XGBoost: Theory and Practice

September 10, 2025

1 Introduction

XGBoost is an efficient and scalable implementation of gradient boosted decision trees (GBDTs). It improves training speed, regularization, and accuracy through second-order optimization, tree sparsity-aware split finding, shrinkage, and subsampling.

2 Theory and Formulas

Gradient boosting fits an additive model $F_M(\mathbf{x}) = \sum_{m=1}^{M} f_m(\mathbf{x})$ of shallow trees by stage-wise optimization. XGBoost minimizes a regularized objective

$$\mathcal{L} = \sum_{i=1}^{n} \ell\big(y_i, \hat{y}_i\big) + \sum_{m=1}^{M} \Omega(f_m), \quad \Omega(f) = \gamma T + \tfrac{1}{2}\lambda \|w\|^2, \tag{1}$$

where T is the number of leaves and w are leaf scores. Using a second-order Taylor expansion of the loss around current predictions yields per-node sums of gradients g_i and Hessians h_i; the split gain for left/right partitions L, R is

$$\text{Gain} = \tfrac{1}{2}\left(\frac{\big(\sum_{i\in L} g_i\big)^2}{\sum_{i\in L} h_i + \lambda} + \frac{\big(\sum_{i\in R} g_i\big)^2}{\sum_{i\in R} h_i + \lambda} - \frac{\big(\sum_{i\in L\cup R} g_i\big)^2}{\sum_{i\in L\cup R} h_i + \lambda}\right) - \gamma. \tag{2}$$

Regularization via λ, γ, shrinkage (learning rate), column/row subsampling, and maximum depth/leaf constraints control complexity and reduce overfitting.

3 Applications and Tips

- **Scaling and types:** handles numeric and one-hot encoded categorical features; no scaling required for trees.
- **Key hyperparameters:** `n_estimators`, `max_depth`, `learning_rate`, `subsample`, `colsample_bytree`, `reg_alpha/reg_lambda`.
- **Early stopping:** use validation with `eval_set` and `early_stopping_rounds`.
- **Imbalance:** set `scale_pos_weight` or use stratified sampling.
- **Interpretation:** start with built-in importances; prefer permutation or SHAP for robust insights.

4 Python Practice

Run the script in this chapter directory to generate figures into `figures/`.

Listing 1: Generate XGBoost figures

```
python gen_xgboost_figures.py
```

Listing 2: gen_xgboost_figures.py

```
"""
Figure generator for the XGBoost chapter.

Generates illustrative figures and saves them into the chapter's 'figures/'
folder next to this script, regardless of current working directory.

Requirements:
- Python 3.8+
- numpy, matplotlib, scikit-learn
- xgboost (optional; falls back to scikit-learn GradientBoosting if missing)

Install (if needed):
  pip install numpy matplotlib scikit-learn xgboost

This script avoids newer or experimental APIs for broader compatibility.
"""
from __future__ import annotations

import os
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

try:
    import xgboost as xgb  # type: ignore
    HAS_XGB = True
except Exception:
    xgb = None
    HAS_XGB = False

from sklearn.datasets import make_moons, make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import log_loss

try:
    from sklearn.ensemble import GradientBoostingClassifier
except Exception:
    GradientBoostingClassifier = None  # type: ignore


def _ensure_figures_dir(path: str | None = None) -> str:
    """Create figures directory under this chapter regardless of CWD."""
    if path is None:
        base = os.path.dirname(os.path.abspath(__file__))
        path = os.path.join(base, "figures")
```

```
    os.makedirs(path, exist_ok=True)
    return path


def _plot_decision_boundary(ax, clf, X, y, title: str):
    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
    xx, yy = np.meshgrid(
        np.linspace(x_min, x_max, 400), np.linspace(y_min, y_max, 400)
    )
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
    cmap_light = ListedColormap(["#FFEEEE", "#EEEEFF"])
    cmap_bold = ListedColormap(["#E74C3C", "#3498DB"])
    ax.contourf(xx, yy, Z, cmap=cmap_light, alpha=0.8, levels=np.unique(Z).size)
    ax.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold, edgecolors="k", s=20)
    ax.set_title(title)
    ax.set_xlabel("Feature 1")
    ax.set_ylabel("Feature 2")


def _xgb_classifier(**kwargs):
    if HAS_XGB:
        params = dict(
            n_estimators=200,
            max_depth=3,
            learning_rate=0.1,
            subsample=1.0,
            colsample_bytree=1.0,
            objective="binary:logistic",
            tree_method="hist",
            random_state=0,
            n_jobs=0,
        )
        params.update(kwargs)
        return xgb.XGBClassifier(**params)
    else:
        if GradientBoostingClassifier is None:
            raise RuntimeError("Neither xgboost nor GradientBoostingClassifier available.")
        params = dict(
            n_estimators=kwargs.get("n_estimators", 200),
            max_depth=kwargs.get("max_depth", 3),
            learning_rate=kwargs.get("learning_rate", 0.1),
            random_state=0,
        )
        return GradientBoostingClassifier(**params)


def fig_xgb_decision_boundary_2class(out_dir: str) -> str:
    np.random.seed(0)
    X, y = make_moons(n_samples=500, noise=0.3, random_state=0)
    clf = _xgb_classifier()
    clf.fit(X, y)
```

```python

    fig, ax = plt.subplots(figsize=(5.5, 4.5), dpi=150)
    title = "XGBoost boundary (max_depth=3, lr=0.1)" if HAS_XGB else "GBDT boundary (fallback)"
    _plot_decision_boundary(ax, clf, X, y, title)
    out_path = os.path.join(out_dir, "xgb_decision_boundary_2class.png")
    fig.tight_layout()
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_xgb_learning_rate_compare(out_dir: str) -> str:
    np.random.seed(1)
    X, y = make_moons(n_samples=550, noise=0.3, random_state=1)
    models = [
        (_xgb_classifier(learning_rate=0.05, n_estimators=400), "learning_rate=0.05, n_estimators=400"),
        (_xgb_classifier(learning_rate=0.3, n_estimators=150), "learning_rate=0.3, n_estimators=150"),
    ]
    fig, axes = plt.subplots(1, 2, figsize=(9.5, 4.2), dpi=150, sharex=True, sharey=True)
    for ax, (m, title) in zip(axes, models):
        m.fit(X, y)
        label = ("XGBoost: " if HAS_XGB else "GBDT: ") + title
        _plot_decision_boundary(ax, m, X, y, label)
    fig.suptitle("Effect of learning_rate with trees budget")
    out_path = os.path.join(out_dir, "xgb_learning_rate_compare.png")
    fig.tight_layout(rect=[0, 0.03, 1, 0.95])
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_xgb_max_depth_compare(out_dir: str) -> str:
    np.random.seed(2)
    X, y = make_moons(n_samples=600, noise=0.32, random_state=2)
    models = [
        (_xgb_classifier(max_depth=2, n_estimators=250), "max_depth=2"),
        (_xgb_classifier(max_depth=4, n_estimators=250), "max_depth=4"),
        (_xgb_classifier(max_depth=8, n_estimators=250), "max_depth=8"),
    ]
    fig, axes = plt.subplots(1, 3, figsize=(12.5, 4.2), dpi=150, sharex=True, sharey=True)
    for ax, (m, title) in zip(axes, models):
        m.fit(X, y)
        label = ("XGBoost: " if HAS_XGB else "GBDT: ") + title
        _plot_decision_boundary(ax, m, X, y, label)
    fig.suptitle("Effect of max_depth")
    out_path = os.path.join(out_dir, "xgb_max_depth_compare.png")
    fig.tight_layout(rect=[0, 0.03, 1, 0.95])
    fig.savefig(out_path)
    plt.close(fig)
```

```python
    return out_path


def fig_xgb_feature_importances(out_dir: str) -> str:
    X, y = make_classification(
        n_samples=800,
        n_features=10,
        n_informative=4,
        n_redundant=3,
        n_repeated=0,
        random_state=7,
        shuffle=True,
    )
    clf = _xgb_classifier(n_estimators=300, max_depth=4, learning_rate=0.1)
    clf.fit(X, y)
    importances = getattr(clf, "feature_importances_", None)
    if importances is None:
        # Fallback: uniform zeros to avoid crash
        importances = np.zeros(X.shape[1], dtype=float)

    fig, ax = plt.subplots(figsize=(7.0, 4.0), dpi=160)
    idx = np.arange(importances.size)
    ax.bar(idx, importances, color="#F39C12")
    ax.set_xticks(idx)
    ax.set_xticklabels([f"f{i}" for i in idx])
    ax.set_ylabel("importance")
    title = "XGBoost feature importances" if HAS_XGB else "GBDT feature
        importances"
    ax.set_title(title)
    ax.set_ylim(0, max(0.25, float(importances.max()) + 0.05))
    for i, v in enumerate(importances):
        ax.text(i, v + 0.01, f"{v:.2f}", ha="center", va="bottom", fontsize=8)
    out_path = os.path.join(out_dir, "xgb_feature_importances.png")
    fig.tight_layout()
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_xgb_eval_logloss_curve(out_dir: str) -> str:
    np.random.seed(3)
    X, y = make_classification(
        n_samples=1200,
        n_features=15,
        n_informative=5,
        n_redundant=5,
        random_state=3,
    )
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3,
        random_state=3)

    if HAS_XGB:
        clf = _xgb_classifier(n_estimators=300, learning_rate=0.1, max_depth
            =4)
```

```python
        clf.fit(X_train, y_train, eval_set=[(X_train, y_train), (X_val, y_val)], eval_metric="logloss", verbose=False)
        res = clf.evals_result()
        tr = np.array(res.get("validation_0", {}).get("logloss", []), dtype=float)
        va = np.array(res.get("validation_1", {}).get("logloss", []), dtype=float)
    else:
        # Fallback using staged decision on GradientBoosting
        clf = _xgb_classifier(n_estimators=300, learning_rate=0.1, max_depth=3)
        clf.fit(X_train, y_train)
        tr_list, va_list = [], []
        # GradientBoostingClassifier provides staged_predict_proba
        if hasattr(clf, "staged_predict_proba"):
            for y_tr_prob, y_va_prob in zip(clf.staged_predict_proba(X_train), clf.staged_predict_proba(X_val)):
                tr_list.append(log_loss(y_train, y_tr_prob))
                va_list.append(log_loss(y_val, y_va_prob))
        else:
            # Last resort: single-point curves
            y_tr_prob = clf.predict_proba(X_train)
            y_va_prob = clf.predict_proba(X_val)
            tr_list = [log_loss(y_train, y_tr_prob)]
            va_list = [log_loss(y_val, y_va_prob)]
        tr, va = np.array(tr_list), np.array(va_list)

    fig, ax = plt.subplots(figsize=(6.8, 4.2), dpi=160)
    ax.plot(np.arange(1, len(tr) + 1), tr, label="train logloss", color="#2ECC71")
    ax.plot(np.arange(1, len(va) + 1), va, label="valid logloss", color="#E74C3C")
    ax.set_xlabel("n_trees")
    ax.set_ylabel("logloss")
    ax.set_title("Evaluation curve (logloss vs trees)")
    ax.legend()
    ax.grid(True, linestyle=":", alpha=0.4)
    out_path = os.path.join(out_dir, "xgb_eval_logloss_curve.png")
    fig.tight_layout()
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def main():
    out_dir = _ensure_figures_dir(None)
    generators = [
        fig_xgb_decision_boundary_2class,
        fig_xgb_learning_rate_compare,
        fig_xgb_max_depth_compare,
        fig_xgb_feature_importances,
        fig_xgb_eval_logloss_curve,
    ]
    print("Generating figures into:", os.path.abspath(out_dir))
```

```
    if not HAS_XGB:
        print("xgboost not found; falling back to GradientBoostingClassifier
            where possible.")
    for gen in generators:
        try:
            p = gen(out_dir)
            print("Saved:", p)
        except Exception as e:
            print("Failed generating", gen.__name__, ":", e)


if __name__ == "__main__":
    main()
```

5 Result

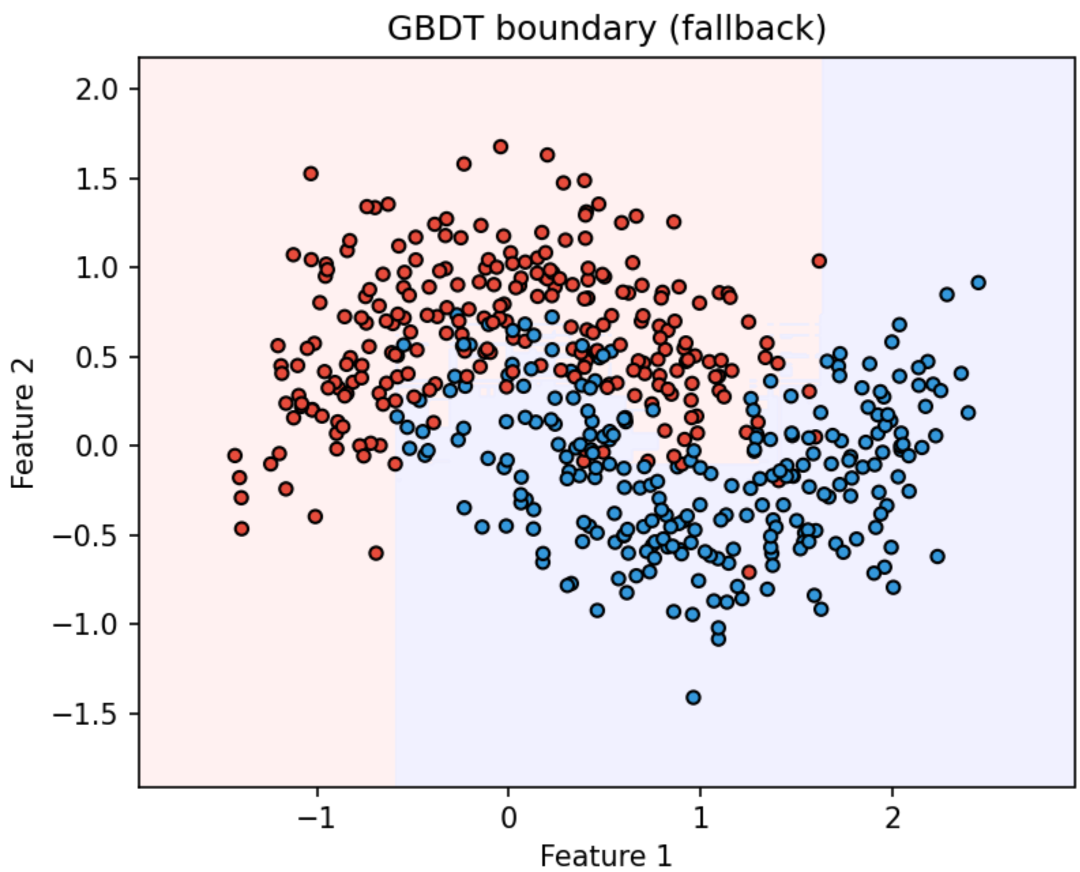

Figure 1: XGBoost decision boundary on a 2-class dataset.

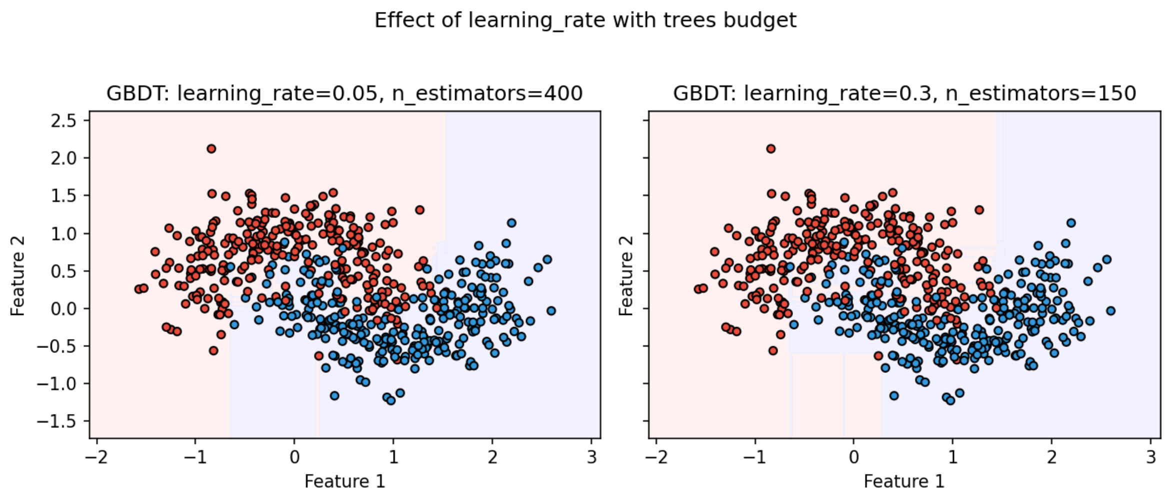

Figure 2: Learning rate effect with a budget of trees.

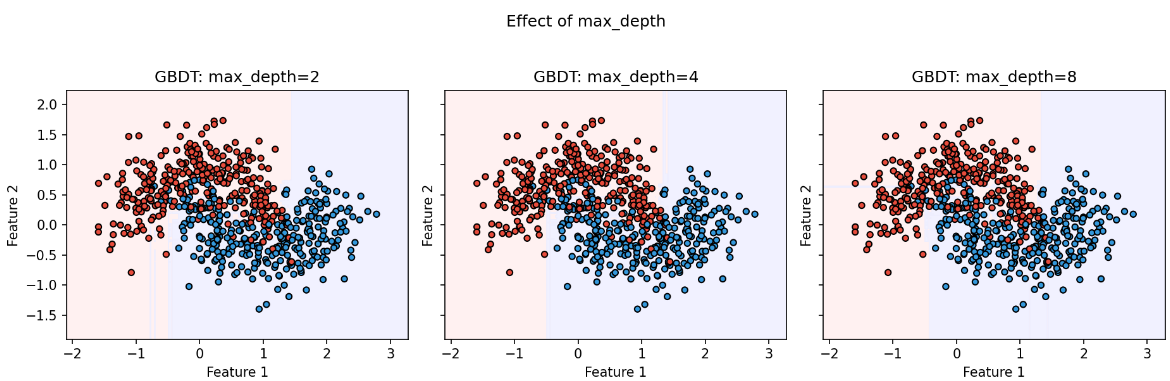

Figure 3: Decision boundaries under different max_depth.

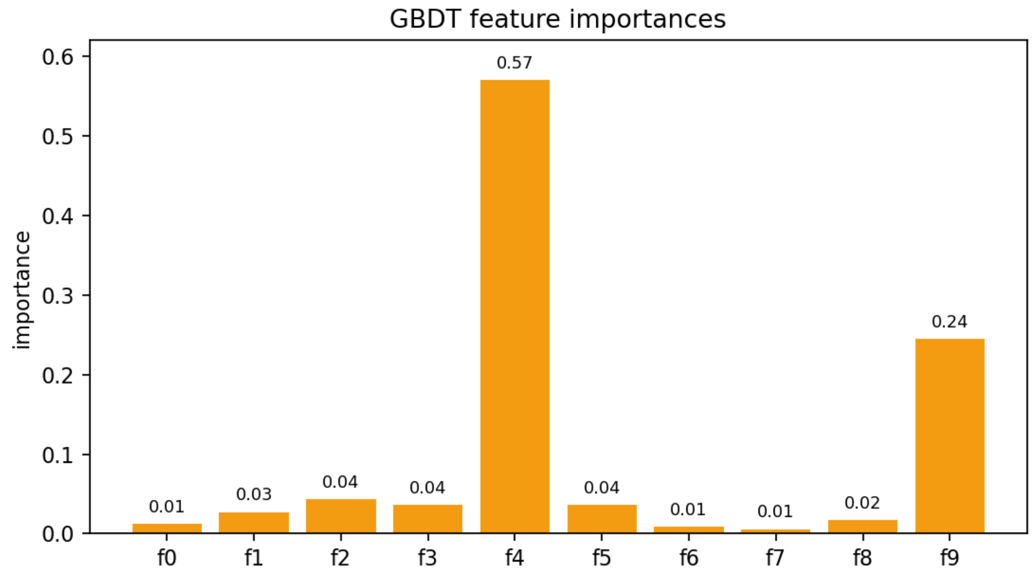

Figure 4: Feature importances from XGBoost.

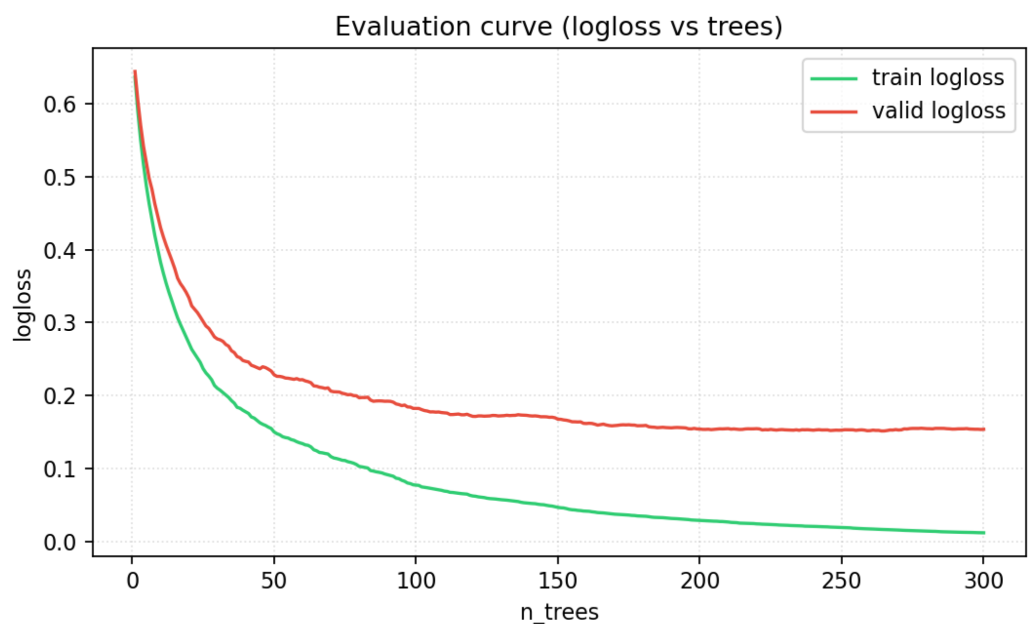

Figure 5: Training/validation logloss vs number of trees.

6 Summary

XGBoost combines efficient tree boosting with strong regularization and advanced split finding. With careful tuning of depth, learning rate, and sampling, it delivers state-of-the-art performance on many tabular tasks.