DATA+AI SUMMIT 2022

Fugue Tune:

Distributed Hybrid Hyperparameter Tuning

Jun Liu

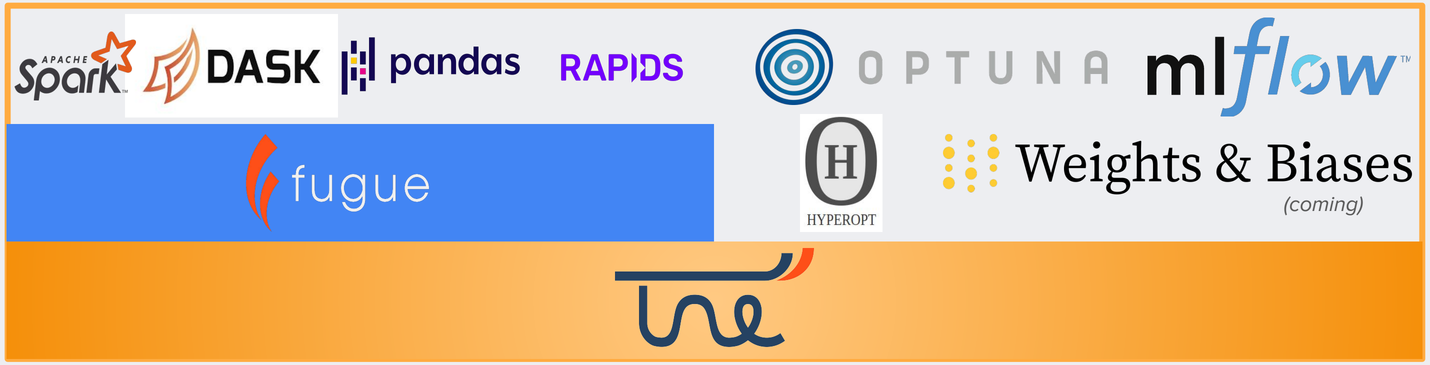
Senior Data Scientist, Lyft

Fugue-Tune:

Distributed Hybrid Hyperparameter Tuning

Tune is an abstraction layer for general parameter tuning. It has integrated existing hyperparameter tuning frameworks such as **Optuna** and **Hyperopt**, ML lifecycle management frameworks such as **MLflow**, and provided a simple, flexible and scalable interface on top of them.

Tune is built on **Fugue**, a unifier layer for distributed computing. So Tune can seamlessly run on any backend supported by Fugue, such as **Spark**, **Dask** and local.



Agenda

Fugue Tune: Distributed Hybrid Hyperparameter Tuning

Introduction

- Hyperparameter Optimization in ML

Fugue Tune

- The concept of **Hybrid Search Space**
- **Distributed** Hybrid Hyperparameter Tuning
- Integration with existing HPO and ML lifecycle management frameworks

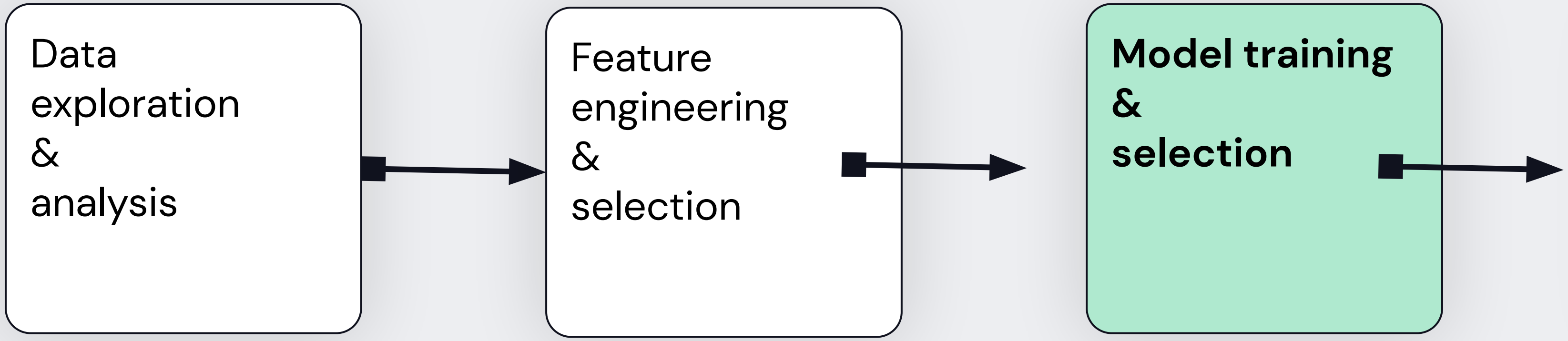
Demo on databricks

- Construct Hybrid Search Spaces with simple operations
- Distribute Tuning with **Spark** and track results with **MLflow**
- General ML objective tuning using **GreyKite**

Hyperparameter Optimization In Machine Learning

Hyperparameter Tuning In Machine Learning

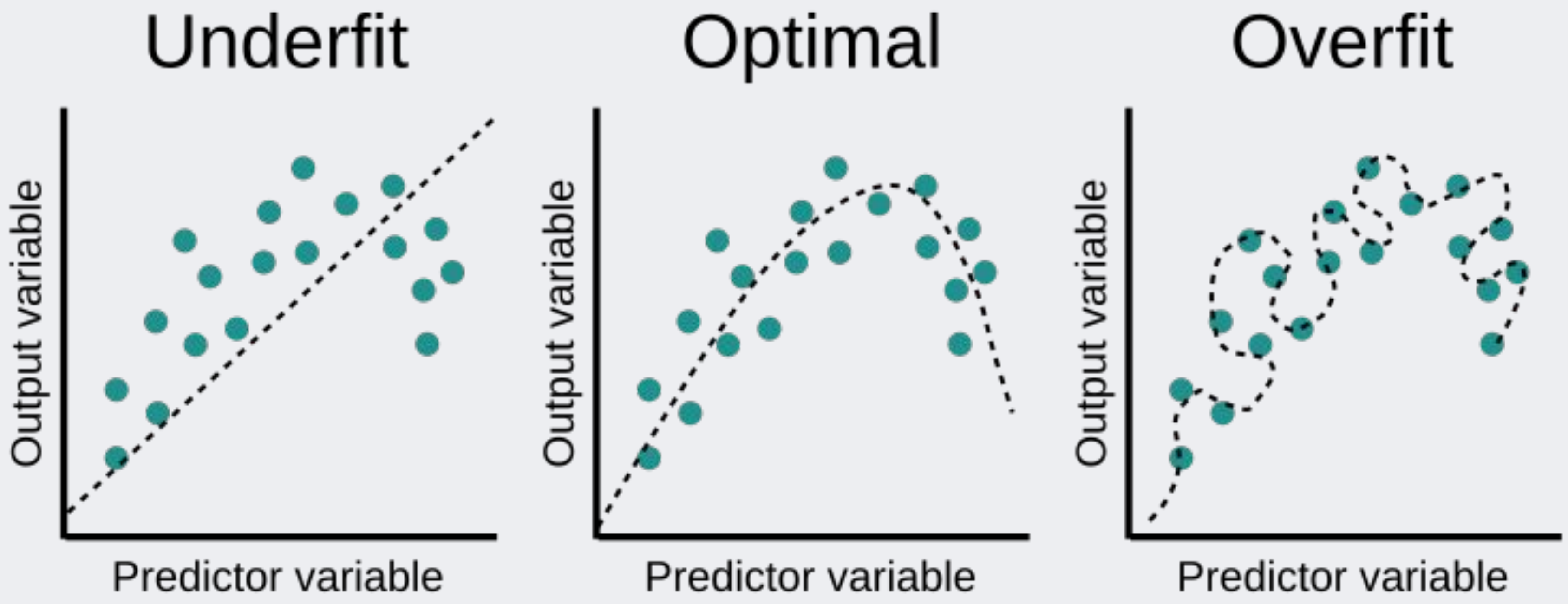
A critical step in the ML modeling workflow



- Objective selection
- Method selection
- **Hyperparameter tuning**
    - **Helps model generalize to the problem and dataset**
- …

Hyperparameter Tuning In Machine Learning

Sub-optimal hyperparameters → sub-optimal model performance



e.g. tree depth:	too low	**just right** 👍	too high

Image source: educative.io

The Essence of Hyperparameter Optimization

From a given **search space** (input range of hyperparameters), find a set

- `learning_rate = ?`
- `n_estimators = ?`
- `max_depth = ?`
- …

that **optimizes**

some **objective function** that

- Takes the input hyperparameter set
- Run the ML model
- Returns the cross validation score

Example:

California Housing Prices

Han is working on the California housing price prediction problem on Kaggle.

He looked over the discussion board and noticed that many people are using **XGBoost** and **LightGBM**.

Han decided to try both and take the best result.

Because XGBoost takes longer training time than LightGBM, Han decided to use grid search to tune XGBoost and Bayesian optimization to tune LightGBM.

If you were Han, how would you design this search space and tuning flow?

dmlc XGBoost

LightGBM

Define Objective Function:

Takes input modeling algorithm, train, evaluate and return a model score

```
def objective(model:Any, **hp:Any) -> float:
    model_ins = model(**hp)
    x = train.iloc[:,:-1]
    y = train.iloc[:,-1]
    scores = cross_val_score(model_ins, x, y, cv=3,
                             scoring=make_scorer(mean_absolute_percentage_error))
    return scores.mean()
```

Define Search Space:

Intuitively

Space 1:

- Model = XGBoost
- Try n_estimators in grid (100, 200, 300)

Space 2:

- Model = LightGBM
- Do BO on n_estimators in range (100, 400)

Call an optimizer to find the best parameters from the union of Space 1 and Space 2

Define Search Space:

Reality

Optuna

- Grid search, random search and BO are **exclusive** to each other. Users need to define separate objective functions to use more than one method.
- To do Grid search, parameters need to be declared both inside and outside the objective and in **different** ways.

```
def xgboost_objective(trial):
    train, _ = get_housing(fetch_california_housing)
    params = {
        "n_estimators": trial.suggest_int("n_estimators", 100, 300),
    }
    return objective(train, XGBRegressor, **params)

def lgbm_objective(trial):
    train, _ = get_housing(fetch_california_housing)
    params = {
        "n_estimators": trial.suggest_int("n_estimators", 10, 400),
    }
    return objective(train, LGBMRegressor, **params)

xgb_space = {"n_estimators": [100, 200, 300]}
xgb_study = optuna.create_study(sampler=optuna.samplers.GridSampler(xgb_space))
xgb_study.optimize(xgboost_objective)

lgbm_study = optuna.create_study()
lgbm_study.optimize(lgbm_objective, n_trials=20)

if xgb_study.best_value < lgbm_study.best_value:
    result = dict(model = XGBRegressor, **xgb_study.best_params)
    metric = xgb_study.best_value
else:
    result = dict(model = LGBMRegressor, **lgbm_study.best_params)
    metric = lgbm_study.best_value
```

Existing Frameworks vs. Fugue-Tune

```python
def xgboost_objective(trial):
    train, _ = get_housing(fetch_california_housing)
    params = {
        "n_estimators": trial.suggest_int("n_estimators", 100, 300),
    }
    return objective(train, XGBRegressor, **params)

def lgbm_objective(trial):
    train, _ = get_housing(fetch_california_housing)
    params = {
        "n_estimators": trial.suggest_int("n_estimators", 10, 400),
    }
    return objective(train, LGBMRegressor, **params)

xgb_space = {"n_estimators": [100, 200, 300]}
xgb_study = optuna.create_study(sampler=optuna.samplers.GridSampler(xgb_space))
xgb_study.optimize(xgboost_objective)

lgbm_study = optuna.create_study()
lgbm_study.optimize(lgbm_objective, n_trials=20)

if xgb_study.best_value < lgbm_study.best_value:
    result = dict(model = XGBRegressor, **xgb_study.best_params)
    metric = xgb_study.best_value
else:
    result = dict(model = LGBMRegressor, **lgbm_study.best_params)
    metric = lgbm_study.best_value
```

```python
lgbm_space = Space(model=LGBMRegressor, n_estimators=RandInt(10,400))
xgb_space = Space(model=XGBRegressor, n_estimators=Grid(100,200,300))

result = suggest_for_noniterative_objective(
    objective       = objective,
    space           = lgbm_space + xgb_space,
    local_optimizer = OptunaLocalOptimizer(max_iter=20)
)
```

Define Search Space:

Fugue Tune

- Model search, grid search, random search and BO can be **combined intuitively**
- **Zero redundancy** on defining parameters
- One expression for all underlying frameworks (e.g. Optuna, HyperOpt)

```
lgbm_space = Space(model=LGBMRegressor, n_estimators=RandInt(10,400))
xgb_space = Space(model=XGBRegressor, n_estimators=Grid(100,200,300))

result = suggest_for_noniterative_objective(
    objective       = objective,
    space           = lgbm_space + xgb_space,
    local_optimizer = OptunaLocalOptimizer(max_iter=20)
)
```

The concept of Hybrid Search Space

Grid Search

Exhaustively searches through a set of specified choices

```
space = Space(
    a = 1
    b = Grid(2, 3)
    c = Grid("x", "y")
)
```

Generated search space:

```
{"a": 1, "b": 2, "c": "x"}
{"a": 1, "b": 2, "c": "y"}
{"a": 1, "b": 3, "c": "x"}
{"a": 1, "b": 3, "c": "y"}
```

Pros: deterministic, interpretable, even coverage, good for categorical parameters
Cons: inefficient, complexity can increase exponentially

Random Search

Generates and evaluates a specified number of random inputs

```
space = Space(
    a = 1
    b = Rand(2, 3)
    c = Choice("x", "y")
).sample(4)
```

Generated search space:

```
{"a": 1, "b": 2.25, "c": "x"}
{"a": 1, "b": 2.11, "c": "y"}
{"a": 1, "b": 2.67, "c": "x"}
{"a": 1, "b": 2.84, "c": "x"}
```

Pros: complexity and distribution are controlled, good for continuous variables
Cons: not deterministic, normally requires large number of samples, number of iterations limited by time/resources

Bayesian Optimization

Iteratively searches based on previous observations

```
space = Space(
    a = 1
    b = Rand(2, 3)
)
```

Generated search space:

```
{"a": 1, "b": BO in (2,3)}
```

Pros: automated guided search, better result in fewer evaluations
Cons: sequential operations can not be distributed and may take more time

Hybrid Search Space

Customizing search space and use mixed type of methods

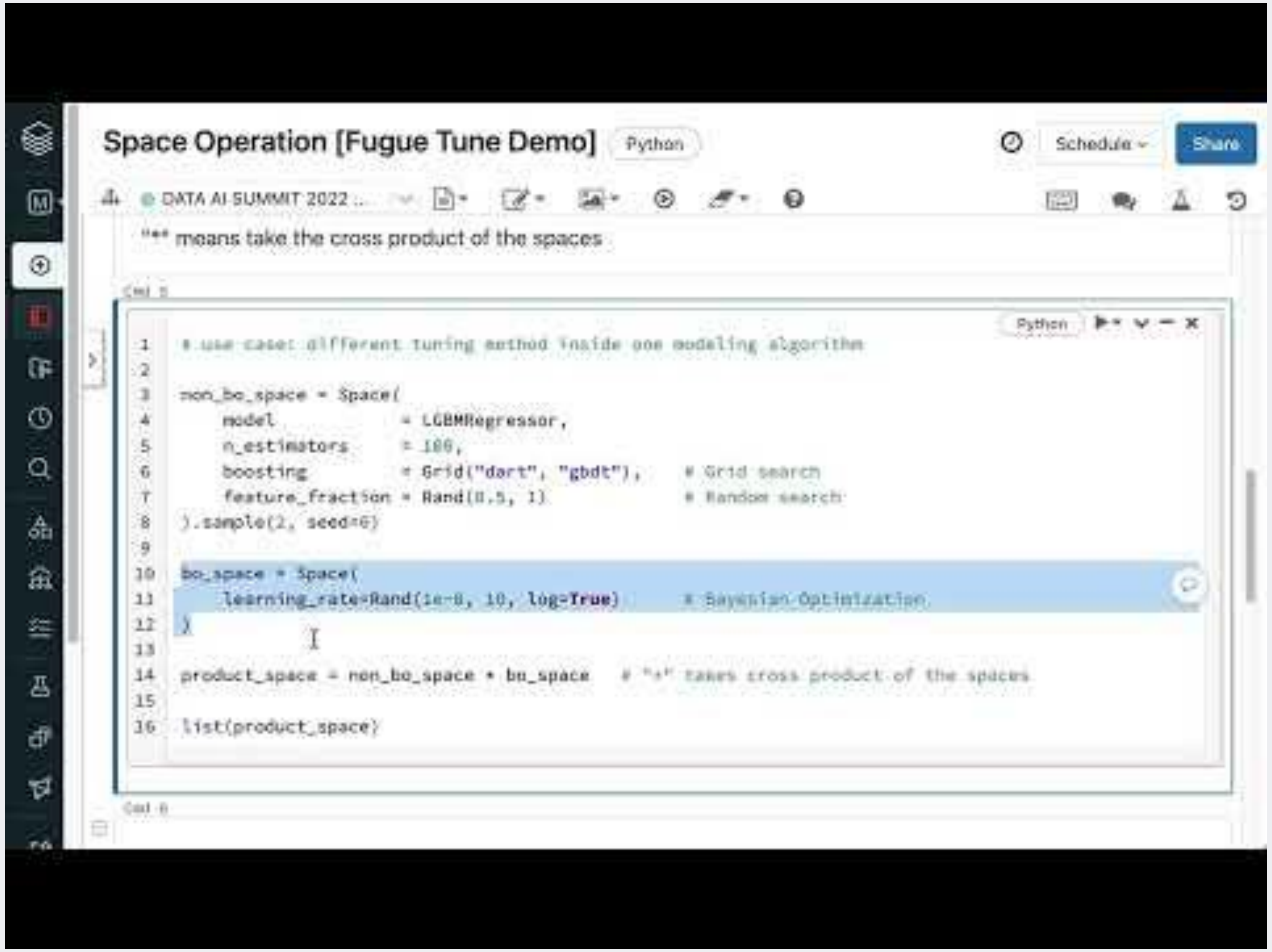
```
rand_space = Space(
     a = Rand(1, 2)
).sample(2)
grid_space = Space(
     b = Grid("x", "y")
)
bo_space   = Space(
     c = Rand(2, 3)
)
space = (rand_space + grid_space) * bo_space
```

Generated search space:

```
{"a": 1.2, "c": bo in (2,3)}
{"a": 1.7, "c": bo in (2,3)}
{"b": "x", "c": bo in (2,3)}
{"b": "y", "c": bo in (2,3)}
```

Bayesian optimization as a second tuning layer on top of Random and Grid Search.

Demo: Hybrid Parameter Search Space



- https://www.youtube.com/watch?v=Po2AFbKde5E&t=2s

Example 1: Union Space

"+" means take the union of the spaces

```
# use case: different tuning method on different modeling algorithms

xgb_grid = Space(model=XGBRegressor, n_estimatores=Grid(50,150))
lgbm_random = Space(model=LGBMRegressor, n_estimatores=RandInt(100,200)).sample(3)
catboost_bo = Space(model=CatBoostRegressor, n_estimatores=RandInt(100,200))

union_space = xgb_grid + lgbm_random + catboost_bo
```

Example 2: Cross Product Space

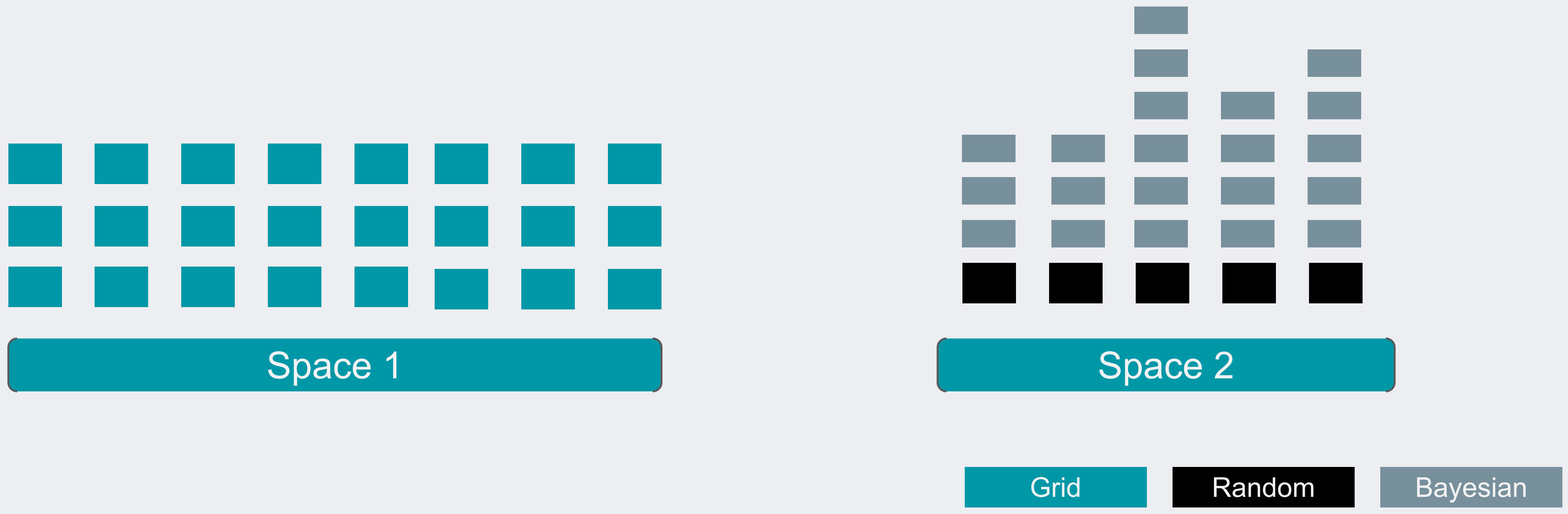
"*" means take the cross product of the spaces

```
# use case: different tuning method inside one modeling algorithm

non_bo_space = Space(
    model=LGBMRegressor,
    boosting=Grid("dart", "gbdt"),     # Grid search
    feature_fraction=Rand(0.5, 1)      # Random search
).sample(2, seed=0)
bo_space = Space(learning_rate=Rand(1e-8, 10, log=True)) # Bayesian Optimization

product_space = non_bo_space * bo_space
```

Distributed HPO on Hybrid Search Space

Distributed HPO

on Hybrid Search Space



Fugue-Tune

Distribute the tuning jobs to Spark/Dask with one parameter

```
def objective(model:Any, **hp:Any) -> float:
    model_ins = model(**hp)
    x = train.iloc[:,:-1]
    y = train.iloc[:,-1]
    scores = cross_val_score(model_ins, x, y, cv=3,
                             scoring=make_scorer(mean_absolute_percentage_error))
    return scores.mean()

lgbm_space = Space(model=LGBMRegressor, n_estimators=RandInt(10,400))
xgb_space = Space(model=XGBRegressor, n_estimators=Grid(100,200,300))

result = suggest_for_noniterative_objective(
    objective        = objective,
    space            = lgbm_space + xgb_space,
    local_optimizer  = HyperoptLocalOptimizer(max_iter=20),
    execution_engine= spark
)
```

- Set execution engine to your **spark session**
- Fugue will take care the backend and parallelize everything that could be parallelized

Integration with existing HPO and ML lifecycle management frameworks

Fugue-Tune:

Monitor tuning jobs on MLflow with one parameter change

```python
def objective(model:Any, **hp:Any) -> float:
    model_ins = model(**hp)
    x = train.iloc[:,:-1]
    y = train.iloc[:,-1]
    scores = cross_val_score(model_ins, x, y, cv=3,
                             scoring=make_scorer(mean_absolute_percentage_error))
    return scores.mean()

lgbm_space = Space(model=LGBMRegressor, n_estimators=RandInt(10,400))
xgb_space = Space(model=XGBRegressor, n_estimators=Grid(100,200,300))

result = suggest_for_noniterative_objective(
    objective       = objective,
    space           = lgbm_space + xgb_space,
    local_optimizer = HyperoptLocalOptimizer(max_iter=20),
    execution_engine= spark,
    logger          = "mlflow"
)
```

Use *MLflow* as logging backend with one parameter change

Fugue-Tune:

Track tuning results with MLflow experiments

				Metrics >			Parameters <	
☐	↓ Start Time	Duration	Run Name	OBJECTIVE_MI	mean_test_CO	mean_test_MA	changepoints	custom
☐	⊟ 12 hours ago	1.1h	bd82e	5.075	0.729	0.434	{'changepo...	{'fit_algorit...
☐	11 hours ago	25.7min	{'trial_id': '...	5.573	0.771	0.47	{'changepo...	{'fit_algorit...
☐	11 hours ago	10.2min	{'trial_id': 'f...	5.096	0.745	0.439	{'changepo...	{'fit_algorit...
☐	11 hours ago	7.9min	{'trial_id': '...	5.075	0.729	0.434	{'changepo...	{'fit_algorit...
☐	11 hours ago	27.0min	{'trial_id': '...	5.513	0.771	0.465	{'changepo...	{'fit_algorit...
☐	11 hours ago	30.4min	{'trial_id': '...	5.848	0.774	0.491	{'changepo...	{'fit_algorit...
☐	11 hours ago	12.6min	{'trial_id': '...	5.129	0.744	0.442	{'changepo...	{'fit_algorit...
☐	11 hours ago	31.8min	{'trial_id': '...	5.541	0.772	0.468	{'changepo...	{'fit_algorit...
☐	12 hours ago							
☐	12 hours ago							
☐	12 hours ago							

- Highly organized logging with nested structure
  - One suggest method will generate one parent run
  - All the sub trials are logged as sub runs
  - Parent run takes the best result from all the sub runs

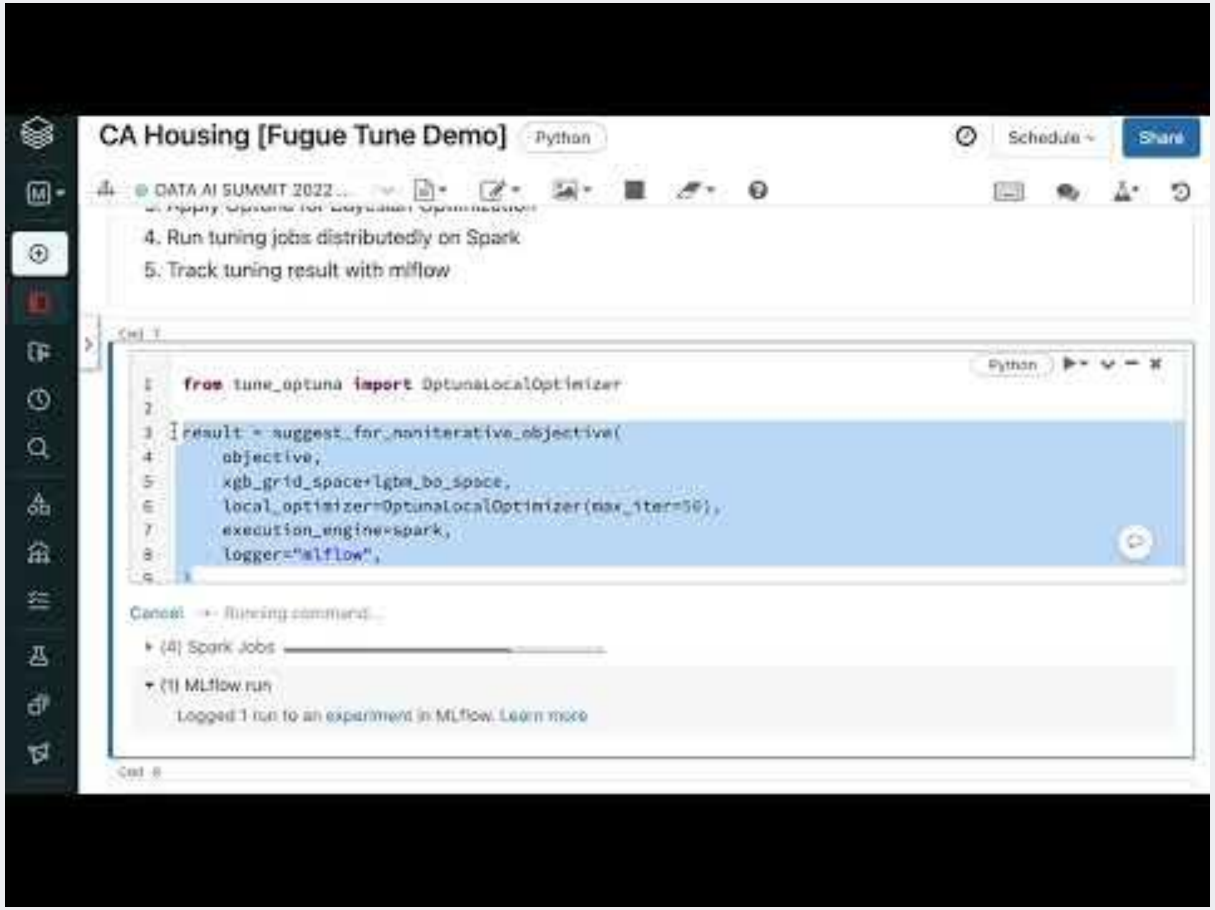
Fugue-Tune:

Switch between HPO libraries seamlessly

```
def objective(model:Any, **hp:Any) -> float:
    model_ins = model(**hp)
    x = train.iloc[:,:-1]
    y = train.iloc[:,-1]
    scores = cross_val_score(model_ins, x, y, cv=3,
                             scoring=make_scorer(mean_absolute_percentage_error))
    return scores.mean()

lgbm_space = Space(model=LGBMRegressor, n_estimators=RandInt(10,400))
xgb_space = Space(model=XGBRegressor, n_estimators=Grid(100,200,300))

result = suggest_for_noniterative_objective(
    objective       = objective,
    space           = lgbm_space + xgb_space,
    local_optimizer = HyperoptLocalOptimizer(max_iter=20)
)
```

Switch to HyperOpt for BO in one parameter change

Demo: Distribute Tuning with Spark and result monitoring with MLflow



- https://www.youtube.com/watch?v=LOkROeqJG1M

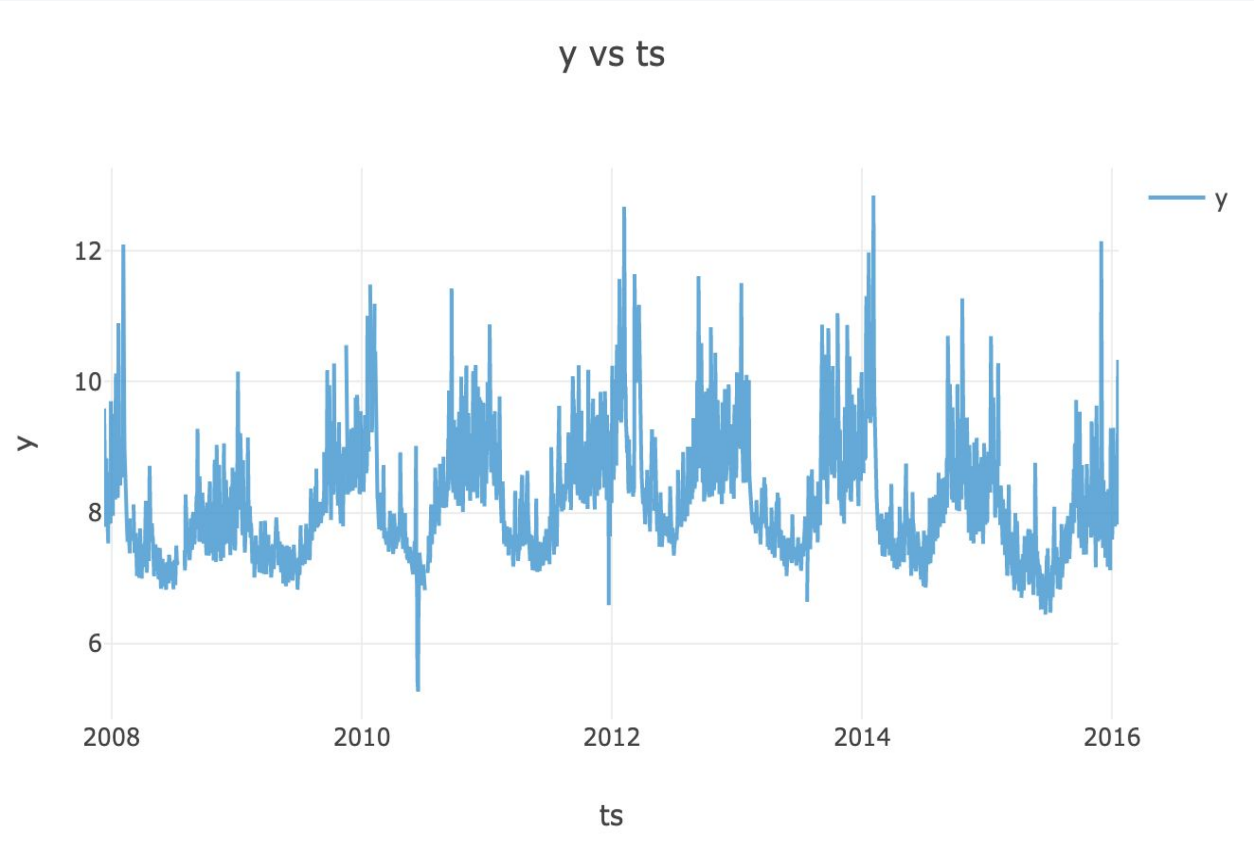
Demo: Distribute Tuning with Spark and result monitoring with MLflow

```
# Run tuning distributedly on Spark, track the result with mlflow and apply Optuna
for Bayesian Optimization.

result = suggest_for_noniterative_objective(
    objective,
    xgb_space + lgbm_space,
    local_optimizer=HyperoptLocalOptimizer(max_iter=50),
    execution_engine="spark",
    logger="mlflow",
)
```

Final Demo: General ML objective tuning using GreyKite

General ML objective tuning using GreyKite

Forecasting Peyton Manning Wiki Daily Log Page View

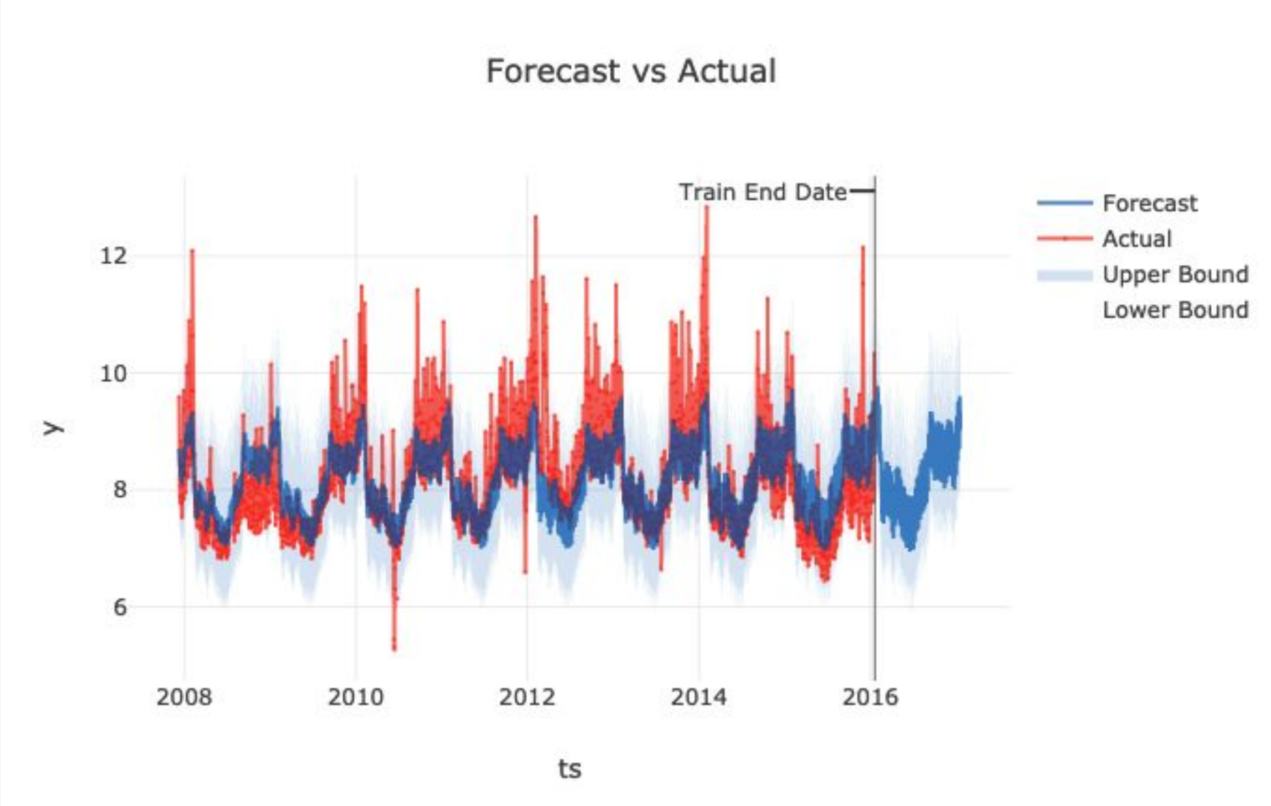


Dataset Info:

- Time column "ts" ranges from 2007-12-10 to 2016-01-20
- Value column "y" ranges from 5.26 to 12.84
- Time series cross validation
- Last year to test
- Metric: MAPE

General ML objective tuning using GreyKite

Forecasting Peyton Manning Wiki Daily Log Page View



Common Parameters to tune:

- Datetime derivatives
- Growth
- Trend
- Seasonality
- Events
- Autoregression method
- Interactions
- …

GREYKITE

Demo: General ML Objective Tuning using GreyKite

- https://www.youtube.com/watch?v=kXB8uXIQ85O

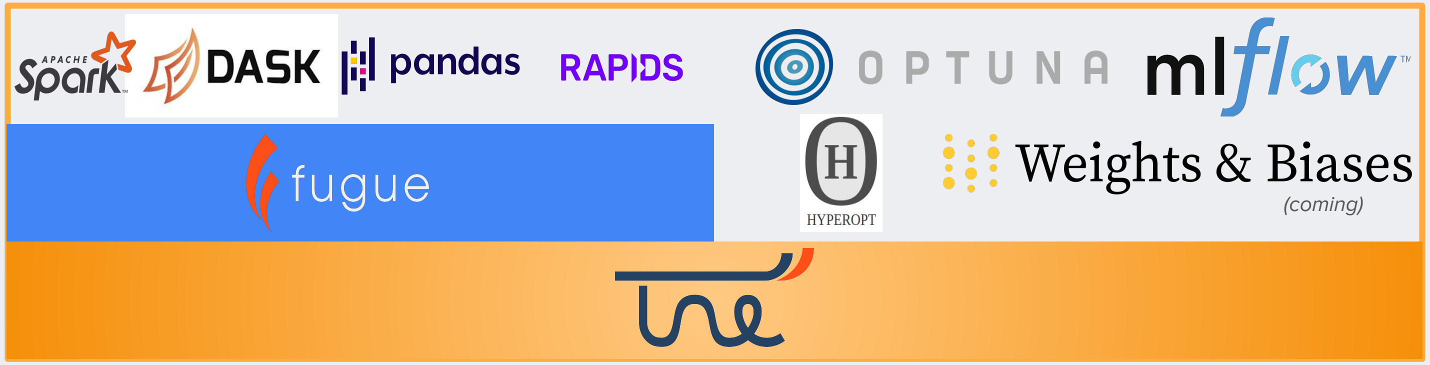
Summary

Fugue Tune: Distributed Hybrid Hyperparameter Tuning

What we have covered today:

- Definition and construction of hybrid search space
- Distributed model evaluation and result monitoring
- Distributed hybrid hyperparameter tuning on complex ML objective functions

Fugue Tune provides an simple, flexible, and scalable interface for distributed hybrid parameter tuning. It helps achieve key functionalities in machine learning with minimized amount of code.



Presentation Materials

Fugue Tune: Distributed Hybrid Hyperparameter Tuning

`pip install tune` (https://github.com/fugue-project/tune)

`pip install fugue` (https://github.com/fugue-project/fugue)

Space Operation Demo

- Notebook: https://www.kaggle.com/liujun4/tune-demo-1-space-operation
- Recording: https://www.youtube.com/watch?v=Po2AFbKde5E&t=2s

CA Housing Demo

- Notebook: https://www.kaggle.com/code/liujun4/tune-vs-optuna
- Recording: https://www.youtube.com/watch?v=LOkROeqJG1M

Greykite Demo

- Notebook: https://www.kaggle.com/liujun4/tune-demo-2-general-ml-objective-tuning-greykite
- Recording: https://www.youtube.com/watch?v=kXB8uXIQ850

DATA+AI SUMMIT 2022

Thank you

Jun Liu

Senior Data Scientist