Boosting Algorithms

for Regression Analysis

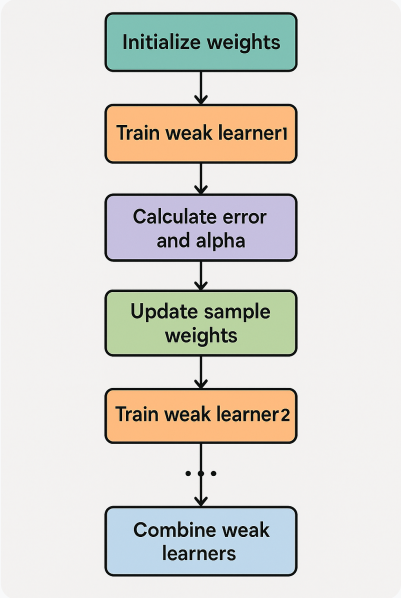
AdaBoost • XGBoost • LightGBM

What is Boosting?

- **Ensemble technique** combining multiple weak learners sequentially
- Each learner **corrects errors** of the previous models
- Converts **weak learners into a strong learner**
- **Reduces bias and variance** to improve performance
- Works by focusing on **difficult samples** in each iteration

Key Concept: Boosting algorithms build models sequentially, with each new model learning from the mistakes of previous ones, gradually improving prediction accuracy.

AdaBoost for Regression



AdaBoost Sequential Learning Process

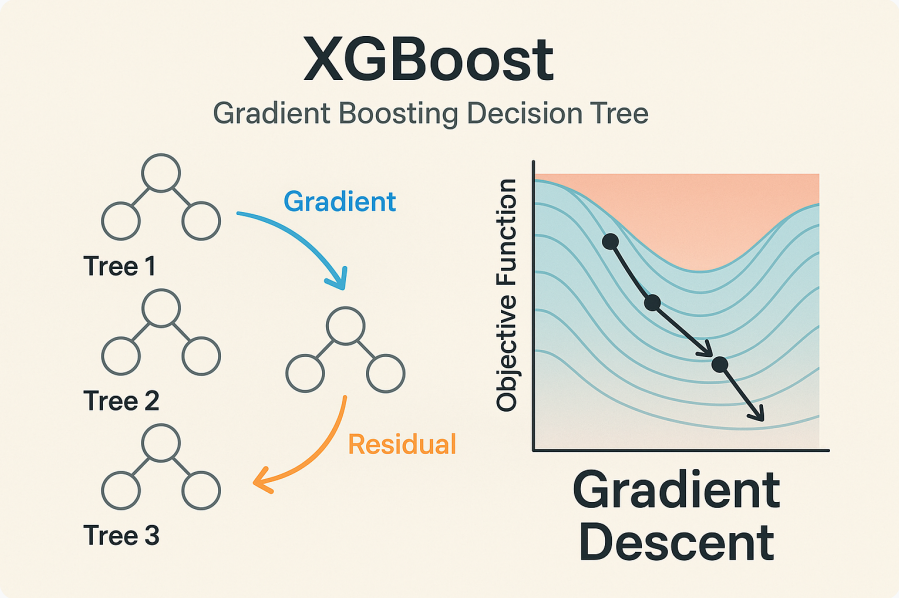
Key Features:

- Originally designed for classification, adapted for regression (AdaBoost.R)
- **Assigns weights** to samples and fits weak regressors iteratively
- **Focuses on difficult samples** with larger residuals
- Base learners are often **shallow decision trees**
- Simple and interpretable but may underperform on complex data

Code Example:

```
from sklearn.ensemble import AdaBoostRegressor from sklearn.tree import DecisionTreeRegressor # AdaBoost with Decision Tree base estimator ada_regressor = AdaBoostRegressor( estimator=DecisionTreeRegressor(max_depth=1), n_estimators=50, learning_rate=1.0 ) ada_regressor.fit(X_train, y_train)
```

XGBoost for Regression



XGBoost Gradient Boosting Structure

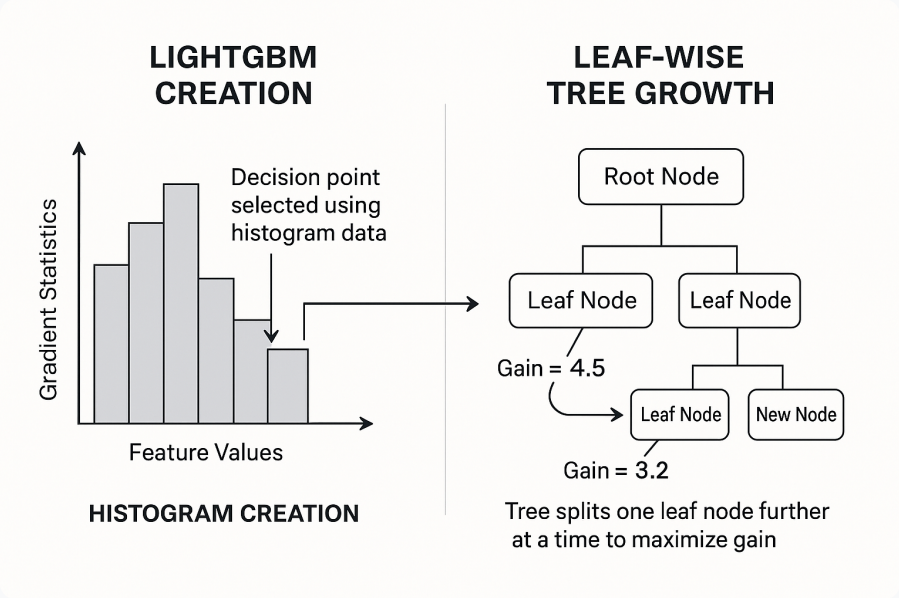
Key Features:

- **Extreme Gradient Boosting** (XGBoost)
- **Optimized gradient boosting** algorithm
- Handles **missing values, regularization, parallel processing**
- Uses **second-order gradient (Hessian)** information
- Highly efficient and accurate on large structured datasets
- Supports **early stopping** to prevent overfitting

Code Example:

```
import xgboost as xgb # XGBoost Regressor xgb_regressor =
xgb.XGBRegressor( objective='reg:squarederror',
n_estimators=100, learning_rate=0.1, max_depth=6 )
xgb_regressor.fit(X_train, y_train)
```

LightGBM for Regression



LightGBM Histogram-based Algorithm

Key Features:

- **Light Gradient Boosting Machine** (LightGBM)
- Uses **histogram-based algorithms** for speed
- **Gradient-based One-Side Sampling (GOSS)** to focus on hard examples
- **Efficient with large datasets** and lower memory usage
- Supports **categorical features natively**
- Also supports **early stopping and parallel learning**

Code Example:

```
import lightgbm as lgb # LightGBM Regressor lgb_regressor =
lgb.LGBMRegressor( boosting_type='gbdt', n_estimators=100,
learning_rate=0.1, objective='regression' )
lgb_regressor.fit(X_train, y_train)
```

Comparison Summary

Feature	AdaBoost	XGBoost	LightGBM
Base Learners	Shallow trees	Decision trees (CART)	Decision trees (GOSS)
Speed	Moderate	Fast	Very fast
Accuracy	Good (simple problems)	High (complex data)	High (large data)
Handling Missing	No	Yes	Yes
Memory Usage	Moderate	Moderate	Low
Regularization	Limited	Advanced (L1/L2)	Built-in
Parallel Processing	No	Yes	Yes
Use Case	Basic regression	Challenging regression	Large-scale regression

When to Use Each Algorithm

AdaBoost

- Small to medium datasets
- Simple regression problems
- When interpretability is important
- Limited computational resources

XGBoost

- Medium to large datasets
- Complex regression problems
- When accuracy is the primary goal
- Structured/ tabular data

LightGBM

- Very large datasets
- When speed is critical
- Limited memory resources
- Categorical features present

Key Takeaways

1. **Boosting** sequentially improves weak learners to create strong predictors
2. **AdaBoost** is simple and interpretable for basic regression tasks
3. **XGBoost** offers the best balance of accuracy and features for most problems
4. **LightGBM** excels with large datasets and speed requirements
5. **Choose based on** dataset size, computational constraints, and accuracy needs
6. **Hyperparameter tuning** and **early stopping** are crucial for optimal performance