Лабораторна робота №3. Гіперпараметрична оптимізація та оркестрація ML-пайплайнів з Optuna

0. Передумови та правила академічної доброчесності

Передумови: лабораторна робота охоплює результати ЛР1–ЛР2 (підготовка даних, базове тренування/оцінювання моделі, базове логування в MLflow, структура репозиторію).

Академічна доброчесність:

- Усі запуски експериментів мають бути відтворюваними: фіксуйте seed, версію датасету/ознак, версію коду (commit hash) та конфігурацію запуску.
- Заборонено підмінювати результати (наприклад, "покращувати" метрики вручну). У висновках пояснюйте, *чому* отримали саме такі значення.
- Якщо використовуєте зовнішні матеріали (документацію, приклади з блогів, репозиторії) - додавайте посилання в звіт.

1. Мета роботи

1. Пояснити різницю між параметрами моделі та гіперпараметрами;
2. Спроєктувати простір пошуку (search space) і обґрунтувати його межі;
3. Реалізувати objective function для Optuna та коректно оцінювати модель на validation наборі;
4. Організувати експеримент як відтворюваний пайплайн (конфігурація + код + артефакти);
5. Логувати експерименти в MLflow з використанням nested runs (parent run = study, child run = trial);
6. Інтерпретувати результати HPO і зробити висновок про компроміс якість ↔ час/ресурси.

2. Завдання для виконання

1. Встановити та налаштувати Optuna у віртуальному середовищі.
2. Реалізувати objective function, яку Optuna оптимізуватиме.
3. Описати простір пошуку гіперпараметрів (search space) відповідно до обраної моделі.

4. Створити Study та запустити оптимізацію не менше ніж на 20 спроб (trials).
5. Логувати кожен trial у MLflow як дочірній (nested) run усередині батьківського run.
6. Визначити найкращі гіперпараметри та перетренувати модель з ними, залогувавши фінальну модель як артефакт.
7. (Опційно/за наявності tracking server) Зареєструвати найкращу модель у MLflow Model Registry та перевести версію в стадію Staging.
8. Додати Hydra-конфігурацію (YAML) для параметризації пайплайна.

3. Теоретичні відомості

3.1. Гіперпараметрична оптимізація (HPO)

Гіперпараметри - це налаштування алгоритму/моделі, які задає дослідник *до* навчання (на відміну від параметрів, що навчаються, наприклад ваг/зміщень у нейромережах).

Приклади гіперпараметрів:

- Random Forest: n_estimators, max_depth, min_samples_split, min_samples_leaf.
- Logistic Regression: C (обернена сила регуляризації), solver, penalty.
- SVM: C, kernel, gamma.

Навіщо потрібна HPO:

- Оптимізація гіперпараметрів *може* суттєво покращити метрику якості (ефект залежить від задачі, даних і моделі).
- Ручний підбір параметрів неефективний, погано масштабований і часто важко відтворюваний.
- Систематична оптимізація полегшує обґрунтування вибору конфігурації та контроль експериментів.

3.2. Методи пошуку гіперпараметрів

1. Grid Search - повний перебір усіх комбінацій із заданої сітки.
   - Переваги: простота; гарантує знаходження найкращої точки *в межах сітки*.
   - Недоліки: швидко зростає обчислювальна складність із кількістю параметрів (прокляття розмірності).
2. Random Search - метод випадкового відбору.

   - Переваги: часто ефективніший за Grid Search при обмеженому бюджеті.
   - Недоліки: немає гарантії знаходження близько оптимальних областей при малому числі спроб.
3. Bayesian Optimization - інформований пошук, який будує сурогатну модель залежності “гіперпараметри → метрика” та обирає наступні точки, балансуючи exploration/exploitation.
   - Переваги: швидше знаходить перспективні області за малого бюджету.
   - Недоліки: потребує додаткових припущень/налаштувань та може бути важчим у реалізації.
4. Tree-structured Parzen Estimator (TPE) - популярний варіант байєсівської оптимізації, який використовується в Optuna за замовчуванням для багатьох сценаріїв.

3.3. Optuna - фреймворк для HPO

Optuna - open-source фреймворк (Preferred Networks), який автоматизує гіперпараметричну оптимізацію.

Ключові поняття:

- Study - “контейнер” оптимізації: зберігає всі trials і їх результати.
- Trial - одна спроба з конкретним набором гіперпараметрів.
- Objective function - функція, яку Optuna мінімізує або максимізує; повертає значення метрики (score).
- Sampler - стратегія вибору наступних гіперпараметрів (TPE, Random, Grid).
- Pruner - механізм ранньої зупинки trials за проміжними результатами.

Практичні переваги Optuna:

- Невисокий поріг входу та гнучке API.
- Підтримка відтворюваності через seed у sampler.
- Інтеграція з MLflow, можливість паралельного запуску за наявності спільного сховища (DB).

3.4. MLflow Nested Runs

Для HPO зручно використовувати вкладені runs:

- Parent run - основний запуск оптимізації (HPO експеримент).

- Child runs - окремі trials Optuna, вкладені в parent run.

Це дозволяє:

- Логічно групувати trials за одним експериментом.
- Візуально аналізувати еволюцію метрики та параметрів.
- Спрощувати аудит експерименту (конфігурації, seed, версії коду).

3.5. Hydra для конфігурації

Hydra - фреймворк керування конфігураціями, який дозволяє описувати параметри в YAML і змінювати їх без редагування коду.

Практичні переваги:

- Єдине джерело правди для параметрів експерименту.
- Зручна командна параметризація: python src/optimize.py hpo.n_trials=50 model.type=random_forest.
- Полегшує відтворення експериментів і контроль змін.

3.6. Відтворюваність в HPO

Для відтворюваності результатів критично:

- Фіксувати seed у NumPy/Python та в Optuna sampler.
- Логувати конфігурацію і seed у MLflow.
- Версіонувати код (Git commit hash) і дані (DVC/датасет-версія).

3.7. Як проєктувати простір пошуку (Search Space Design)

Простір пошуку визначає, *що саме* оптимізує Optuna. Типові правила:

- Межі мають бути обґрунтованими (з документації моделі, попередніх експериментів або стандартних значень).
- Уникайте надто широких діапазонів без потреби: це збільшує бюджет trial-ів і ризик “випадкових” покращень.
- Для параметрів масштабу (наприклад, C у логістичній регресії) часто застосовують log-uniform розподіл.
- Для дискретних рішень використовуйте categorical (наприклад, solver, penalty).

3.8. Переоптимізація на validation і “p-hacking”

Якщо багато разів підбирати гіперпараметри по одному validation-розбиттю, можна “переобрати” параметри під шум цього розбиття. Практики пом’якшення:

- використовуйте крос-валідацію (CV) або кілька розбиттів;
- тримайте окремий test набір, який використовується лише один раз - для фінальної оцінки.

3.9. Бюджет оптимізації та рання зупинка (Pruning)

HPO - це компроміс між якістю та витратами. У Optuna можна застосовувати pruner-и, які зупиняють слабкі trial-и раніше, якщо проміжні результати погані. Це особливо корисно для моделей, що тренуються довго.

4. Покрокова інструкція

Крок 1. Встановлення залежностей

Очікуваний результат: середовище Python з встановленими hydra-core, optuna, mlflow, scikit-learn (або іншим ML-пакетом залежно від задачі). Рекомендовано зафіксувати версії в requirements.txt.

Для роботи з MLflow Model Registry рекомендовано використовувати MLflow tracking server з backend store (SQLite/PostgreSQL тощо).

Якщо ви не запускаєте MLflow server, а використовуєте лише локальний ./mlruns, то реєстрація в Model Registry може бути недоступною або обмеженою (залежно від версії MLflow і типу backend store).

Крок 2. Структура проєкту

Очікуваний результат: директорії config/, src/, data/, models/, reports/ створені, а структура узгоджена з інструкцією.

Розширте структуру проєкту з попередніх лабораторних:

```
mlops_lab_project/
├── config/
│   ├── config.yaml
│   ├── model/
│   │   ├── random_forest.yaml
│   │   └── logistic_regression.yaml
│   └── hpo/
│       ├── optuna.yaml
│       ├── random.yaml
│       └── grid.yaml
├── src/
│   ├── prepare.py
│   ├── train.py
│   └── optimize.py          # HPO + MLflow + Hydra
├── data/
│   ├── raw/
│   │   └── dataset.csv
│   └── processed/
│       └── processed_data.pickle
├── models/
│   └── best_model.pkl
├── mlruns/                  # артефакти MLflow (для file store)
├── dvc.yaml
├── dvc.lock
├── requirements.txt
├── .gitignore
└── README.md
```

Крок 3. Підготовка конфігурацій (Hydra)

Очікуваний результат: config/config.yaml визначає sampler, кількість trial, метрику, параметри моделі, seed та налаштування MLflow.

1) Створіть config/config.yaml (базова конфігурація).
2) Створіть окремі конфігурації для моделі та HPO (приклади нижче).

Важливо: Hydra за замовчуванням змінює робочу директорію (створює outputs/...). Щоб не поршувати відносні шляхи до data/, у цій лабораторній рекомендується зафіксувати робочу директорію на корені проєкту через hydra.run.dir.

Крок 4. Реалізація objective function

Очікуваний результат: objective повертає метрику, а всі ключові параметри/метрики логуються в MLflow.

В src/optimize.py реалізуйте:

- вибір моделі (Random Forest або Logistic Regression) на основі конфігурації;
- оптимізацію та вибір гіперпараметрів через trial.suggest_*;
- коректну оцінку (мінімум train/test split; для високого рівня - cross-validation);
- вкладені MLflow runs для кожного trial.

Крок 5. Запуск Optuna Study

Очікуваний результат: після запуску виконується задана кількість trial-ів (наприклад, 20–50), і відображення найкращих параметрів.

У main():

1. Налаштуйте MLflow (tracking_uri, експеримент).
2. Запустіть parent run.
3. Створіть study з потрібним sampler та direction.
4. Виконайте study.optimize(objective, n_trials=...).

Крок 6. Логування у MLflow

Очікуваний результат: у MLflow UI продемонстровано:

- один parent run (Study),
- усередині - child runs (Trials),
- для кожного trial - params + metrics,
- артефакти (наприклад, best_params.json, best_model.pkl).

Для кожного trial логуйте:

- параметри (mlflow.log_params)
- метрики (mlflow.log_metric)
- службові теги (trial_number, sampler, model_type, seed)

У parent run логуйте:

- найкращу метрику та найкращі параметри

- конфігурацію експерименту (як артефакт або текст)
- фінальну модель

Крок 7. Реєстрація найкращої моделі (додатковий)

Очікуваний результат: або (A) модель зареєстрована в Registry, або (B) модель записана в MLflow як артефакт і її можна відтворити.

Якщо використовується MLflow tracking server з backend store, зареєструйте модель у Registry та переведіть її в Staging.

Крок 8. Порівняння sampler-ів

Очікуваний результат: відобразити порівняння мінімум 2 sampler-ів (наприклад, Random vs TPE): best value, середнє/медіана метрики, та коментар про витрати (час/кількість trial).

Запустіть HPO щонайменше для двох sampler-ів (наприклад, TPE та Random), використовуючи однаковий бюджет n_trials, і порівняйте:

- найкращу метрику;
- стабільність (розкид метрики між trials);
- швидкість виходу на "добрі" конфігурації (за графіком best-so-far).

5. Контрольні запитання

1. Чим відрізняються гіперпараметри від параметрів моделі?
2. Назвіть три методи пошуку гіперпараметрів і коротко поясніть принцип роботи кожного.
3. Що таке Trial в Optuna?
4. Поясніть Parent run та Child run у MLflow. Як nested runs допомагають аналізувати HPO?
5. Яку проблему вирішує Hydra в ML-проєкті?
6. У вашому експерименті n_trials=20. Що можна зробити, щоб:
   – зменшити час виконання;
   – підвищити якість пошуку (за тим самим або більшим бюджетом)?
7. Чому nested runs зручніші за логування всіх trials в один run?
8. Навіщо фіксувати seed під час HPO? Що саме треба "сідувати"?
9. Як коректно порівнювати TPE і Random sampler на одному датасеті?
10. Які кроки ви виконаєте перед переведенням моделі зі Staging у Production?

11. Модифікуйте objective function, щоб підтримувати 5-fold cross-validation. Як це вплине на стабільність і час оптимізації?
12. Опишіть ранню зупинку (early stopping) в Optuna для нейромережі (PyTorch/TensorFlow) та роль pruner.
13. Як організувати HPO для кількох моделей (RF, LR, SVM, GBM, NN) з динамічним вибором через Hydra?
14. Як інтегрувати HPO в CI/CD: коли запускати, який бюджет встановлювати, що робити з артефактами?
15. Запропонуйте стратегію тегування/опису версій моделі в MLflow Registry для різних sampler-ів та конфігурацій.

6. Рекомендовані матеріали

- Optuna Documentation: https://optuna.readthedocs.io/
- Optuna GitHub: https://github.com/optuna/optuna
- Hydra Documentation: https://hydra.cc/
- MLflow Documentation (Tracking/Model Registry): https://mlflow.org/docs/latest/index.html

7. Додаток А: приклад src/optimize.py

```
import json
import os
import random
from dataclasses import dataclass
from typing import Any, Dict, Tuple

import joblib
import mlflow
import numpy as np
import optuna
import pandas as pd
from hydra.core.config_store import ConfigStore
from hydra.utils import to_absolute_path
from omegaconf import DictConfig, OmegaConf
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.base import clone
from sklearn.metrics import f1_score, roc_auc_score
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler


def set_global_seed(seed: int) -> None:
```

```
    random.seed(seed)
    np.random.seed(seed)


def load_processed_data(path: str) -> Tuple[np.ndarray, np.ndarray,
np.ndarray, np.ndarray]:
    abs_path = to_absolute_path(path)

    if abs_path.endswith((".pkl", ".pickle")):
        obj = joblib.load(abs_path)
        if isinstance(obj, dict):
            if {"X_train", "X_test", "y_train",
"y_test"}.issubset(obj.keys()):
                return obj["X_train"], obj["X_test"], obj["y_train"],
obj["y_test"]
            if {"X", "y"}.issubset(obj.keys()):
                X = obj["X"]
                y = obj["y"]
            else:
                raise ValueError("Unknown format ({X,y} or
{X_train,X_test,y_train,y_test}).")
        elif isinstance(obj, pd.DataFrame):
            if "target" not in obj.columns:
                raise ValueError("DataFrame should contains 'target'
column.")
            X = obj.drop(columns=["target"]).values
            y = obj["target"].values
        else:
            raise ValueError("Unsupported format (dict or
pandas.DataFrame).")

        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=42, stratify=y if
len(np.unique(y)) > 1 else None
        )
        return X_train, X_test, y_train, y_test

    if abs_path.endswith(".csv"):
        df = pd.read_csv(abs_path)
        if "target" not in df.columns:
            raise ValueError("CSV should contains 'target' column.")
        X = df.drop(columns=["target"]).values
        y = df["target"].values
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=42, stratify=y if
len(np.unique(y)) > 1 else None
        )
        return X_train, X_test, y_train, y_test

    raise ValueError("Supporting  - .pickle/.pkl або .csv.")
```

```
def build_model(model_type: str, params: Dict[str, Any], seed: int) ->
Any:
    if model_type == "random_forest":
        return RandomForestClassifier(random_state=seed, n_jobs=-1,
**params)

    if model_type == "logistic_regression":
        clf = LogisticRegression(random_state=seed, max_iter=500,
**params)
        return Pipeline([("scaler", StandardScaler()), ("clf", clf)])

    raise ValueError(f"Unknown model.type='{model_type}'. Expecting
'random_forest' or 'logistic_regression'.")


def evaluate(model: Any, X_train: np.ndarray, y_train: np.ndarray,
X_test: np.ndarray, y_test: np.ndarray, metric: str) -> float:
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    if metric == "f1":
        return float(f1_score(y_test, y_pred, average="binary" if
len(np.unique(y_test)) == 2 else "weighted"))

    if metric == "roc_auc":
        if hasattr(model, "predict_proba"):
            y_score = model.predict_proba(X_test)[:, 1]
        else:
            y_score = model.decision_function(X_test)
        if len(np.unique(y_test)) > 2:
            return float(roc_auc_score(y_test,
model.predict_proba(X_test), multi_class="ovr", average="weighted"))
        return float(roc_auc_score(y_test, y_score))

    raise ValueError("Unsupported metrics. Use 'f1' or 'roc_auc'.")


def evaluate_cv(model: Any, X: np.ndarray, y: np.ndarray, metric: str,
seed: int, n_splits: int = 5) -> float:
    cv = StratifiedKFold(n_splits=n_splits, shuffle=True,
random_state=seed)
    scores = []
    for train_idx, test_idx in cv.split(X, y):
        X_tr, X_te = X[train_idx], X[test_idx]
        y_tr, y_te = y[train_idx], y[test_idx]
        m = clone(model)
        scores.append(evaluate(m, X_tr, y_tr, X_te, y_te, metric))
    return float(np.mean(scores))


def make_sampler(sampler_name: str, seed: int, grid_space: Dict[str,
Any] = None) -> optuna.samplers.BaseSampler:
```

```
    sampler_name = sampler_name.lower()
    if sampler_name == "tpe":
        return optuna.samplers.TPESampler(seed=seed)
    if sampler_name == "random":
        return optuna.samplers.RandomSampler(seed=seed)
    if sampler_name == "grid":
        if not grid_space:
            raise ValueError("For sampler='grid' need to set
grid_space.")
        return optuna.samplers.GridSampler(search_space=grid_space)
    raise ValueError("sampler should be: tpe, random, grid")


def suggest_params(trial: optuna.Trial, model_type: str, cfg:
DictConfig) -> Dict[str, Any]:
    if model_type == "random_forest":
        space = cfg.hpo.random_forest
        return {
            "n_estimators": trial.suggest_int("n_estimators",
space.n_estimators.low, space.n_estimators.high),
            "max_depth": trial.suggest_int("max_depth",
space.max_depth.low, space.max_depth.high),
            "min_samples_split":
trial.suggest_int("min_samples_split", space.min_samples_split.low,
space.min_samples_split.high),
            "min_samples_leaf": trial.suggest_int("min_samples_leaf",
space.min_samples_leaf.low, space.min_samples_leaf.high),
        }

    if model_type == "logistic_regression":
        space = cfg.hpo.logistic_regression
        return {
            "C": trial.suggest_float("C", space.C.low, space.C.high,
log=True),
            "solver": trial.suggest_categorical("solver",
list(space.solver)),
            "penalty": trial.suggest_categorical("penalty",
list(space.penalty)),
        }

    raise ValueError(f"Unknown model.type='{model_type}'.")


def objective_factory(cfg: DictConfig, X_train, X_test, y_train,
y_test):
    def objective(trial: optuna.Trial) -> float:
        params = suggest_params(trial, cfg.model.type, cfg)
        with mlflow.start_run(nested=True,
run_name=f"trial_{trial.number:03d}"):
            mlflow.set_tag("trial_number", trial.number)
            mlflow.set_tag("model_type", cfg.model.type)
            mlflow.set_tag("sampler", cfg.hpo.sampler)
```

```
            mlflow.set_tag("seed", cfg.seed)

            mlflow.log_params(params)

            model = build_model(cfg.model.type, params=params,
seed=cfg.seed)

            if cfg.hpo.use_cv:
                X = np.concatenate([X_train, X_test], axis=0)
                y = np.concatenate([y_train, y_test], axis=0)
                score = evaluate_cv(model, X, y,
metric=cfg.hpo.metric, seed=cfg.seed, n_splits=cfg.hpo.cv_folds)
            else:
                score = evaluate(model, X_train, y_train, X_test,
y_test, metric=cfg.hpo.metric)

            mlflow.log_metric(cfg.hpo.metric, score)
            return score

    return objective


def register_model_if_enabled(model_uri: str, model_name: str, stage:
str) -> None:
    client = mlflow.tracking.MlflowClient()
    mv = mlflow.register_model(model_uri, model_name)
    client.transition_model_version_stage(name=model_name,
version=mv.version, stage=stage)
    client.set_model_version_tag(model_name, mv.version,
"registered_by", "lab3")
    client.set_model_version_tag(model_name, mv.version, "stage",
stage)


def main(cfg: DictConfig) -> None:
    set_global_seed(cfg.seed)

    mlflow.set_tracking_uri(cfg.mlflow.tracking_uri)
    mlflow.set_experiment(cfg.mlflow.experiment_name)

    X_train, X_test, y_train, y_test =
load_processed_data(cfg.data.processed_path)

    grid_space = None
    if cfg.hpo.sampler.lower() == "grid":
        if cfg.model.type == "random_forest":
            grid_space = {
                "n_estimators":
list(cfg.hpo.grid.random_forest.n_estimators),
                "max_depth":
list(cfg.hpo.grid.random_forest.max_depth),
```

```
                "min_samples_split":
list(cfg.hpo.grid.random_forest.min_samples_split),
                "min_samples_leaf":
list(cfg.hpo.grid.random_forest.min_samples_leaf),
            }
        elif cfg.model.type == "logistic_regression":
            grid_space = {
                "C": list(cfg.hpo.grid.logistic_regression.C),
                "solver":
list(cfg.hpo.grid.logistic_regression.solver),
                "penalty":
list(cfg.hpo.grid.logistic_regression.penalty),
            }

    sampler = make_sampler(cfg.hpo.sampler, seed=cfg.seed,
grid_space=grid_space)

    with mlflow.start_run(run_name="hpo_parent") as parent_run:
        mlflow.set_tag("model_type", cfg.model.type)
        mlflow.set_tag("sampler", cfg.hpo.sampler)
        mlflow.set_tag("seed", cfg.seed)
        mlflow.log_dict(OmegaConf.to_container(cfg, resolve=True),
"config_resolved.json")

        study = optuna.create_study(direction=cfg.hpo.direction,
sampler=sampler)
        objective = objective_factory(cfg, X_train, X_test, y_train,
y_test)
        study.optimize(objective, n_trials=cfg.hpo.n_trials)

        best_trial = study.best_trial
        mlflow.log_metric(f"best_{cfg.hpo.metric}",
float(best_trial.value))
        mlflow.log_dict(best_trial.params, "best_params.json")

        best_model = build_model(cfg.model.type,
params=best_trial.params, seed=cfg.seed)
        best_score = evaluate(best_model, X_train, y_train, X_test,
y_test, metric=cfg.hpo.metric)
        mlflow.log_metric(f"final_{cfg.hpo.metric}", best_score)

        os.makedirs("models", exist_ok=True)
        joblib.dump(best_model, "models/best_model.pkl")
        mlflow.log_artifact("models/best_model.pkl")

        if cfg.mlflow.log_model:
            import mlflow.sklearn
            mlflow.sklearn.log_model(best_model,
artifact_path="model")

            if cfg.mlflow.register_model:
                model_uri = f"runs:/{parent_run.info.run_id}/model"
```

```
                register_model_if_enabled(model_uri,
cfg.mlflow.model_name, stage=cfg.mlflow.stage)

import hydra

@hydra.main(version_base=None, config_path="../config",
config_name="config")
def hydra_entry(cfg: DictConfig) -> None:
    main(cfg)

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

8. Додаток B: Приклад config/config.yaml

Це приклад, який ви можете адаптувати під власний датасет/модель. Зверніть увагу на hydra.run.dir: ., щоб відносні шляхи працювали стабільно.

```
seed: 42

mlflow:
  tracking_uri: "http://127.0.0.1:5000"
  experiment_name: "HPO_Lab3"
  log_model: true
  register_model: false                    # true for tracking server
+ backend store
  model_name: "BestOptimizedModel"
  stage: "Staging"

data:
  processed_path: "data/processed/processed_data.pickle"

model:
  type: "random_forest"

hpo:
  n_trials: 20
  sampler: "tpe"                           # tpe | random | grid
  metric: "f1"                             # f1 | roc_auc
  direction: "maximize"
  use_cv: false
  cv_folds: 5

  random_forest:
    n_estimators: { low: 50, high: 300 }
    max_depth: { low: 3, high: 15 }
    min_samples_split: { low: 2, high: 10 }
    min_samples_leaf: { low: 1, high: 5 }

  logistic_regression:
    C: { low: 1.0e-3, high: 1.0e2 }
```

```
    solver: ["liblinear", "lbfgs"]
    penalty: ["l2"]

  # GridSampler
  grid:
    random_forest:
      n_estimators: [50, 100, 200]
      max_depth: [5, 10, 15]
      min_samples_split: [2, 5, 10]
      min_samples_leaf: [1, 2, 4]
    logistic_regression:
      C: [0.01, 0.1, 1.0, 10.0]
      solver: ["liblinear"]
      penalty: ["l2"]

hydra:
  run:
    dir: .
  output_subdir: null
```