Hypernets

Jul 17, 2023

Home:

CHAPTER 1

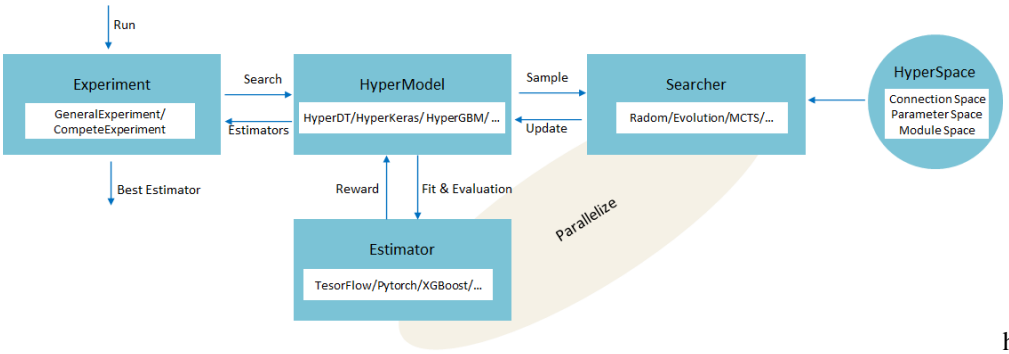
Hypernets: A General Automated Machine Learning Framework

Hypernets is a general AutoML framework that can meet various needs such as feature engineering, hyperparameter optimization, and neural architecture search, thereby helping users achieve the end-to-end automated machine learning pipelines.

1.1 Overview

Hypernets is a general automated search framework, based on which it can implement automatic optimization tools for various machine learning frameworks and libraries, including deep learning frameworks such as tensorflow, keras, pytorch, and machine learning libraries like sklearn, lightgbm, xgboost, etc. We introduced an abstract search space representation, taking into account the requirements of hyperparameter optimization and neural architecture search(NAS), making Hypernets a general framework that can adapt to various automated machine learning needs.

The figure below shows conceptual model of Hypernets.



hypernets_conceptual_model

1.1.1 Key Components

HyperSpace

The space of all feasible solutions for a model is called **Search Space**. HyperSpace is an abstract representation of the search space composed of `Parameter Space`, `Connection Space`, and `Module Space`. The general form of HyperSpace is a DAG, so it can represent ML pipeline and neural network architecture very flexibly.

Seacher

Search algorithms that looking for a optimal solution in `HyperSpace` and generating samples for `HyperModel`.

HyperModel

High-level interface for users to perform model search and training, as long as the defined search space and training data are passed in to get the best model. HyperModel is an abstract class that needs to implement a dedicated HyperModel for different frameworks or domains. For example, `HyperKeras` is used to automatically search for neural networks built with keras, and `HyperGBM` is used to automatically optimize ML pipeline composed of sklearn, xgboost, and lightgbm. . . .

Estimator

A specific `HyperModle` needs to be paired with a dedicated `Estimator` to fit and evaluate the sample given by the `HyperModel`. This sample may be a set of hyperparameters, a network architecture, or a mixture of them.

Experiment

The playground to prepare training and testing data, and search the optimized estimator with HyperModel.

Tabular Toolbox

A general tabular data computing layer. At present, we provide the implementations of pandas, cudf and dask data types.

1.2 Quick-Start

1.2.1 Installation

Python version 3.6 or above is necessary before installing Hypernets.

Conda

Install Hypernets with `conda` from the channel *conda-forge*:

```
conda install -c conda-forge hypernets
```

Pip

Install Hypernets with `pip` command:

```
pip install hypernets
```

Optional, to run Hypernets in JupyterLab notebooks, install Hypernets and JupyterLab with command:

```
pip install hypernets[notebook]
```

- Optional, to support experiment visualization base on web, install with command:

```
pip install hypernets[board]
```

Optional, to run Hypernets in distributed Dask cluster, install Hypernets with command:

```
pip install hypernets[dask]
```

Optional, to support simplified Chinese in feature generation, install `jieba` package before run Hypernets, or install Hypernets with command:

```
pip install hypernets[zhcn]
```

Optional, install all Hypernets components and dependencies with one command:

```
pip install hypernets[all]
```

‘‘

Verify installation:

```
python -m hypernets.examples.smoke_testing
```

1.2.2 Getting started

In current version, we provide `PlainModel` (a plain `HyperModel` implementation), which can be used for hyper-parameter tuning with sklearn machine learning algorithms.

Basically, to search the best model only needs 4 steps:

- Step 1. Define `Search Space`
- Step 2. Select a `Searcher`
- Step 3. Select a `HyperModel`
- Step 4. Search and get the best model

Define Search space

Firstly, we define a search space for hyper-parameters of *DecisionTreeClassifier* and *MLPClassifier*:

```
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier

from hypernets.core import get_random_state
from hypernets.core.ops import ModuleChoice, HyperInput, ModuleSpace
```

(continues on next page)

(continued from previous page)

```
from hypernets.core.search_space import HyperSpace, Choice, Int


def my_search_space(enable_dt=True, enable_mlp=True):
    space = HyperSpace()

    with space.as_default():
        hyper_input = HyperInput(name='input1')

        estimators = []
        if enable_dt:
            estimators.append(dict(
                cls=DecisionTreeClassifier,
                criterion=Choice(["gini", "entropy"]),
                splitter=Choice(["best", "random"]),
                max_depth=Choice([None, 3, 5, 10, 20, 50]),
                random_state=get_random_state(),
            ))

        if enable_mlp:
            estimators.append(dict(
                cls=MLPClassifier,
                max_iter=Int(500, 5000, step=500),
                activation=Choice(['identity', 'logistic', 'tanh', 'relu']),
                solver=Choice(['lbfgs', 'sgd', 'adam']),
                learning_rate=Choice(['constant', 'invscaling', 'adaptive']),
                random_state=get_random_state(),
            ))

        modules = [ModuleSpace(name=f'{e["cls"].__name__}', **e) for e in estimators]
        outputs = ModuleChoice(modules)(hyper_input)
        space.set_inputs(hyper_input)

    return space
```

Training with PlainModel

Turning and scoring with `my_search_space` for `heart_disease_uci` dataset.

```
def train():
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import classification_report
    from hypernets.core.callbacks import SummaryCallback
    from hypernets.examples.plain_model import PlainModel
    from hypernets.searchers import make_searcher
    from hypernets.tabular.datasets import dsutils

    X = dsutils.load_heart_disease_uci()
    y = X.pop('target')

    X_train, X_eval, y_train, y_eval = \
        train_test_split(X, y, test_size=0.3)

    # make MCTS searcher
```

(continues on next page)

(continued from previous page)

```
    searcher = make_searcher('mcts', my_search_space, optimize_direction='max')
    callbacks = [SummaryCallback()]

    # create HyperModel and do 'search' action
    hm = PlainModel(searcher=searcher, reward_metric='f1', callbacks=callbacks)
    hm.search(X_train, y_train, X_eval, y_eval, )

    # get best estimator
    best = hm.get_best_trial()
    estimator = hm.final_train(best.space_sample, X_train, y_train)

    # scoring
    y_pred = estimator.predict(X_eval)
    print(classification_report(y_eval, y_pred))


if __name__ == '__main__':
    train()
```

Run the example, we will get console output:

```
17:36:56 I hypernets.u.common.py 147 - 2 class detected, {0, 1}, so inferred as a␣
↪[binary classification] task
17:36:56 I hypernets.c.meta_learner.py 22 - Initialize Meta Learner: dataset_
↪id:e10ae1d61123d55062f7d3b64b79a6e1
17:36:56 I hypernets.c.callbacks.py 235 -
Trial No:1
----------------------------------------------------------------
(0) Module_ModuleChoice_1.hp_or:                              0
(1) DecisionTreeClassifier.criterion:                      gini
(2) DecisionTreeClassifier.splitter:                       best
(3) DecisionTreeClassifier.max_depth:                         5
----------------------------------------------------------------
...

              precision    recall  f1-score   support

           0       0.68      0.67      0.67        42
           1       0.72      0.73      0.73        49

    accuracy                           0.70        91
   macro avg       0.70      0.70      0.70        91
weighted avg       0.70      0.70      0.70        91
```

1.3 Search Space

For hyper-parameter optimization, the search space might be a closed subset of Euclidean space composed of discrete or continuous values. For neural architecture search(NAS), The network architecture is composed of graphs generated by various layers through connections, therefore all feasible graphs constitute the search space.

Hypernets introduced an abstract search space representation and taken into account both of the above requirements, so that it can use an unified primitive to define various search spaces.

- illustration of the search space in Hypernets

1.3.1 HyperSpace

HyperSpace is an abstract representation of the search space composed of `Parameter Space`, `Connection Space`, and `Module Space`. The general form of HyperSpace is a directed acyclic graph (DAG), consisting of connecting nodes and edges. Each node contains a set of hyperparameters. In some cases, the objective function for optimatization has only one node, like the lightGBM model, which we only need to optiizea a few hyper-parameters.

1.3.2 Parameter Space

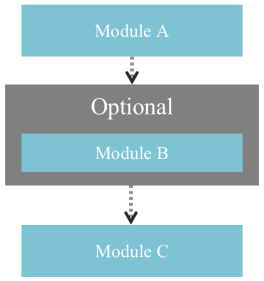
ParameterSpace can be divided into two categories: mutable and immutable. The mutable parameter space is always followed with user defined values. For example, `Real(0.1, 0.9)`, `Choice(['relu','tanh'])`, etc. The immutable includes `Constant`,`Dynamic`, and `Cascade`.

- Mutable: `Int`, `Real`, `Bool`, `Choice`, `MultipleChoice`
- Immutable: `Constant`, `Dynamic`, `Cascade`

1.3.3 Connection Space

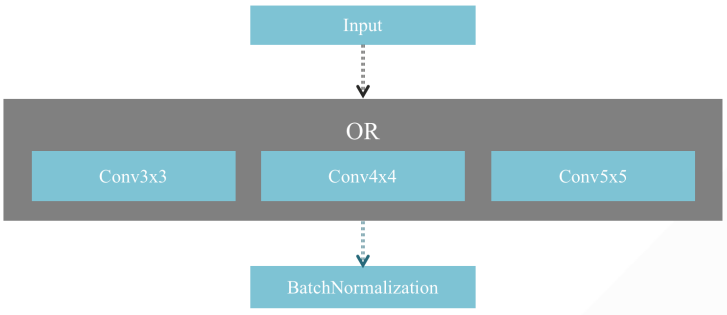
ConnectionSpace is used to represent possible connections between modules.

- Optional: Whether to insert a module between two modules.



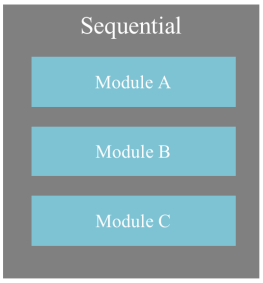

```
from hypernets.frameworks.keras.layers import Dense, Input, Dropout
from hypernets.core.ops import Optional

module_a = Input()
module_b = Dropout()
optional_dropout = Optional(module_b, keep_link=True)(module_a)
module_c = Dense()(optional_dropout)
```

- ModuleChoice: There are several optional modules on a node, choose one of them.



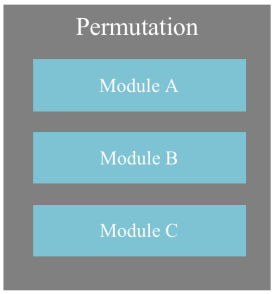
```
from hypernets.frameworks.keras.layers import Input, BatchNormalization
from hypernets.core.ops import ModuleChoice

module_a = Input()
or_conv_pool = ModuleChoice([sepconv5x5(),sepconv3x3(),avgpooling3x3()])(module_a)
module_c = BatchNormalization()(or_conv_pool)
```

- Sequential: A series of modules connected in order.




```
from hypernets.frameworks.keras.layers import Dense, Input, BatchNormalization
from hypernets.core.ops import Sequential

module_a = Input()
module_b = Dense()
module_c = BatchNormalization()
Sequential([module_a,module_b,module_c])
```

- Permutation: Choose one of the possible permutations and connect the modules in order.



Possible permutations：
A -> B -> C
A -> C -> B
B -> A -> C
B -> C -> A
C -> A -> B
C -> B -> A

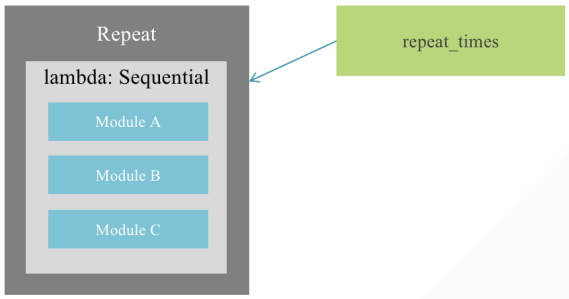
```
from hypernets.frameworks.keras.layers import Dense, BatchNormalization, Dropout,␣
↪Activation
from hypernets.core.search_space import Choice
from hypernets.core.ops import Permutation, Sequential, Optional

dense = Dense(units=Choice([100, 300, 500, 1000]))
act = Activation(activation=Choice(['relu', 'tanh']))
optional_bn = Optional(BatchNormalization(), keep_link=True)
dropout = Dropout(rate=Choice([0, 0.1, 0.2, 0.3, 0.5]))

# Use `Permutation` to try different arrangements of act, optional_bn, dropout
# optional_bn is optional module and will be skipped when hp_use_bn is False
perm_act_bn_dropout = Permutation([act, optional_bn, dropout])
```

(continues on next page)

(continued from previous page)

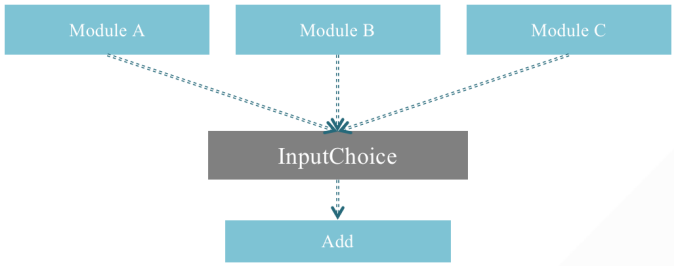
```
# Use `Sequential` to connect dense and perm_act_bn_dropout in order
seq = Sequential([dense, perm_act_bn_dropout])
```

- Repeat: The module is repeated multiple times and connected in order.




```
from hypernets.frameworks.keras.layers import Dense, BatchNormalization, Dropout,␣
↪Activation
from hypernets.core.search_space import Choice
from hypernets.core.ops import Permutation, Sequential, Optional, Repeat

dense = Dense(units=Choice([100, 300, 500, 1000]))
act = Activation(activation=Choice(['relu', 'tanh']))
optional_bn = Optional(BatchNormalization(), keep_link=True)
dropout = Dropout(rate=Choice([0, 0.1, 0.2, 0.3, 0.5]))

# Use `Permutation` to try different arrangements of act, optional_bn, dropout
# optional_bn is optional module and will be skipped when hp_use_bn is False
perm_act_bn_dropout = Permutation([act, optional_bn, dropout])

repeat_seq = Repeat(module_fn=lambda :Sequential([dense, perm_act_bn_dropout]),␣
↪repeat_times=Choice([2,3,4]))
```

- InputChoice: A modules has multiple upstream modules, choose one or more connections.




```
from hypernets.frameworks.keras.layers import Input,Dense, BatchNormalization,␣
↪Dropout, Activation
from hypernets.core.search_space import Choice
from hypernets.core.ops import Permutation, Sequential, Optional, Repeat, InputChoice

module_a = Input()
module_b = Input()
```

(continues on next page)

(continued from previous page)

```
module_c = Input()
ic = InputChoice(inputs=[module_a,module_b,module_a], num_chosen_most=2)([module_a,
↪module_b,module_c])
add = Add()(ic)
```

1.3.4 Module Space

ModuleSpace represents an optimizable function with a set of hyper-parameters. Unlike ParameterSpace and ConnectionSpace, ModuleSpace is related to frameworks or libraries. For different `HyperModel`, it is necessary to define a corresponding series of ModuleSpace.

1.4 Searchers

1.4.1 Single-objective Optimization

MCTSSearcher

Monte-Carlo Tree Search (MCTS) extends the celebrated Multi-armed Bandit algorithm to tree-structured search spaces. The MCTS algorithm iterates over four phases: selection, expansion, playout and backpropagation.

- Selection: In each node of the tree, the child node is selected after a Multi-armed Bandit strategy, e.g. the UCT (Upper Confidence bound applied to Trees) algorithm.
- Expansion: The algorithm adds one or more nodes to the tree. This node corresponds to the first encountered position that was not added in the tree.
- Playout: When reaching the limits of the visited tree, a roll-out strategy is used to select the options until reaching a terminal node and computing the associated reward.
- Backpropagation: The reward value is propagated back, i.e. it is used to update the value associated to all nodes along the visited path up to the root node.

Code example

```
from hypernets.searchers import MCTSSearcher

searcher = MCTSSearcher(search_space_fn, use_meta_learner=False, max_node_space=10,␣
↪candidates_size=10, optimize_direction='max')
```

Required Parameters

- *space_fn*: callable, A search space function which when called returns a `HyperSpace` instance.

Optional Parameters

- *policy*: hypernets.searchers.mcts_core.BasePolicy, (default=None), The policy for *Selection* and *Backpropagation* phases, `UCT` by default.
- *max_node_space*: int, (default=10), Maximum space for node expansion
- *use_meta_learner*: bool, (default=True), Meta-learner aims to evaluate the performance of unseen samples based on previously evaluated samples. It provides a practical solution to accurately estimate a search branch with many simulations without involving the actual training.
- *candidates_size*: int, (default=10), The number of samples for the meta-learner to evaluate candidate paths when roll out

- *optimize_direction*: 'min' or 'max', (default='min'), Whether the search process is approaching the maximum or minimum reward value.
- *space_sample_validation_fn*: callable or None, (default=None), Used to verify the validity of samples from the search space, and can be used to add specific constraint rules to the search space to reduce the size of the space.

EvolutionSearcher

Evolutionary algorithm (EA) is a subset of evolutionary computation, a generic population-based metaheuristic optimization algorithm. An EA uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection. Candidate solutions to the optimization problem play the role of individuals in a population, and the fitness function determines the quality of the solutions (see also loss function). Evolution of the population then takes place after the repeated application of the above operators.

Code example

```
from hypernets.searchers import EvolutionSearcher

searcher = EvolutionSearcher(search_space_fn, population_size=20, sample_size=5,␣
↪optimize_direction='min')
```

Required Parameters

- *space_fn*: callable, A search space function which when called returns a `HyperSpace` instance
- *population_size*: int, Size of population
- *sample_size*: int, The number of parent candidates selected in each cycle of evolution

Optional Parameters

- *regularized*: bool, (default=False), Whether to enable regularized
- *use_meta_learner*: bool, (default=True), Meta-learner aims to evaluate the performance of unseen samples based on previously evaluated samples. It provides a practical solution to accurately estimate a search branch with many simulations without involving the actual training.
- *candidates_size*: int, (default=10), The number of samples for the meta-learner to evaluate candidate paths when roll out
- *optimize_direction*: 'min' or 'max', (default='min'), Whether the search process is approaching the maximum or minimum reward value.
- *space_sample_validation_fn*: callable or None, (default=None), Used to verify the validity of samples from the search space, and can be used to add specific constraint rules to the search space to reduce the size of the space.

RandomSearcher

As its name suggests, Random Search uses random combinations of hyperparameters.

Code example

```
from hypernets.searchers import RandomSearcher
searcher = RandomSearcher(search_space_fn, optimize_direction='min')
```

Required Parameters

- *space_fn*: callable, A search space function which when called returns a `HyperSpace` instance

Optional Parameters

- *optimize_direction*: 'min' or 'max', (default='min'), Whether the search process is approaching the maximum or minimum reward value.
- *space_sample_validation_fn*: callable or None, (default=None), Used to verify the validity of samples from the search space, and can be used to add specific constraint rules to the search space to reduce the size of the space.

1.4.2 Multi-objective optimization

NSGA-II: Non-dominated Sorting Genetic Algorithm

NSGA-II is a dominate-based genetic algorithm used for multi-objective optimization. It rank individuals into levels according to the dominance relationship then calculate crowded-distance within a level. The ranking levels and crowded-distance are used to sort individuals in population and keep population size to be stable.

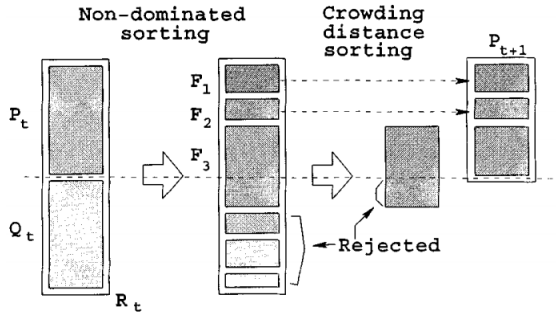


Fig. 2. NSGA-II procedure.

NSGAIISearcher code example:

```
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.preprocessing import LabelEncoder
>>> from hypernets.core.random_state import set_random_state, get_random_state
>>> from hypernets.examples.plain_model import PlainSearchSpace, PlainModel
>>> from hypernets.model.objectives import create_objective
>>> from hypernets.searchers.genetic import create_recombination
>>> from hypernets.searchers.nsga_searcher import NSGAIISearcher
>>> from hypernets.tabular.datasets import dsutils
>>> from hypernets.tabular.sklearn_ex import MultiLabelEncoder
>>> from hypernets.utils import logging as hyn_logging
>>> hyn_logging.set_level(hyn_logging.WARN)
>>> set_random_state(1234)
```

```
>>> df = dsutils.load_bank().head(1000)
>>> df['y'] = LabelEncoder().fit_transform(df['y'])
>>> df.drop(['id'], axis=1, inplace=True)
>>> X_train, X_test = train_test_split(df, test_size=0.2, random_state=1234)
>>> y_train = X_train.pop('y')
>>> y_test = X_test.pop('y')
>>> random_state = get_random_state()
>>> search_space = PlainSearchSpace(enable_dt=True, enable_lr=False, enable_nn=True)
```

```
>>> rs = NSGAIISearcher(search_space,  objectives=[create_objective('auc'), create_
↪objective('nf')],
>>>                     recombination=create_recombination('single_point', random_
↪state=random_state),
```

(continues on next page)

(continued from previous page)

```
>>>                     population_size=5,
>>>                     random_state=random_state)
>>> rs
NSGAIISearcher(objectives=[PredictionObjective(name=auc, scorer=make_scorer(roc_auc_
↪score, needs_threshold=True), direction=max), NumOfFeatures(name=nf, sample_
↪size=2000, direction=min)], recombination=SinglePointCrossOver(random_
↪state=RandomState(MT19937))), mutation=SinglePointMutation(random_
↪state=RandomState(MT19937), proba=0.7)), survival=<hypernets.searchers.nsga_
↪searcher.RankAndCrowdSortSurvival object at 0x000002851D8A4910>), random_
↪state=RandomState(MT19937)
```

```
>>> hk = PlainModel(rs, task='binary', transformer=MultiLabelEncoder)
>>> hk.search(X_train, y_train, X_test, y_test, max_trials=10)
>>> rs.get_population()[:3]
[NSGAIndividual(scores=[0.768788682581786, 0.125], rank=0, n=0, distance=inf),
 NSGAIndividual(scores=[0.7992926613616268, 0.1875], rank=0, n=0, distance=inf),
 NSGAIndividual(scores=[0.617816091954023, 0.1875], rank=1, n=0, distance=inf)]
```

References:

[1] Deb, Kalyanmoy, et al. "A fast and elitist multiobjective genetic algorithm: NSGA-II." IEEE transactions on evolutionary computation 6.2 (2002): 182-197.

MOEA/D: Multiobjective Evolutionary Algorithm Based on Decomposition

MOEA/D is a decomposition-based genetic algorithm framework used for multi-objective optimization. It decomposes multi-objective optimization problem into several sub optimization problem in different directions. One an excellent solution for a sub problem is obtained it will share the genes with it's neighbors since the neighboring sub problems are similar, thus, this mechanism can accelerate convergence process. One more thing, it's a framework that can support several decomposition approaches for different situations, now we supported:

- Weighted Sum Approach: straight and effective approach
- Tchebycheff Approach : working in case of the solution space is concavity
- Penalty-based boundary intersection approach(PBI): suitable for high-dimensional solution spaces

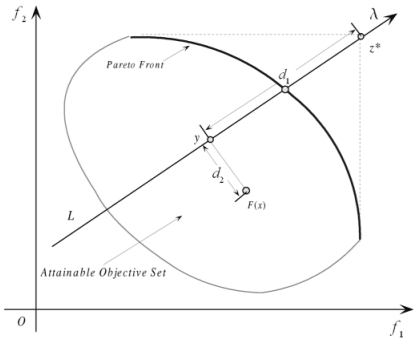


Fig. 2. Illustration of the penalty-based boundary intersection approach.

MOEADSearcher code example:

```
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.preprocessing import LabelEncoder
>>> from hypernets.core.random_state import set_random_state, get_random_state
```

(continues on next page)

(continued from previous page)

```
>>> from hypernets.examples.plain_model import PlainSearchSpace, PlainModel
>>> from hypernets.model.objectives import create_objective
>>> from hypernets.searchers.genetic import create_recombination
>>> from hypernets.searchers.moead_searcher import MOEADSearcher
>>> from hypernets.tabular.datasets import dsutils
>>> from hypernets.tabular.sklearn_ex import MultiLabelEncoder
>>> from hypernets.utils import logging as hyn_logging
>>> hyn_logging.set_level(hyn_logging.WARN)
>>> set_random_state(1234)
```

```
>>> df = dsutils.load_bank().head(1000)
>>> df['y'] = LabelEncoder().fit_transform(df['y'])
>>> df.drop(['id'], axis=1, inplace=True)
>>> X_train, X_test = train_test_split(df, test_size=0.2, random_state=1234)
>>> y_train = X_train.pop('y')
>>> y_test = X_test.pop('y')
```

```
>>> random_state = get_random_state()
>>> search_space = PlainSearchSpace(enable_dt=True, enable_lr=False, enable_nn=True)
>>> rs = MOEADSearcher(search_space,  objectives=[create_objective('logloss'), create_
↪objective('nf')],
>>>                    recombination=create_recombination('single_point', random_
↪state=random_state),
>>>                    random_state=random_state)
>>> rs
MOEADSearcher(objectives=[PredictionObjective(name=logloss, scorer=make_scorer(log_
↪loss, needs_proba=True), direction=min), NumOfFeatures(name=nf, sample_size=2000,␣
↪direction=min)], n_neighbors=2, recombination=SinglePointCrossOver(random_
↪state=RandomState(MT19937)), mutation=SinglePointMutation(random_
↪state=RandomState(MT19937), proba=0.3), population_size=6)
```

```
>>> hk = PlainModel(rs, task='binary', transformer=MultiLabelEncoder)
>>> hk.search(X_train, y_train, X_test, y_test, max_trials=10)
>>> rs.get_population()[:3]
[Individual(dna=DAG_HyperSpace_1, scores=[10.632877749789559, 0.1875], random_
↪state=RandomState(MT19937)),
 Individual(dna=DAG_HyperSpace_1, scores=[0.4372370852623173, 1.0], random_
↪state=RandomState(MT19937)),
 Individual(dna=DAG_HyperSpace_1, scores=[6.494675998141714, 0.6875], random_
↪state=RandomState(MT19937))]
```

References:

[1] Zhang, Qingfu, and Hui Li. “MOEA/D: A multiobjective evolutionary algorithm based on decomposition.” IEEE Transactions on evolutionary computation 11.6 (2007): 712-731.

R-Dominance: dominance relation for multicriteria decision making

R-NSGA-II is a variant of NSGA-II, used for multi-objective optimization but considering the decision preferences of decision-makers(DMs). It comprehensively considers the pareto dominance relationship and the reference points provided by DMs to search for non-dominated solutions near reference points to assist users in making decisions.

RNSGAIISearcher code example:

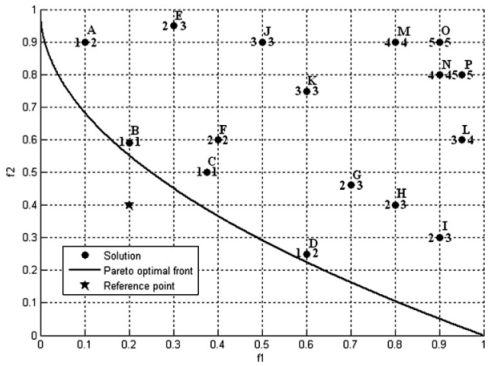


Fig. 3. Non-ε-dominated sorting ($\delta = 0.3$).

```
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.preprocessing import LabelEncoder
>>> from hypernets.core.random_state import set_random_state, get_random_state
>>> from hypernets.examples.plain_model import PlainSearchSpace, PlainModel
>>> from hypernets.model.objectives import create_objective
>>> from hypernets.searchers.genetic import create_recombination
>>> from hypernets.searchers.nsga_searcher import RNSGAIISearcher
>>> from hypernets.tabular.datasets import dsutils
>>> from hypernets.tabular.sklearn_ex import MultiLabelEncoder
>>> from hypernets.utils import logging as hyn_logging
>>> hyn_logging.set_level(hyn_logging.WARN)
>>> set_random_state(1234)
```

```
>>> df = dsutils.load_bank().head(1000)
>>> df['y'] = LabelEncoder().fit_transform(df['y'])
>>> df.drop(['id'], axis=1, inplace=True)
>>> X_train, X_test = train_test_split(df, test_size=0.2, random_state=1234)
>>> y_train = X_train.pop('y')
>>> y_test = X_test.pop('y')
```

```
>>> random_state = get_random_state()
>>> search_space = PlainSearchSpace(enable_dt=True, enable_lr=False, enable_nn=True)
>>> rs = RNSGAIISearcher(search_space,  objectives=[create_objective('logloss'),␣
↪create_objective('nf')],
>>>                      ref_point=np.array([0.2, 0.3]),
>>>                      recombination=create_recombination('single_point', random_
↪state=random_state),
>>>                      random_state=random_state)
>>> rs
RNSGAIISearcher(objectives=[PredictionObjective(name=logloss, scorer=make_scorer(log_
↪loss, needs_proba=True), direction=min), NumOfFeatures(name=nf, sample_size=2000,␣
↪direction=min)], recombination=SinglePointCrossOver(random_
↪state=RandomState(MT19937))), mutation=SinglePointMutation(random_
↪state=RandomState(MT19937), proba=0.7)), survival=RDominanceSurvival(ref_point=[0.2␣
↪0.3], weights=[0.5, 0.5], threshold=0.3, random_state=RandomState(MT19937))),␣
↪random_state=RandomState(MT19937)
```

```
>>> hk = PlainModel(rs, task='binary', transformer=MultiLabelEncoder)
>>> hk.search(X_train, y_train, X_test, y_test, max_trials=10)
```

(continues on next page)

(continued from previous page)

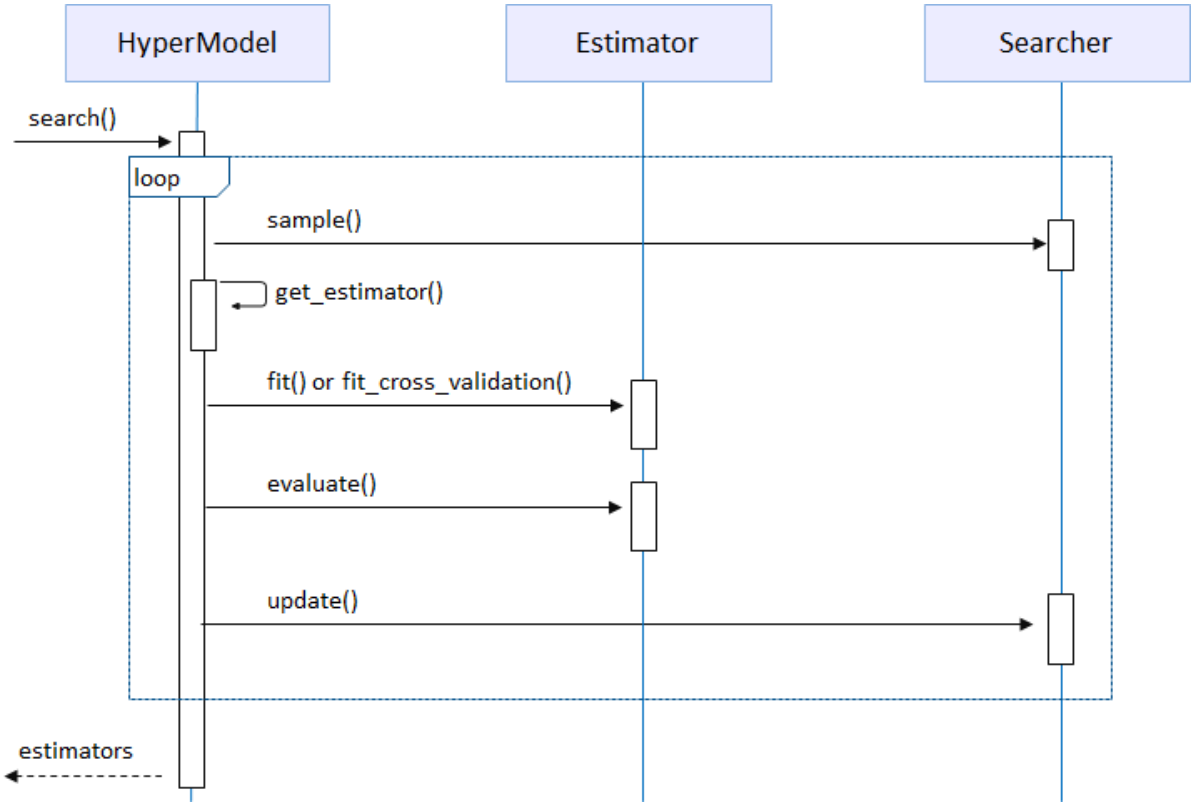
```
>>> rs.get_population()[:3]
[NSGAIndividual(scores=[10.632877749789559, 0.1875], rank=-1, n=-1, distance=-1.0),
 NSGAIndividual(scores=[0.4372370852623173, 1.0], rank=-1, n=-1, distance=-1.0),
 NSGAIndividual(scores=[6.494675998141714, 0.6875], rank=-1, n=-1, distance=-1.0)]
```

References:

[1] Said, Lamjed Ben, Slim Bechikh, and Khaled Ghédira. "The r-dominance: a new dominance relation for interactive evolutionary multicriteria decision making." IEEE transactions on Evolutionary Computation 14.5 (2010): 801-818.

1.5 HyperModel

HyperModel is an abstract class that needs to implement a dedicated HyperModel for different frameworks or domains. HyperModel explore hyper-parameters sample from Searcher, fit and evaluate Estimator, then reward the metric score to Searcher for optimization. The figure below shows HyperModel search sequence.



1.5.1 Customize HyperModel

To customize HyerModel, two components are required:

- HyperModel: subclass of *hypernets.model.HyperModel*, create newer Estimator instance with searched space sample, and load trained estimator from storage.
- Estimator: subclass of *hypernets.model.Estimator*, the core component for model fitting/evaluation/prediction/persistence.

You can reference *hypernets.examples.plain_model.PlainModel* and *hypernets.examples.plain_model.PlainEstimator* as start point. See DeepTables, HyperGBM, HyperKeras for more details.

1.6 Neural Architecture Search

Deep Learning has enabled remarkable progress over the last years on a variety of tasks, such as CV, NLP, and machine translation. It is crucial to discover novel neural architectures. However, currently most of them have been developed manually by human experts. Neural Architecture Search (NAS) has emerged as a promising tool to alleviate human effort in this trial and error design process.

NAS has demonstrated much success in automating neural network architecture design for various tasks, such as image recognition and language modeling. Representative works include NASNet, ENAS, ProxylessNAS, DARTS, One-Shot NAS, Regularized Evolution, AlphaX, etc.

However, most of these works are usually for specific use-cases, and their search space, search strategy and estimation strategy are often intertwined, making it difficult to reuse the code and make further innovations on it.

In Hypernets, we propose an abstract architecture, fully decoupling Search Space, Search Strategy, and Performance Estimation Strategy so that each part is relatively independent and can be reused to accelerate innovations and engineering of NAS algorithms.

The three problems of NAS: Search Space, Search Strategy, and Performance Estimation Strategy, which correspond to `HyperSpace`, `Searcher`, and `Estimator` in Hypernets respectively.

We'll use some definitions from HyperKeras in this chapter's examples. Install it with `pip`:

```
pip install hyperkeras
```

1.6.1 Define A DNN Search Space

```
# define a DNN search space
from hyperkeras.layers import Dense, Input, BatchNormalization, Dropout, Activation
from hypernets.core.ops import Permutation, Sequential, Optional, Repeat
from hypernets.core.search_space import HyperSpace, Bool, Choice, Real, Dynamic
import itertools


def dnn_block(hp_dnn_units, hp_reduce_factor, hp_seq, hp_use_bn, hp_dropout, hp_
↪activation, step):
    # The value of a `Dynamic` is computed as a function of the values of the␣
↪ParameterSpace it depends on.
    block_units = Dynamic(
        lambda_fn=lambda units, reduce_factor: units if step == 0 else units *␣
↪(reduce_factor ** step),
        units=hp_dnn_units, reduce_factor=hp_reduce_factor)

    dense = Dense(units=block_units)
    act = Activation(activation=hp_activation)
    optional_bn = Optional(BatchNormalization(), keep_link=True, hp_opt=hp_use_bn)
    dropout = Dropout(rate=hp_dropout)

    # Use `Permutation` to try different arrangements of act, optional_bn, dropout
    # optional_bn is optional module and will be skipped when hp_use_bn is False
    perm_act_bn_dropout = Permutation([act, optional_bn, dropout], hp_seq=hp_seq)
```

(continues on next page)

(continued from previous page)

```
    # Use `Sequential` to connect dense and perm_act_bn_dropout in order
    seq = Sequential([dense, perm_act_bn_dropout])
    return seq


def dnn_search_space(input_shape, output_units, output_activation, units_choices=[200,
↪ 500, 1000],
                     reduce_facotr_choices=[1, 0.8, 0.5], layer_num_choices=[2, 3, 4],
                     activation_choices=['relu', 'tanh']):
    space = HyperSpace()
    with space.as_default():
        hp_dnn_units = Choice(units_choices)
        hp_reduce_factor = Choice(reduce_facotr_choices)
        hp_use_bn = Bool()
        if len(activation_choices) == 1:
            hp_activation = activation_choices[0]
        else:
            hp_activation = Choice(activation_choices)

        hp_dropout = Real(0., 0.5, step=0.1)
        hp_seq = Choice([seq for seq in itertools.permutations(range(3))])

        input = Input(shape=input_shape)
        backbone = Repeat(
            lambda step: dnn_block(hp_dnn_units, hp_reduce_factor, hp_seq, hp_use_bn,␣
↪hp_dropout, hp_activation, step),
            repeat_times=layer_num_choices)(input)
        output = Dense(units=output_units, activation=output_activation)(backbone)
    return space


# Search the best model in search space defined above
from hypernets.searchers.random_searcher import RandomSearcher
from hypernets.core.callbacks import SummaryCallback
from hyperkeras.hyper_keras import HyperKeras
import numpy as np

rs = RandomSearcher(lambda: dnn_search_space(input_shape=10, output_units=2, output_
↪activation='sigmoid'),
                    optimize_direction='max')
hk = HyperKeras(rs, optimizer='adam', loss='sparse_categorical_crossentropy',␣
↪metrics=['accuracy'],
                callbacks=[SummaryCallback()])

x = np.random.randint(0, 10000, size=(100, 10))
y = np.random.randint(0, 2, size=(100), dtype='int')

hk.search(x, y, x, y, max_trials=3)
assert hk.get_best_trial()
```

1.6.2 Define A CNN Search Space

```
import itertools

from hyperkeras.layers import Dense, Input, BatchNormalization, Activation, Conv2D,␣
↪MaxPooling2D, AveragePooling2D, Flatten
```

(continues on next page)

(continued from previous page)

```
from hypernets.core.search_space import HyperSpace, Bool, Choice, Dynamic
from hypernets.core.ops import Permutation, Sequential, Optional, Repeat, ModuleChoice


def conv_block(block_no, hp_pooling, hp_filters, hp_kernel_size, hp_bn_act, hp_use_bn,
↪ hp_activation, strides=(1, 1)):
    def conv_bn(step):
        conv = Conv2D(filters=conv_filters, kernel_size=hp_kernel_size,␣
↪strides=strides, padding='same')
        act = Activation(activation=hp_activation)
        optional_bn = Optional(BatchNormalization(), keep_link=True, hp_opt=hp_use_bn)

        # Use `Permutation` to try different arrangements of act, optional_bn
        # optional_bn is optional module and will be skipped when hp_use_bn is False
        perm_act_bn = Permutation([optional_bn, act], hp_seq=hp_bn_act)
        seq = Sequential([conv, perm_act_bn])
        return seq

    if block_no < 2:
        repeat_num_choices = [2]
        multiplier = 1
    else:
        repeat_num_choices = [3, 4, 5]
        multiplier = 2 ** (block_no - 1)

    conv_filters = Dynamic(lambda filters: filters * multiplier, filters=hp_filters)
    conv = Repeat(conv_bn, repeat_times=repeat_num_choices)
    pooling = ModuleChoice([MaxPooling2D(padding='same'), AveragePooling2D(padding=
↪'same')], hp_or=hp_pooling)
    block = Sequential([conv, pooling])
    return block


def cnn_search_space(input_shape, output_units, output_activation='softmax', block_
↪num_choices=[2, 3, 4, 5, 6],
                     activation_choices=['relu'], filters_choices=[32, 64], kernel_
↪size_choices=[(1, 1), (3, 3)]):
    space = HyperSpace()
    with space.as_default():
        hp_use_bn = Bool()
        hp_pooling = Choice(list(range(2)))
        hp_filters = Choice(filters_choices)
        hp_kernel_size = Choice(kernel_size_choices)
        hp_fc_units = Choice([1024, 2048, 4096])
        if len(activation_choices) == 1:
            hp_activation = activation_choices[0]
        else:
            hp_activation = Choice(activation_choices)
        hp_bn_act = Choice([seq for seq in itertools.permutations(range(2))])

        input = Input(shape=input_shape)
        blocks = Repeat(
            lambda step: conv_block(
                block_no=step,
                hp_pooling=hp_pooling,
                hp_filters=hp_filters,
                hp_kernel_size=hp_kernel_size,
```

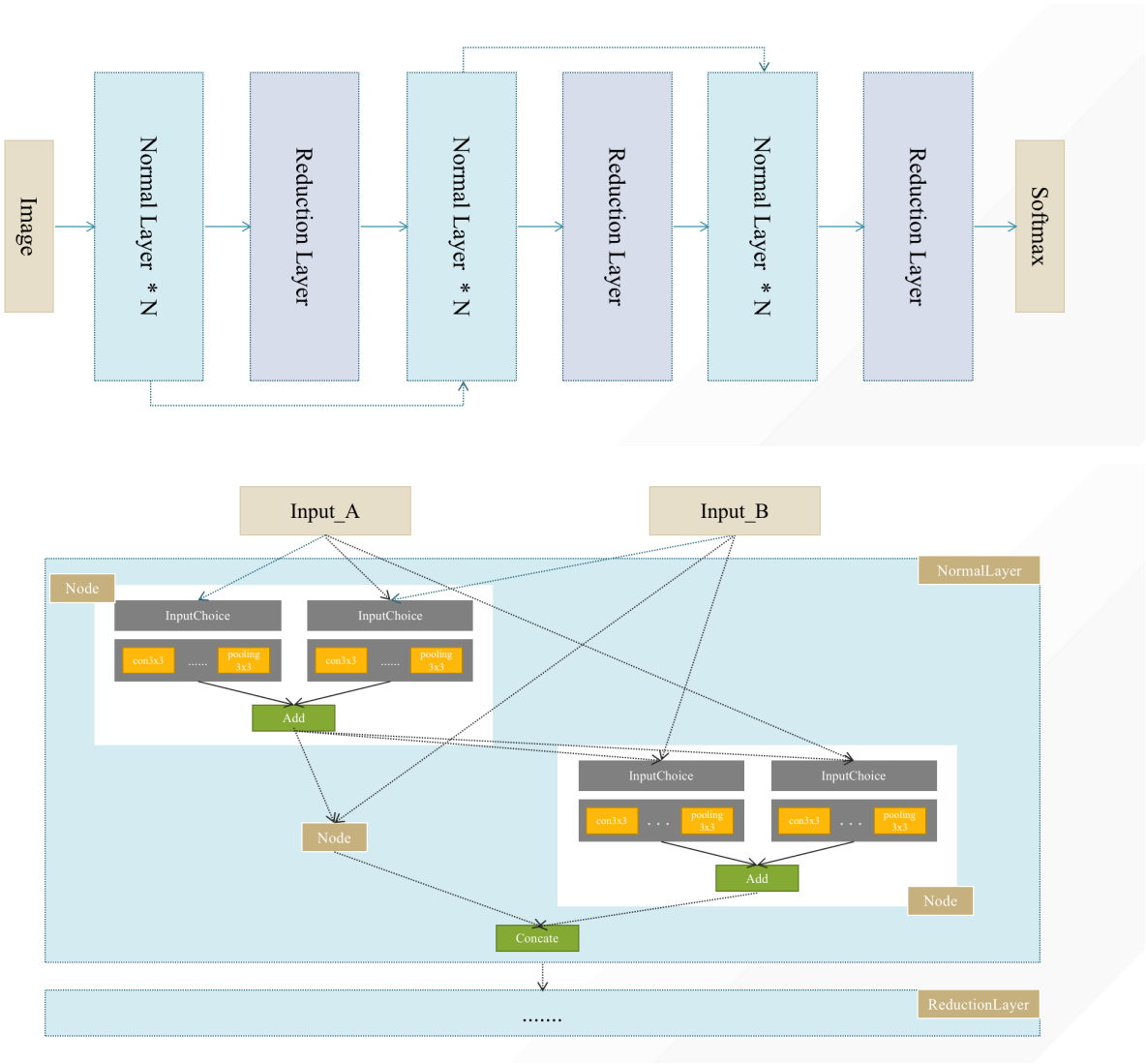
(continues on next page)

(continued from previous page)

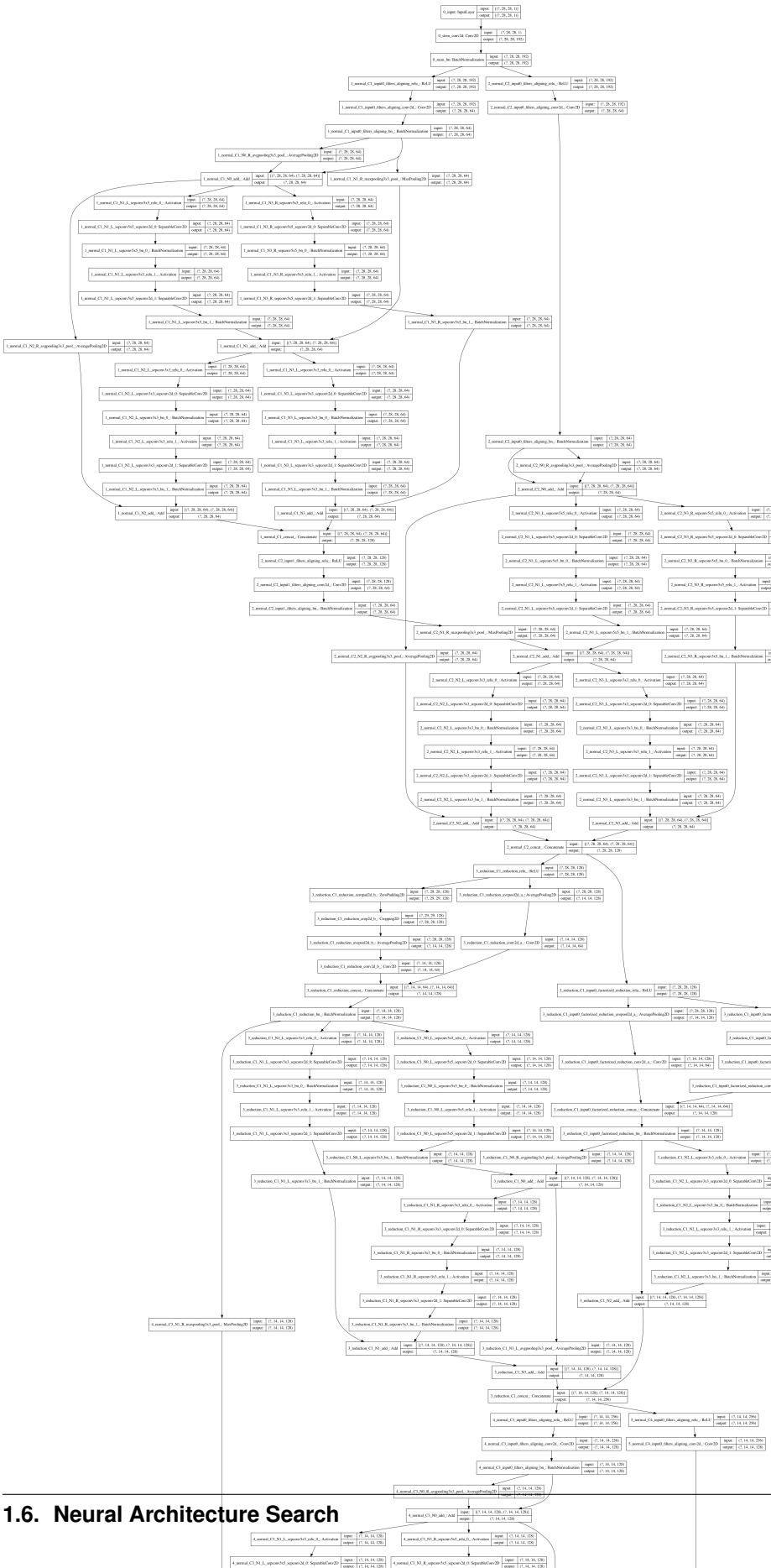
```
                hp_use_bn=hp_use_bn,
                hp_activation=hp_activation,
                hp_bn_act=hp_bn_act),
            repeat_times=block_num_choices)(input)
        x = Flatten()(blocks)
        x = Dense(units=hp_fc_units, activation=hp_activation, name='fc1')(x)
        x = Dense(units=hp_fc_units, activation=hp_activation, name='fc2')(x)
        x = Dense(output_units, activation=output_activation, name='predictions')(x)
    return space


# Search the best model in search space defined above on mnist dataset

from hypernets.searchers.random_searcher import RandomSearcher
from hypernets.core.callbacks import SummaryCallback
from hyperkeras.hyper_keras import HyperKeras
import numpy as np
import tensorflow as tf

rs = RandomSearcher(
    lambda: cnn_search_space(input_shape=(28, 28, 1),
                             output_units=10,
                             output_activation='softmax',
                             block_num_choices=[2, 3, 4, 5],
                             filters_choices=[32, 64, 128],
                             kernel_size_choices=[(1, 1), (3, 3)]),
    optimize_direction='max')
hk = HyperKeras(rs, optimizer='adam', loss='categorical_crossentropy', metrics=[
↪'accuracy'],
                callbacks=[SummaryCallback()])

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Rescale the images from [0,255] to the [0.0,1.0] range.
x_train, x_test = x_train[..., np.newaxis] / 255.0, x_test[..., np.newaxis] / 255.0
y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)

hk.search(x_train, y_train, x_test, y_test, max_trials=10, epochs=10)
assert hk.get_best_trial()
```

1.6.3 Define An ENAS Micro Search Space

- Illustration of ENAS



- An ENAS architecture sample



```
# define an ENAS micro search space
from hyperkeras.search_space.enas_layers import SafeMerge, Identity, CalibrateSize
from hyperkeras.layers import BatchNormalization, Activation, Add, MaxPooling2D,␣
↪AveragePooling2D, \
    SeparableConv2D, Conv2D, GlobalAveragePooling2D, Dense, Dropout
from hypernets.core.ops import ModuleChoice, InputChoice, ConnectLooseEnd
from hypernets.core.search_space import ModuleSpace


def sepconv2d_bn(no, name_prefix, kernel_size, filters, strides=(1, 1), data_
↪format=None, x=None):
    relu = Activation(activation='relu', name=f'{name_prefix}relu_{no}_')
    if x is not None and isinstance(x, ModuleSpace):
        relu(x)
    sepconv2d = SeparableConv2D(
        filters=filters,
        kernel_size=kernel_size,
        strides=strides,
        padding='same',
        data_format=data_format,
        name=f'{name_prefix}sepconv2d_{no}'
    )(relu)
    bn = BatchNormalization(name=f'{name_prefix}bn_{no}_')(sepconv2d)
    return bn


def sepconv3x3(name_prefix, filters, strides=(1, 1), data_format=None):
    name_prefix = name_prefix + 'sepconv3x3_'
    sep1 = sepconv2d_bn(0, name_prefix, kernel_size=(3, 3), filters=filters,␣
↪strides=strides, data_format=data_format)
    sep2 = sepconv2d_bn(1, name_prefix, kernel_size=(3, 3), filters=filters,␣
↪strides=strides, data_format=data_format,
                        x=sep1)
    return sep2


def sepconv5x5(name_prefix, filters, strides=(1, 1), data_format=None):
    name_prefix = name_prefix + 'sepconv5x5_'
    sep1 = sepconv2d_bn(0, name_prefix, kernel_size=(5, 5), filters=filters,␣
↪strides=strides, data_format=data_format)
    sep2 = sepconv2d_bn(1, name_prefix, kernel_size=(5, 5), filters=filters,␣
↪strides=strides, data_format=data_format,
                        x=sep1)
    return sep2


def maxpooling3x3(name_prefix, filters, strides=(1, 1), data_format=None):
    name_prefix = name_prefix + 'maxpooling3x3_'
    max = MaxPooling2D(pool_size=(3, 3), strides=strides, padding='same', data_
↪format=data_format,
                       name=f'{name_prefix}pool_')
    return max


def avgpooling3x3(name_prefix, filters, strides=(1, 1), data_format=None):
    name_prefix = name_prefix + 'avgpooling3x3_'
    avg = AveragePooling2D(pool_size=(3, 3), strides=strides, padding='same', data_
↪format=data_format,
```

(continues on next page)

(continued from previous page)

```
                            name=f'{name_prefix}pool_')
    return avg


def identity(name_prefix):
    return Identity(name=f'{name_prefix}identity')


def add(x1, x2, name_prefix, filters):
    return Add(name=f'{name_prefix}add_')([x1, x2])


def conv_cell(hp_dict, type, cell_no, node_no, left_or_right, inputs, filters, is_
↪reduction=False, data_format=None):
    assert isinstance(inputs, list)
    assert all([isinstance(m, ModuleSpace) for m in inputs])
    name_prefix = f'{type}_C{cell_no}_N{node_no}_{left_or_right}_'

    input_choice_key = f'{type[2:]}_N{node_no}_{left_or_right}_input_choice'
    op_choice_key = f'{type[2:]}_N{node_no}_{left_or_right}_op_choice'
    hp_choice = hp_dict.get(input_choice_key)
    ic1 = InputChoice(inputs, 1, hp_choice=hp_choice)(inputs)
    if hp_choice is None:
        hp_dict[input_choice_key] = ic1.hp_choice

    # hp_strides = Dynamic(lambda_fn=lambda choice: (2, 2) if is_reduction and␣
↪choice[0] <= 1 else (1, 1),
    #                      choice=ic1.hp_choice)
    hp_strides = (1, 1)
    hp_or = hp_dict.get(op_choice_key)
    or1 = ModuleChoice([sepconv5x5(name_prefix, filters, strides=hp_strides, data_
↪format=data_format),
                        sepconv3x3(name_prefix, filters, strides=hp_strides, data_
↪format=data_format),
                        avgpooling3x3(name_prefix, filters, strides=hp_strides, data_
↪format=data_format),
                        maxpooling3x3(name_prefix, filters, strides=hp_strides, data_
↪format=data_format),
                        identity(name_prefix)], hp_or=hp_or)(ic1)

    if hp_or is None:
        hp_dict[op_choice_key] = or1.hp_or

    return or1


def conv_node(hp_dict, type, cell_no, node_no, inputs, filters, is_reduction=False,␣
↪data_format=None):
    op_left = conv_cell(hp_dict, type, cell_no, node_no, 'L', inputs, filters, is_
↪reduction, data_format)
    op_right = conv_cell(hp_dict, type, cell_no, node_no, 'R', inputs, filters, is_
↪reduction, data_format)
    name_prefix = f'{type}_C{cell_no}_N{node_no}_'
    return add(op_left, op_right, name_prefix, filters)


def conv_layer(hp_dict, type, cell_no, inputs, filters, node_num, is_reduction=False,␣
↪data_format=None):
```

(continues on next page)

(continued from previous page)

```
    name_prefix = f'{type}_C{cell_no}_'

    if inputs[0] == inputs[1]:
        c1 = c2 = CalibrateSize(0, filters, name_prefix, data_format)(inputs[0])
    else:
        c1 = CalibrateSize(0, filters, name_prefix, data_format)(inputs)
        c2 = CalibrateSize(1, filters, name_prefix, data_format)(inputs)
    inputs = [c1, c2]
    all_nodes = [c1, c2]
    for node_no in range(node_num):
        node = conv_node(hp_dict, type, cell_no, node_no, inputs, filters, is_
↪reduction, data_format)
        inputs.append(node)
        all_nodes.append(node)
    cle = ConnectLooseEnd(all_nodes)(all_nodes)
    concat = SafeConcatenate(filters, name_prefix, name=name_prefix + 'concat_')(cle)
    return concat


def stem_op(input, filters, data_format=None):
    conv = Conv2D(
        filters=filters * 3,
        kernel_size=(3, 3),
        strides=(1, 1),
        padding='same',
        data_format=data_format,
        name=f'0_stem_conv2d'
    )(input)
    bn = BatchNormalization(name=f'0_stem_bn')(conv)
    return bn


def auxiliary_head():
    pass


def classfication(x, classes, dropout_rate=0, data_format=None):
    x = Activation('relu')(x)
    x = GlobalAveragePooling2D(data_format=data_format)(x)
    if dropout_rate > 0:
        x = Dropout(dropout_rate)(x)
    x = Dense(classes, activation='softmax')(x)
    return x


from hyperkeras.search_space.enas_common_ops import *
from hyperkeras.layers import Input
from hyperkeras.search_space.enas_layers import FactorizedReduction
from hypernets.core.search_space import HyperSpace


def enas_micro_search_space(arch='NRNR', input_shape=(28, 28, 1), init_filters=64,
                            node_num=4, data_format=None,
                            classes=10, classification_dropout=0,
                            hp_dict={}):
    space = HyperSpace()
    with space.as_default():
```

(continues on next page)

(continued from previous page)

```
        input = Input(shape=input_shape, name='0_input')
        stem = stem_op(input, init_filters, data_format)
        node0 = stem
        node1 = stem
        # node0 = input
        # node1 = input
        reduction_no = 0
        normal_no = 0

        for l in arch:
            if l == 'N':
                normal_no += 1
                type = 'normal'
                cell_no = normal_no
                is_reduction = False
            else:
                reduction_no += 1
                type = 'reduction'
                cell_no = reduction_no
                is_reduction = True
            filters = (2 ** reduction_no) * init_filters

            if is_reduction:
                node0 = node1
                node1 = FactorizedReduction(filters, f'{type}_C{cell_no}_', data_
↪format)(node1)
            x = conv_layer(hp_dict, f'{normal_no + reduction_no}_{type}', cell_no,␣
↪[node0, node1], filters, node_num,
                           is_reduction)
            node0 = node1
            node1 = x
        logit = classfication(x, classes, classification_dropout, data_format)
        space.set_inputs(input)
    return space


# Search the best model in search space defined above on mnist dataset
import tensorflow as tf
import numpy as np

from hypernets.core.callbacks import SummaryCallback
from hypernets.core.ops import *
from hyperkeras.search_space.enas_micro import enas_micro_search_space
from hyperkeras.hyper_keras import HyperKeras
from hypernets.searchers.random_searcher import RandomSearcher

rs = RandomSearcher(
    lambda: enas_micro_search_space(arch='NR', hp_dict={}),
    optimize_direction='max')
hk = HyperKeras(rs, optimizer='adam', loss='categorical_crossentropy', metrics=[
↪'accuracy'],
                callbacks=[SummaryCallback()])

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Rescale the images from [0,255] to the [0.0,1.0] range.
x_train, x_test = x_train[..., np.newaxis] / 255.0, x_test[..., np.newaxis] / 255.0
```

(continues on next page)

(continued from previous page)

```
y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)
print("Number of original training examples:", len(x_train))
print("Number of original test examples:", len(x_test))

# sample for speed up
samples = 10000
hk.search(x_train[:samples], y_train[:samples], x_test[:int(samples / 10)], y_
↪test[:int(samples / 10)],
          max_trials=10, epochs=3)
assert hk.get_best_trial()
```

Console output:

```
/Users/jack/opt/anaconda3/envs/hypernets/bin/python /Users/jack/workspace/aps/
↪Hypernets/tests/keras/run_enas.py
/Users/jack/opt/anaconda3/envs/hypernets/lib/python3.6/site-packages/lightgbm/__init__
↪.py:48: UserWarning: Starting from version 2.2.1, the library file in distribution␣
↪wheels for macOS is built by the Apple Clang (Xcode_8.3.3) compiler.
This means that in case of installing LightGBM from PyPI via the ``pip install␣
↪lightgbm`` command, you don't need to install the gcc compiler anymore.
Instead of that, you need to install the OpenMP library, which is required for␣
↪running LightGBM on the system with the Apple Clang compiler.
You can install the OpenMP library by the following command: ``brew install libomp``.
  "You can install the OpenMP library by the following command: ``brew install␣
↪libomp``.", UserWarning)
Number of original training examples: 60000
Number of original test examples: 10000
Initialize Meta Learner: dataset_id:(10000, 28, 28, 1)
2020-07-09 12:39:44.508558: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your␣
↪CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2␣
↪FMA
2020-07-09 12:39:44.524010: I tensorflow/compiler/xla/service/service.cc:168] XLA␣
↪service 0x7fedc7d8bc50 initialized for platform Host (this does not guarantee that␣
↪XLA will be used). Devices:
2020-07-09 12:39:44.524026: I tensorflow/compiler/xla/service/service.cc:176]  ␣
↪StreamExecutor device (0): Host, Default Version

Trial No:1

----------------------------------------------------------------
(0) Module_InputChoice_1.hp_choice:                         [0]
(1) Module_InputChoice_2.hp_choice:                         [0]
(2) Module_ModuleChoice_1.hp_or:                                1
(3) Module_ModuleChoice_2.hp_or:                                1
(4) Module_InputChoice_3.hp_choice:                         [1]
(5) Module_InputChoice_4.hp_choice:                         [2]
(6) Module_ModuleChoice_3.hp_or:                                1
(7) Module_ModuleChoice_4.hp_or:                                3
(8) Module_InputChoice_5.hp_choice:                         [1]
(9) Module_InputChoice_6.hp_choice:                         [2]
(10) Module_ModuleChoice_5.hp_or:                               1
(11) Module_ModuleChoice_6.hp_or:                               2
(12) Module_InputChoice_7.hp_choice:                        [1]
(13) Module_InputChoice_8.hp_choice:                        [0]
(14) Module_ModuleChoice_7.hp_or:                               3
(15) Module_ModuleChoice_8.hp_or:                               1
```

(continues on next page)

(continued from previous page)

```
(16) Module_InputChoice_10.hp_choice:                    [1]
(17) Module_InputChoice_9.hp_choice:                     [0]
(18) Module_ModuleChoice_10.hp_or:                           0
(19) Module_ModuleChoice_9.hp_or:                            4
(20) Module_InputChoice_11.hp_choice:                    [2]
(21) Module_InputChoice_12.hp_choice:                    [0]
(22) Module_ModuleChoice_11.hp_or:                           1
(23) Module_ModuleChoice_12.hp_or:                           4
(24) Module_InputChoice_13.hp_choice:                    [2]
(25) Module_InputChoice_14.hp_choice:                    [3]
(26) Module_ModuleChoice_13.hp_or:                           1
(27) Module_ModuleChoice_14.hp_or:                           3
(28) Module_InputChoice_15.hp_choice:                    [2]
(29) Module_InputChoice_16.hp_choice:                    [1]
(30) Module_ModuleChoice_15.hp_or:                           3
(31) Module_ModuleChoice_16.hp_or:                           0
----------------------------------------------------------------
Model: "model"
_________________________________________________________________
↪_____________
Layer (type)                 Output Shape              Param #     Connected to
↪
=================================================================
0_input (InputLayer)         [(None, 28, 28, 1)]       0
↪
_________________________________________________________________
↪_____________
0_stem_conv2d (Conv2D)       (None, 28, 28, 192)       1920        0_input[0][0]
↪
_________________________________________________________________
↪_____________
0_stem_bn (BatchNormalization)  (None, 28, 28, 192)    768         0_stem_conv2d[0][0]
↪
_________________________________________________________________
↪_____________
1_normal_C1_input0_filters_alig (None, 28, 28, 192)    0           0_stem_bn[0][0]
↪
_________________________________________________________________
↪_____________
1_normal_C1_input0_filters_alig (None, 28, 28, 64)     12352       1_normal_C1_input0_
↪filters_aligni
_________________________________________________________________
↪_____________
1_normal_C1_input0_filters_alig (None, 28, 28, 64)     256         1_normal_C1_input0_
↪filters_aligni
_________________________________________________________________
↪_____________
1_normal_C1_N0_L_sepconv3x3_rel (None, 28, 28, 64)     0           1_normal_C1_input0_
↪filters_aligni
_________________________________________________________________
↪_____________
1_normal_C1_N0_R_sepconv3x3_rel (None, 28, 28, 64)     0           1_normal_C1_input0_
↪filters_aligni
_________________________________________________________________
↪_____________
1_normal_C1_N0_L_sepconv3x3_sep (None, 28, 28, 64)     4736        1_normal_C1_N0_L_
↪sepconv3x3_relu_
```

(continues on next page)

(continued from previous page)

```
_________________________________________________________________
↪_____________
1_normal_C1_N0_R_sepconv3x3_sep (None, 28, 28, 64)   4736        1_normal_C1_N0_R_
↪sepconv3x3_relu_
_________________________________________________________________
↪_____________
1_normal_C1_N1_L_sepconv3x3_rel (None, 28, 28, 64)   0           1_normal_C1_input0_
↪filters_aligni
_________________________________________________________________
↪_____________
1_normal_C1_N0_L_sepconv3x3_bn_ (None, 28, 28, 64)   256         1_normal_C1_N0_L_
↪sepconv3x3_sepco
_________________________________________________________________
↪_____________
1_normal_C1_N0_R_sepconv3x3_bn_ (None, 28, 28, 64)   256         1_normal_C1_N0_R_
↪sepconv3x3_sepco
_________________________________________________________________
↪_____________
1_normal_C1_N2_L_sepconv3x3_rel (None, 28, 28, 64)   0           1_normal_C1_input0_
↪filters_aligni
_________________________________________________________________
↪_____________
1_normal_C1_N3_R_sepconv3x3_rel (None, 28, 28, 64)   0           1_normal_C1_input0_
↪filters_aligni
_________________________________________________________________
↪_____________
1_normal_C1_N1_L_sepconv3x3_sep (None, 28, 28, 64)   4736        1_normal_C1_N1_L_
↪sepconv3x3_relu_
_________________________________________________________________
↪_____________
1_normal_C1_N0_L_sepconv3x3_rel (None, 28, 28, 64)   0           1_normal_C1_N0_L_
↪sepconv3x3_bn_0_
_________________________________________________________________
↪_____________
1_normal_C1_N0_R_sepconv3x3_rel (None, 28, 28, 64)   0           1_normal_C1_N0_R_
↪sepconv3x3_bn_0_
_________________________________________________________________
↪_____________
1_normal_C1_N2_L_sepconv3x3_sep (None, 28, 28, 64)   4736        1_normal_C1_N2_L_
↪sepconv3x3_relu_
_________________________________________________________________
↪_____________
1_normal_C1_N3_R_sepconv3x3_sep (None, 28, 28, 64)   4736        1_normal_C1_N3_R_
↪sepconv3x3_relu_
_________________________________________________________________
↪_____________
1_normal_C1_N1_L_sepconv3x3_bn_ (None, 28, 28, 64)   256         1_normal_C1_N1_L_
↪sepconv3x3_sepco
_________________________________________________________________
↪_____________
1_normal_C1_N0_L_sepconv3x3_sep (None, 28, 28, 64)   4736        1_normal_C1_N0_L_
↪sepconv3x3_relu_
_________________________________________________________________
↪_____________
1_normal_C1_N0_R_sepconv3x3_sep (None, 28, 28, 64)   4736        1_normal_C1_N0_R_
↪sepconv3x3_relu_
_________________________________________________________________
↪_____________
```

(continues on next page)

(continued from previous page)

```
1_normal_C1_N2_L_sepconv3x3_bn_ (None, 28, 28, 64)   256         1_normal_C1_N2_L_
↪sepconv3x3_sepco
_____________
↪_____________
1_normal_C1_N3_R_sepconv3x3_bn_ (None, 28, 28, 64)   256         1_normal_C1_N3_R_
↪sepconv3x3_sepco
_____________
↪_____________
1_normal_C1_N1_L_sepconv3x3_rel (None, 28, 28, 64)   0           1_normal_C1_N1_L_
↪sepconv3x3_bn_0_
_____________
↪_____________
1_normal_C1_N0_L_sepconv3x3_bn_ (None, 28, 28, 64)   256         1_normal_C1_N0_L_
↪sepconv3x3_sepco
_____________
↪_____________
1_normal_C1_N0_R_sepconv3x3_bn_ (None, 28, 28, 64)   256         1_normal_C1_N0_R_
↪sepconv3x3_sepco
_____________
↪_____________
1_normal_C1_N2_L_sepconv3x3_rel (None, 28, 28, 64)   0           1_normal_C1_N2_L_
↪sepconv3x3_bn_0_
_____________
↪_____________
1_normal_C1_N3_R_sepconv3x3_rel (None, 28, 28, 64)   0           1_normal_C1_N3_R_
↪sepconv3x3_bn_0_
_____________
↪_____________
1_normal_C1_N1_L_sepconv3x3_sep (None, 28, 28, 64)   4736        1_normal_C1_N1_L_
↪sepconv3x3_relu_
_____________
↪_____________
1_normal_C1_N0_add_ (Add)       (None, 28, 28, 64)   0           1_normal_C1_N0_L_
↪sepconv3x3_bn_1_
                                                                 1_normal_C1_N0_R_
↪sepconv3x3_bn_1_
_____________
↪_____________
1_normal_C1_N2_L_sepconv3x3_sep (None, 28, 28, 64)   4736        1_normal_C1_N2_L_
↪sepconv3x3_relu_
_____________
↪_____________
1_normal_C1_N3_R_sepconv3x3_sep (None, 28, 28, 64)   4736        1_normal_C1_N3_R_
↪sepconv3x3_relu_
_____________
↪_____________
1_normal_C1_N1_L_sepconv3x3_bn_ (None, 28, 28, 64)   256         1_normal_C1_N1_L_
↪sepconv3x3_sepco
_____________
↪_____________
1_normal_C1_N1_R_maxpooling3x3_ (None, 28, 28, 64)   0           1_normal_C1_N0_add_
↪[0][0]
_____________
↪_____________
1_normal_C1_N2_L_sepconv3x3_bn_ (None, 28, 28, 64)   256         1_normal_C1_N2_L_
↪sepconv3x3_sepco
_____________
↪_____________
```

(continues on next page)

(continued from previous page)

```
1_normal_C1_N2_R_avgpooling3x3_ (None, 28, 28, 64)   0           1_normal_C1_N0_add_
↪[0][0]
_____________________________________________________________________________
↪_____________
1_normal_C1_N3_L_maxpooling3x3_ (None, 28, 28, 64)   0           1_normal_C1_input0_
↪filters_aligni
_____________________________________________________________________________
↪_____________
1_normal_C1_N3_R_sepconv3x3_bn_ (None, 28, 28, 64)   256         1_normal_C1_N3_R_
↪sepconv3x3_sepco
_____________________________________________________________________________
↪_____________
1_normal_C1_N1_add_ (Add)       (None, 28, 28, 64)   0           1_normal_C1_N1_L_
↪sepconv3x3_bn_1_
                                                                 1_normal_C1_N1_R_
↪maxpooling3x3_po
_____________________________________________________________________________
↪_____________
1_normal_C1_N2_add_ (Add)       (None, 28, 28, 64)   0           1_normal_C1_N2_L_
↪sepconv3x3_bn_1_
                                                                 1_normal_C1_N2_R_
↪avgpooling3x3_po
_____________________________________________________________________________
↪_____________
1_normal_C1_N3_add_ (Add)       (None, 28, 28, 64)   0           1_normal_C1_N3_L_
↪maxpooling3x3_po
                                                                 1_normal_C1_N3_R_
↪sepconv3x3_bn_1_
_____________________________________________________________________________
↪_____________
1_normal_C1_concat_ (Concatenat (None, 28, 28, 192)  0           1_normal_C1_N1_add_
↪[0][0]
                                                                 1_normal_C1_N2_add_
↪[0][0]
                                                                 1_normal_C1_N3_add_
↪[0][0]
_____________________________________________________________________________
↪_____________
reduction_C1_relu_ (ReLU)       (None, 28, 28, 192)  0           1_normal_C1_concat_
↪[0][0]
_____________________________________________________________________________
↪_____________
reduction_C1_reduction_zeropad2 (None, 29, 29, 192)  0           reduction_C1_relu_
↪[0][0]
_____________________________________________________________________________
↪_____________
reduction_C1_reduction_crop2d_b (None, 28, 28, 192)  0           reduction_C1_
↪reduction_zeropad2d_
_____________________________________________________________________________
↪_____________
reduction_C1_reduction_avepool2 (None, 14, 14, 192)  0           reduction_C1_relu_
↪[0][0]
_____________________________________________________________________________
↪_____________
reduction_C1_reduction_avepool2 (None, 14, 14, 192)  0           reduction_C1_
↪reduction_crop2d_b_[
_____________________________________________________________________________
↪_____________
```

(continues on next page)

(continued from previous page)

```
reduction_C1_reduction_conv2d_a (None, 14, 14, 64)   12352       reduction_C1_
↪reduction_avepool2d_
_____________
↪_____________
reduction_C1_reduction_conv2d_b (None, 14, 14, 64)   12352       reduction_C1_
↪reduction_avepool2d_
_____________
↪_____________
reduction_C1_reduction_concat_  (None, 14, 14, 128)  0           reduction_C1_
↪reduction_conv2d_a_[
                                                                 reduction_C1_
↪reduction_conv2d_b_[
_____________
↪_____________
2_reduction_C1_relu_ (ReLU)     (None, 28, 28, 192)  0           1_normal_C1_concat_
↪[0][0]
_____________
↪_____________
reduction_C1_reduction_bn_ (Bat (None, 14, 14, 128)  512         reduction_C1_
↪reduction_concat_[0]
_____________
↪_____________
2_reduction_C1_input0_factorize (None, 29, 29, 192)  0           2_reduction_C1_relu_
↪[0][0]
_____________
↪_____________
2_reduction_C1_N0_R_sepconv5x5_ (None, 14, 14, 128)  0           reduction_C1_
↪reduction_bn_[0][0]
_____________
↪_____________
2_reduction_C1_input0_factorize (None, 28, 28, 192)  0           2_reduction_C1_
↪input0_factorized_
_____________
↪_____________
2_reduction_C1_N0_R_sepconv5x5_ (None, 14, 14, 128)  19712       2_reduction_C1_N0_R_
↪sepconv5x5_re
_____________
↪_____________
2_reduction_C1_input0_factorize (None, 14, 14, 192)  0           2_reduction_C1_relu_
↪[0][0]
_____________
↪_____________
2_reduction_C1_input0_factorize (None, 14, 14, 192)  0           2_reduction_C1_
↪input0_factorized_
_____________
↪_____________
2_reduction_C1_N0_R_sepconv5x5_ (None, 14, 14, 128)  512         2_reduction_C1_N0_R_
↪sepconv5x5_se
_____________
↪_____________
2_reduction_C1_input0_factorize (None, 14, 14, 64)   12352       2_reduction_C1_
↪input0_factorized_
_____________
↪_____________
2_reduction_C1_input0_factorize (None, 14, 14, 64)   12352       2_reduction_C1_
↪input0_factorized_
_____________
↪_____________
```

(continues on next page)

(continued from previous page)

```
2_reduction_C1_N0_R_sepconv5x5_ (None, 14, 14, 128)  0           2_reduction_C1_N0_R_
↪sepconv5x5_bn
_____________
↪_____________
2_reduction_C1_input0_factorize (None, 14, 14, 128)  0           2_reduction_C1_
↪input0_factorized_
                                                                 2_reduction_C1_
↪input0_factorized_
_____________
↪_____________
2_reduction_C1_N0_R_sepconv5x5_ (None, 14, 14, 128)  19712       2_reduction_C1_N0_R_
↪sepconv5x5_re
_____________
↪_____________
2_reduction_C1_input0_factorize (None, 14, 14, 128)  512         2_reduction_C1_
↪input0_factorized_
_____________
↪_____________
2_reduction_C1_N0_R_sepconv5x5_ (None, 14, 14, 128)  512         2_reduction_C1_N0_R_
↪sepconv5x5_se
_____________
↪_____________
2_reduction_C1_N0_add_ (Add)     (None, 14, 14, 128)  0           2_reduction_C1_
↪input0_factorized_
                                                                 2_reduction_C1_N0_R_
↪sepconv5x5_bn
_____________
↪_____________
2_reduction_C1_N1_L_sepconv3x3_ (None, 14, 14, 128)  0           2_reduction_C1_N0_
↪add_[0][0]
_____________
↪_____________
2_reduction_C1_N1_L_sepconv3x3_ (None, 14, 14, 128)  17664       2_reduction_C1_N1_L_
↪sepconv3x3_re
_____________
↪_____________
2_reduction_C1_N2_L_sepconv3x3_ (None, 14, 14, 128)  0           2_reduction_C1_N0_
↪add_[0][0]
_____________
↪_____________
2_reduction_C1_N1_L_sepconv3x3_ (None, 14, 14, 128)  512         2_reduction_C1_N1_L_
↪sepconv3x3_se
_____________
↪_____________
2_reduction_C1_N3_R_sepconv5x5_ (None, 14, 14, 128)  0           reduction_C1_
↪reduction_bn_[0][0]
_____________
↪_____________
2_reduction_C1_N2_L_sepconv3x3_ (None, 14, 14, 128)  17664       2_reduction_C1_N2_L_
↪sepconv3x3_re
_____________
↪_____________
2_reduction_C1_N1_L_sepconv3x3_ (None, 14, 14, 128)  0           2_reduction_C1_N1_L_
↪sepconv3x3_bn
_____________
↪_____________
2_reduction_C1_N3_R_sepconv5x5_ (None, 14, 14, 128)  19712       2_reduction_C1_N3_R_
↪sepconv5x5_re
```

(continues on next page)

(continued from previous page)

```
_____
↪_____________
2_reduction_C1_N2_L_sepconv3x3_ (None, 14, 14, 128)  512         2_reduction_C1_N2_L_
↪sepconv3x3_se
_____
↪_____________
2_reduction_C1_N1_L_sepconv3x3_ (None, 14, 14, 128)  17664       2_reduction_C1_N1_L_
↪sepconv3x3_re
_____
↪_____________
2_reduction_C1_N3_R_sepconv5x5_ (None, 14, 14, 128)  512         2_reduction_C1_N3_R_
↪sepconv5x5_se
_____
↪_____________
2_reduction_C1_N2_L_sepconv3x3_ (None, 14, 14, 128)  0           2_reduction_C1_N2_L_
↪sepconv3x3_bn
_____
↪_____________
2_reduction_C1_N1_L_sepconv3x3_ (None, 14, 14, 128)  512         2_reduction_C1_N1_L_
↪sepconv3x3_se
_____
↪_____________
2_reduction_C1_N3_R_sepconv5x5_ (None, 14, 14, 128)  0           2_reduction_C1_N3_R_
↪sepconv5x5_bn
_____
↪_____________
2_reduction_C1_N2_L_sepconv3x3_ (None, 14, 14, 128)  17664       2_reduction_C1_N2_L_
↪sepconv3x3_re
_____
↪_____________
2_reduction_C1_N1_add_ (Add)    (None, 14, 14, 128)  0           2_reduction_C1_N1_L_
↪sepconv3x3_bn
                                                                 2_reduction_C1_
↪input0_factorized_
_____
↪_____________
2_reduction_C1_N3_R_sepconv5x5_ (None, 14, 14, 128)  19712       2_reduction_C1_N3_R_
↪sepconv5x5_re
_____
↪_____________
2_reduction_C1_N2_L_sepconv3x3_ (None, 14, 14, 128)  512         2_reduction_C1_N2_L_
↪sepconv3x3_se
_____
↪_____________
2_reduction_C1_N2_R_maxpooling3 (None, 14, 14, 128)  0           2_reduction_C1_N1_
↪add_[0][0]
_____
↪_____________
2_reduction_C1_N3_L_maxpooling3 (None, 14, 14, 128)  0           2_reduction_C1_N0_
↪add_[0][0]
_____
↪_____________
2_reduction_C1_N3_R_sepconv5x5_ (None, 14, 14, 128)  512         2_reduction_C1_N3_R_
↪sepconv5x5_se
_____
↪_____________
2_reduction_C1_N2_add_ (Add)    (None, 14, 14, 128)  0           2_reduction_C1_N2_L_
↪sepconv3x3_bn
```

(continues on next page)

(continued from previous page)

```
                                                                    2_reduction_C1_N2_R_
↪maxpooling3x3
_________________________________________________________________
↪_____________
2_reduction_C1_N3_add_ (Add)     (None, 14, 14, 128)  0           2_reduction_C1_N3_L_
↪maxpooling3x3
                                                                    2_reduction_C1_N3_R_
↪sepconv5x5_bn
_________________________________________________________________
↪_____________
2_reduction_C1_concat_ (Concate (None, 14, 14, 256)  0           2_reduction_C1_N2_
↪add_[0][0]
                                                                    2_reduction_C1_N3_
↪add_[0][0]
_________________________________________________________________
↪_____________
Module_Activation_1 (Activation (None, 14, 14, 256)  0           2_reduction_C1_
↪concat_[0][0]
_________________________________________________________________
↪_____________
Module_GlobalAveragePooling2D_1 (None, 256)          0           Module_Activation_
↪1[0][0]
_________________________________________________________________
↪_____________
Module_Dense_1 (Dense)           (None, 10)           2570        Module_
↪GlobalAveragePooling2D_1[0
==================================================================================================
Total params: 271,818
Trainable params: 267,466
Non-trainable params: 4,352
_________________________________________________________________
↪_____________
trial begin
Train on 10000 samples
Epoch 1/3
10000/10000 [==============================] - 322s 32ms/sample - loss: 0.3171 -␣
↪accuracy: 0.9157
Epoch 2/3
10000/10000 [==============================] - 321s 32ms/sample - loss: 0.0870 -␣
↪accuracy: 0.9755
Epoch 3/3
10000/10000 [==============================] - 319s 32ms/sample - loss: 0.0584 -␣
↪accuracy: 0.9825
1000/1000 [==============================] - 8s 8ms/sample - loss: 0.3346 - accuracy:␣
↪0.8820
trial end. reward:0.8820000290870667, improved:True, elapsed:971.3707249164581
Total elapsed:971.4891712665558
----------------------------------------------------------------
space signatures: {'76001d37233837206b28fd9999cb2e75'}
----------------------------------------------------------------

Trial No:2
......
```

1.6.4 References

[1] Pham H, Guan M Y, Zoph B, et al. Efficient neural architecture search via parameter sharing[J]. arXiv preprint arXiv:1802.03268, 2018.

[2] Elsken T, Metzen J H, Hutter F. Neural architecture search: A survey[J]. arXiv preprint arXiv:1808.05377, 2018.

1.7 Experiment

Experiment is the playground to prepare training and testing data, and search the optimized estimator with HyperModel. Use *experiment* with the following steps:

- Prepare datasets
  - training data(X_train, y_train): required
  - evaluation data(X_eval, y_eval): optional, will be split from training data if not provided
  - testing data(X_test only): optional, the sample of the prediction data, is used to optimize the experiment
- Create HyperModel instance with your selected implementation
  - The *PlainModel* is provided as a plain implementation in this project.
  - More implementations can be found from DeepTables, HyperGBM, HyperKeras, etc.
- Create experiment instance with the prepared datasets and HyperModel instance and other arguments required by your selected experiment implementation (*GeneralExperiment* , *CompeteExperiment* , or your customized implementation).
- Call experiment's *.run()* to search the optimized estimator.

1.7.1 GeneralExperiment

GeneralExperiment is the basic implementation of experiment. Example code:

```
from sklearn.model_selection import train_test_split

from hypernets.experiment import GeneralExperiment
from hypernets.searchers import make_searcher
from hypernets.tabular.datasets import dsutils


def create_hyper_model(reward_metric='auc', optimize_direction='max'):
    from hypernets.examples.plain_model import PlainModel, PlainSearchSpace

    search_space = PlainSearchSpace()
    searcher = make_searcher('random', search_space_fn=search_space, optimize_
↪direction=optimize_direction)
    hyper_model = PlainModel(searcher=searcher, reward_metric=reward_metric,␣
↪callbacks=[])

    return hyper_model


def general_experiment_with_heart_disease():
    hyper_model = create_hyper_model()
```

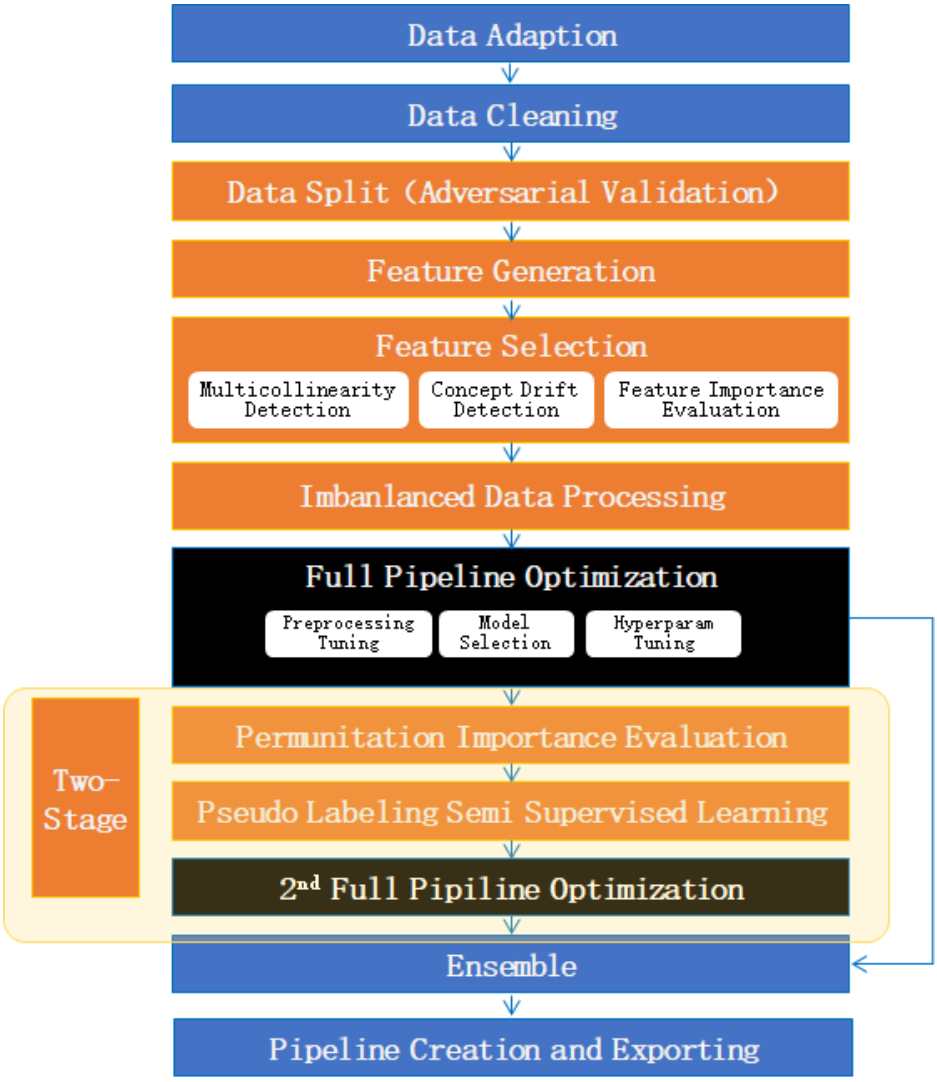
(continues on next page)

(continued from previous page)

```
    X = dsutils.load_heart_disease_uci()
    y = X.pop('target')

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

    experiment = GeneralExperiment(hyper_model, X_train, y_train, eval_size=0.3)
    estimator = experiment.run(max_trials=5)

    score = estimator.evaluate(X_test, y_test, metrics=['auc', 'accuracy', 'f1',
↪'recall', 'precision'])
    print('evaluate score:', score)


if __name__ == '__main__':
    general_experiment_with_heart_disease()
```

1.7.2 CompeteExperiment

CompeteExperiment is an implementation of experiment with many advanced features for tabular data, such as data cleaning, feature generation, feature selection, semi-supervised machine learning, two-stages searching etc.



Quick start

Basically, use *CompeteExperiment* with default settings just like *GeneralExperiment*.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

from hypernets.experiment import CompeteExperiment
from hypernets.tabular.datasets import dsutils
from hypernets.tabular.metrics import calc_score


def create_hyper_model(reward_metric='auc', optimize_direction='max'):
    from hypernets.core.callbacks import SummaryCallback
    from hypernets.examples.plain_model import PlainModel, PlainSearchSpace
    from hypernets.searchers import make_searcher
    from hypernets.tabular.sklearn_ex import MultiLabelEncoder

    search_space = PlainSearchSpace(enable_dt=True, enable_lr=True, enable_nn=False)
```

(continues on next page)

(continued from previous page)

```
    searcher = make_searcher('random', search_space_fn=search_space, optimize_
↪direction=optimize_direction)
    hyper_model = PlainModel(searcher=searcher, reward_metric=reward_metric,␣
↪callbacks=[SummaryCallback()],
                             transformer=MultiLabelEncoder)

    return hyper_model


def experiment_with_bank_data(row_count=3000):
    X = dsutils.load_bank()
    if row_count is not None:
        X = X.head(row_count)
    X['y'] = LabelEncoder().fit_transform(X['y'])
    y = X.pop('y')

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
↪state=9527)

    experiment = CompeteExperiment(create_hyper_model(), X_train, y_train, max_
↪trials=10)
    estimator = experiment.run()

    preds = estimator.predict(X_test)
    proba = estimator.predict_proba(X_test)

    score = calc_score(y_test, preds, proba, metrics=['auc', 'accuracy', 'f1', 'recall
↪', 'precision'])
    print('evaluate score:', score)
    assert score


if __name__ == '__main__':
    experiment_with_bank_data()
```

Set the Number of Search Trials

One can set the max search trial number by adjusting *max_trials*.

The following codes set the max trial times as 300:

```
hyper_model = create_hyperModel()
experiment = CompeteExperiment(hyper_model, max_trials=300, ...)
```

Use Cross Validation

Users can apply cross validation in the experiment by manually setting parameter *cv*. Setting *cv* as 'False' will lead the experiment to avoid using cross validation and apply train_test_split instead. On the other hand, when *cv* is *True*, the experiment will use cross validation where the number of folds can be adjusted through the parameter *num_folds*. The default value of *num_folds* is 3.

Example code when *cv=True*:

```
hyper_model = create_hyperModel()
experiment = CompeteExperiment(hyper_model, cv=True, num_folds=5, ...)
```

Evaluation dataset

When *cv=False*, training model will require evaluating its performance additionally on evaluation dataset. This can be done by setting *X_eval* and *y_eval* when creating *CompeteExperiment*. For example:

```
df = dsutils.load_blood()
X = df.copy()
y = X.pop(target)
X_train, X_eval, y_train, y_eval = train_test_split(X, y , test_size=0.3)
hyper_model = create_hyperModel()
experiment = CompeteExperiment(hyper_model, X_train=X_train,y_train=y_train, X_eval=X_
↪eval, y_eval=y_eval, ...)
```

If the *X_eval* or *y_eval* is None, the experiment object will split the *X_train* and *y_train* to get an evaluation dataset, whose size can be adjusted by setting *eval_size*:

```
df = dsutils.load_blood()
X = df.copy()
y = X.pop(target)
hyper_model = create_hyperModel()
experiment = CompeteExperiment(hyper_model, X_train=X, y_train=y, eval_size=0.3, ...)
```

Set the Evaluation Criterion

The default evaluation criterion is *accuracy* for classification task is, and *rmse* for regression task. Other criterions can be set by *reward_metric*. For example:

```
hyper_model = create_hyperModel()
experiment = CompeteExperiment(hyper_model, reward_metric='auc', ...)
```

Set the Early Stopping

One can set the early stopping strategy with settings of *early_stopping_round*, *early_stopping_time_limit* and *early_stopping_reward*.

The following code sets the max searching time as 3 hours:

```
hyper_model = create_hyperModel()
experiment = CompeteExperiment(hyper_model, max_trials=300, early_stopping_time_
↪limit=3600 * 3, ...)
```

Choose a Searcher

One can choose a specific searcher for the experiment by setting the parameter *searcher*.

```
hyper_model = create_hyperModel()
experiment = CompeteExperiment(hyper_model, searcher='random', ...)
```

Furthermore, you can customize a new searcher object for experiment, for an example:

```
from hypernets.searchers import MCTSSearcher

my_searcher = MCTSSearcher(lambda: search_space_general(n_estimators=100),
                           max_node_space=20,
                           optimize_direction='max')
hyper_model = create_hyperModel()
experiment = CompeteExperiment(hyper_model, searcher=my_searcher, ...)
```

Ensemble Models

CompeteExperiment automatically turns on the model ensemble function to get a better model when created. It will ensemble the best 20 models while the number for ensembling can be changed by setting *ensemble_size* as the following code, where *ensemble_size=0* means to disable ensembling.

```
hyper_model = create_hyperModel()
experiment = CompeteExperiment(hyper_model, ensemble_size=10, ...)
```

Data Adaption

This step supports Pandas/Cuml data types only, relevant parameters:

- data_adaption(default True). Whether to enable data adaption.
- data_adaption_memory_limit(default 0.05). If float, should be between 0.0 and 1.0 and represent the proportion of the system free memory. If int, represents the absolute byte number of memory.
- data_adaption_min_cols(default 0.3. If float, should be between 0.0 and 1.0 and represent the proportion of the original dataframe column number. If int, represents the absolute column number.
- data_adaption_target(default None)Where to run the next steps. 'cuml' or 'cuda', adapt training data into cuml datatypes and run next steps on nvidia GPU Devices. None, not change the training data types.

Data cleaning

CompeteExperiment performs data cleaning with DataCleaner in Hypernets. Note that this step can not be disabled but can be adjusted with DataCleaner in the following ways

- nan_chars value or list, (default None), replace some characters with np.nan
- correct_object_dtype bool, (default True), whether correct the data types
- drop_constant_columns bool, (default True), whether drop constant columns
- drop_duplicated_columns bool, (default False), whether delete repeated columns
- drop_idness_columns bool, (default True), whether drop id columns
- drop_label_nan_rows bool, (default True), whether drop rows with target values np.nan
- replace_inf_values (default np.nan), which values to replace np.nan with
- drop_columns list, (default None), drop which columns
- reserve_columns list, (default None), reserve which columns when performing data cleaning
- reduce_mem_usage bool, (default False), whether try to reduce the memory usage
- int_convert_to bool, (default 'float'), transform int to other typesNone for no transformation

If nan is represented by '\N' in datausers can replace '\N' back to np.nan when performing data cleaning as follows:

```
hyper_model = create_hyperModel()
experiment = CompeteExperiment(hyper_model, data_cleaner_args={'nan_chars': r'\N'}, ..
↪.)
...
```

Feature generation

CompeteExperiment is capable of performing feature generation, which can be turned on by setting *feature_generation=True* when creating experiment with *make_experiment*. There are several options:

- feature_generation_continuous_colslist (default None)), continuous feature, inferring automatically if set as None.
- feature_generation_categories_colslist (default None)), categorical feature, need to be set explicitly, *CompeteExperiment* can not perform automatic inference for this one.
- feature_generation_datetime_colslist (default None), datetime feature, inferring automatically if set as None.
- feature_generation_latlong_colslist (default None), latitude and longtitude feature, inferring automatically if set as None.
- feature_generation_text_colslist (default None), text feature, inferring automatically if set as None.
- feature_generation_trans_primitiveslist (default None), transformations for feature generation, inferring automatically if set as None.

When feature_generation_trans_primitives=None, *CompeteExperiment* will automatically infer the types used for transforming based on the default features. Specifically, different transformations will be adopted for different types:

- continuous_cols None, need to be set explicitly.
- categories_cols cross_categorical.
- datetime_cols month, week, day, hour, minute, second, weekday, is_weekend.
- latlong_cols haversine, geohash
- text_colstfidf

An example code for enabling feature generation:

```
hyper_model = create_hyperModel()
experiment = CompeteExperiment(hyper_model, feature_generation=True, ...)
```

Please see featuretools for more information.

Collinearity detection

There will often be some highly relevant features which are not informative but are more seen as noises. They are not very useful. On the contrary, the dataset will be affected by drifts of these features more heavily.

It is possible to handle these collinear features with *CompeteExperiment*. This can be simply enabled by setting *collinearity_detection=True* when creating experiment.

Example code for using collinearity detection

```
hyper_model = create_hyperModel()
experiment = CompeteExperiment(hyper_model, collinearity_detection=True, ...)
...
```

Drift detection

Concept drift is one of the major challenge for machine learning. The model will often perform worse in practice due to the fact that the data distributions will change along with time. To handle this problem, *CompeteExperiment* adopts Adversarial Validation to detect whether there is any drifted features and drop them to maintain a good performance.

To enable drift detection, one needs to set *drift_detection=True* when creating experiment and provide *X_test*.

Relevant parameters:

- drift_detection_remove_shift_variable : bool, (default=True), whether to detect the stability of every column first.
- drift_detection_variable_shift_threshold : float, (default=0.7), stability socres higher than this value will be dropped.
- drift_detection_threshold : float, (default=0.7), detecting scores higher than this value will be dropped.
- drift_detection_remove_size : float, (default=0.1), ratio of columns to be dropped.
- drift_detection_min_features : int, (default=10), the minimal number of columns to be reserved.
- drift_detection_num_folds : int, (default=5), the number of folds for cross validation.

An code example:

```
from io import StringIO
import pandas as pd
from hypergbm import make_experiment
from hypernets.tabular.datasets import dsutils

test_data = """
Recency,Frequency,Monetary,Time
2,10,2500,64
4,5,1250,23
4,9,2250,46
4,5,1250,23
4,8,2000,40
2,12,3000,82
11,24,6000,64
2,7,1750,46
4,11,2750,61
1,7,1750,57
2,11,2750,79
2,3,750,16
4,5,1250,26
2,6,1500,41
"""

df = dsutils.load_blood()
X = df.copy()
y = X.pop(target)
test_df = pd.read_csv(StringIO(test_data))
hyper_model = create_hyperModel()
experiment = CompeteExperiment(hyper_model, X_train=X, y_train=y, X_test=test_df,
                               drift_detection=True, ...)

...
```

Feature selection

CompeteExperiment evaluates the feature importance by training a pre-defined model. Then it chooses the most important ones among them to continue the model training.

To enable feature selection, one needs to set *feature_selection=True* when creating experiment. Relevant parameters:

- feature_selection_strategystr, selection strategies(default threshold), can be chose from *threshold*, *number* and *quantile*.
- feature_selection_thresholdfloat, (default 0.1), selection threshold when the strategy is *threshold*, features with scores higher than this threshold will be selected.
- feature_selection_quantilefloat, (default 0.2), selection threshold when the strategy is *quantile*, features with scores higher than this threshold will be selected.
- feature_selection_numberint or float, (default 0.8), selection numbers when the strategy is *number*.

An example code:

```
hyper_model = create_hyperModel()
experiment = CompeteExperiment(hyper_model,
                               feature_selection=True,
                               feature_selection_strategy='quantile',
                               feature_selection_quantile=0.3,
                               ...)
```

UnderSampling pre-search

Normally, hyperparameter optimization utilizes all training data. However, this will cost a huge amount of time for a large dataset. To alleviate this problem, one can perform a pre-search with only a part of data to try more model parameters in the same amount of time. Better parameters will then be used for training with the whole data to obtain the optimal parameters.

To enable feature selection, one needs to set *down_sample_search=True* when creating experiment. Relevant parameters:

- down_sample_search_sizeint, float(0.0~1.0) or dict (default 0.1), number of examples used for pre-search.
- down_sample_search_time_limitint, (default early_stopping_time_limit*0.33), time limit for pre-search.
- down_sample_search_max_trialsint, (default max_trials*3), max trail numbers for pre-search.

An example code:

```
hyper_model = create_hyperModel()
experiment = CompeteExperiment(hyper_model,
                               down_sample_search=True,
                               down_sample_search_size=0.2,
                               ...)
```

The second stage feature selection

CompeteExperiment supports continuing data processing with the trained model, which is officially called *Two-stage search*. There are two types of Two-stage processing supported by *CompeteExperiment*: Two-stage feature selection and pseudo label which will be covered in the rest of this section.

In *CompeteExperiment*, the second stage feature selection is to choose models with good performances in the first stage, and use *permutation_importance* to evaluate them to give better features.

To enable the second stage feature selection, one needs to set *feature_reselection=True* when creating experiment. Relevant parameters:

- feature_reselection_estimator_sizeint, (default=10), the number of models to be used for evaluating the importances of feature (top n best models in the first stage).
- feature_reselection_strategystr, selection strategy(default threshold), available selection strategies include *threshold*, *number*, *quantile*.
- feature_reselection_thresholdfloat, (default 1e-5), threshold when the selection strategy is *threshold*, importance scores higher than this values will be choosed.
- feature_reselection_quantilefloat, (default 0.2), threshold when the selection strategy is *quantile*, importance scores higher than this values will be choosed.
- feature_reselection_numberint or float, (default 0.8), the number of features to be selected when the strategy is *number*.

An example code:

```
hyper_model = create_hyperModel()
experiment = CompeteExperiment(hyper_model,
                               feature_reselection=True,
                               ...)
```

Please refer to scikit-learn for more information about *permutation_importance*.

Pseudo label

Pseudo label is a kind of semi-supervised machine learning method. It will assign labels predicted by the model trained in the first stage to some examples in test data. Then examples with higher confidence values than a threshold will be added into the trainig set to train the model again.

To enable feature selection, one needs to set *pseudo_labeling=True* when creating experiment. Relevant parameters:

- pseudo_labeling_strategystr, selection strategy(default threshold), available strategies include *threshold*, *number* and *quantile*.
- pseudo_labeling_proba_thresholdfloat(default 0.8), threshold when the selection strategy is *threshold*, confidence scores higher than this values will be chose.
- pseudo_labeling_proba_quantilefloat(default 0.8), threshold when the selection strategy is *quantile*, importance scores higher than this values will be chose.
- pseudo_labeling_sample_numberfloat(0.0~1.0) or int (default 0.2), the number of top features to be selected when the strategy is *number*.
- pseudo_labeling_resplitbool(default=False), whether split training and validation set after adding pseudo label examples. If set as False, all examples with pseudo labels will be added into training set to train the model. Otherwise, experiment will perform training set and validation set splitting for the new dataset with pseudo labels.

An example code:

```
from hypergbm import make_experiment

X_test=...
hyper_model = create_hyperModel()
experiment = CompeteExperiment(hyper_model,
                               X_test=X_test,
```

(continues on next page)

(continued from previous page)

```
                                pseudo_labeling=True,
                                ...)
```

Note: Pseudo label is only valid for classification task.

Change the log level

The progress messages during training can be printed by setting *log_level* (*str* or *int*) to change the log level. Please refer to the *logging* package for more details. Besides, more thorough messages will show when *verbose* is set as *1*.

The following codes sets the log level to 'INFO':

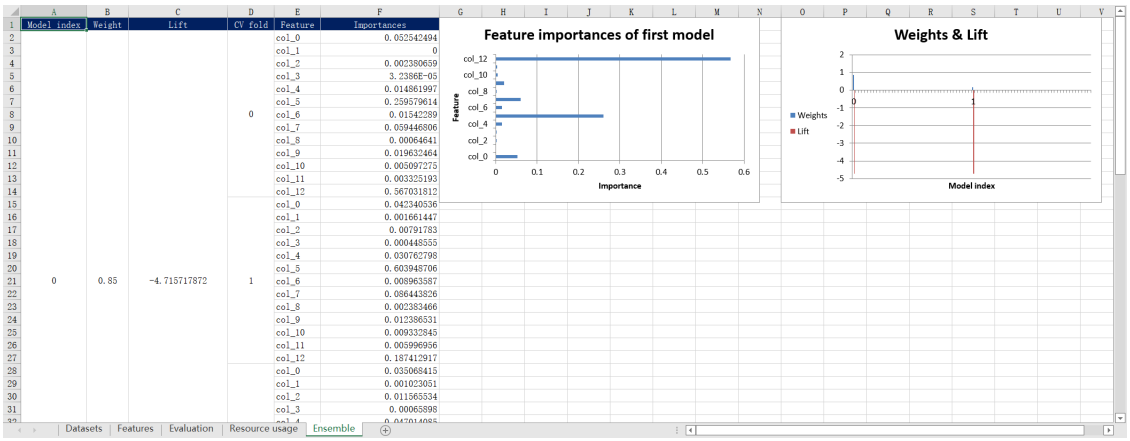
```
hyper_model = create_hyperModel()
experiment = CompeteExperiment(hyper_model, log_level='INFO', verbose=1, ...)
```

Export experiment report

If you want to export the experiment report in Excel format after training, you can set the argument as *report_render= 'excel'*. The sample codes for displaying the experiment report are shown as follows:

```
from sklearn.model_selection import train_test_split

from hypernets.examples.plain_model import PlainModel, PlainSearchSpace
from hypernets.experiment import make_experiment
from hypernets.tabular.datasets import dsutils

df = dsutils.load_boston()

df_train, df_eval = train_test_split(df, test_size=0.2)
search_space=PlainSearchSpace(enable_lr=False, enable_nn=False, enable_dt=False,␣
↪enable_dtr=True)

experiment = make_experiment(PlainModel, df_train,
                             target='target',
                             search_space=search_space,
                             report_render='excel')
estimator = experiment.run(max_trials=3)
print(estimator)
```

An excel report file named *report.xlsx* will be generated in the current directory. The report include the information of the engineering features, evaluation scores, resource usage and ensemble models, as well as the automatically generated plots. One example is shown below:



You can also change the file path of the output report by setting the argument `report_render_options`, here is the example code of how to set output report to `/tmp/report.xlsx`:

```
...
experiment = make_experiment(...,
                             report_render='excel',
                             report_render_options={'file_path': "/tmp/report.xlsx"})
...
```

Experiment visualization in jupyter notebook

Hypernets support two new features of experiment visualization in notebooks:

1. visualization of the experiment configurations
2. visualization of the dataset information

These features are optional tools and do not influence the experiment results.

To install the notebook visualization tool, use the command:

```
pip install hypernets[notebook]
```

If you have not installed the notebook widget, you can install it by command:

```
pip install hboard-widget
```

Here is an example of how to use these features in jupyter notebook :

1. Import the required modules

```
from sklearn.model_selection import train_test_split

from hypernets.examples.plain_model import PlainModel, PlainSearchSpace
from hypernets.experiment import make_experiment
from hypernets.tabular.datasets import dsutils
```

2. Create a hypernets experiment

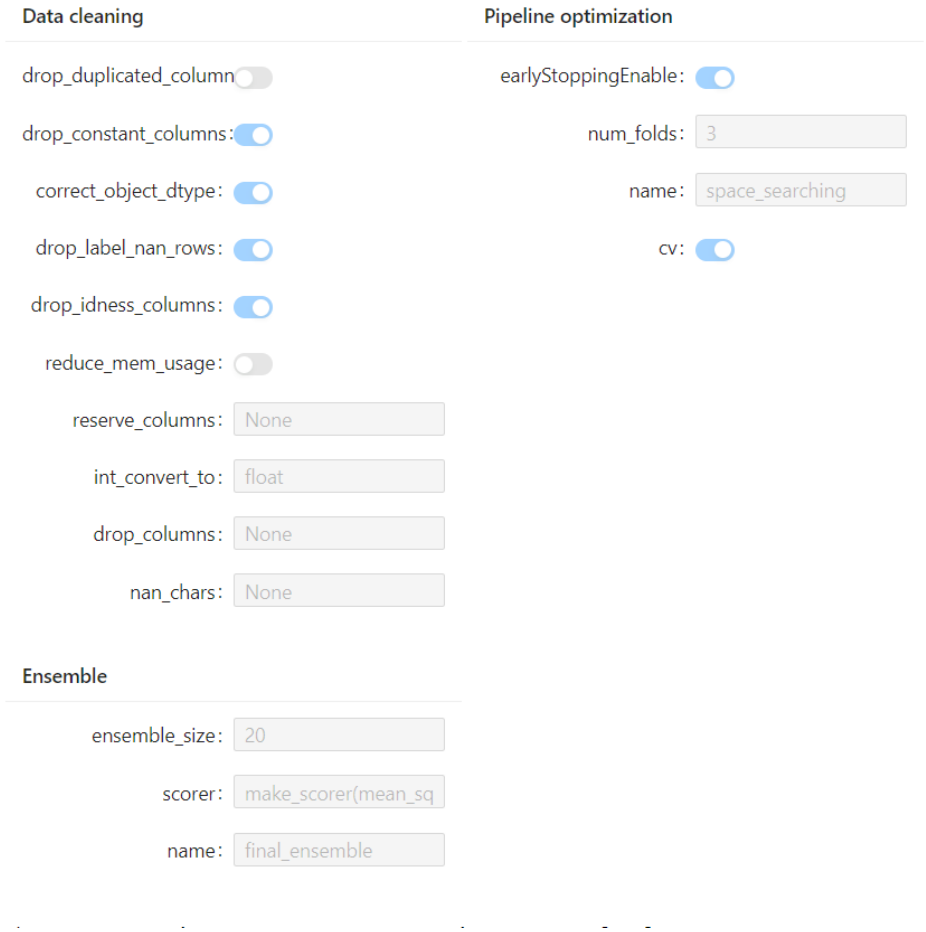
```
df_train, df_eval = train_test_split(df, test_size=0.2)
search_space=PlainSearchSpace(enable_lr=False, enable_nn=False, enable_dt=False,␣
↪enable_dtr=True)
```

(continues on next page)

(continued from previous page)

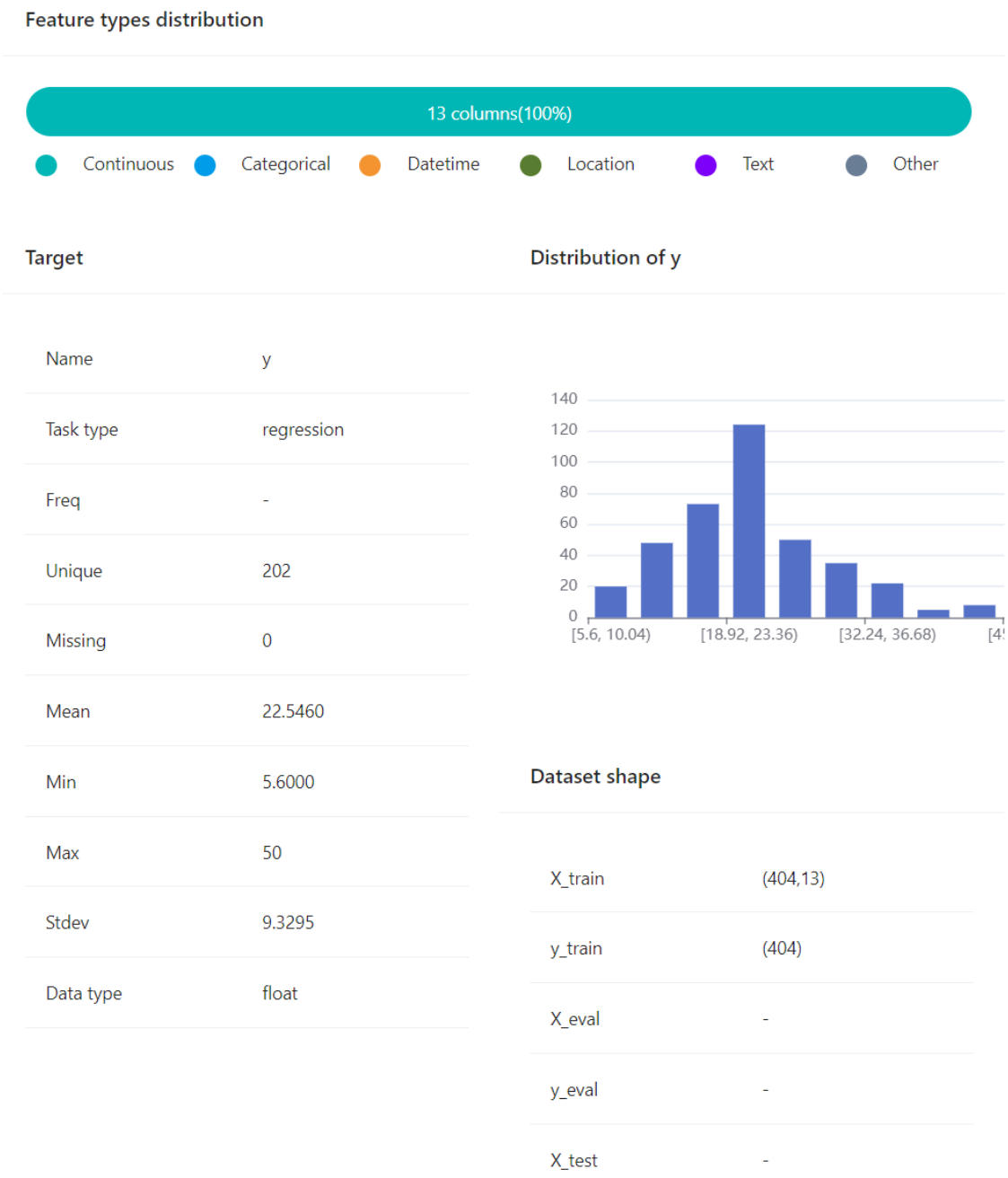
```
experiment = make_experiment(PlainModel, df_train,
                             target='target',
                             search_space=search_space)
experiment
```

3. The experiment configurations are automatically displayed, which provides a convenient reference for further modification.



4. The dataset information can also be easily visualized by command:

```
experiment.plot_dataset()
```



You can find the notebook example at hypernets_experiment_notebook_visualization.ipynb

1.8 Hyperctl

Hyperctl is a general tool for multi-job management, which includes but not limit to training, testing and comparison. It is packaged under Hypernets and intended to provide convenience to every developing stage.

1.8.1 Concepts

Job

A command line job that accepts parameters. Hyperctl provides the python API to read the parameters of the job, so this command line can execute a python script and use the API to obtain the parameters to complete the job.

Batch

A batch of jobs. All the status files and output files of jobs in the same batch are in the working directory of the batch.

Scheduler

Job scheduler, which schedules jobs in batch to run on appropriate machine resources and manages computing resources.

Backend

The backend for running jobs can run in a stand-alone mode or multiple remote nodes through SSH protocol.

1.8.2 Quick start

Run batch using command line tool

After installing `hypernets`, you could see the following description by typing `hyperctl`, which includes four arguments `run`, `generate`, `batch`, `job`:

```
$ hyperctl
usage: hyperctl [-h] [--log-level LOG_LEVEL] [-error] [-warn] [-info] [-debug] {run,
↪generate,batch,job} ...

hyperctl command is used to manage jobs

positional arguments:
  {run,generate,batch,job}
    run                 run jobs
    generate            generate specific jobs json file
    batch               batch operations
    job                 job operations

optional arguments:
  -h, --help            show this help message and exit

Console outputs:
  --log-level LOG_LEVEL
                        logging level, default is INFO
  -error                alias of "--log-level=ERROR"
  -warn                 alias of "--log-level=WARN"
  -info                 alias of "--log-level=INFO"
  -debug                alias of "--log-level=DEBUG"

```
```

Take using Hyperctl to tuning the `tol` parameter of `sklearn.linear_model.LogisticRegression` as an example, create a job python script `~/sklearn_iris_example.py` with following content:

```
import os
import pickle as pkl
```

(continues on next page)
```

(continued from previous page)

```
from hypernets import hyperctl
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

job_params = hyperctl.get_job_params()  # read job params as a dict from hyperctl

tol=job_params['tol']

X, y = datasets.load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
↪state=8086)

lr = LogisticRegression(tol=tol)
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)

print(f"tol: {tol}, accuracy_score: {accuracy_score(y_pred, y_test)}")

# persist assets
report_dir = "~/report/"
os.makedirs(report_dir)

with open(os.path.join(reportdir, f"model_tol_{tol}.pkl"), 'wb') as f:
    pkl.dump(lr, f)
```

Hyperctl uses the JSON format file to define the jobs, for example create a file named `batch.json` and configures 2 jobs then set the parameter `tol` to 1 and 100 respectively:

```
{
    "name": "sklearn_iris_example",
    "jobs": [{
            "name": "tol_1",
            "params": {
                "tol": 1
            },
            "command": "python ~/sklearn_iris_example.py"
        },
        {
            "name": "tol_100",
            "params": {
                "tol": 100
            },
            "command": "python ~/sklearn_iris_example.py"
        }
    ]
}
```

Note: Make sure that the python used by the command in the job has `scikit-learn` installed

Run the job with command:

```
$ hyperctl run --config ./batch.json
```

After the task finished, view the output log file:

```
~/hyperctl-batches-working-dir/sklearn_iris_example/tol_1/stdout
--------------------------------------------------------------------
tol: 1, accuracy_score: 0.9333333333333333
```

```
~/hyperctl-batches-working-dir/sklearn_iris_example/tol_100/stdout
--------------------------------------------------------------------
tol: 100, accuracy_score: 0.36666666666666664
```

Run batch using API

Using API is more flexible than the command line tool to manage batch.

```
from hypernets.hyperctl.appliation import BatchApplication
from hypernets.hyperctl.batch import Batch

batch = Batch(name="remote-batch-example", data_dir="~/hyperctl/remote-batch-example",
↪ job_command="python ~/sklearn_iris_example.py")

batch.add_job(name='job1', params={"tol": 1})
batch.add_job(name='job2', params={"tol": 2})
batch.add_job(name='job3', params={"tol": 3})


backend_conf = {
        "type": "remote",
        "machines": [
            {
                "connection": { 'hostname': "172.20.30.105", 'username': "hyperctl",
↪'password': "hyperctl"}  # modify to your host configuration
            }, {
                "connection": { 'hostname': "172.20.30.106", 'username': "hyperctl",
↪'password': "hyperctl"}  # modify to your host configuration
            }, {
                "connection": { 'hostname': "172.20.30.107", 'username': "hyperctl",
↪'password': "hyperctl"}  # modify to your host configuration
            }
        ]
    }


app = BatchApplication(batch, server_host="172.20.30.105",  # modify to your host␣
↪configuration
                       server_port=8061,
                       scheduler_exit_on_finish=True,
                       scheduler_interval=1000,
                       backend_conf=backend_conf,
                       independent_tmp=True,
                       scheduler_callbacks=[ConsoleCallback()])

app.start()
```

Debug job in development stage

Job scripts need to be scheduled through *BatchApplication* to run, so that they run in two separate processes, This brings inconvenience to the development and debugging of the job script. At this time, we can inject test parameters into the job to run the job script directly:

```
import os
import pickle as pkl
from hypernets import hyperctl
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

test_params = {  # define your test params
    "tol": 1
}

api.inject(params=mock_params)  # inject test params, and NOTE that remove it when␣
↪you no longer debug or it can not read params from BatchApplication

job_params = hyperctl.get_job_params()

tol=job_params['tol']

X, y = datasets.load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
↪state=8086)

lr = LogisticRegression(tol=tol)
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)

print(f"tol: {tol}, accuracy_score: {accuracy_score(y_pred, y_test)}")

# persist assets
report_dir = "~/report/"
os.makedirs(report_dir)

with open(os.path.join(reportdir, f"model_tol_{tol}.pkl"), 'wb') as f:
    pkl.dump(lr, f)
```

And now we can run or debug the job script directly, but note that we need remove `api.inject(params=mock_params)` if needs to receive params from `BatchApplication`.

Generate jobs from template

Hyperctl generates jobs config in batch by arranging and combining parameters based on the configuration template, the generated file can be used to run the batch. Here is an example of how to use template file to generate batch config file . First create a template file `job-template.yml` with following content:

```
params:
    learning_rate: [0.1,0.2]
    max_depth: [3, 5]
command: python3 cli.py
```

Then execute command to generate batch config file:

```
$ hyperctl generate --template ./job-template.yml --output ./batch.json
```

Here is the generated `batch.json` file:

```
{
    "name": "eVqNV5Ut1",
    "job": [{
        "name": "eaqNV5Ut1",
        "params": {
            "learning_rate": 0.1,
            "max_depth": 3
        },
        "command": "python3 cli.py"
    }, {
        "name": "ebqNV5Ut1",
        "params": {
            "learning_rate": 0.1,
            "max_depth": 5
        },
        "command": "python3 cli.py"
    }, {
        "name": "ecqNV5Ut1",
        "params": {
            "learning_rate": 0.2,
            "max_depth": 3
        },
        "command": "python3 cli.py"
    }, {
        "name": "edqNV5Ut1",
        "params": {
            "learning_rate": 0.2,
            "max_depth": 5
        },
        "command": "python3 cli.py"
    }]
}
```

1.8.3 Batch configuration file references

Examples

LocalBackend

```
{
    "name": "local_backend_example",
    "jobs": [
        {
            "name": "job1",
            "params": {
                "param1": 1
            },
            "command": "sleep 3"
        }
    ],
```

(continues on next page)

(continued from previous page)

```
    "backend": {
        "type": "local"
    }
}
```

RemoteSSHBackend

```
{
    "name": "local_backend_example",
    "jobs": [
        {
            "name": "job1",
            "params": {
                "param1": 1
            },
            "command": "sleep 3"
        }
    ],
    "backend": {
        "type": "remote",
        "machines": [
            {
                "connection": {
                    "hostname": "host1",
                    "username": "hyperctl",
                    "password": "hyperctl"
                }
            }
        ]
    },
    "server": {
      "host": "192.168.10.206"
    }
}
```

Configuration references

BatchApplicationConfig

Field Name	Type	Description
name	`str`, required	batch name, should be unique in a batch.
jobs	list[*JobConfig*], required	Jobs to run.
backend	*BackendConfig*, optional	platform where the jobs running on, default is *LocalBackendConfig* .
server	*ServerConfig*, optional	server setting.
scheduler	*SchedulerConfig*, optional	scheduler setting.
batches_data_dir	`str`, optional	batches working directory, where to store output files of batches, hyperctl will create a sub-directory by the batch name for every batch in this directory. default read from environment by key `HYPERCTL_BATCHES_DATA_DIR`, if do not set in environments using `~/hyperctl-batches-data-dir`.
version	`str`, optional	if is None, use the currently running version, default is None.

JobConfig

Field Name	Type	Description
name	`str`, optional	str, unique in batch, optional, if is null will generate a uuid as job name, recommended that you specify one, with the name of the batch name, the executed job can be skipped when the batch is re-executed
params	`dict`, required	job params, it can be obtained through API `hypernets.hyperctl.get_job_params`
command	`str`, required	command to the the job, if execute a file, recommend use absolute path or path relative to {execution.working_dir}
working_dir	`str`, optional	working dir to run the `command`, default is {batches_data_dir}/{batch_name}/{job_name}

Note: A job write output file to `{batches_data_dir}/{batch_name}/{job_name}`, it usually contains files:

- stdout: standard output
- stderr: standard error
- run.sh: shell script to run the job

BackendConfig

Is one of :

- *LocalBackendConfig*
- *RemoteBackendConfig*

LocalBackendConfig

Running batch in standalone mode, please refer to the example *LocalBackend*.

Field Name	Type	Description
type	`"local"`	
environments	`dict`, optional	Environments setting will export for the job process.

RemoteBackendConfig

Hyperctl supports parallel jobs in remote machines, this mode uses multiple machines to speed up the progress of the batch. It distributes jobs to remote nodes through the SSH protocol, which requires that the nodes running tasks remotely need to run SSH services and provide connection accounts. Please refer to the example *RemoteSSHBackend*

Field Name	Type	Description
machines	list[*RemoteMachineConfig*], required	Connection and configuration information of remote machines.

RemoteMachineConfig

Field Name	Type	Description
connection	*SHHConnectionConfig*, required	Connection information for the remote machine.
environments	`dict`, optional	Environments setting will export for the job process.

SHHConnectionConfig

Field Name	Type	Description
hostname	`hostname`, required	IP or hostname of remote machine.
username	`username`, required	username of remote machine.
password	`password`, required	password of remote machine.

ServerConfig

Field Name	Type	Description
host	`str`, optional	where to bind for the http server, it's should be IP address that can be accessed in remote machines if is remote backend, otherwise, the job will fail because the api server cannot be accessed, default is localhost.
port	`int`, optional	http server port, default is 8060

SchedulerConfig

Field Name	Type	Description
interval	`int`, optional	Scheduling interval, the unit is milliseconds, default value is 5000
exit_on_finish	`boolean`, optional	whether to exit the process when all jobs are finished, default is false

1.8.4 Job template configuration file references

Examples

Basic example

Refer to *Generate jobs from template* .

Configuration references

JobTemplateConfig

Field Name	Type	Description
name	`str`, required	refer to `BatchApplicationConfig.name`
params	`dict[str, list]`, required	job params list, used to arrange and combine to generate jobs config.
command	`str`, required	refer to `JobConfig.command`
working_dir	`dict[str, list]`, required	
backend	*BackendConfig*, optional	refer to `BatchApplicationConfig.backend`
batches_data_dir	`str`, optional	refer to `BatchApplicationConfig.batches_data_dir`
server	*ServerConfig*, optional	refer to `BatchApplicationConfig.server`
scheduler	*SchedulerConfig*, optional	refer to `BatchApplicationConfig.scheduler`
version	`str`, optional	refer to `BatchApplicationConfig.version`

1.9 hypernets package

1.9.1 Subpackages

hypernets.experiment package

Submodules

hypernets.experiment.cfg module

hypernets.experiment.compete module

class hypernets.experiment.compete.**CompeteExperiment**(*hyper_model*, *X_train*, *y_train*, *X_eval=None*, *y_eval=None*, *X_test=None*, *eval_size=0.3*, *train_test_split_strategy=None*, *cv=None*, *num_folds=3*, *task=None*, *id=None*, *callbacks=None*, *random_state=None*, *scorer=None*, *data_adaption=None*, *data_adaption_target=None*, *data_adaption_memory_limit=0.05*, *data_adaption_min_cols=0.3*, *data_cleaner_args=None*, *feature_generation=False*, *feature_generation_trans_primitives=None*, *feature_generation_max_depth=1*, *feature_generation_categories_cols=None*, *feature_generation_continuous_cols=None*, *feature_generation_datetime_cols=None*, *feature_generation_latlong_cols=None*, *feature_generation_text_cols=None*, *collinearity_detection=False*, *drift_detection=True*, *drift_detection_remove_shift_variable=True*, *drift_detection_variable_shift_threshold=0.7*, *drift_detection_threshold=0.7*, *drift_detection_remove_size=0.1*, *drift_detection_min_features=10*, *drift_detection_num_folds=5*, *feature_selection=False*, *feature_selection_strategy=None*, *feature_selection_threshold=None*, *feature_selection_quantile=None*, *feature_selection_number=None*, *down_sample_search=None*, *down_sample_search_size=None*, *down_sample_search_time_limit=None*, *down_sample_search_max_trials=None*, *ensemble_size=20*, *feature_reselection=False*, *feature_reselection_estimator_size=10*, *feature_reselection_strategy=None*, *feature_reselection_threshold=1e-05*, *feature_reselection_quantile=None*, *feature_reselection_number=None*,

A powerful experiment strategy for AutoML with a set of advanced features.

There are still many challenges in the machine learning modeling process for tabular data, such as imbalanced data, data drift, poor generalization ability, etc. This challenges cannot be completely solved by pipeline search, so we introduced in HyperNets a more powerful tool is *CompeteExperiment*. *CompeteExperiment* is composed of a series of steps and *Pipeline Search* is just one step. It also includes advanced steps such as data cleaning, data drift handling, two-stage search, ensemble etc.

static get_creators(*hyper_model*, *X_train*, *y_train*, *X_test=None*, *X_eval=None*, *y_eval=None*, *down_sample_search=False*)

get_data_character()

run(***kwargs*)

to_estimator(*X_train*, *y_train*, *X_test*, *X_eval*, *y_eval*, *steps*)

class hypernets.experiment.compete.**DaskEnsembleStep**(*experiment*, *name*, *scorer=None*, *ensemble_size=7*)

Bases: *hypernets.experiment.compete.EnsembleStep*

get_ensemble_predictions(*trials*, *ensemble*)

class hypernets.experiment.compete.**DaskPseudoLabelStep**(*experiment*, *name*, *estimator_builder_name*, *strategy=None*, *proba_threshold=None*, *proba_quantile=None*, *sample_number=None*, *resplit=False*)

Bases: *hypernets.experiment.compete.PseudoLabelStep*

merge_pseudo_label(*X_train*, *y_train*, *X_eval*, *y_eval*, *X_pseudo*, *y_pseudo*, ***kwargs*)

class hypernets.experiment.compete.**DataAdaptionStep**(*experiment*, *name*, *target=None*, *memory_limit=0.05*, *min_cols=0.3*)

Bases: *hypernets.experiment.compete.FeatureSelectStep*

compact_by_columns(*X_train*, *y_train*, *X_test*, *X_eval*, *y_eval*, *frac*)

compact_by_rows(*X_train*, *y_train*, *X_test*, *X_eval*, *y_eval*, *frac*)

fit_transform(*hyper_model*, *X_train*, *y_train*, *X_test=None*, *X_eval=None*, *y_eval=None*, ***kwargs*)

get_tool_box_with_target(**data*)

sample(*X*, *y*, *frac*)

class hypernets.experiment.compete.**DataCleanStep**(*experiment*, *name*, *data_cleaner_args=None*, *cv=False*, *train_test_split_strategy=None*)

Bases: *hypernets.experiment.compete.FeatureSelectStep*

as_local()

cache_transform(*hyper_model*, *X_train*, *y_train*, *X_test=None*, *X_eval=None*, *y_eval=None*, ***kwargs*)

fit_transform(***kwargs*)

get_fitted_params()

get_params(*deep=True*)

Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type dict

transform(*X*, *y=None*, ***kwargs*)

class hypernets.experiment.compete.**DriftDetectStep**(*experiment*, *name*, *remove_shift_variable*, *variable_shift_threshold*, *threshold*, *remove_size*, *min_features*, *num_folds*)

Bases: *hypernets.experiment.compete.FeatureSelectStep*

fit_transform(***kwargs*)

get_fitted_params()

class hypernets.experiment.compete.**EnsembleStep**(*experiment*, *name*, *scorer=None*, *ensemble_size=7*)

Bases: *hypernets.experiment.compete.EstimatorBuilderStep*

build_estimator(*hyper_model*, *X_train*, *y_train*, *X_eval=None*, *y_eval=None*, ***kwargs*)

get_ensemble(*estimators*, *X_train*, *y_train*)

get_ensemble_predictions(*trials*, *ensemble*)

select_trials(*hyper_model*)

select trials to ensemble from hyper_model (and it's history)

class hypernets.experiment.compete.**EstimatorBuilderStep**(*experiment*, *name*)

Bases: *hypernets.experiment.compete.ExperimentStep*

build_estimator(*hyper_model*, *X_train*, *y_train*, *X_test=None*, *X_eval=None*, *y_eval=None*, ***kwargs*)

fit_transform(*hyper_model*, *X_train*, *y_train*, *X_test=None*, *X_eval=None*, *y_eval=None*, ***kwargs*)

get_fitted_params()

is_transform_skipped()

transform(*X*, *y=None*, ***kwargs*)

class hypernets.experiment.compete.**ExperimentStep**(*experiment*, *name*)

Bases: sklearn.base.BaseEstimator

STATUS_FAILED = 1

STATUS_NONE = -1

STATUS_RUNNING = 10

STATUS_SKIPPED = 2

STATUS_SUCCESS = 0

elapsed_seconds

fit_transform(*hyper_model*, *X_train*, *y_train*, *X_test=None*, *X_eval=None*, *y_eval=None*, ***kwargs*)

get_fitted_params()

is_transform_skipped()

step_progress(**args*, ***kwargs*)

task

transform(*X*, *y=None*, ***kwargs*)

class hypernets.experiment.compete.**FeatureGenerationStep**(*experiment*, *name*, *trans_primitives=None*, *continuous_cols=None*, *datetime_cols=None*, *categories_cols=None*, *latlong_cols=None*, *text_cols=None*, *max_depth=1*, *feature_selection_args=None*)

Bases: *hypernets.experiment.compete.TransformerAdaptorStep*

get_fitted_params()

is_transform_skipped()

class hypernets.experiment.compete.**FeatureImportanceSelectionStep**(*experiment*, *name*, *strategy*, *threshold*, *quantile*, *number*)

Bases: *hypernets.experiment.compete.FeatureSelectStep*

fit_transform(***kwargs*)

get_fitted_params()

class hypernets.experiment.compete.**FeatureSelectStep**(*experiment*, *name*)

Bases: *hypernets.experiment.compete.ExperimentStep*

cache_transform(*hyper_model*, *X_train*, *y_train*, *X_test=None*, *X_eval=None*, *y_eval=None*, ***kwargs*)

get_fitted_params()

is_transform_skipped()

selected_features

transform(*X*, *y=None*, ***kwargs*)

unselected_features

class hypernets.experiment.compete.**FinalTrainStep**(*experiment*, *name*, *retrain_on_wholedata=False*)

Bases: *hypernets.experiment.compete.EstimatorBuilderStep*

build_estimator(*hyper_model*, *X_train*, *y_train*, *X_test=None*, *X_eval=None*, *y_eval=None*, ***kwargs*)

class hypernets.experiment.compete.**MOOFinalStep**(*experiment*, *name*)

Bases: *hypernets.experiment.compete.EstimatorBuilderStep*

build_estimator(*hyper_model*, *X_train*, *y_train*, *X_test=None*, *X_eval=None*, *y_eval=None*, ***kwargs*)

class hypernets.experiment.compete.**MulticollinearityDetectStep**(*experiment*, *name*)

Bases: *hypernets.experiment.compete.FeatureSelectStep*

fit_transform(***kwargs*)

get_fitted_params()

class hypernets.experiment.compete.**PermutationImportanceSelectionStep**(*experiment*, *name*, *scorer*, *estimator_size*, *strategy*, *threshold*, *quantile*, *number*)

Bases: *hypernets.experiment.compete.FeatureSelectStep*

fit_transform(*hyper_model*, *X_train*, *y_train*, *X_test=None*, *X_eval=None*, *y_eval=None*, ***kwargs*)

get_fitted_params()

class hypernets.experiment.compete.**PseudoLabelStep**(*experiment*, *name*, *estimator_builder_name*, *strategy=None*, *proba_threshold=None*, *proba_quantile=None*, *sample_number=None*, *resplit=False*)

Bases: *hypernets.experiment.compete.ExperimentStep*

fit_transform(*hyper_model*, *X_train*, *y_train*, *X_test=None*, *X_eval=None*, *y_eval=None*, ***kwargs*)

get_fitted_params()

is_transform_skipped()

merge_pseudo_label(*X_train*, *y_train*, *X_eval*, *y_eval*, *X_pseudo*, *y_pseudo*, ***kwargs*)

static stat_pseudo_label(*y_pseudo*, *classes*)

transform(*X*, *y=None*, ***kwargs*)

class hypernets.experiment.compete.**SpaceSearchStep**(*experiment*, *name*, *cv=False*, *num_folds=3*)

Bases: *hypernets.experiment.compete.ExperimentStep*

estimate_time_limit(*total_time_limit*)

static find_early_stopping_callback(*cbs*)

fit_transform(*hyper_model*, *X_train*, *y_train*, *X_test=None*, *X_eval=None*, *y_eval=None*, ***kwargs*)

from_fitted_step(*fitted_step*)

get_fitted_params()

is_transform_skipped()

search(*X_train*, *y_train*, *X_test=None*, *X_eval=None*, *y_eval=None*, ***kwargs*)

transform(*X*, *y=None*, ***kwargs*)

class hypernets.experiment.compete.**SpaceSearchWithDownSampleStep**(*experiment*, *name*, *cv=False*, *num_folds=3*, *size=None*, *max_trials=None*, *time_limit=None*)

Bases: *hypernets.experiment.compete.SpaceSearchStep*

static create_playback_searcher(*history*)

down_sample(*X_train*, *y_train*, *X_eval*, *y_eval*)

from_fitted_step(*fitted_step*)

search(*X_train*, *y_train*, *X_test=None*, *X_eval=None*, *y_eval=None*, ***kwargs*)

class hypernets.experiment.compete.**StepNames**

Bases: object

DATA_ADAPTION = 'data_adaption'

DATA_CLEAN = 'data_clean'

DRIFT_DETECTION = 'drift_detection'

ENSEMBLE = 'ensemble'

FEATURE_GENERATION = 'feature_generation'

FEATURE_IMPORTANCE_SELECTION = 'feature_selection'

FEATURE_RESELECTION = 'feature_reselection'

FINAL_ENSEMBLE = 'final_ensemble'

FINAL_MOO = 'final_moo'

FINAL_SEARCHING = 'two_stage_searching'

FINAL_TRAINING = 'final_train'

MULITICOLLINEARITY_DETECTION = 'multicollinearity_detection'

PSEUDO_LABELING = 'pseudo_labeling'

SPACE_SEARCHING = 'space_searching'

TRAINING = 'training'

class hypernets.experiment.compete.**SteppedExperiment**(*steps*, **args*, ***kwargs*)

Bases: hypernets.experiment._experiment.Experiment

find_step(*fn*, *until_step_name=None*, *index=False*)

get_step(*name*)

get_step_index(*name_or_index*, *default*)

static to_estimator(*X_train*, *y_train*, *X_test*, *X_eval*, *y_eval*, *steps*)

train(*hyper_model*, *X_train*, *y_train*, *X_test*, *X_eval=None*, *y_eval=None*, ***kwargs*)

Run an experiment

Parameters

- **hyper_model** (*HyperModel*) –
- **X_train** –
- **y_train** –
- **X_test** –
- **X_eval** –
- **y_eval** –

class hypernets.experiment.compete.**TransformerAdaptorStep**(*experiment*, *name*, *transformer_creator*, ***kwargs*)

Bases: *hypernets.experiment.compete.ExperimentStep*

cache_transform(*hyper_model*, *X_train*, *y_train*, *X_test=None*, *X_eval=None*, *y_eval=None*, ***kwargs*)

fit_transform(***kwargs*)

transform(*X*, *y=None*, ***kwargs*)

hypernets.experiment.compete.**evaluate_oofs**(*hyper_model*, *ensemble_estimator*, *y_train*, *metrics*)

hypernets.experiment.general module

class hypernets.experiment.general.**GeneralExperiment**(*hyper_model*, *X_train*, *y_train*, *X_eval=None*, *y_eval=None*, *X_test=None*, *eval_size=0.3*, *task=None*, *id=None*, *callbacks=None*, *random_state=9527*)

Bases: hypernets.experiment._experiment.Experiment

train(*hyper_model*, *X_train*, *y_train*, *X_test*, *X_eval=None*, *y_eval=None*, ***kwargs*)

Run an experiment

hypernets.experiment.job module

class hypernets.experiment.job.**CompeteExperimentJobCreator**

Bases: *hypernets.experiment.job.ExperimentJobCreator*

create_and_run_experiment()

create_experiment_with_params(*make_kwargs*, *job_working_dir*)

static set_default_eval_dir(*job_working_dir*, *make_kwargs*)

static set_default_render_path(*job_working_dir*, *make_kwargs*)

class hypernets.experiment.job.**ExperimentJobCreator**

Bases: object

create_and_run_experiment()

hypernets.experiment.report module

class hypernets.experiment.report.**ExcelReportRender**(*file_path: str = './report.xlsx'*, *theme='default'*)

Bases: *hypernets.experiment.report.ReportRender*

MAX_CELL_LENGTH = 50

static log_skip_sheet(*name*)

render(*experiment_meta: hypernets.experiment._extractor.ExperimentMeta*, ***kwargs*)

Render report data into a excel file

Parameters

- **experiment_meta** – if part of {experiment_report} is empty maybe skip to create sheet.
- **kwargs** –

class hypernets.experiment.report.**FeatureTrans**(*feature*, *method*, *stage*, *reason*, *remark*)

Bases: tuple

feature

Alias for field number 0

method

Alias for field number 1

reason

Alias for field number 3

remark

Alias for field number 4

stage

Alias for field number 2

class hypernets.experiment.report.**FeatureTransCollector**(*steps: List[hypernets.experiment._extractor.StepMeta]*)

Bases: object

METHOD_ADD = 'add'

METHOD_DROP = 'drop'

collect()

get_handler(*step_class_name*)

class hypernets.experiment.report.**ReportRender**(***kwargs*)

Bases: object

render(*experiment_meta: hypernets.experiment._extractor.ExperimentMeta*, ***kwargs*)

class hypernets.experiment.report.**Theme**(*theme_name*)

Bases: object

get_header_style()

get_row_diff_style()

hypernets.experiment.report.**get_render**(*name*)

Module contents

hypernets.searchers package

Submodules

hypernets.searchers.evolution_searcher module

class hypernets.searchers.evolution_searcher.**EvolutionSearcher**(*space_fn*, *population_size*, *sample_size*, *regularized=False*, *candidates_size=10*, *optimize_direction=<OptimizeDirection.Min: 'min'>*, *use_meta_learner=True*, *space_sample_validation_fn=None*, *random_state=None*)

Bases: hypernets.core.searcher.Searcher

Evolutionary Algorithm

Parameters

- **space_fn** (*callable, required*) – A search space function which when called returns a *HyperSpace* instance
- **population_size** (*int, required*) – Size of population
- **sample_size** (*int, required*) – The number of parent candidates selected in each cycle of evolution
- **regularized** (*bool*) – (default=False), Whether to enable regularized
- **candidates_size** (*int, (default=10)*) – The number of samples for the meta-learner to evaluate candidate paths when roll out
- **optimize_direction** (*'min' or 'max', (default='min')*) – Whether the search process is approaching the maximum or minimum reward value.
- **use_meta_learner** (*bool, (default=True)*) – Whether to enable meta leaner. Meta-learner aims to evaluate the performance of unseen samples based on previously evaluated samples. It provides a practical solution to accurately estimate a search branch with many simulations without involving the actual training.
- **space_sample_validation_fn** (*callable or None, (default=None)*) – Used to verify the validity of samples from the search space, and can be used to add specific constraint rules to the search space to reduce the size of the space.

References

Real, Esteban, et al. "Regularized evolution for image classifier architecture search." Proceedings of the aaai conference on artificial intelligence. Vol. 33. 2019.

parallelizable

population_size

sample(*space_options=None*)

summary()

update_result(*space_sample*, *result*)

class hypernets.searchers.evolution_searcher.**Individual**(*space_sample*, *reward*)
Bases: object

mutate()

class hypernets.searchers.evolution_searcher.**Population**(*size=50*, *optimize_direction=<OptimizeDirection.Minimize: 'min'>*, *random_state=None*)
Bases: object

append(*space_sample*, *reward*)

eliminate(*num=1*, *regularized=False*)

initializing

length

mutate(*parent_space*, *offspring_space*)

sample_best(*sample_size*)

shuffle()

hypernets.searchers.genetic module

class hypernets.searchers.genetic.**Individual**(*dna*, *scores*, *random_state*)
Bases: object

class hypernets.searchers.genetic.**Recombination**(*random_state*)
Bases: object

check_parents(*ind1: hypernets.searchers.genetic.Individual*, *ind2: hypernets.searchers.genetic.Individual*)

do(*ind1: hypernets.searchers.genetic.Individual*, *ind2: hypernets.searchers.genetic.Individual*, *out_space: hypernets.core.search_space.HyperSpace*)

class hypernets.searchers.genetic.**ShuffleCrossOver**(*random_state*)
Bases: *hypernets.searchers.genetic.Recombination*

do(*ind1: hypernets.searchers.genetic.Individual*, *ind2: hypernets.searchers.genetic.Individual*, *out_space: hypernets.core.search_space.HyperSpace*)

class hypernets.searchers.genetic.**SinglePointCrossOver**(*random_state*)
Bases: *hypernets.searchers.genetic.Recombination*

do(*ind1: hypernets.searchers.genetic.Individual*, *ind2: hypernets.searchers.genetic.Individual*, *out_space: hypernets.core.search_space.HyperSpace*)

class hypernets.searchers.genetic.**SinglePointMutation**(*random_state*, *proba=0.7*)
Bases: object

do(*sample_space*, *out_space*, *proba=None*)

class hypernets.searchers.genetic.**UniformCrossover**(*random_state*)
Bases: *hypernets.searchers.genetic.Recombination*

do(*ind1: hypernets.searchers.genetic.Individual*, *ind2: hypernets.searchers.genetic.Individual*, *out_space: hypernets.core.search_space.HyperSpace*)

hypernets.searchers.genetic.**create_recombination**(*name*, *random_state*, ***kwargs*)

hypernets.searchers.grid_searcher module

class hypernets.searchers.grid_searcher.**GridSearcher**(*space_fn*, *optimize_direction=<OptimizeDirection.Minimize: 'min'>*, *space_sample_validation_fn=None*, *n_expansion=5*)
Bases: hypernets.core.searcher.Searcher

export()

get_best()

parallelizable

reset()

sample(*space_options=None*)

update_result(*space*, *result*)

hypernets.searchers.grid_searcher.**test_parameter_grid**(*self*)

hypernets.searchers.mcts_core module

class hypernets.searchers.mcts_core.**BasePolicy**
Bases: object

back_propagation(*node*, *reward*)

selection(*node*)

class hypernets.searchers.mcts_core.**MCNode**(*id*, *name*, *param_sample*, *parent=None*, *tree=None*, *is_terminal=False*, *random_state=None*)
Bases: object

add_child(*param_sample*)

children

depth

expanded

expansion(*param_space*, *max_space*)

info()

is_leaf

is_terminal

random_sample()

set_parent(*parent*)

set_terminal()

class hypernets.searchers.mcts_core.**MCTree**(*space_fn*, *policy*, *max_node_space*)
Bases: object

back_propagation(*node*, *reward*, *is_simulation=False*)

current_node

expansion(*node*)

node_to_space(*node*)

path_to_node(*node*)

roll_out(*space_sample*, *node*)

selection_and_expansion()

simulation(*node*)

class hypernets.searchers.mcts_core.**UCT**(*exploration_bonus=0.6*)
Bases: *hypernets.searchers.mcts_core.BasePolicy*

back_propagation(*node*, *reward*, *is_simulation=False*)

selection(*node*)

hypernets.searchers.mcts_searcher module

class hypernets.searchers.mcts_searcher.**MCTSSearcher**(*space_fn*, *policy=None*, *max_node_space=10*, *candidates_size=10*, *optimize_direction=<OptimizeDirection.Minimize: 'min'>*, *use_meta_learner=True*, *space_sample_validation_fn=None*)
Bases: hypernets.core.searcher.Searcher

Parameters

- **space_fn** (*Callable*) – A search space function which when called returns a *HyperSpace* object.
- **policy** (*hypernets.searchers.mcts_core.BasePolicy, (default=None)*) – The policy for *Selection* and *Backpropagation* phases, *UCT* by default.
- **max_node_space** (*int, (default=10)*) – Maximum space for node expansion
- **candidates_size** (*int, (default=10)*) – The number of samples for the meta-learner to evaluate candidate paths when roll out
- **optimize_direction** (*'min' or 'max', (default='min')*) – Whether the search process is approaching the maximum or minimum reward value
- **use_meta_learner** (*bool, (default=True)*) – Meta-learner aims to evaluate the performance of unseen samples based on previously evaluated samples. It provides a practical solution to accurately estimate a search branch with many simulations without involving the actual training

- **space_sample_validation_fn** (*Callable or None, (default=None)*) – Used to verify the validity of samples from the search space, and can be used to add specific constraint rules to the search space to reduce the size of the space

References

[1] Wang, Linnan, et al. "Alphax: exploring neural architectures with deep neural networks and monte carlo tree search." arXiv preprint arXiv:1903.11059 (2019).

[2] Browne, Cameron B., et al. "A survey of monte carlo tree search methods." IEEE Transactions on Computational Intelligence and AI in games 4.1 (2012): 1-43.

export()

get_best()

max_node_space

parallelizable()

reset()

sample(*space_options=None*)

summary()

update_result(*space_sample*, *result*)

hypernets.searchers.moead_searcher module

class hypernets.searchers.moead_searcher.**Decomposition**(***kwargs*)

Bases: object

static adaptive_normalization(*F*, *ideal*, *nadir*)

do(*scores: numpy.ndarray*, *weight_vector: numpy.ndarray*, *Z: numpy.ndarray*, *ideal: numpy.ndarray*, *nadir: numpy.ndarray*, ***kwargs*)

class hypernets.searchers.moead_searcher.**MOEADSearcher**(*space_fn*, *objectives*, *n_sampling=5*, *n_neighbors=2*, *recombination=None*, *mutate_probability=0.7*, *decomposition=None*, *space_sample_validation_fn=None*, *random_state=None*)

Bases: *hypernets.searchers.moo.MOOSearcher*

An implementation of "MOEA/D".

Parameters

- **space_fn** (*callable, required*) – A search space function which when called returns a *HyperSpace* instance
- **objectives** (*List[Objective], optional, (default to NumOfFeatures instance)*) – The optimization objectives.
- **n_sampling** (*int, optional, default to 5.*) – The number of samples in each objective, it affects the number of optimization objectives after decomposition:

  $N = C_{samples+objectives-1}^{objectives-1}$

- **n_neighbors** (*int, optional, default to 3.*) – Number of neighbors to crossover.
- **recombination** (Recombination, *optional, default to instance of SinglePointCrossOver*) – the strategy to recombine DNA of parents to generate offspring. Builtin strategies:
  - ShuffleCrossOver
  - UniformCrossover
  - SinglePointCrossOver
- **decomposition** (Decomposition, *optional, default to instance of TchebicheffDecomposition*) – The strategy to decompose multi-objectives optimization problem and calculate scores for the sub problem, now supported:
  - TchebicheffDecomposition
  - PBIDecomposition
  - WeightedSumDecomposition

  Due to the possible differences in dimension of objectives, normalization will be performed on the scores, the formula:

  $f_i' = \frac{f_i - z_i^*}{z_i^{nad} - z^* + \epsilon}$

mutate_probability: float, optional, default to 0.7 the probability of genetic variation for offspring, when the parents can not recombine, it will definitely mutate a gene for the generated offspring.

space_sample_validation_fn: callable or None, (default=None) used to verify the validity of samples from the search space, and can be used to add specific constraint rules to the search space to reduce the size of the space.

random_state: np.RandomState, optional used to reproduce the search process

References

[1] Q. Zhang and H. Li, "MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition," in IEEE Transactions on Evolutionary Computation, vol. 11, no. 6, pp. 712-731, Dec. 2007, doi: 10.1109/TEVC.2007.892759.

[2] Das I, Dennis J E. "Normal-boundary intersection: A new method for generating the Pareto surface in nonlinear multicriteria optimization problems[J]." SIAM Journal on Optimization, 1998, 8(3): 631-657.

calc_euler_distance(*vectors*)

distribution_number(*n_samples*, *n_objectives*)
Uniform weighted vectors, an implementation of Normal-boundary intersection.

export()

get_best()

get_historical_population() → List[hypernets.searchers.genetic.Individual]

get_ideal_point()

get_nadir_point()

get_nondominated_set()

get_population() → List[hypernets.searchers.genetic.Individual]

get_reference_point()
calculate Z in tchebicheff decomposition

init_mean_vector_by_NBI(*n_samples*, *n_objectives*)

init_population(*weight_vectors*)

n_objectives

parallelizable

population_size

reset()

sample(*space_options=None*)

update_result(*space*, *result*)

class hypernets.searchers.moead_searcher.**PBIDecomposition**(*penalty=0.5*)
Bases: *hypernets.searchers.moead_searcher.Decomposition*

An implementation of "Boundary Intersection Approach base on penalty"

Parameters **penalty** (*float, optional, default to 0.5*) – Penalty the solution(F) deviates from the weight vector, the larger the value, the faster the convergence.

do(*scores: numpy.ndarray*, *weight_vector: numpy.ndarray*, *Z: numpy.ndarray*, *ideal: numpy.ndarray*, *nadir: numpy.ndarray*, ***kwargs*)

class hypernets.searchers.moead_searcher.**TchebicheffDecomposition**(***kwargs*)
Bases: *hypernets.searchers.moead_searcher.Decomposition*

do(*scores: numpy.ndarray*, *weight_vector*, *Z*, *ideal: numpy.ndarray*, *nadir: numpy.ndarray*, ***kwargs*)

class hypernets.searchers.moead_searcher.**WeightedSumDecomposition**(***kwargs*)
Bases: *hypernets.searchers.moead_searcher.Decomposition*

do(*scores: numpy.ndarray*, *weight_vector*, *Z: numpy.ndarray*, *ideal: numpy.ndarray*, *nadir: numpy.ndarray*, ***kwargs*)

hypernets.searchers.moead_searcher.**create_decomposition**(*name*, ***kwargs*)

hypernets.searchers.moo module

class hypernets.searchers.moo.**MOOSearcher**(*space_fn*, *objectives: List[hypernets.core.objective.Objective]*, ***, *use_meta_learner=True*, *space_sample_validation_fn=None*, ***kwargs*)
Bases: hypernets.core.searcher.Searcher

check_plot()

get_historical_population() → List[hypernets.searchers.genetic.Individual]

get_nondominated_set() → List[hypernets.searchers.genetic.Individual]

get_pareto_nondominated_set()

get_population() → List[hypernets.searchers.genetic.Individual]

kind()
Type of the Searcher, should be one of soo, moo. This property used to avoid having to import MOOSearcher when detecting Searcher type.

plot_population(*figsize=(6, 6)*, ***kwargs*)

hypernets.searchers.nsga_searcher module

class hypernets.searchers.nsga_searcher.**NSGAIISearcher**(*space_fn*, *objectives*, *recombination=None*, *mutate_probability=0.7*, *population_size=30*, *space_sample_validation_fn=None*, *random_state=None*)

Bases: hypernets.searchers.nsga_searcher._NSGAIIBasedSearcher

An implementation of "NSGA-II".

Parameters

- **space_fn** (*callable, required*) – A search space function which when called returns a *HyperSpace* instance
- **objectives** (*List[Objective], optional, (default to NumOfFeatures instance)*) – The optimization objectives.
- **recombination** (Recombination, *required*) – the strategy to recombine DNA of parents to generate offspring. Builtin strategies: - ShuffleCrossOver - UniformCrossover - SinglePointCrossOver
- **mutate_probability** (*float, optional, default to 0.7*) – the probability of genetic variation for offspring, when the parents can not recombine, it will definitely mutate a gene for the generated offspring.
- **population_size** (*int, default to 30*) – size of population
- **space_sample_validation_fn** (*callable or None, (default=None)*) – used to verify the validity of samples from the search space, and can be used to add specific constraint rules to the search space to reduce the size of the space.
- **random_state** (*np.RandomState, optional*) – used to reproduce the search process

References

[1] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," in IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182-197, April 2002, doi: 10.1109/4235.996017.

class hypernets.searchers.nsga_searcher.**RNSGAIISearcher**(*space_fn*, *objectives*, *ref_point=None*, *weights=None*, *dominance_threshold=0.3*, *recombination=None*, *mutate_probability=0.7*, *population_size=30*, *space_sample_validation_fn=None*, *random_state=None*)

Bases: hypernets.searchers.nsga_searcher._NSGAIIBasedSearcher

An implementation of R-NSGA-II which is a variant of NSGA-II algorithm.

Parameters

- **space_fn** (*callable, required*) – A search space function which when called returns a *HyperSpace* instance
- **objectives** (*List[Objective], optional, (default to NumOfFeatures instance)*) – The optimization objectives.
- **ref_point** (*Tuple[float], required*) – user-specified reference point, used to guide the search toward the desired region.
- **weights** (*Tuple[float], optional, default to uniform*) – weights vector, provides more detailed information about what Pareto optimal to converge to.
- **dominance_threshold** (*float, optional, default to 0.3*) – distance threshold, in case of pareto-equivalent, compare distance between two solutions.
- **recombination** (Recombination, *required*) – the strategy to recombine DNA of parents to generate offspring. Builtin strategies: - ShuffleCrossOver - UniformCrossover - SinglePointCrossOver
- **mutate_probability** (*float, optional, default to 0.7*) – the probability of genetic variation for offspring, when the parents can not recombine, it will definitely mutate a gene for the generated offspring.
- **population_size** (*int, default to 30*) – size of population
- **space_sample_validation_fn** (*callable or None, (default=None)*) – used to verify the validity of samples from the search space, and can be used to add specific constraint rules to the search space to reduce the size of the space.
- **random_state** (*np.RandomState, optional*) – used to reproduce the search process

References

[1] L. Ben Said, S. Bechikh and K. Ghedira, "The r-Dominance: A New Dominance Relation for Interactive Evolutionary Multicriteria Decision Making," in IEEE Transactions on Evolutionary Computation, vol. 14, no. 5, pp. 801-818, Oct. 2010, doi: 10.1109/TEVC.2010.2041060.

hypernets.searchers.playback_searcher module

class hypernets.searchers.playback_searcher.**PlaybackSearcher**(*history: hypernets.core.trial.TrialHistory*, *top_n=None*, *reverse=False*, *optimize_direction=<OptimizeDirection.Minim 'min'>*)

Bases: hypernets.core.searcher.Searcher

parallelizable

sample(*space_options=None*)

update_result(*space*, *result*)

hypernets.searchers.random_searcher module

class hypernets.searchers.random_searcher.**RandomSearcher**(*space_fn*, *optimize_direction=<OptimizeDirection.Minimize: 'min'>*, *space_sample_validation_fn=None*)

Bases: hypernets.core.searcher.Searcher

export()

get_best()

parallelizable

reset()

sample(*space_options=None*)

update_result(*space*, *result*)

Module contents

hypernets.searchers.**get_searcher_cls**(*identifier*)

hypernets.searchers.**make_searcher**(*cls*, *search_space_fn*, *optimize_direction='min'*, *objectives=None*, ***kwargs*)

1.9.2 Module contents

1.10 Release Notes

Releasing history:

1.10.1 Version 0.2.5

We add a few new features to this version:

- Toolbox: A general computing layer for tabular data - Provide implementations of pandas, dask and cudf data types
  - DefaultToolbox (Numpy + Pandas + Sklearn)
  - DaskToolbox (DaskCore + DaskML)
  - CumlToolBox (Cupy + Cudf + Cuml)
- HyperCtl: A tool package for multi-job management - Support sequencial jobs with multi-parameter settings - Support parallel jobs in remote multi-machines
- Export experiment report (.xlsx) - Include information of engineering features, ensembled models, evaluation scores, resource usages, etc. - Generate plots automatically

1.10.2 Version 0.3.0

We add a few new features to this version:

- Multi-objectives optimization
  - **optimization algorithm**
    - add MOEA/D(Multiobjective Evolutionary Algorithm Based on Decomposition)
    - add Tchebycheff, Weighted Sum, Penalty-based boundary intersection approach(PBI) decompose approachs
    - add shuffle crossover, uniform crossover, single point crossover strategies for GA based algorithms
    - automatically normalize objectives of different dimensions
    - automatically convert maximization problem to minimization problem
    - add NSGA-II(Non-dominated Sorting Genetic Algorithm)
    - add R-NSGA-II(A new dominance relation for multicriteria decision making)
  - **builtin objectives**
    - number of features
    - prediction performance

1.11 FAQ

1.11.1 How...

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

Index

D

E

F

G

H

I

K

L

M

N

P

R

S

T

U

W