**Сравнение XGBoost, LightGBM и CatBoost: подходы к построению моделей и рекомендации по применению**

**1. Введение**

Градиентный бустинг (Gradient Boosting) — это мощная техника ансамблевого обучения, при которой слабые модели (обычно деревья решений) объединяются в сильную модель. Каждое последующее дерево обучается на ошибках предыдущих. На сегодняшний день наиболее популярные библиотеки для реализации градиентного бустинга: **XGBoost**, **LightGBM** и **CatBoost**. Все они активно применяются в задачах машинного обучения, особенно при работе с табличными данными.

## **2. XGBoost (Extreme Gradient Boosting)**

XGBoost реализует классический градиентный бустинг, где каждая новая модель обучается на ошибках предыдущей. Основная идея — минимизация дифференцируемой функции потерь с помощью градиентного спуска.

XGBoost использует второй порядок производной (в отличие от некоторых других реализаций, использующих только градиент), что повышает точность выбора следующего шага оптимизации. Дополнительно, он включает L1 и L2 регуляризацию, что помогает избежать переобучения.

* Использует жадный алгоритм построения деревьев (анализ всех возможных сплитов).
* Применяет буферизацию и предварительное вычисление градиентов.
* Поддерживает параллельную обработку на уровне признаков при построении дерева.

Построение деревьев:

* Строятся по стратегии "level-wise" — дерево растет по уровням.
* Все листья одного уровня обрабатываются одновременно.
* Такая стратегия приводит к более сбалансированным деревьям и помогает избежать переобучения, но может быть медленнее.

### Когда использовать

* Необходимо высокое качество предсказаний.
* Требуется контроль над переобучением с помощью регуляризации.
* Работа ведётся с разреженными или большими наборами данных.

Пример построения:

import xgboost as xgb

from sklearn.datasets import load\_boston

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

# Загрузка и разделение данных

X, y = load\_boston(return\_X\_y=True)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Преобразование в DMatrix

dtrain = xgb.DMatrix(X\_train, label=y\_train)

dtest = xgb.DMatrix(X\_test, label=y\_test)

# Параметры модели

params = {'objective': 'reg:squarederror', 'max\_depth': 3, 'eta': 0.1}

model = xgb.train(params, dtrain, num\_boost\_round=100)

# Предсказания

preds = model.predict(dtest)

print("MSE:", mean\_squared\_error(y\_test, preds))

**Что делает код:**

1. Загружает датасет о ценах на жильё в Бостоне.
2. Делит данные на обучающую и тестовую выборки.
3. Преобразует данные в формат DMatrix, используемый XGBoost для оптимизации.
4. Определяет параметры модели и обучает её.
5. Строит прогнозы и рассчитывает метрику MSE (среднеквадратичную ошибку).

Практический пример, где можно использовать эту библиотеку: https://www.kaggle.com/c/higgs-boson

## **3. LightGBM (Light Gradient Boosting Machine)**

LightGBM также реализует градиентный бустинг, но с двумя основными оптимизациями:

1. **Gradient-based One-Side Sampling (GOSS)** — выборка по градиенту. Вместо случайного отбора объектов для построения дерева, GOSS сохраняет объекты с наибольшими ошибками (высоким градиентом), что помогает сосредоточиться на трудных примерах.
2. **Exclusive Feature Bundling (EFB)** — объединение разреженных и взаимно исключающих признаков в один, чтобы уменьшить размерность.

Эти методы позволяют ускорить обучение без значительной потери точности.

При построении деревьев использует стратегию "leaf-wise":

* + Вместо роста дерева по уровням выбирается лист с наибольшим потенциалом прироста информации и углубляется.
  + Это дает более точные деревья, но они могут быть несбалансированными и склонными к переобучению.

### Когда использовать

* Работа ведётся с большими наборами данных.
* Требуется высокая скорость обучения и предсказания.
* Необходимо эффективное использование памяти.

Пример построения:

import lightgbm as lgb

from sklearn.datasets import load\_boston

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

X, y = load\_boston(return\_X\_y=True)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

train\_data = lgb.Dataset(X\_train, label=y\_train)

params = {'objective': 'regression', 'metric': 'mse', 'verbose': -1}

model = lgb.train(params, train\_data, num\_boost\_round=100)

preds = model.predict(X\_test)

print("MSE:", mean\_squared\_error(y\_test, preds))

**Что делает код:**

1. Загружает и разбивает датасет на обучающую и тестовую выборки.
2. Формирует объект Dataset для обучения модели.
3. Определяет параметры и обучает модель.
4. Делает предсказания на тестовой выборке и оценивает ошибку с помощью MSE.

Практический пример, где можно использовать эту библиотеку: ttps://www.microsoft.com/en-us/research/project/mslr/

## **4. CatBoost**

CatBoost создавался с прицелом на работу с категориальными данными. Он использует **Ordered Boosting** — модифицированный градиентный бустинг, который устраняет смещение, возникающее при обучении на одной и той же выборке, которую затем используют для оценки.

Особенности:

* Использует статистику по истории (например, таргет-энкодинг) с учетом случайного порядка для кодирования категориальных признаков.
* Благодаря подходу Ordered Target Statistics, избежано смещения, присущего обычному target encoding.
* Внутри применяется симметричная структура дерева (обрезанные деревья), при которой на каждом уровне дерева проверяются те же признаки и пороги для всех узлов.

**Построение деревьев:**

* Строит **симметричные деревья** — все листья на одном уровне имеют одинаковую структуру.
* Это делает модель быстрее на стадии предсказания и способствует лучшей обобщающей способности.

### Когда использовать

* Данные содержат большое количество категориальных признаков.
* Требуется минимальная предварительная обработка.
* Необходимо высокое качество предсказаний с минимальной настройкой модели.

Пример построения:

from catboost import CatBoostRegressor

from sklearn.datasets import load\_boston

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

X, y = load\_boston(return\_X\_y=True)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = CatBoostRegressor(verbose=0)

model.fit(X\_train, y\_train)

preds = model.predict(X\_test)

print("MSE:", mean\_squared\_error(y\_test, preds))

**Что делает код:**

1. Загружает данные и делит их на обучающую и тестовую выборки.
2. Создаёт и обучает модель CatBoostRegressor с настройками по умолчанию.
3. Делает предсказания на тестовой выборке.
4. Рассчитывает среднеквадратичную ошибку между реальными и предсказанными значениями.

Практически пример, где можно применить данную библиотеку: https://www.kaggle.com/c/amazon-employee-access-challenge