DATA SCIENCE & ML

Gradient Boosting with XGBoost and LightGBM

A practitioner's guide to winning with tabular data

Houssam Kodad

PDF · DATAFORGE BOOKS

Questions about this sample or the full edition: **support@dataforgebooks.com**

CONTENTS

Table of Contents

CHAPTER 01

How Boosting Actually Works

On tabular data — the rows and columns that run most businesses — gradient-boosted trees remain stubbornly hard to beat. Neural networks dominate images and text, but for credit risk, churn, demand, pricing and the thousand other tabular problems that actually pay the bills, a well-tuned boosting model is usually the strongest baseline and frequently the final answer. This book is about using that tool with real understanding rather than by ritual.

This chapter explains how boosting actually works, because the mechanism is what turns parameter tuning from superstition into judgement. We cover additive modelling on residuals, the role of the gradient and the loss function, and why deliberately weak, shallow trees combine into such a strong predictor. We also meet XGBoost and LightGBM and the design choices that make them fast.

Understanding the engine pays off immediately. Once you grasp why more trees with a lower learning rate generally help, why depth trades capacity against overfitting, and what early stopping is really doing, the otherwise bewildering list of hyperparameters resolves into a small number of levers you can reason about. That reasoning, not a copied configuration, is what makes the difference in practice.

Additive Models and Residuals

Boosting builds its model one small tree at a time, and the core idea is beautifully simple. The first tree makes a rough prediction. The second tree is trained to predict the errors the first one made. The third corrects what remains, and so on. The final prediction is the sum of all these small corrections — an additive model assembled greedily, each step cleaning up after the last.

This sequential, error-correcting structure is both why boosting is so effective and why it overfits if left unchecked. Each tree chips away at the residual error, so with enough trees and enough depth the model can fit the training data almost perfectly, including its noise. Nearly every tuning decision in this book is, in some form, about letting the model fit the signal while stopping it from memorising the noise.

Gradients and Loss Functions

The "gradient" in gradient boosting is precise and worth understanding. Rather than fitting each new tree to the raw residuals, you fit it to the negative gradient of a chosen loss function. For squared-error loss that gradient happens to be the residual, which is why the simple intuition holds —

but the framework generalises to any differentiable loss, and that generality is its real power.

Swap the loss and you change what the model optimises without changing the algorithm. Log loss gives you well-calibrated classification. Quantile loss produces prediction intervals. A custom asymmetric loss encodes a business cost where over- and under-prediction differ. Recognising that boosting is gradient descent performed in the space of functions is the conceptual key that lets you bend it to the actual problem in front of you.

```
import xgboost as xgb

model = xgb.XGBRegressor(
    n_estimators=600,       # many small corrections...
    learning_rate=0.03,     # ...each shrunk to avoid overfitting
    max_depth=4,            # shallow trees = deliberately weak learners
    subsample=0.8,          # row sampling adds regularisation
    colsample_bytree=0.8,   # column sampling decorrelates trees
)
model.fit(X_train, y_train, eval_set=[(X_val, y_val)], early_stopping_rounds=50)
```

Trees as Weak Learners

A single shallow decision tree is a weak learner: on its own it predicts only a little better than chance. That weakness is the point. Because each tree captures just a small piece of the structure, the ensemble can combine hundreds of them to model complex interactions without any single tree overfitting badly, and the gradual accumulation keeps the whole model under control.

This is why depth is such a consequential setting. Shallow trees capture low-order interactions and lean on the ensemble for complexity; deep trees capture rich interactions individually but overfit far more readily. The interplay between tree depth, the number of trees, and the learning rate is the central tuning trade-off of the entire field, and we return to it in chapter after chapter.

XGBoost and LightGBM Under the Hood

XGBoost and LightGBM implement the same fundamental algorithm but make different engineering choices, and knowing them helps you pick and tune. Both bin continuous features into histograms to find splits quickly rather than scanning every value. Where they differ most is tree growth: XGBoost grows level by level by default, while LightGBM grows leaf by leaf, expanding wherever the loss drops most.

That single difference has practical consequences. LightGBM's leaf-wise growth is often faster and more accurate but more prone to overfitting on small data, which is why it controls complexity through a maximum number of leaves rather than depth. Understanding these internals turns the two libraries from interchangeable black boxes into tools you can choose between deliberately, and we compare them honestly throughout.

The Parameters That Matter

The parameter lists for these libraries are intimidatingly long, but a handful do almost all the work, and the rest are refinements. The learning rate and the number of trees together set how finely and how far the model fits. Tree depth or leaf count sets the complexity of each step. Subsampling of rows and columns adds regularisation and decorrelates the trees. Master these and you can tune effectively before touching anything else.

There is a reliable relationship worth memorising: a lower learning rate with more trees almost always generalises better than the reverse, at the cost of training time. So the standard recipe is to fix a small learning rate, use early stopping to choose the number of trees automatically, and then tune complexity and sampling. We make this concrete with worked examples rather than leaving you to guess.

Validation and Early Stopping

Because boosting will happily overfit given enough trees, you need a principled way to know when to stop adding them, and early stopping is it. You hold out a validation set, watch the validation metric as trees are added, and stop when it stops improving. This both prevents overfitting and removes one parameter from your search by choosing the number of trees for you.

The catch is that early stopping is only as trustworthy as the validation set behind it. A leaky split — where information about the validation rows has crept into training — produces an optimistic stopping point and a model that disappoints in production. We treat validation design with the seriousness it deserves, because in boosting the validation strategy is not a detail; it is what stands between you and quiet overfitting.

Interpreting Boosted Models

A common objection to boosted trees is that they are black boxes, but modern tools make them surprisingly interpretable. Built-in feature importances give a rough ranking, though they can mislead. Permutation importance is more trustworthy, measuring how much performance drops when

a feature is shuffled. And SHAP values attribute each individual prediction to its features, turning the ensemble into something you can explain row by row.

Interpretation is not just for satisfying stakeholders; it is a debugging tool. A feature with implausibly high importance often signals leakage. A SHAP plot that contradicts domain knowledge points to a data problem. We use these techniques throughout to understand, trust, and improve models — because a model you can explain is one you can defend and one you can fix.

This is a free sample

You've reached the end of the sample chapter.

Get the complete book — every chapter, fully worked — at **dataforgebooks.com**.

FULL EDITION · 164 PAGES · PDF

Read the full title at **dataforgebooks.com**

Questions? support@dataforgebooks.com