Fine-tuning the hyper parameters for a classifier in HEP

Miguel Fernández Gómez

3rd COMCHA lectures

Instituto Galego de Física de Altas Enerxías

A quick reminder and notation

A quick reminder and notation

- We want to train a model to help us differentiate between output classes

A quick reminder and notation

- We want to train a model to help us differentiate between output classes
- We need to split the dataset to achieve the tasks

A quick reminder and notation

- We want to train a model to help us differentiate between output classes
- We need to split the dataset to achieve the tasks

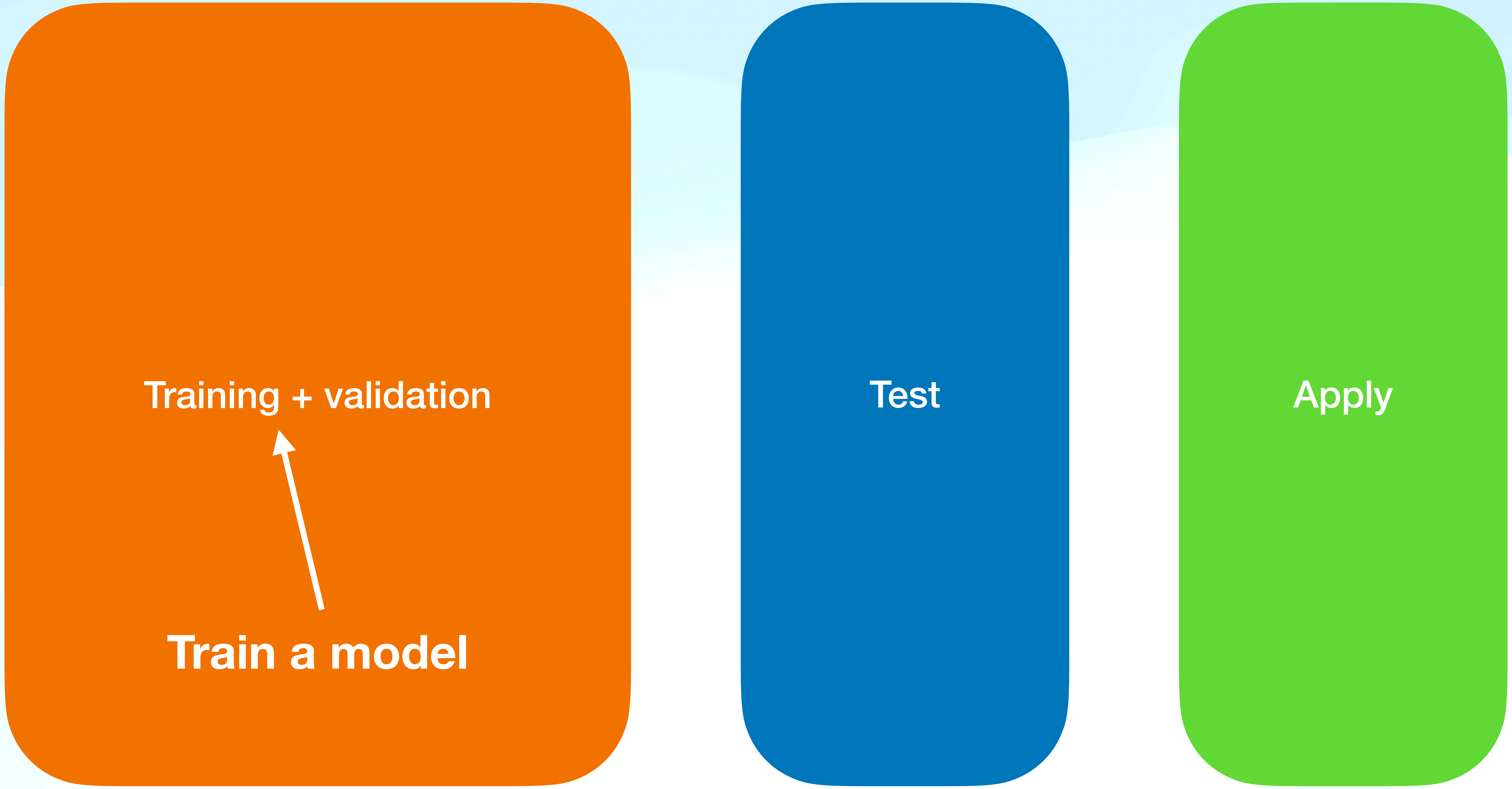
Training + validation

Test

Apply

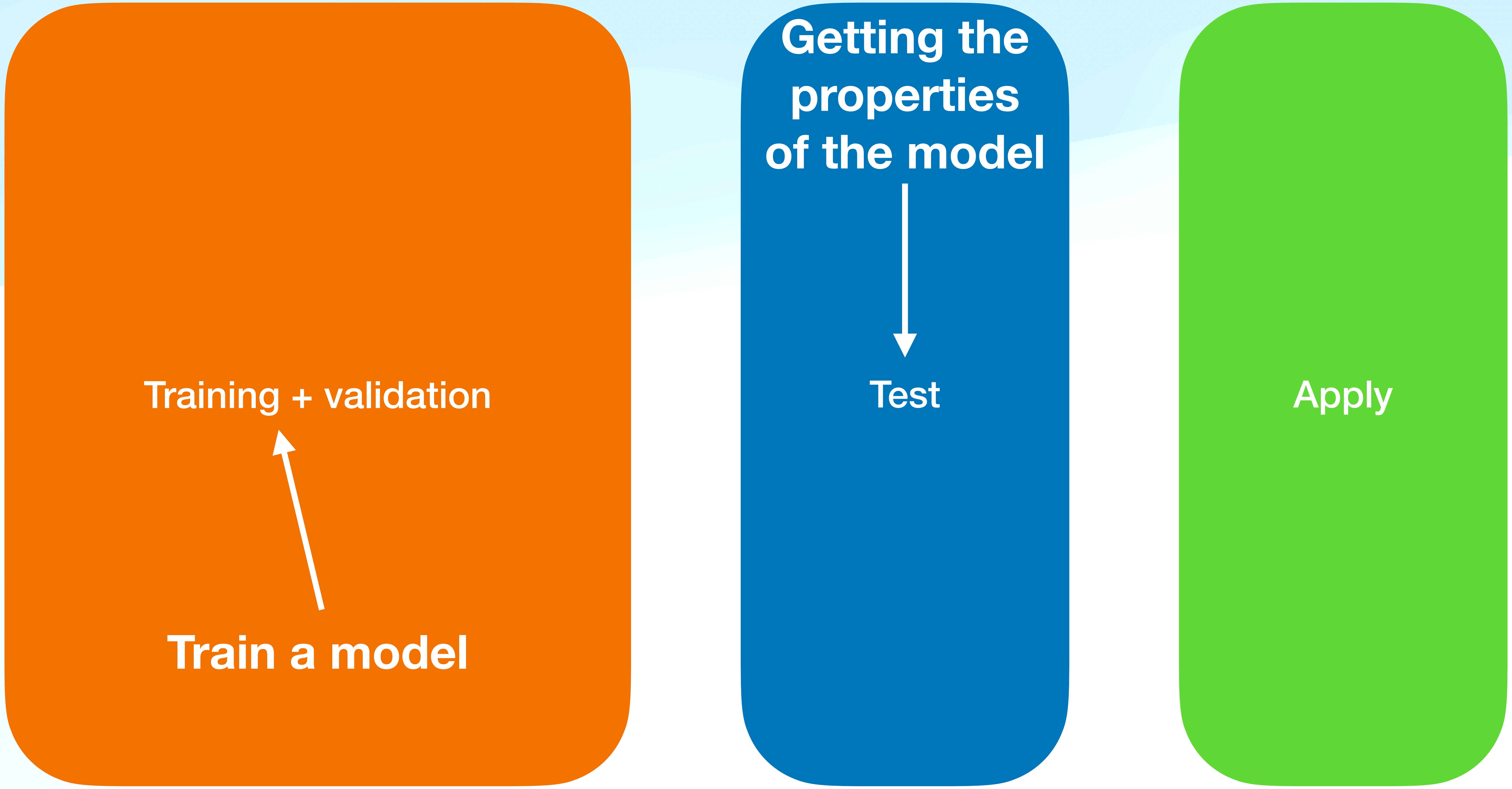
A quick reminder and notation

- We want to train a model to help us differentiate between output classes
- We need to split the dataset to achieve the tasks



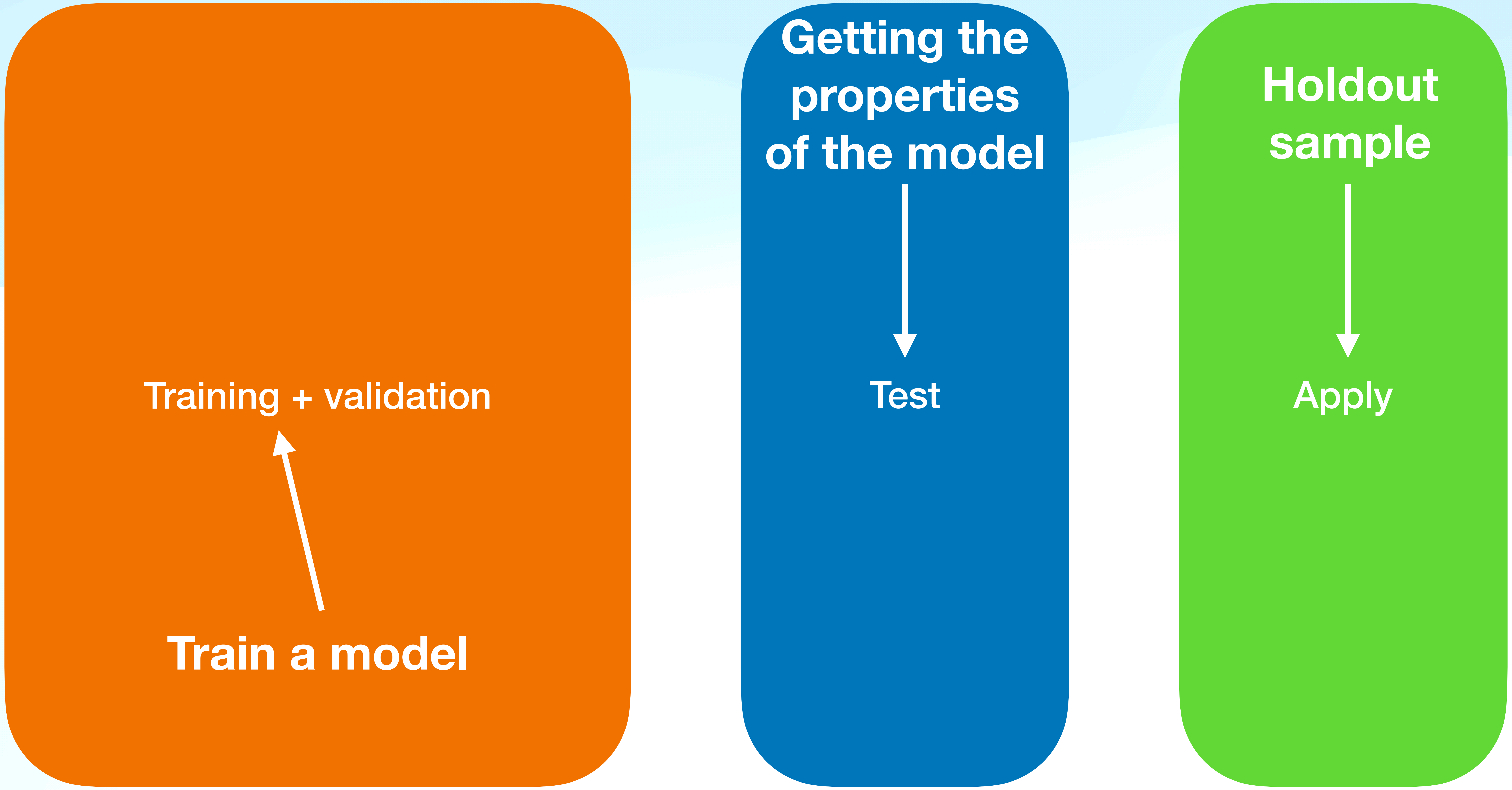
A quick reminder and notation

- We want to train a model to help us differentiate between output classes
- We need to split the dataset to achieve the tasks



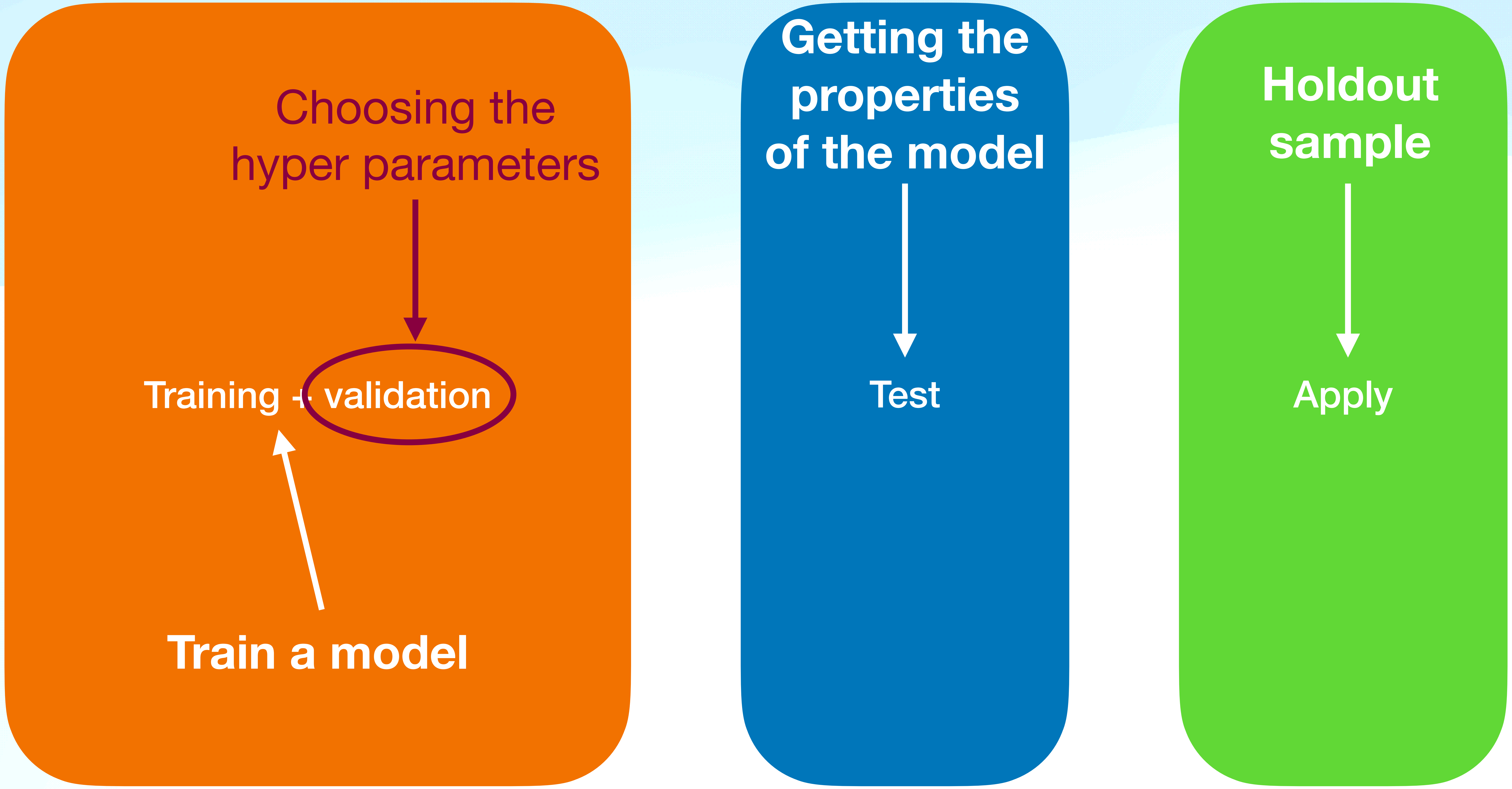
A quick reminder and notation

- We want to train a model to help us differentiate between output classes
- We need to split the dataset to achieve the tasks



A quick reminder and notation

- We want to train a model to help us differentiate between output classes
- We need to split the dataset to achieve the tasks



Here’s a common situation…

```
from sklearn.ensemble import AdaBoostClassifier

clf = AdaBoostClassifier(
```

Here's a common situation…

```python
from sklearn.ensemble import AdaBoostClassifier

clf = AdaBoostClassifier(
```

The problem of choosing hyper parameters

The problem of choosing hyper parameters

- Classifiers are used in HEP, e.g., to separate between signal-like and bkg-like events

The problem of choosing hyper parameters

- Classifiers are used in HEP, e.g., to separate between signal-like and bkg-like events

- BDTs are the go-to tools

The problem of choosing hyper parameters

- Classifiers are used in HEP, e.g., to separate between signal-like and bkg-like events
- BDTs are the go-to tools
- Options are vast → how do we know what's best?

The problem of choosing hyper parameters

- Classifiers are used in HEP, e.g., to separate between signal-like and bkg-like events
- BDTs are the go-to tools
- Options are vast → how do we know what's best?
- First: Define what's "best"

The problem of choosing hyper parameters

- Classifiers are used in HEP, e.g., to separate between signal-like and bkg-like events
- BDTs are the go-to tools
- Options are vast → how do we know what's best?
- First: Define what's "best"
    - Test score

The problem of choosing hyper parameters

- Classifiers are used in HEP, e.g., to separate between signal-like and bkg-like events
- BDTs are the go-to tools
- Options are vast → how do we know what’s best?
- First: Define what’s “best”
  - Test score
  - AUC

The problem of choosing hyper parameters

- Classifiers are used in HEP, e.g., to separate between signal-like and bkg-like events
- BDTs are the go-to tools
- Options are vast → how do we know what's best?
- First: Define what's "best"
    - Test score
    - AUC
    - Other metrics (precision, recall, specificity, etc.)

Scikit-learn and hyper parameter searches

Scikit-learn and hyper parameter searches

- Python's sklearn introduces several possible estimators (AdaBoost, XGBoost, DecisionTreeClassifier, etc.)

Scikit-learn and hyper parameter searches

- Python's sklearn introduces several possible estimators (AdaBoost, XGBoost, DecisionTreeClassifier, etc.)
- Each possible estimator has its own set of possible hyper parameters

Scikit-learn and hyper parameter searches

- Python’s sklearn introduces several possible estimators (AdaBoost, XGBoost, DecisionTreeClassifier, etc.)
- Each possible estimator has its own set of possible hyper parameters
- So how do we choose?

Scikit-learn and hyper parameter searches

- Python's sklearn introduces several possible estimators (AdaBoost, XGBoost, DecisionTreeClassifier, etc.)
- Each possible estimator has its own set of possible hyper parameters
- So how do we choose?
- ANSWER: HalvingRandomSearchCV

Scikit-learn and hyper parameter searches

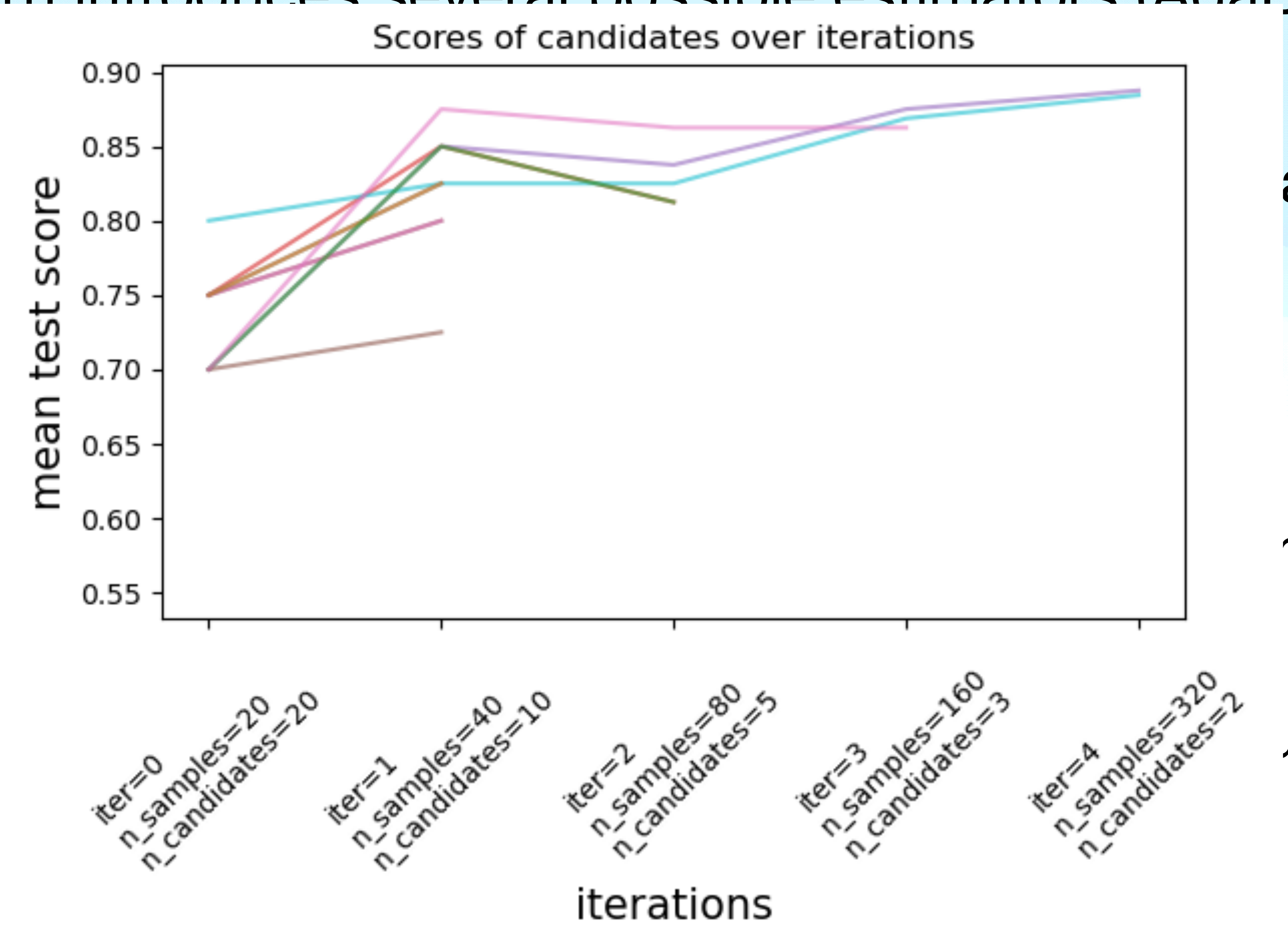
- Python's sklearn introduces several possible estimators (AdaBoost, XGBoost, DecisionTreeClassifier, etc.)
- Each possible estimator has its own set of possible hyper parameters
- So how do we choose?
- ANSWER: HalvingRandomSearchCV
- This new feature implements a state-of-the-art technique in choosing hyper parameters named successive halving

Scikit-learn and hyper parameter searches

- Python's sklearn introduces several possible estimators (AdaBoost, XGBoost, DecisionTreeClassifier, etc.)
- Each possible estimator has its own set of possible hyper parameters
- So how do we choose?
- ANSWER: HalvingRandomSearchCV
- This new feature implements a state-of-the-art technique in choosing hyper parameters named successive halving
- Successive halving allows us to allocate more and more resources to the models that are working best

Scikit-learn and hyper parameter searches

- Python's sklearn introduces several possible estimators (AdaBoost, XGBoost, DecisionTreeC
- Each possible ameters
- So how do we
- ANSWER: Hal
- This new featu hoosing hyper parameters na
- Successive ha urces to the models that ar



Scikit-learn’s successive halving

Scikit-learn’s successive halving

- IMPORTANT: Still in development

Scikit-learn’s successive halving

- IMPORTANT: Still in development
- Consider: estimator E with hyperparameter grid of N possibilities

Scikit-learn's successive halving

- IMPORTANT: Still in development
- Consider: estimator *E* with hyperparameter grid of *N* possibilities
- We have *n* events to train, divided into n_c target classes

Scikit-learn's successive halving

- IMPORTANT: Still in development
- Consider: estimator *E* with hyperparameter grid of *N* possibilities
- We have *n* events to train, divided into n_c target classes
- We divide the process in p iterations

Scikit-learn's successive halving

- IMPORTANT: Still in development
- Consider: estimator *E* with hyperparameter grid of *N* possibilities
- We have *n* events to train, divided into n_c target classes
- We divide the process in p iterations
- On each iteration, we k-fold

Scikit-learn's successive halving

- IMPORTANT: Still in development
- Consider: estimator *E* with hyperparameter grid of *N* possibilities
- We have *n* events to train, divided into n_c target classes
- We divide the process in p iterations
- On each iteration, we k-fold
  - Default: Stratified folding with $k = 5$ and test size = 0.2

Scikit-learn's successive halving

- IMPORTANT: Still in development
- Consider: estimator *E* with hyperparameter grid of *N* possibilities
- We have *n* events to train, divided into n_c target classes
- We divide the process in p iterations
- On each iteration, we k-fold
  - Default: Stratified folding with $k = 5$ and test size = 0.2
- The *i*-th iteration uses q events, where

Scikit-learn's successive halving

- IMPORTANT: Still in development
- Consider: estimator *E* with hyperparameter grid of *N* possibilities
- We have *n* events to train, divided into n_c target classes
- We divide the process in p iterations
- On each iteration, we k-fold
  - Default: Stratified folding with $k = 5$ and test size = 0.2
- The *i*-th iteration uses q events, where

$$q = \min\left(2 \times n_c \times k \times f^i,\, n\right);\ f = 3 \text{ (default)}$$

Scikit-learn’s successive halving (2)

Scikit-learn's successive halving (2)

- On the first iteration, by default we use n_p parameters

Scikit-learn's successive halving (2)

- On the first iteration, by default we use n_p parameters
    - HalvingRandomSearch: $n_p = n // q(i = 0)$

Scikit-learn's successive halving (2)

- On the first iteration, by default we use n_p parameters
  - HalvingRandomSearch: $n_p = n//q(i = 0)$
  - HalvingGridSearch: $n_p = N$

Scikit-learn's successive halving (2)

- On the first iteration, by default we use n_p parameters
  - HalvingRandomSearch: $n_p = n//q(i = 0)$
  - HalvingGridSearch: $n_p = N$
- Number of iterations: $p = 1 + floor(\log_f n_p)$

Scikit-learn's successive halving (2)

- On the first iteration, by default we use n_p parameters
  - HalvingRandomSearch: $n_p = n//q(i = 0)$
  - HalvingGridSearch: $n_p = N$
- Number of iterations: $p = 1 + floor(\log_f n_p)$
- At the end of each iteration, we keep $ceil(n_p/f)$

Scikit-learn's successive halving (2)

- On the first iteration, by default we use n_p parameters
  - HalvingRandomSearch: $n_p = n//q(i = 0)$
  - HalvingGridSearch: $n_p = N$
- Number of iterations: $p = 1 + floor(\log_f n_p)$
- At the end of each iteration, we keep $ceil(n_p/f)$
- Notice: In the end, N doesn't really matter for random halving

Example

Example

- Take sklearn's digits dataset

Example

- Take sklearn's digits dataset
    - $n = 1797; \quad n_c = 10$

Example

- Take sklearn's digits dataset
  - $n = 1797; \quad n_c = 10$
- We want to use sklearn's AdaBoostClassifier, which takes in:

Example

- Take sklearn's digits dataset
  - $n = 1797; \quad n_c = 10$
- We want to use sklearn's AdaBoostClassifier, which takes in:
  - A base estimator → We will avoid this for now (default: None)

Example

- Take sklearn's digits dataset
  - $n = 1797; \quad n_c = 10$
- We want to use sklearn's AdaBoostClassifier, which takes in:
  - A base estimator → We will avoid this for now (default: None)
  - Number of estimators → Let us take [20, 250] in steps of 5 (default: 50)

Example

- Take sklearn’s digits dataset
    - $n = 1797; \quad n_c = 10$
- We want to use sklearn’s AdaBoostClassifier, which takes in:
    - A base estimator → We will avoid this for now (default: None)
    - Number of estimators → Let us take [20, 250] in steps of 5 (default: 50)
    - Learning rate → [0.01, 0.5] in steps of 0.01 (default: 0.1)

Example

- Take sklearn's digits dataset
  - $n = 1797; \quad n_c = 10$
- We want to use sklearn's AdaBoostClassifier, which takes in:
  - A base estimator → We will avoid this for now (default: None)
  - Number of estimators → Let us take [20, 250] in steps of 5 (default: 50)
  - Learning rate → [0.01, 0.5] in steps of 0.01 (default: 0.1)
- Parameter grid: $N = 2350$ possibilities

Example

- Take sklearn's digits dataset
  - $n = 1797; \quad n_c = 10$
- We want to use sklearn's AdaBoostClassifier, which takes in:
  - A base estimator → We will avoid this for now (default: None)
  - Number of estimators → Let us take [20, 250] in steps of 5 (default: 50)
  - Learning rate → [0.01, 0.5] in steps of 0.01 (default: 0.1)
- Parameter grid: $N = 2350$ possibilities
  - $q(i = 0) = 100; \quad n_p = 17$

Example

- Take sklearn's digits dataset
  - $n = 1797; \quad n_c = 10$
- We want to use sklearn's AdaBoostClassifier, which takes in:
  - A base estimator → We will avoid this for now (default: None)
  - Number of estimators → Let us take [20, 250] in steps of 5 (default: 50)
  - Learning rate → [0.01, 0.5] in steps of 0.01 (default: 0.1)
- Parameter grid: $N = 2350$ possibilities
  - $q(i=0) = 100; \quad n_p = 17$
- Taking $f = 3$ → $p = 3$ iterations

Example

- Take sklearn's digits dataset
  - $n = 1797; \quad n_c = 10$
- We want to use sklearn's AdaBoostClassifier, which takes in:
  - A base estimator → We will avoid this for now (default: None)
  - Number of estimators → Let us take [20, 250] in steps of 5 (default: 50)
  - Learning rate → [0.01, 0.5] in steps of 0.01 (default: 0.1)
- Parameter grid: $N = 2350$ possibilities
  - $q(i = 0) = 100; \quad n_p = 17$
- Taking $f = 3 \rightarrow p = 3$ iterations

$$i = 0$$

$$N = 17, \quad q = 100$$

Example

- Take sklearn's digits dataset
  - $n = 1797; \quad n_c = 10$
- We want to use sklearn's AdaBoostClassifier, which takes in:
  - A base estimator → We will avoid this for now (default: None)
  - Number of estimators → Let us take [20, 250] in steps of 5 (default: 50)
  - Learning rate → [0.01, 0.5] in steps of 0.01 (default: 0.1)
- Parameter grid: $N = 2350$ possibilities
  - $q(i = 0) = 100; \quad n_p = 17$
- Taking $f = 3$ → $p = 3$ iterations

$i = 0$ | $N = 17, \quad q = 100$

$i = 1$ | $N = 6, \quad q = 300$

Example

- Take sklearn's digits dataset
  - $n = 1797; \quad n_c = 10$
- We want to use sklearn's AdaBoostClassifier, which takes in:
  - A base estimator → We will avoid this for now (default: None)
  - Number of estimators → Let us take [20, 250] in steps of 5 (default: 50)
  - Learning rate → [0.01, 0.5] in steps of 0.01 (default: 0.1)
- Parameter grid: $N = 2350$ possibilities
  - $q(i = 0) = 100; \quad n_p = 17$
- Taking $f = 3$ → $p = 3$ iterations

$i = 0$ | $N = 17, \quad q = 100$

$i = 1$ | $N = 6, \quad q = 300$

$i = 2$ | $N = 2, \quad q = 900$

HalvingRandomSearchCV

sklearn.model_selection.HalvingRandomSearchCV

class `sklearn.model_selection.`**`HalvingRandomSearchCV`**(*estimator, param_distributions, *, n_candidates='exhaust', factor=3, resource='n_samples', max_resources='auto', min_resources='smallest', aggressive_elimination=False, cv=5, scoring=None, refit=True, error_score=nan, return_train_score=True, random_state=None, n_jobs=None, verbose=0*) ¶ [source]

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.HalvingRandomSearchCV.html

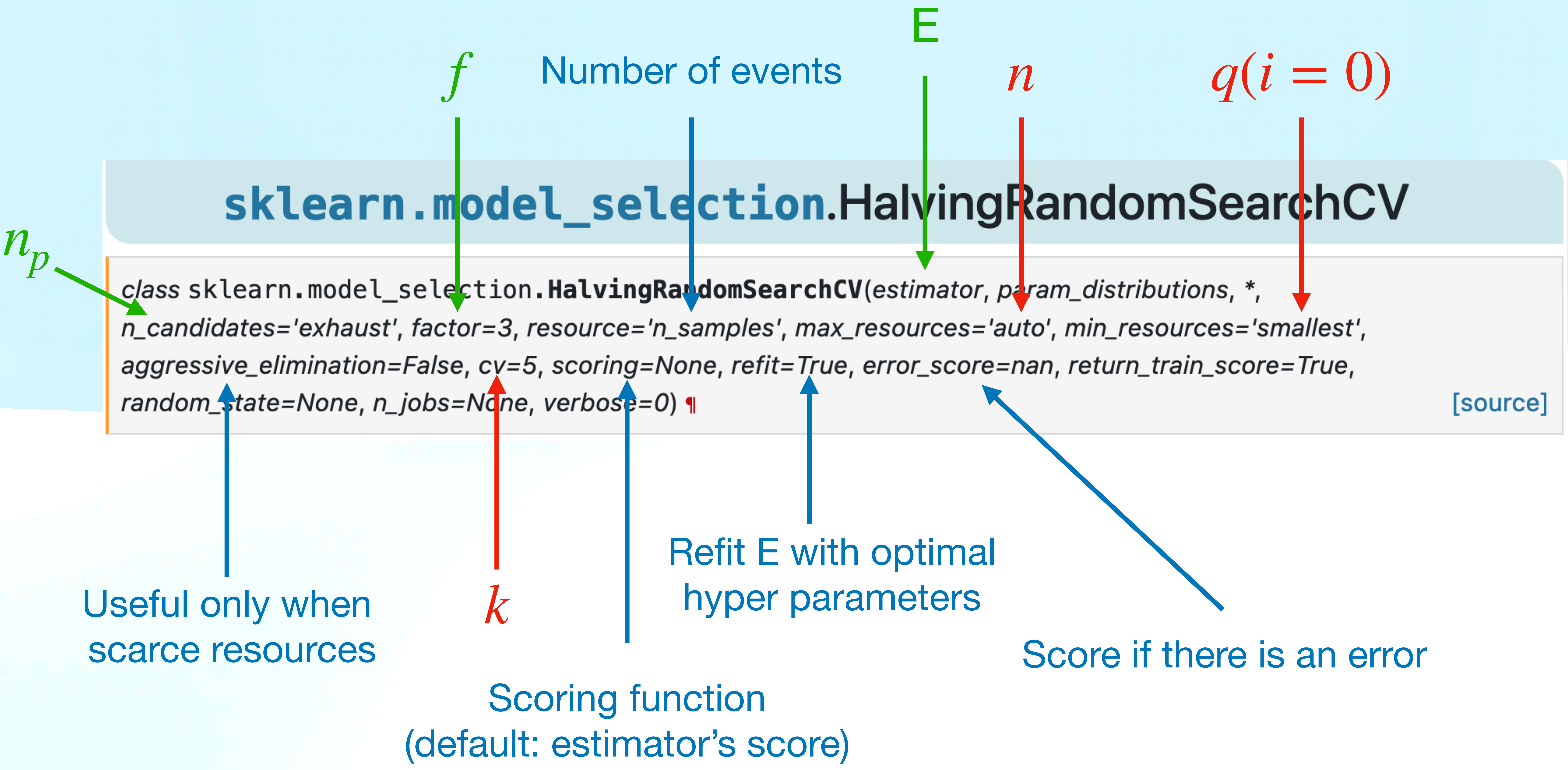
HalvingRandomSearchCV

sklearn.model_selection.HalvingRandomSearchCV

class sklearn.model_selection.**HalvingRandomSearchCV**(*estimator*, *param_distributions*, ***, *n_candidates='exhaust'*, *factor=3*, *resource='n_samples'*, *max_resources='auto'*, *min_resources='smallest'*, *aggressive_elimination=False*, *cv=5*, *scoring=None*, *refit=True*, *error_score=nan*, *return_train_score=True*, *random_state=None*, *n_jobs=None*, *verbose=0*) ¶ [source]

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.HalvingRandomSearchCV.html

HalvingRandomSearchCV



HalvingRandomSearchCV

f

Number of events

E

n

$q(i = 0)$

n_p

sklearn.model_selection.HalvingRandomSearchCV

class `sklearn.model_selection.`**`HalvingRandomSearchCV`**(*estimator*, *param_distributions*, ***, *n_candidates='exhaust'*, *factor=3*, *resource='n_samples'*, *max_resources='auto'*, *min_resources='smallest'*, *aggressive_elimination=False*, *cv=5*, *scoring=None*, *refit=True*, *error_score=nan*, *return_train_score=True*, *random_state=None*, *n_jobs=None*, *verbose=0*) ¶ [source]

Useful only when scarce resources

k

Scoring function (default: estimator's score)

Refit E with optimal hyper parameters

Score if there is an error

Practical examples

- Two simple examples on sklearn's digits dataset:

https://github.com/miguel-fernandez/comcha23/blob/main/halving.py

https://github.com/miguel-fernandez/comcha23/blob/main/halving2.py

Optuna

- Optuna is a more generic approach to the search of hyper parameters
- It can also implement successive halving inside
- General concept:
  - Define a function to maximize or minimise (e.g., the accuracy)
  - Define a hyper parameter space (can be even more generic than in sklearn)
  - Do random approaches, exploiting the areas where the function looks more promising

Optuna’s language

- Every hyperparameter search is called a *study*
- We define an *objective function* to minimize/maximize
- Each study is comprised of several *tries*
  - Each try is a random sampling on the hyperparameter space
  - A hyperparameter combination is chosen
  - The objective function outputs a value, which optuna uses to learn in subsequent trials

Optuna example

- Basic example:

  https://github.com/miguel-fernandez/comcha23/blob/main/optuna_test.py

- Challenge:
  - Implement halving2.py on optuna

Backup

Some minor gripes with sklearn…

- No easy implementation of a holdout sample (very easy to train-test split)
- No easy implementation of spectator variables

- This, combined with the desire to create easier access to Hyperparameter fine-tuning led me to putting together a still-in-development package:

https://github.com/miguel-fernandez/hep-mva