

Machine Learning: Module 5

Dr. Likhit Nayak

1 Boosting Methods: Exponential Loss and AdaBoost

1.1 Key Concepts

In this section, we move beyond individual models to **Ensemble Methods**. The core philosophy here is the "wisdom of crowds": a group of estimators often performs better than a single estimator.

- **Ensemble Learning:** A paradigm where multiple models (hypotheses) are combined. There are two main families:
 1. **Bagging (Bootstrap Aggregating):** Trains models in parallel on different subsets of data (e.g., Random Forest). It primarily reduces *variance* (overfitting).
 2. **Boosting:** Trains models sequentially. It primarily reduces *bias* (underfitting) while also keeping variance in check.
- **Boosting:** The fundamental idea of boosting is *iterative correction*. If we have a model that misclassifies certain points, the next model in the sequence should focus specifically on those "hard" points. We convert a collection of *weak learners* into a single *strong learner*.
- **Weak Learner:** A classifier $h(x)$ is "weak" if its error rate is only slightly better than random guessing.

$$\text{Error} < 0.5 \quad (\text{for binary classification}).$$

The most common weak learner used in Boosting is the **Decision Stump**: a decision tree with a depth of 1 (a single split on a single feature). Stumps have high bias but are very fast to compute.

- **AdaBoost (Adaptive Boosting):** Proposed by Freund and Schapire. It is "adaptive" because it tweaks the distribution of the training data for the next learner based on the performance of the current learner.
 - If a sample is misclassified, its weight is *increased*.
 - If a sample is correctly classified, its weight is *decreased*.

This forces the next weak learner to focus on the difficult examples.

- **Exponential Loss:** In classification, we ideally want to minimize the 0-1 Loss (1 if wrong, 0 if right), but this is non-differentiable and difficult to optimize. AdaBoost optimizes a smooth, convex surrogate called Exponential Loss:

$$L(y, f(x)) = \exp(-yf(x))$$

Here, $y \in \{-1, 1\}$.

- If the prediction is correct (y and $f(x)$ have same sign), $yf(x) > 0$, so Loss < 1 (decaying to 0).
- If the prediction is wrong (y and $f(x)$ have different signs), $yf(x) < 0$, so Loss > 1 (growing exponentially).

It creates a heavy penalty for large negative margins (confident wrong predictions).

1.2 The AdaBoost Algorithm (Detailed Walkthrough)

The goal is to learn a sign function $H(x) = \text{sign}(\sum \alpha_m h_m(x))$. We derive this via forward stagewise additive modeling using exponential loss.

Let the training data be $(x_1, y_1), \dots, (x_N, y_N)$ with labels $y_i \in \{-1, 1\}$.

1. **Initialize sample weights:** At the start, every data point is equally important.

$$w_i^{(1)} = \frac{1}{N}, \quad i = 1, \dots, N.$$

2. **Iterate for $m = 1, \dots, M$:**

- (a) **Train a weak learner:** Find $h_m(x)$ that minimizes the weighted error. The learner doesn't treat all data points as equal; it prioritizes points with high $w_i^{(m)}$.
- (b) **Calculate weighted error:** The error ε_m is the sum of weights of the misclassified samples.

$$\varepsilon_m = \sum_{i=1}^N w_i^{(m)} \mathbb{I}(y_i \neq h_m(x_i)).$$

Note: Since we use weak learners, we expect $\varepsilon_m < 0.5$. If $\varepsilon_m \geq 0.5$, we flip the predictions or stop.

- (c) **Compute learner weight (α_m):** This determines how much "say" classifier h_m has in the final vote.

$$\alpha_m = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_m}{\varepsilon_m} \right).$$

Interpretation:

- If $\varepsilon_m \rightarrow 0$ (very accurate), $\alpha_m \rightarrow \infty$ (high trust).
- If $\varepsilon_m \rightarrow 0.5$ (random guess), $\alpha_m \rightarrow 0$ (no trust).

- (d) **Update sample weights:** We need to increase weights for misclassified samples and decrease for correct ones.

$$w_i^{(m+1)} = w_i^{(m)} \exp(-\alpha_m y_i h_m(x_i)).$$

Breakdown:

- **Correct ($y_i = h_m(x_i)$):** Term becomes $\exp(-\alpha_m)$. Since $\alpha_m > 0$, this is < 1 . The weight **decreases**.
- **Incorrect ($y_i \neq h_m(x_i)$):** Term becomes $\exp(\alpha_m)$. This is > 1 . The weight **increases**.

- (e) **Normalize weights:** To ensure w remains a valid probability distribution summing to 1.

$$Z_m = \sum_{i=1}^N w_i^{(m)} \exp(-\alpha_m y_i h_m(x_i)), \quad w_i^{(m+1)} = \frac{w_i^{(m)}}{Z_m} \cdots.$$

3. **Construct the final classifier:** The final hypothesis is a weighted majority vote.

$$H(x) = \text{sign} \left(\sum_{m=1}^M \alpha_m h_m(x) \right).$$

2 Numerical Optimization via Gradient Boosting

2.1 Key Concepts

While AdaBoost was derived specifically for Exponential Loss, **Gradient Boosting Machines (GBM)** generalize boosting to *any* differentiable loss function.

- **Gradient Descent in Function Space:** In standard optimization, we update parameters $\theta \leftarrow \theta - \eta \nabla_{\theta} L$. In Gradient Boosting, we treat the function values $F(x_i)$ as parameters. We want to adjust $F(x_i)$ to reduce the loss. The "gradient" tells us the direction to move $F(x_i)$.

- **Additive Model:** We build the prediction function $F(x)$ sequentially:

$$F_M(x) = F_0(x) + \sum_{m=1}^M \eta h_m(x).$$

Here, $h_m(x)$ is not fitting the data labels y directly; it is fitting the *errors* of the previous step. η (learning rate/shrinkage) prevents overfitting.

- **Pseudo-Residuals:** The mathematical definition of the "error" remains to be solved for general loss functions. The negative gradient of the loss with respect to the prediction is called the pseudo-residual.

$$r_{im} = - \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F=F_{m-1}}.$$

Intuitively, r_{im} is the direction we need to move the prediction $F(x_i)$ to minimize the loss.

2.2 The Gradient Boosting Algorithm

1. **Initialize:** $F_0(x) = \arg \min_{\gamma} \sum_i L(y_i, \gamma)$. (e.g., the mean of y for regression).
2. **Loop** for $m = 1, \dots, M$:
 - (a) **Compute Residuals:** Calculate the negative gradient r_{im} for all $i = 1 \dots N$.
 - (b) **Fit Weak Learner:** Train a regression tree $h_m(x)$ to predict r_{im} given x_i . *Note: Even for classification, the weak learner in GBM is often a regressor fitting the gradient value.*
 - (c) **Update Model:** Add the new tree to the ensemble:

$$F_m(x) = F_{m-1}(x) + \eta h_m(x).$$

3. **Output:** The final additive model $F_M(x)$.

Special Case: Squared Error Loss

If we choose Mean Squared Error (MSE) as our loss function:

$$L(y, F) = \frac{1}{2}(y - F)^2$$

The derivative is:

$$\frac{\partial L}{\partial F} = -(y - F)$$

The negative gradient (pseudo-residual) is:

$$r_{im} = -(-(y_i - F_{m-1}(x_i))) = y_i - F_{m-1}(x_i)$$

This results in standard **residual fitting**. Thus, Gradient Boosting with MSE is equivalent to fitting trees on the residuals $y - \hat{y}$.

3 Introduction to Reinforcement Learning (RL)

Reinforcement Learning is distinct from Supervised Learning (where we have ground truth labels) and Unsupervised Learning (where we look for hidden structure). RL is about **learning from interaction** to achieve a goal.

3.1 Key Concepts

- **The RL Loop:**
 - At time t , the **Agent** observes state S_t .
 - The Agent selects an **Action** A_t based on a policy.
 - The **Environment** responds with a **Reward** R_{t+1} and a new state S_{t+1} .
- **Policy (π):** The agent's brain/strategy. It maps states to actions.
- **Return (G_t):** The goal is not just immediate reward, but cumulative reward over time.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- **Discount Factor ($\gamma \in [0, 1]$):** Determines the present value of future rewards.
 - $\gamma \rightarrow 0$: Myopic (cares only about immediate reward).
 - $\gamma \rightarrow 1$: Far-sighted. Also ensures mathematical convergence for infinite horizons.

3.2 Markov Processes (MP)

A Markov Process is a sequence of random states S_1, S_2, \dots with the **Markov Property**:

$$P(S_{t+1}|S_t, S_{t-1}, \dots, S_1) = P(S_{t+1}|S_t)$$

"The future is independent of the past given the present." The state S_t captures all relevant information from history. An MP is defined by (\mathcal{S}, P) , where P is the state transition matrix.

3.3 Markov Random Processes (MRP)

An MRP introduces **Value** to the states. It is a Markov Process with rewards attached. Defined by tuple $(\mathcal{S}, P, R, \gamma)$.

- $R(s) = \mathbb{E}[R_{t+1}|S_t = s]$ is the immediate reward for being in state s .
- The **Value Function** $V(s)$ is the expected return starting from state s :

$$V(s) = \mathbb{E}[G_t|S_t = s]$$

- **Bellman Equation for MRP:** Decomposes value into immediate reward + discounted value of next state:

$$V(s) = R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s)V(s')$$

3.4 Markov Decision Processes (MDP)

An MDP adds **Agency** (Actions). Defined by tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$. The transition now depends on action: $P(s'|s, a)$. The reward now depends on action: $R(s, a)$.

3.5 State-Value and Action-Value Functions

In an MDP, value depends on the policy π used.

- **State-Value Function** $V^\pi(s)$: How good is it to be in state s following policy π ?

$$V^\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$

- **Action-Value Function** $Q^\pi(s, a)$: How good is it to take action a in state s , and *then* follow policy π ?

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

4 K-Armed Bandit

4.1 The K-Armed Bandit Problem

This is a simplified RL setting (a "one-state" MDP).

- **Scenario:** You face a slot machine (bandit) with K arms. Pulling an arm gives a reward drawn from a probability distribution unknown to you.
- **Goal:** Maximize total reward over a fixed number of time steps T .
- **True Action-Value** ($q_*(a)$): The actual mean reward of arm a .
- **Estimated Action-Value** ($Q_t(a)$): Our current guess of how good arm a is, usually calculated by averaging the rewards received so far from that arm.
- **The Dilemma (Exploration vs. Exploitation):**
 - **Exploit:** Pull the arm with the highest current $Q_t(a)$ to maximize immediate reward.
 - **Explore:** Pull a random/less-tried arm to gather data. The "best" arm might be one we haven't tried enough yet.
- **ϵ -Greedy Strategy:** A simple algorithm to balance this.
 - With probability $1 - \epsilon$: Choose best arm (Exploit).
 - With probability ϵ : Choose random arm (Explore).
- **Regret** (L_T): The difference between the reward we could have gotten (if we knew the best arm from the start) and what we actually got.

$$L_T = \sum_{t=1}^T (q_*(a^*) - q_*(a_t))$$

Minimizing regret is the primary theoretical goal in Bandit problems.