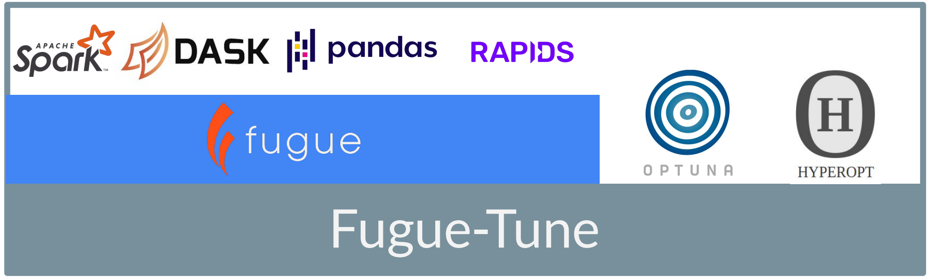
Fugue-Tune: A Simple Interface for Distributed Hyperparameter Tuning

Presented By: Jun Liu

Summary

- **Tune** is an abstraction layer for general parameter tuning. It has integrated existing hyperparameter tuning frameworks such as Optuna and Hyperopt and provided a **scalable and simple interface** on top of them. It is built on **fugue** so it can seamlessly run on any backend supported by Fugue, such as Spark, Dask and local.
- `pip install tune`
    - https://github.com/fugue-project/tune
- `pip install fugue`
    - https://github.com/fugue-project/fugue



Agenda

- **Introduction**: Hyperparameter Optimization in ML
- **Fugue Tune**
    - The concept of ***Space***
    - Distributed Hyperparameter Tuning
    - Feature Highlights
- **Demo**
    - Search Space Operations
    - General ML objective tuning using GreyKite

Hyperparameter Optimization In Machine Learning

HPO is a critical step in the ML modeling workflow

Data exploration & analysis → Feature engineering & selection → **Model training & selection** → Model development & deployment

- Objective selection
- Method selection
- **Hyperparameter tuning**
- ...

Recap: The Essence of Hyperparameter Optimization

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

- `learning_rate = ?`
- `n_estimators = ?`
- `n_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?

Define objective

```
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

Search Space 1:
- Model = XGBoost
- Try n_estimators in grid (100, 200, 300)

Search Space 2:
- Model = LightGBM
- Do BO on n_estimators in range (100, 400)

Find the best parameters in the union of space 1 and space 2

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.

```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
```

Fugue-Tune: A Simple Interface for Distributed Hyperparameter Tuning

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)
)
```

Existing Frameworks vs. Fugue-Tune

```
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)
)
```

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)

Search *Space*

Grid Search

```
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

```
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

```
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

```
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)}
```

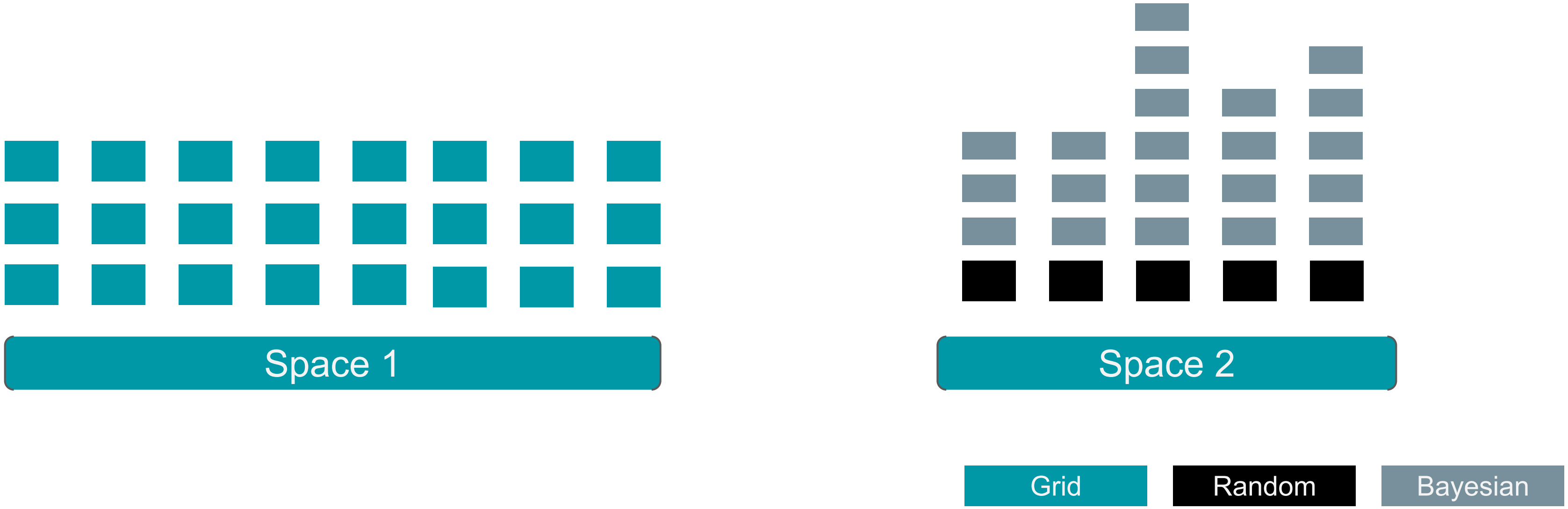
Bayesian optimization as a second tuning layer on top of Random and Grid Search.

Demo:

- Search Spaces Operations

Distributed HPO on Hybrid Search Space

Hybrid Search Space



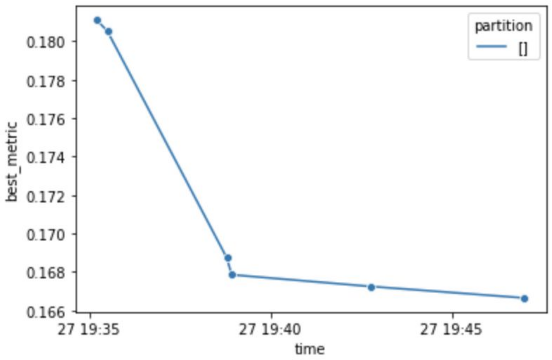
Distribute the tuning jobs to Spark/Dask with one parameter

```
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 = OptunaLocalOptimizer(max_iter=20),
    execution_engine="spark"
)
```

- Use a string to represent your **spark session**
- Fugue will take care the backend and parallelize everything that could be parallelized

Monitor tuning result at real time

```
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 = OptunaLocalOptimizer(max_iter=20),
    execution_engine="dask",
    execution_engine_conf={"callback":True},
    monitor="ts"
)
```



- Fugue lets workers communicate with driver at realtime
  - `ts` to monitor the up-to-date best metric collected
  - `hist` to monitor the histogram of metrics collected

Feature Highlights

1. Define objectives in native Python function

```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()
```

- Could pass in any parameters
  - Hyperparameter
  - Model
  - Dataframe
- Tune will convert simple function to `tune` compatible projects

2. Use Metadata to keep track of multiple metrics

```
def objective(model:Any, **hp:Any) -> Tuple[float, Dict[str,Any]]:
    model_ins = model(**hp)
    x = train.iloc[:,:-1]
    y = train.iloc[:,-1]
    scoring = {"MAE": make_scorer(mean_absolute_error), "MAPE": make_scorer(mean_absolute_percentage_error)}
    scores = cross_validate(model_ins, x, y, cv=3,
                            scoring=scoring)
    print(scores)
    return float(scores["test_MAPE"].mean()), {"mean_MAE": scores["test_MAE"].mean()}
```

e.g. model itself, business metrics, ...

```
print(result[0].metric, result[0].trial.params, result[0].metadata)
```

```
0.167867079442689 {'model': <class 'lightgbm.sklearn.LGBMRegressor'>, 'n_estimators': 393}
{'mean_MAE': 0.3024627517963494}
```

3. Switch between HPO libraries seamlessly

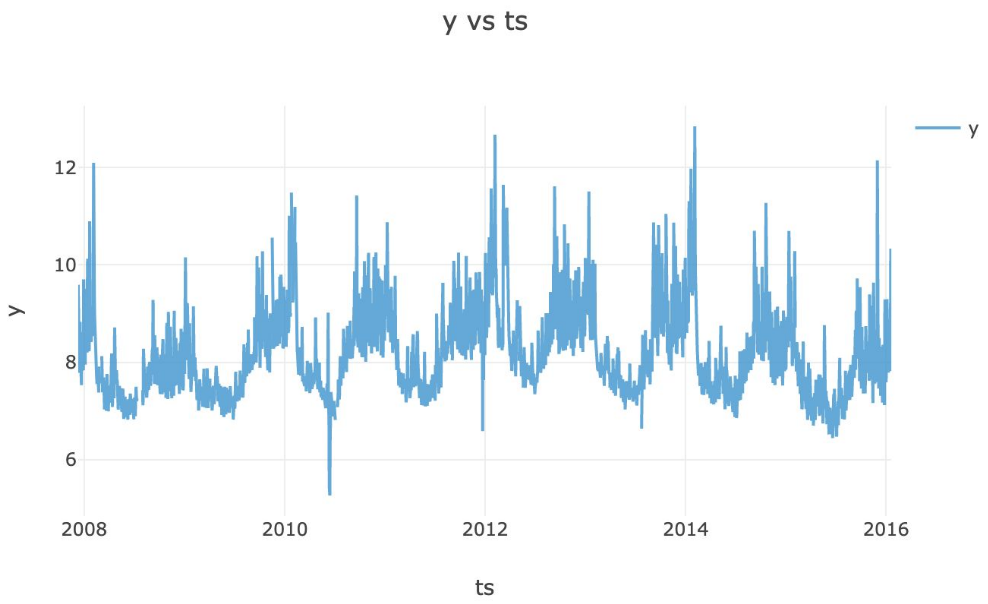
```python
def objective(model:Any, **hp:Any) -> float:
    model_ins = model(**hp)
    x = train.iloc[:,:-1]
    y = train.iloc[:,-1]
    scores = cross_val_score(model_ins, x, y, cv=3,
                             scoring=make_scorer(mean_absolute_percentage_error))
    return scores.mean()

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)
)
```

Switch to HyperOpt for BO in one parameter change

Demo:

- General ML objective tuning using GreyKite

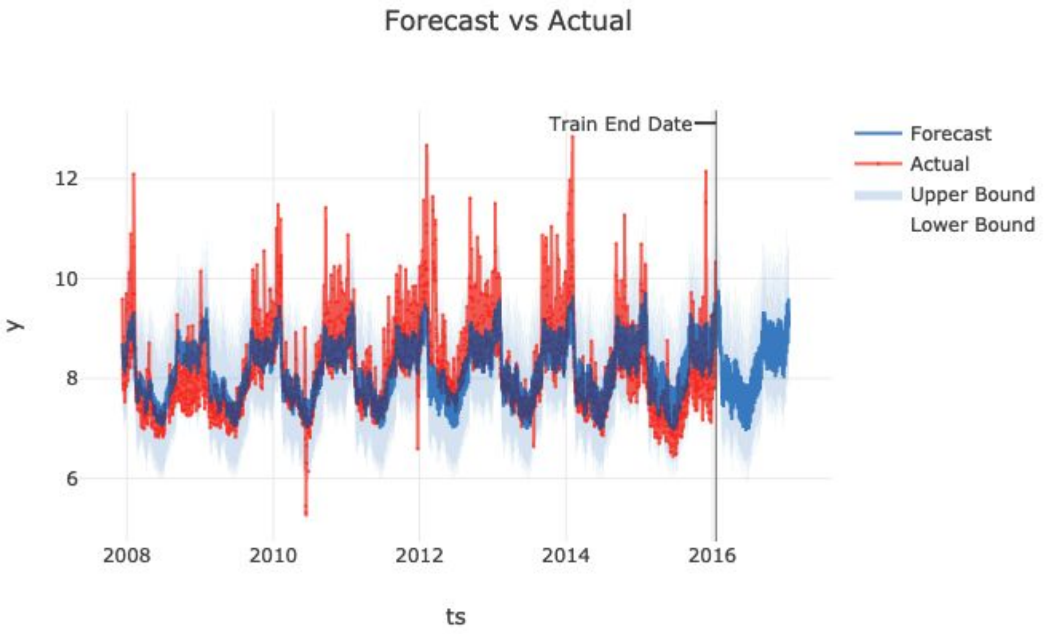
Peyton Manning Wiki Daily Log Page View



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

Peyton Manning Wiki Daily Log Page View



Common Parameters to tune in Greykite:

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

Fugue-Tune: A Simple Interface for Distributed HPO

- **Framework and Method agnostic**
  - Use with any ML framework
  - Search on Hybrid Space
  - Switch between libraries like Hyperopt and Optuna without code change
- **Scalable and automated**
  - Tune both locally and distributedly without code change
- **Platform agnostic**
  - Works for backends such as Spark, Dask and local

with a simple and intuitive interface

```
pip install tune
```

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

- Space Operation Demo: https://www.kaggle.com/liujun4/tune-demo-1-space-operation
- Greykite Demo: https://www.kaggle.com/liujun4/tune-demo-2-general-ml-objective-tuning-greykite

Thank you!