evolearn

Release 0.1

Hindy Yuen Shing Yan

Apr 09, 2022

CONTENTS:


evolearn

CHAPTER

ONE

INSTALLATION

To use evolearn, first install it using pip:

```
(.venv) $ pip install evolearn
```

CHAPTER

TWO

GENETIC HYPERPARAMETER TUNING CV

To perform hyperparameter tuning using genetic algoritm, you need to first import other modules from

1) `evolearn.hyperparameter_tuning.initialization`
2) `evolearn.hyperparameter_tuning.evaluation`
3) `evolearn.hyperparameter_tuning.selection`
4) `evolearn.hyperparameter_tuning.mating`
5) `evolearn.hyperparameter_tuning.reproduction`
6) `evolearn.hyperparameter_tuning.mutation`
7) `evolearn.hyperparameter_tuning.environment` (optional)
8) `evolearn.hyperparameter_tuning.genetic_hyperparameter_tuning`

Although the modules from `environment` are optional for you to determine to use them in your search or not, the searching might end up stopping early or not finding the ideal results. These modules can help to prevent pre-mature convergence and also control other hyperparameters for GA.

For example:

```
>>> from evolearn.hyperparameter_tuning.initialization import Genes
>>> from evolearn.hyperparameter_tuning.evaluation import FitnessFunction
>>> from evolearn.hyperparameter_tuning.selection import (RankSelection,
                                                          RouletteWheelSelection,
                                                          SteadyStateSelection,
                                                          TournamentSelection,
                                                          StochasticUniversalSampling,
                                                          BoltzmannSelection
                                                          )
>>> from evolearn.hyperparameter_tuning.mating import MatingFunction
>>> from evolearn.hyperparameter_tuning.reproduction import (KPointCrossover,
                                                             LinearCombinationCrossover,
                                                             FitnessProportionateAverage
                                                             )
>>> from evolearn.hyperparameter_tuning.mutation import (Boundary,
                                                         Shrink
                                                         )
>>> from evolearn.hyperparameter_tuning.environment import (AdaptiveReproduction,
                                                            AdaptiveMutation,
                                                            Elitism
                                                            )
```

(continues on next page)

(continued from previous page)

```
>>> from evolearn.hyperparameter_tuning.genetic_hyperparameter_tuning import␣
↪GenesSearchCV
>>> from sklearn.ensemble import RandomForestRegressor
>>> search_space_rf = {
              'max_depth':(1, 16, 'uniform'),
              'n_estimators':(100, 1000, 'uniform'),
              'criterion':('squared_error', 'absolute_error', 'poisson')
          }
>>> opt = GenesSearchCV(
          n_gen=10,
          initialization_fn=Genes(search_space=search_space_rf, pop_size=30),
          fitness_fn=FitnessFunction(
              estimator=RandomForestRegressor(n_jobs=-1),
              cv=3,
              scoring='neg_mean_absolute_error',
          ),
          selection_fn=StochasticUniversalSampling(.7),
          mating_fn=MatingFunction(increst_prevention=False),
          reproduction_fn=KPointCrossover(1),
          mutation_fn=Shrink(),
          adaptive_population=AdaptiveReproduction(10),
          elitism=Elitism(),
          adaptive_mutation=AdaptiveMutation()
      )
>>> opt.fit(X_train, y_train)
Max Fitness: -2023.200579609583
{'max_depth': 5, 'n_estimators': 561, 'criterion': 'absolute_error'}
```

The choices of `selection_fn`, `reproduction_fn`, `mutation_fn` are actually up to your personal preference. One can pick what they believe are most benefit to their searching preocess.

CHAPTER

THREE

GENETIC FEATURE SELECTION

To perform feature selection using genetic algoritm, you need to first import other modules from

1) `evolearn.feature_selection.initialization`
2) `evolearn.feature_selection.evaluation`
3) `evolearn.feature_selection.selection`
4) `evolearn.feature_selection.mating`
5) `evolearn.feature_selection.reproduction`
6) `evolearn.feature_selection.mutation`
7) `evolearn.feature_selection.environment` (optional)
8) `evolearn.feature_selection.genetic_hyperparameter_tuning`

The modules looks similar to those modules from the `GenesSearchCV` section, but in fact their internal mechanisim work slightly differently. You need to be ware of importing the wrong modules when using genetic feature selection.

For example:

```
>>> from evolearn.feature_selection.initialization import Genes
>>> from evolearn.feature_selection.evaluation import FitnessFunction
>>> from evolearn.feature_selection.selection import (RankSelection,
                                                      RouletteWheelSelection,
                                                      SteadyStateSelection,
                                                      TournamentSelection,
                                                      StochasticUniversalSampling,
                                                      BoltzmannSelection
                                                      )
>>> from evolearn.feature_selection.mating import MatingFunction
>>> from evolearn.feature_selection.reproduction import KPointCrossover
>>> from evolearn.feature_selection.mutation import (BitStringMutation,
                                                     ExchangeMutation,
                                                     ShiftMutation
                                                     )
>>> from evolearn.feature_selection.environment import (AdaptiveReproduction,
                                                     AdaptiveMutation,
                                                     Elitism
                                                     )
>>> from evolearn.feature_selection.genetic_feature_selection import␣
↪GeneticFeatureSelection
>>> from sklearn.ensemble import RandomForestRegressor
```

(continues on next page)

(continued from previous page)

```
>>> opt = GeneticFeatureSelection(
        n_gen=10,
        initialization_fn=Genes(pop_size=50),
        fitness_fn=FitnessFunction(
            estimator=RandomForestRegressor(n_jobs=-1),
            cv=3,
            scoring='neg_mean_absolute_error'
        ),
        selection_fn=RouletteWheelSelection(.7),
        mating_fn=MatingFunction(),
        reproduction_fn=KPointCrossover(k=4),
        mutation_fn=BitStringMutation(),
        adaptive_population=None,
        elitism=None,
        adaptive_mutation=None
    )
>>> opt.fit(X_train, y_train)
>>> print(opt.best_fitness_)
>>> print(opt.best_params_)
-2797.7245589631652
{'age': True, 'sex': False, 'bmi': True, 'children': True, 'smoker': True, 'region':␣
↪False}
```

CHAPTER

FOUR

HYPERPARAMETER TUNING

4.1 GenesSearchCV

- `n_gen`: int
  - Maximum number of generation (or loop) GenesSearchCV will run.
- `initialization_fn`
  - Class object to generate solution candidates.
- `fitness_fn`
  - Class object to evalute the fitness of solution candidates.
- `selection_fn`
  - Class object to evalute the fitness of solution candidates.
  - **Can either be:**
    - **hyperparameter_tuning.selection.RankSelection,**
      - hyperparameter_tuning.selection.RouletteWheelSelection,
      - hyperparameter_tuning.selection.SteadyStateSelection,
      - hyperparameter_tuning.selection.TournamentSelection,
      - hyperparameter_tuning.selection.StochasticUniversalSampling,
      - hyperparameter_tuning.selection.BoltzmannSelection
- `mating_fn`
  - Class object to pair the solution candidates for reproduction.
- `reproduction_fn`
  - Class object to reproduce child population.
  - **Can either be**
    - hyperparameter_tuning.reproduction.KPointCrossover,
    - hyperparameter_tuning.reproduction.LinearCombinationCrossover,
    - hyperparameter_tuning.reproduction.FitnessProportionateAverage
- `mutation_fn`
  - Class object to mutate the child population.
  - **Can either be**

        * hyperparameter_tuning.mutation.Boundary,
        * hyperparameter_tuning.mutation.Shrink

- ``adaptive_population``=None
  - Class object to adaptively change the mating rate of the mating_fn.
- ``elitism``=None
  - Class object to perform elites selection, ace comparison and elites' traits induction.
- ``adaptive_mutation``=None
  - Class object to adaptively change the mutation probaility of the mutation_fn.

4.1.1 Initialization

4.1.2 Genes

- `search_space`: dict
  - Defines the search range of the algorithm. Where keys are parameter names (strings) and values are int, float or str. Represents search spaceover parameters of the provided estimator.
- `pop_size`: int
  - Size of the initial population.

4.1.3 Evaluation

4.1.4 FitnessFunction

- `estimator`: BaseEstimator
  - A object of that type is instantiated for each search point. This object is assumed to implement the scikit-learn estimator api. Either estimator needs to provide a `score` function, or `scoring` must be passed.
- `cv`: int
  - **cross-validation generator or an iterable, optional Determines the cross-validation splitting strategy. Possible inputs f**

        * None, to use the default 3-fold cross validation,
        * integer, to specify the number of folds in a *(Stratified)KFold*,
        * An object to be used as a cross-validation generator.
        * An iterable yielding train, test splits.

For integer/None inputs, if the estimator is a classifier and `y` is either binary or multiclass, `StratifiedKFold` is used. In all other cases, `KFold` is used.

- `scoring`: str

  - callable or None, default=None A string (see model evaluation documentation) or a scorer callable object / function with signature `scorer(estimator, X, y)`. If `None`, the `score` method of the estimator is used.

4.1.5 Selection

4.1.6 RankSelection

- `pct_survivors`: int, float
  - Argument that controls the number of survivors.

4.1.7 RouletteWheelSelection

- `pct_survivors`: int, float
  - Argument that controls the number of survivors.

4.1.8 SteadyStateSelection

- `elimination_ratio`: float [default=.3]
  - Determine how many candidates are eliminated.

4.1.9 TournamentSelection

- `k`: int [default=2]
  - Argument that controls the number of participants in each tourament.
- `preserve_remainders`: bool [default=True]
  - If True, the remaining individuals not selected for tournament will survive the selection process.

4.1.10 StochasticUniversalSampling

- `pct_survivors`: int, float
  - Argument that controls the number of survivors.

4.1.11 BoltzmannSelection

- `pct_survivors`: float
  - Argument that controls the number of survivors.
- `T0`: int, float
  - Initial Temperature to calculate Boltzmann probability. A number between [5, 100].
- `a`: int, float
  - Alpha, a constant between [0, 1].

4.1.12 Mating

4.1.13 MatingFunction

- `cr_proba`: int, float [default=1]
  - Percentage of survived population. Determines how many couples are paired during mating.
- `increst_prevention`: bool [default=True]
  - If True, solution candidates sharing the same parents will be paired together.

4.1.14 Reproduction

4.1.15 KPointCrossover

- `k`: int
  - Number of times of the chromosomes being splitted.
- `c_pt`: int, str [default='random']
  - If int, c_pt will be the position index of the splitting points. If str, the splitting point location where be randomly determined. If 'random', the splitting point will be randomly picked.

LinearCombinationCrossover * `a`: float

- Alpha, a constant to determine the scale of combinations.

FitnessProportionateAverage No parameters required to instantiate.

4.1.16 Mutation

Boundary * `epsilon`: float [default=.15]

- Mutation rate that determines if genes will mutate or not.

Shrink * `epsilon`: float [default=.15]

- Mutation rate that determines if genes will mutate or not.
- `prior`: str [default='normal']
  - Determines the probability distribution of sampling.

4.1.17 Environment

4.1.18 AdaptiveReproduction

- `pop_cap`: int [default=None]
  - Maximum population size.

4.1.19 AdaptiveMutation

- `a`: int, float [default=.2]
  - Alpha, a constant to adjust the self-adaptive mutation rate.

4.1.20 Elitism

- `pct`: int, float [default=.05]
  - Percentage of population being selected as elites.

CHAPTER

FIVE

FEATURE SELECTION

5.1 GeneticFeatureSelectionCV

- `n_gen`: int
  - Maximum number of generation (or loop) GenesSearchCV will run.
- `initialization_fn`
  - Class object to generate solution candidates.
- `fitness_fn`
  - Class object to evalute the fitness of solution candidates.
- `selection_fn`
  - Class object to evalute the fitness of solution candidates.
  - **Can either be:**
    - **optimization.selection.RankSelection,**
      - optimization.selection.RouletteWheelSelection,
      - optimization.selection.SteadyStateSelection,
      - optimization.selection.TournamentSelection,
      - optimization.selection.StochasticUniversalSampling,
      - optimization.selection.BoltzmannSelection
- `mating_fn`
  - Class object to pair the solution candidates for reproduction.
- `reproduction_fn`
  - Class object to reproduce child population.
  - **Can either be**
    - optimization.reproduction.KPointCrossover,
    - optimization.reproduction.LinearCombinationCrossover,
    - optimization.reproduction.FitnessProportionateAverage
- `mutation_fn`
  - Class object to mutate the child population.
  - **Can either be**

        * optimization.mutation.Boundary,
        * optimization.mutation.Shrink

- ``adaptive_population``=None
  - Class object to adaptively change the mating rate of the mating_fn.
- ``elitism``=None
  - Class object to perform elites selection, ace comparison and elites' traits induction.
- ``adaptive_mutation``=None
  - Class object to adaptively change the mutation probaility of the mutation_fn.

5.1.1 Initialization

5.1.2 Genes

- `search_space`: dict
  - Defines the search range of the algorithm. Where keys are parameter names (strings) and values are int, float or str. Represents search spaceover parameters of the provided estimator.
- `pop_size`: int
  - Size of the initial population.

5.1.3 Evaluation

5.1.4 FitnessFunction

- `estimator`: BaseEstimator
  - A object of that type is instantiated for each search point. This object is assumed to implement the scikit-learn estimator api. Either estimator needs to provide a `score` function, or `scoring` must be passed.
- `cv`: int
  - **cross-validation generator or an iterable, optional Determines the cross-validation splitting strategy. Possible inputs f**

        * None, to use the default 3-fold cross validation,
        * integer, to specify the number of folds in a *(Stratified)KFold*,
        * An object to be used as a cross-validation generator.
        * An iterable yielding train, test splits.

For integer/None inputs, if the estimator is a classifier and `y` is either binary or multiclass, `StratifiedKFold` is used. In all other cases, `KFold` is used.

- `scoring`: str
  - callable or None, default=None A string (see model evaluation documentation) or a scorer callable object / function with signature `scorer(estimator, X, y)`. If `None`, the `score` method of the estimator is used.

5.1.5 Selection

5.1.6 RankSelection

- `pct_survivors`: int, float
  - Argument that controls the number of survivors.

5.1.7 RouletteWheelSelection

- `pct_survivors`: int, float
  - Argument that controls the number of survivors.

5.1.8 SteadyStateSelection

- `elimination_ratio`: float [default=.3]
  - Determine how many candidates are eliminated.

5.1.9 TournamentSelection

- `k`: int [default=2]
  - Argument that controls the number of participants in each tourament.
- `preserve_remainders`: bool [default=True]
  - If True, the remaining individuals not selected for tournament will survive the selection process.

5.1.10 StochasticUniversalSampling

- `pct_survivors`: int, float
  - Argument that controls the number of survivors.

5.1.11 BoltzmannSelection

- `pct_survivors`: float
  - Argument that controls the number of survivors.
- `T0`: int, float
  - Initial Temperature to calculate Boltzmann probability. A number between [5, 100].
- `a`: int, float
  - Alpha, a constant between [0, 1].

5.1.12 Mating

5.1.13 MatingFunction

- `cr_proba`: int, float [default=1]
  - Percentage of survived population. Determines how many couples are paired during mating.
- `increst_prevention`: bool [default=True]
  - If True, solution candidates sharing the same parents will be paired together.

5.1.14 Reproduction

5.1.15 KPointCrossover

- `k`: int
  - Number of times of the chromosomes being splitted.
- `c_pt`: int, str [default='random']
  - If int, c_pt will be the position index of the splitting points. If str, the splitting point location where be randomly determined. If 'random', the splitting point will be randomly picked.

5.1.16 Mutation

5.1.17 BitStringMutation

- `epsilon`: float [default=.15]
  - Mutation rate that determines if genes will mutate or not.

5.1.18 ExchangeMutation

- `epsilon`: float [default=.15]
  - Mutation rate that determines if genes will mutate or not.

5.1.19 ShiftMutation

- `epsilon`: float [default=.15]
  - Mutation rate that determines if genes will mutate or not.

5.1.20 Environment

5.1.21 AdaptiveReproduction

- `pop_cap`: int [default=None]
  - Maximum population size.

5.1.22 AdaptiveMutation

- `a`: int, float [default=.2]
  - Alpha, a constant to adjust the self-adaptive mutation rate.

5.1.23 Elitism

- `pct`: int, float [default=.05]
  - Percentage of population being selected as elites.

CHAPTER

SIX

WARNINGS

6.1 Population Decline Warning

- The `Population Decline Warning` indicates the current population size has become smaller than the previous generation, which might leads to premature convergence

6.2 Elitism Failed Warning

- The `Elitism Failed Warning` indicates the number of elites selected was zero due to round issue.

6.3 Low Population Warning

- The `Low Population Warning` indicates the initial population size might be too smaller. Which might leads to premature convergence.

6.4 Low Population Diversity Warning

- The `Low Population Diversity Warning` indicates most of the candidates in the current generation were reproduced by the same parents. Which might leads to premature convergence.