates whether the feature increased or decreased the prediction relative to the baseline.
 - *Color Coding*: Red represents high feature values; blue represents low feature values.
- **Feature Pruning**: Features tightly clustered around zero across all instances contribute no predictive value and can be safely eliminated.

Waterfall Plot

- **Local Explanation**: Deconstructs an individual prediction starting from the base value $E[f(X)]$.

- **Cumulative Shifts:** Sequential arrows display how each feature value moves the individual prediction higher (positive shift) or lower (negative shift) to arrive at $f(x)$.

Dependence Plot

- **Feature Relationship:** Plots a feature's raw values (x-axis) against its corresponding SHAP values (y-axis) to display linear or non-linear effects.
 - **Interaction Effects:** Vertical dispersion of points at a single x-value reveals interactions with a second feature (represented by hue). Helps identify non-linear thresholds for feature engineering.
-

Module 5: Hyperparameter Tuning in Random Forest (Notebook 5)

Principles of Hyperparameter Tuning

- **Objective:** Finding parameter configurations that optimize model generalization by striking the right balance between bias and variance.
- **Bias-Variance Diagnostics:**
 - *High Variance (Overfitting):* High training performance combined with poor validation performance. Addressed by restricting tree depth (`max_depth`), increasing `min_samples_leaf`, or reducing sample ratios.
 - *High Bias (Underfitting):* Poor performance on both training and validation sets. Addressed by increasing capacity (deeper trees, more features).

Tuning Strategies & Diagnostics

- **GridSearchCV Strategy:** Evaluates all Cartesian combinations across a specified parameter grid.
 - **Evaluating Search Boundaries:**
 - If optimal parameters land on the boundary of the search grid (e.g., `max_depth = 20` when grid max is 20), the search space is constrained.
 - *Action:* Expand parameter boundaries, try unconstrained parameters (`max_depth: None`), or adopt broader random searches (`RandomizedSearchCV`) followed by fine-grained grid searches.
-

Module 6: Natural Language Processing Overview (Notebook 6)

Core NLP Pipeline

- **Definition & Purpose:** Enabling computers to understand, interpret, and generate human language.
- **Low-Level Preprocessing:**
 - *Tokenization*: Splitting text into words, subwords, or sentences.
 - *Text Normalization*: Lowercasing, removing punctuation/stop words, and applying Stemming or Lemmatization.
- **Text Vectorization:**
 - *Bag-of-Words (BoW)*: Frequencies of word occurrences; ignores word order and semantics.
 - *TF-IDF*: Evaluates term importance relative to the corpus:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \log \left(\frac{N}{\text{DF}(t)} \right)$$

Module 7: Big Data Handling Strategies (Notebook 7)

Memory Management & Out-of-Core Processing

- **Pandas Bottlenecks:** Single-threaded execution, row-oriented memory allocation, and high memory overhead (often requiring $3\text{--}5\times$ the dataset size in RAM).
- **Chunking Strategies:** Loading and processing large files in smaller batches (`pd.read_csv(..., chunksize=N)`) to stay within memory limits.

Modern High-Performance Data Frameworks

- **Polars Overview:** A multi-threaded, blazingly fast Dataframe library written in Rust.
- **Columnar Memory vs. Row-Wise:**
 - *Row-Wise (Pandas)*: Stores data contiguously by row; inefficient for column filtering and aggregation.
 - *Columnar (Apache Arrow / Polars)*: Stores contiguous data blocks by column. Maximizes CPU cache locality, vectorization (SIMD), and parallel processing.
- **Lazy Evaluation:** Polars builds an execution graph to optimize queries, filter predicates early, and drop unused columns before reading data into memory.