

---

# **sklearn-genetic**

**Manuel Calzolari**

**Jan 26, 2024**



# CONTENTS

<b>1</b>	<b>Installation</b>	<b>3</b>
1.1	Dependencies . . . . .	3
1.2	User installation . . . . .	3
<b>2</b>	<b>Examples</b>	<b>5</b>
<b>3</b>	<b>API Reference</b>	<b>7</b>
<b>4</b>	<b>Indices and tables</b>	<b>11</b>
	<b>Python Module Index</b>	<b>13</b>
	<b>Index</b>	<b>15</b>



**sklearn-genetic** is a genetic feature selection module for scikit-learn.

Genetic algorithms mimic the process of natural selection to search for optimal values of a function.



## INSTALLATION

### 1.1 Dependencies

sklearn-genetic requires:

- Python ( $\geq 3.7$ )
- scikit-learn ( $\geq 1.0$ )
- deap ( $\geq 1.0.2$ )
- numpy
- multiprocessing

### 1.2 User installation

The easiest way to install sklearn-genetic is using `pip`

```
pip install sklearn-genetic
```

or `conda`

```
conda install -c conda-forge sklearn-genetic
```





## EXAMPLES

Noisy (non informative) features are added to the iris data and genetic feature selection is applied.

```
import random
import numpy as np
from sklearn import datasets, linear_model
from genetic_selection import GeneticSelectionCV

# When using multiple processes (n_jobs != 1), protect the entry point of the program if
# necessary
if __name__ == "__main__":

    # Set seed for reproducibility
    random.seed(42)
    np.random.seed(42)

    iris = datasets.load_iris()

    # Some noisy data not correlated
    E = np.random.uniform(0, 0.1, size=(len(iris.data), 20))

    X = np.hstack((iris.data, E))
    y = iris.target

    estimator = linear_model.LogisticRegression(solver="liblinear", multi_class="ovr")

    selector = GeneticSelectionCV(
        estimator,
        cv=5,
        verbose=1,
        scoring="accuracy",
        max_features=5,
        n_population=50,
        crossover_proba=0.5,
        mutation_proba=0.2,
        n_generations=40,
        crossover_independent_proba=0.5,
        mutation_independent_proba=0.05,
        tournament_size=3,
        n_gen_no_change=10,
        caching=True,
```

(continues on next page)

(continued from previous page)

```
        n_jobs=-1,  
    )  
    selector = selector.fit(X, y)  
  
    print(selector.support_)
```

## API REFERENCE

```
class genetic_selection.GeneticSelectionCV(estimator, cv=None, scoring=None, fit_params=None,  
                                         max_features=None, min_features=None, verbose=0,  
                                         n_jobs=1, n_population=300, crossover_proba=0.5,  
                                         mutation_proba=0.2, n_generations=40,  
                                         crossover_independent_proba=0.1,  
                                         mutation_independent_proba=0.05, tournament_size=3,  
                                         n_gen_no_change=None, caching=False)
```

Feature selection with genetic algorithm.

### Parameters

- **estimator** (*object*) – A supervised learning estimator with a *fit* method.
- **cv** (*int*, *cross-validation generator or an iterable*, *optional*) – Determines the cross-validation splitting strategy. Possible inputs for *cv* are:
  - *None*, to use the default 3-fold cross-validation,
  - *integer*, to specify the number of folds.
  - An object to be used as a cross-validation generator.
  - An iterable yielding train/test splits.For integer/*None* inputs, if *y* is binary or multiclass, *StratifiedKfold* used. If the estimator is a classifier or if *y* is neither binary nor multiclass, *Kfold* is used.
- **scoring** (*string*, *callable or None*, *optional*, *default: None*) – A string (see model evaluation documentation) or a scorer callable object / function with signature *scorer(estimator, X, y)*.
- **fit\_params** (*dict*, *optional*) – Parameters to pass to the *fit* method.
- **max\_features** (*int or None*, *optional*) – The maximum number of features selected.
- **min\_features** (*int or None*, *optional*) – The minimum number of features selected.
- **verbose** (*int*, *default=0*) – Controls verbosity of output.
- **n\_jobs** (*int*, *default 1*) – Number of cores to run in parallel. Defaults to 1 core. If *n\_jobs=-1*, then number of jobs is set to number of cores.
- **n\_population** (*int*, *default=300*) – Number of population for the genetic algorithm.
- **crossover\_proba** (*float*, *default=0.5*) – Probability of crossover for the genetic algorithm.
- **mutation\_proba** (*float*, *default=0.2*) – Probability of mutation for the genetic algorithm.

- **n\_generations** (*int*, *default=40*) – Number of generations for the genetic algorithm.
- **crossover\_independent\_proba** (*float*, *default=0.1*) – Independent probability for each attribute to be exchanged, for the genetic algorithm.
- **mutation\_independent\_proba** (*float*, *default=0.05*) – Independent probability for each attribute to be mutated, for the genetic algorithm.
- **tournament\_size** (*int*, *default=3*) – Tournament size for the genetic algorithm.
- **n\_gen\_no\_change** (*int*, *default=None*) – If set to a number, it will terminate optimization when best individual is not changing in all of the previous **n\_gen\_no\_change** number of generations.
- **caching** (*boolean*, *default=False*) – If True, scores of the genetic algorithm are cached.

**n\_features\_**

The number of selected features with cross-validation.

**Type**

int

**support\_**

The mask of selected features.

**Type**

array of shape [n\_features]

**generation\_scores\_**

The maximum cross-validation score for each generation.

**Type**

array of shape [n\_generations]

**estimator\_**

The external estimator fit on the reduced dataset.

**Type**

object

**Examples**

An example showing genetic feature selection.

```
>>> import numpy as np
>>> from sklearn import datasets, linear_model
>>> from genetic_selection import GeneticSelectionCV
>>> iris = datasets.load_iris()
>>> E = np.random.uniform(0, 0.1, size=(len(iris.data), 20))
>>> X = np.hstack((iris.data, E))
>>> y = iris.target
>>> estimator = linear_model.LogisticRegression(solver="liblinear", multi_class="ovr
↪")
>>> selector = GeneticSelectionCV(estimator, cv=5)
>>> selector = selector.fit(X, y)
>>> selector.support_
array([ True  True  True  True False False False False False False False False])
```

(continues on next page)

(continued from previous page)

```
False False False False False False False False False False False False],  
dtype=bool)
```

**fit**(*X*, *y*, *groups=None*)

Fit the GeneticSelectionCV model and the underlying estimator on the selected features.

#### Parameters

- **X** (*array-like, sparse matrix*), *shape* = [*n\_samples*, *n\_features*] – The training input samples.
- **y** (*array-like*, *shape* = [*n\_samples*]) – The target values.
- **groups** (*array-like, shape* = [*n\_samples*], *optional*) – Group labels for the samples used while splitting the dataset into train/test set. Only used in conjunction with a “Group” *cv* instance (e.g., *GroupKFold*).

**predict**(*X*)

Reduce *X* to the selected features and then predict using the underlying estimator.

#### Parameters

**X** (*array of shape* [*n\_samples*, *n\_features*]) – The input samples.

#### Returns

**y** – The predicted target values.

#### Return type

array of shape [*n\_samples*]

**score**(*X*, *y*)

Reduce *X* to the selected features and return the score of the underlying estimator.

#### Parameters

- **X** (*array of shape* [*n\_samples*, *n\_features*]) – The input samples.
- **y** (*array of shape* [*n\_samples*]) – The target values.

**set\_fit\_request**(*\**, *groups: bool | None | str = '\$UNCHANGED\$'*) → *GeneticSelectionCV*

Request metadata passed to the *fit* method.

Note that this method is only relevant if *enable\_metadata\_routing=True* (see *sklearn.set\_config()*). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- **True**: metadata is requested, and passed to *fit* if provided. The request is ignored if metadata is not provided.
- **False**: metadata is not requested and the meta-estimator will not pass it to *fit*.
- **None**: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- **str**: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (*sklearn.utils.metadata\_routing.UNCHANGED*) retains the existing request. This allows you to change the request for some parameters and not others.

New in version 1.3.

---

**Note:** This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a Pipeline. Otherwise it has no effect.

---

**Parameters**

**groups** (*str, True, False, or None, default=sklearn.utils.metadata\_routing.UNCHANGED*) – Metadata routing for groups parameter in fit.

**Returns**

**self** – The updated object.

**Return type**

object

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`





## PYTHON MODULE INDEX

### g

`genetic_selection`, [7](#)



## INDEX

### E

`estimator_` (*genetic\_selection.GeneticSelectionCV* attribute), 8

### F

`fit()` (*genetic\_selection.GeneticSelectionCV* method), 9

### G

`generation_scores_` (*genetic\_selection.GeneticSelectionCV* attribute), 8

`genetic_selection`  
module, 7

`GeneticSelectionCV` (class in *genetic\_selection*), 7

### M

module  
    *genetic\_selection*, 7

### N

`n_features_` (*genetic\_selection.GeneticSelectionCV* attribute), 8

### P

`predict()` (*genetic\_selection.GeneticSelectionCV* method), 9

### S

`score()` (*genetic\_selection.GeneticSelectionCV* method), 9

`set_fit_request()` (*genetic\_selection.GeneticSelectionCV* method), 9

`support_` (*genetic\_selection.GeneticSelectionCV* attribute), 8