

XGBoost

mlumina.xgboost

Самый зрелый и наиболее близкий к документации модуль. Обучение и подбор гиперпараметров XGBoost, локально и через `xgboost_ray`, с опциональным логированием в MLflow.

Файлы: `src/mlumina/xgboost/{load_dataset,train_xgboost,tune_xgboost}.py`. Публичный API (`src/mlumina/xgboost/__init__.py`): `xgboost_load_dataset`, `XGBoostModel`, `XGBoostTuningModel`.

???????? ???? ??

```
from mlumina.xgboost import xgboost_load_dataset

# Для распределённого обучения (RayDMatrix)
dtrain, deval = xgboost_load_dataset(
    train_path="data/train.csv",
    eval_path="data/eval.csv",
    target="elapsed_time_sec",
    categorical_cols=["vcd", "nom_point"],
    ray_data=True,
)

# Для локального обучения (xgb.DMatrix)
dtrain, deval = xgboost_load_dataset(..., ray_data=False)
```

`xgboost_load_dataset(train_path, eval_path, target="elapsed_time_sec", categorical_cols=None, ray_data=True)`:

- читает CSV через `pandas`;
- колонки из `categorical_cols` кодирует `TfidfVectorizer(max_features=1000)` (несмотря на название "категориальные" — это TF-IDF, а не one-hot/ordinal encoding);
- остальные колонки передаются как есть (`remainder="passthrough"`);
- целевая колонка (`target`) извлекается отдельно.

⚠ Значения по умолчанию (`target="elapsed_time_sec"`, `categorical_cols=["vcd", "nom_point"]`) — это параметры конкретного датасета-примера, а не универсальный дефолт. Для своих данных их всегда нужно передавать явно.

????????

????????

```
from mlumina.xgboost import XGBoostModel

booster, metrics = XGBoostModel.train(
    dtrain=dtrain,
    deval=deval,
    objective="reg:squarederror",
    tree_method="hist",
    subsample=0.8,
    colsample_bytree=0.8,
    num_boost_round=500,
    error_func="rmse",
)

print(f"RMSE: {metrics['rmse']}")
```

`XGBoostModel.train(...)` — статический метод, обёртка над `xgboost.train`. Возвращает `(booster, {error_func: значение_на_последней_итерации})`.

???????????????? Ray

```
import ray
from xgboost_ray import RayParams
from mlumina.xgboost import XGBoostModel

ray.init(address="ray://localhost:10001")

ray_params = RayParams(num_actors=4, cpus_per_actor=2, gpus_per_actor=0)

model = XGBoostModel(ray_params=ray_params, mlflow_uri="http://localhost:5001")
booster, metrics = model.train_ray(
    dtrain=dtrain, # должен быть RayDMatrix (ray_data=True при загрузке)
```

```

deval=deval,
objective="reg:squarederror",
tree_method="hist",
subsample=0.8,
colsample_bytree=0.8,
num_boost_round=500,
error_func="rmse",
)

```

`train_ray` — метод экземпляра, использует `xgboost_ray.train` и `self._ray_params`. Требуется, чтобы `ray.init()` был вызван заранее. `mlflow_uri`, переданный в конструктор, сам по себе в `XGBoostModel.train_ray/train` не используется — эти методы не логируют в MLflow, только возвращают `(booster, metrics)` вызывающему коду. Логирование в MLflow (включая сохранение модели) реализовано в `XGBoostTuningModel.run_optimization/run_optimization_ray`, см. ниже.

?????? ?????????????????? (XGBoostTuningModel)

Наследует `XGBoostModel`.

```

from mlumina.xgboost import XGBoostTuningModel, xgboost_load_dataset

dtrain, deval = xgboost_load_dataset(..., ray_data=False)

tuner = XGBoostTuningModel(
    name="my_experiment",
    mlflow_uri="http://localhost:5001",
    objective="reg:squarederror",
    eval_metric="rmse",
    learning_rate=(0.01, 0.3),
    subsample=(0.6, 1.0),
    colsample_bytree=(0.6, 1.0),
    lambda_=(1e-8, 10.0),
    mode="min",
    tree_method="hist",
    num_samples=100,
    num_boost_round=500,
    max_depth=10,
)

```

)

```
results = tuner.run_optimization(dtrain, deval)          # локальная оптимизация Optuna
best_params, best_score = tuner.get_best_result()
```

- `run_optimization(dtrain, deval=None)` — локальный поиск через `optuna`. Если `deval` не передан — использует `xgb.cv`; иначе `train/eval split` с ранней остановкой. При `mlflow_uri` — вызывает `mlflow.set_tracking_uri(mlflow_uri)`, включает автологирование (`mlflow.xgboost.autolog`) и в конце логирует финальную модель (`mlflow.xgboost.log_model`), переобученную на лучших параметрах. Возвращает `{"best_params": ..., "best_score": ..., "study": optuna.Study}`.
- `run_optimization_ray(dtrain, deval)` — распределённый поиск через `ray.tune` + `OptunaSearch`, с `PlacementGroupFactory` под ресурсы Ray-акторов. При `mlflow_uri`: `MLflowLoggerCallback` логирует параметры/метрики каждого `trial`'а, а после завершения подбора лучшая конфигурация переобучается на полных данных через `train_ray` и логируется как модель (`mlflow.xgboost.log_model`) в отдельном `run`'е `{name}_final` — аналогично `run_optimization`. Возвращает `ray.tune.ResultGrid`.
- `get_best_result()` — возвращает `(best_params, best_score)` по результатам последнего запуска оптимизации.

```
import ray
from xgboost_ray import RayParams
from mlumina.xgboost import XGBoostTuningModel, xgboost_load_dataset

ray.init(address="ray://localhost:10001")
dtrain, deval = xgboost_load_dataset(..., ray_data=True)

tuner = XGBoostTuningModel(
    name="distributed_experiment",
    mlflow_uri="http://localhost:5001",
    objective="reg:squarederror",
    eval_metric="rmse",
    learning_rate=(0.01, 0.3),
    subsample=(0.6, 1.0),
    colsample_bytree=(0.6, 1.0),
    lambda_=(1e-8, 10.0),
    mode="min",
    tree_method="hist",
    num_samples=100,
    num_boost_round=500,
    max_depth=10,
```

```
ray_params=RayParams(num_actors=4, cpus_per_actor=2, gpus_per_actor=0),
)

results = tuner.run_optimization_ray(dtrain, deval)
best_params, best_score = tuner.get_best_result()
```

??????-driven ???????

`configs/settings.yaml` описывает `training.xgboost` (фиксированные параметры обучения) и `tune.xgboost` (диапазоны для тюнинга), а также `data.train_file`/`data.eval_file`. Пример полного цикла (локально + через Ray, train + tune) — `examples/xgboost/example.py`, использующий `mlumina.core.MLuminaConfig`.

Revision #1

Created 2026-09-23 06:40:03 UTC by Маслов Алексей

Updated 2026-09-23 06:40:04 UTC by Маслов Алексей