XÂY DỰNG MÔ HÌNH MÁY HỌC

NỘI DUNG

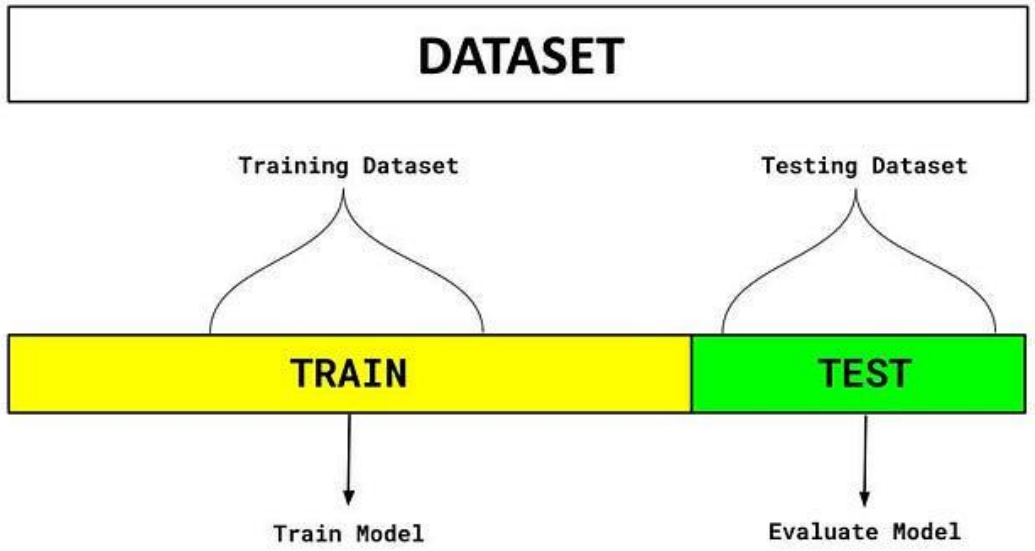
I. Model Selection và Evaluation
   A. Holdout Validation
   B. Cross Validation
   C. Hyper-parameters Optimization
   D. Evaluation Metric

II. Handle Imbalanced Data Method
   A. Resampling
   B. Cost sensitive learning
   C. Tools

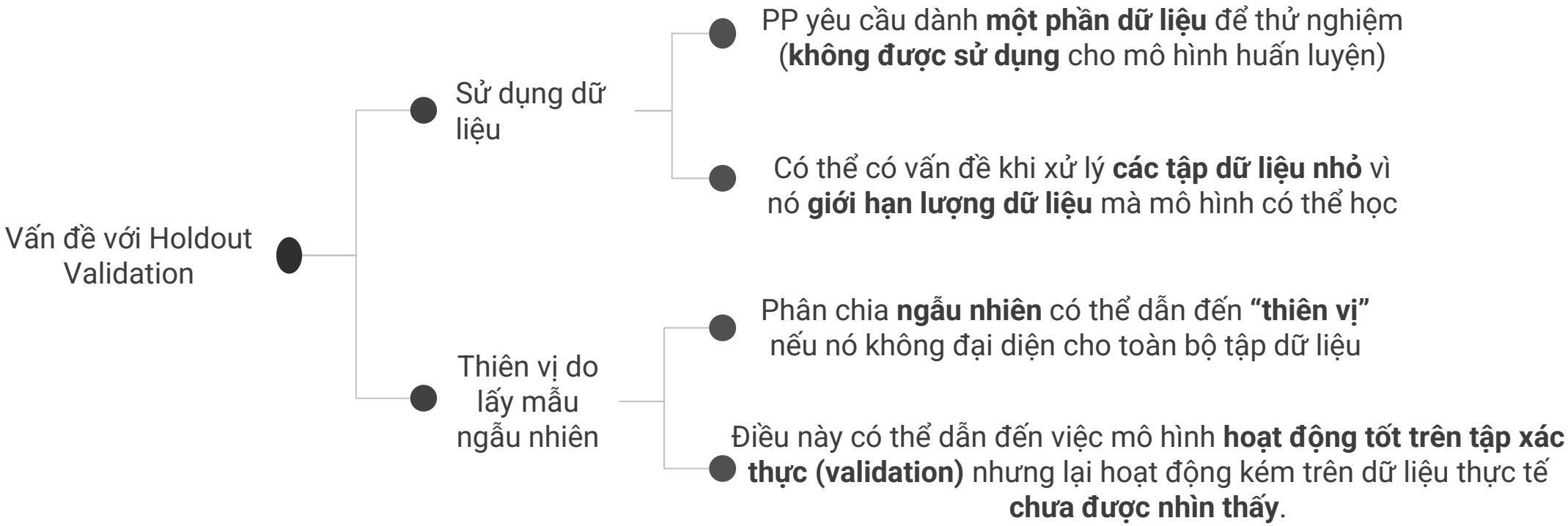
III. Error Analysis Model

Holdout Validation

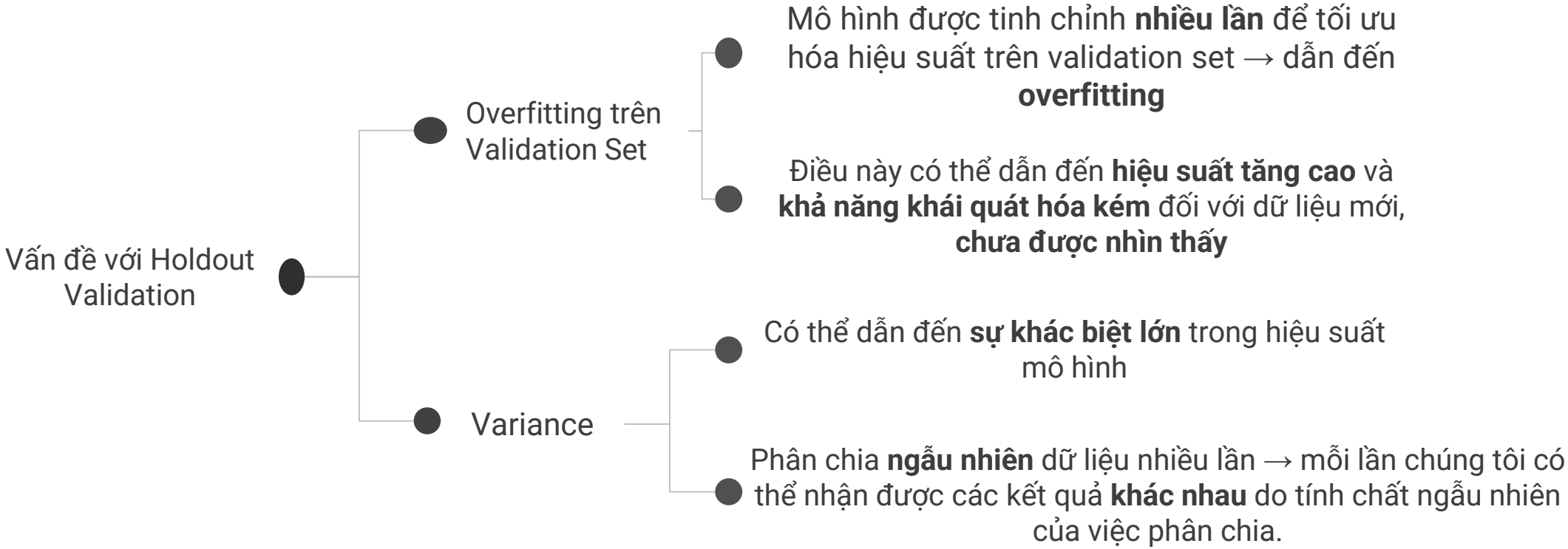
- Một phương pháp đơn giản để đánh giá các mô hình trên dữ liệu chưa biết
- Chia dư liệu hiện tại thành các tập con riêng biệt để tạo thành tập huấn luyện và tập xác thực (và có thể tập kiểm tra)



Vấn đề với Holdout Validation

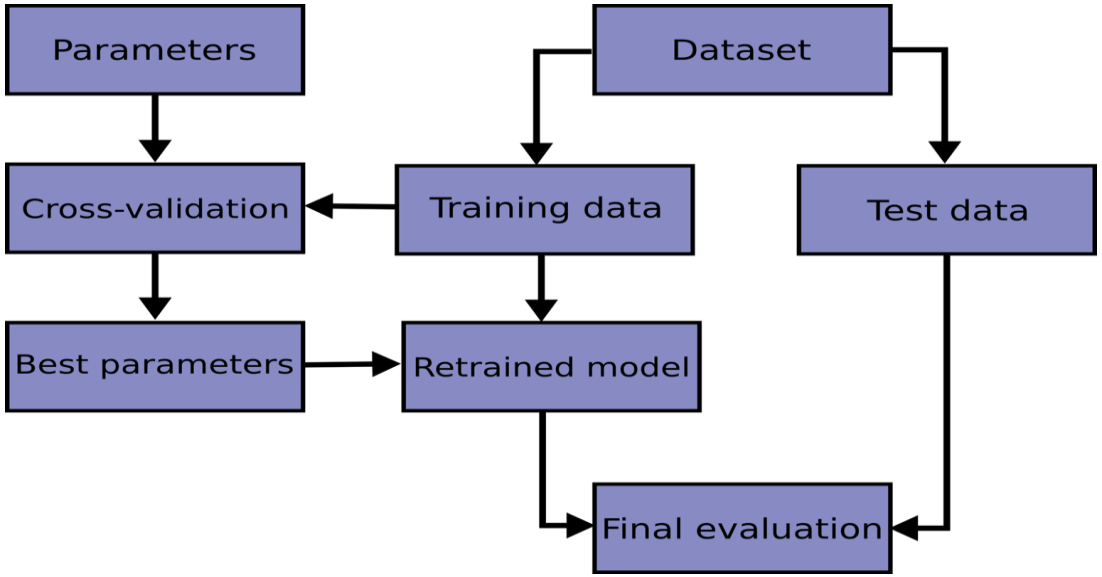


Vấn đề với Holdout Validation



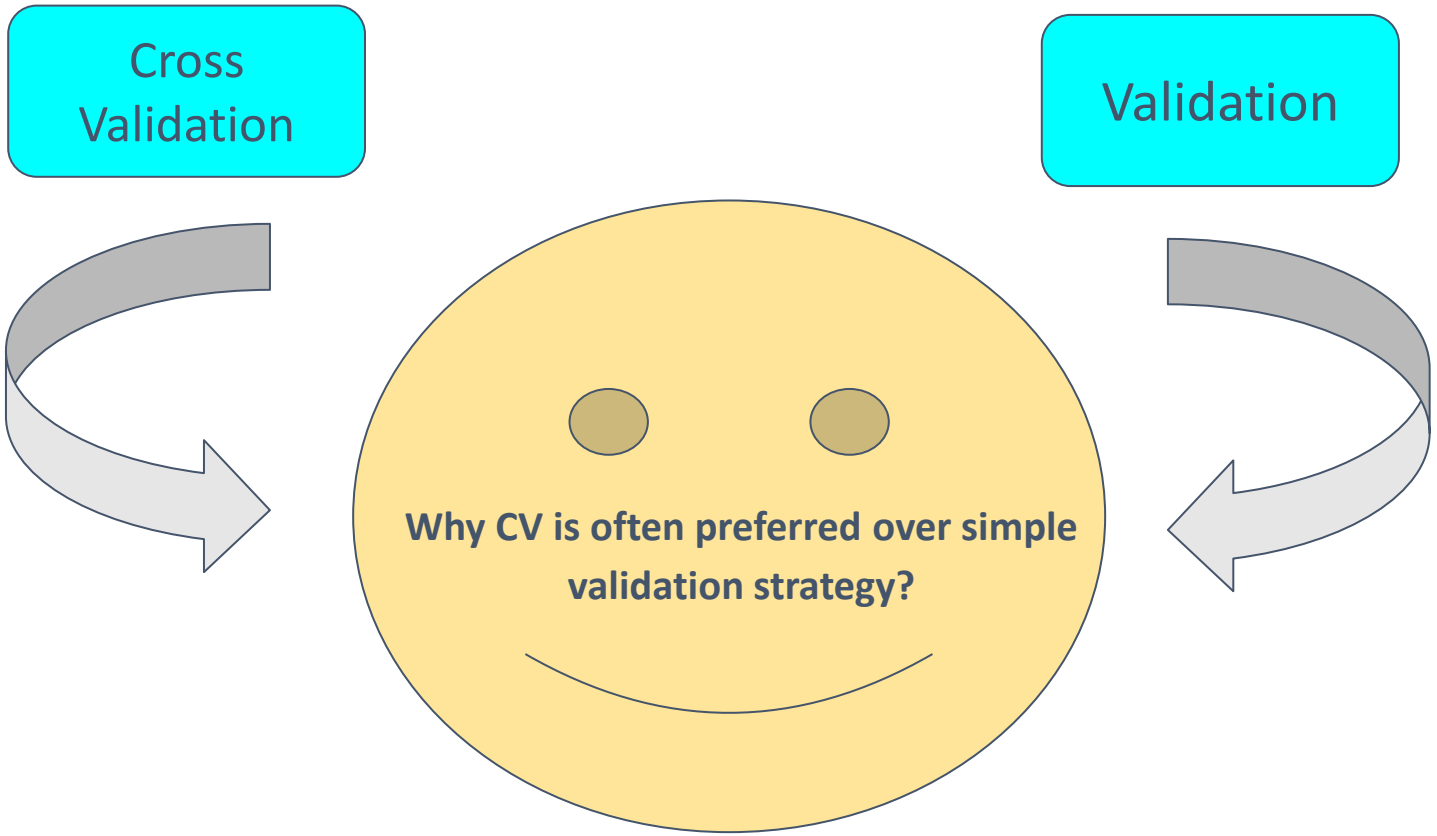
Cross Validation là gì?

- Cross validation (CV) là một kỹ thuật được sử dụng trong ML và thống kê để đánh giá hiệu suất của mô hình trên dữ liệu chưa nhìn thấy
- CV là phương pháp lấy mẫu lại sử dụng các phần khác nhau của dữ liệu để kiểm tra và huấn luyện mô hình trên các lần lặp khác nhau.

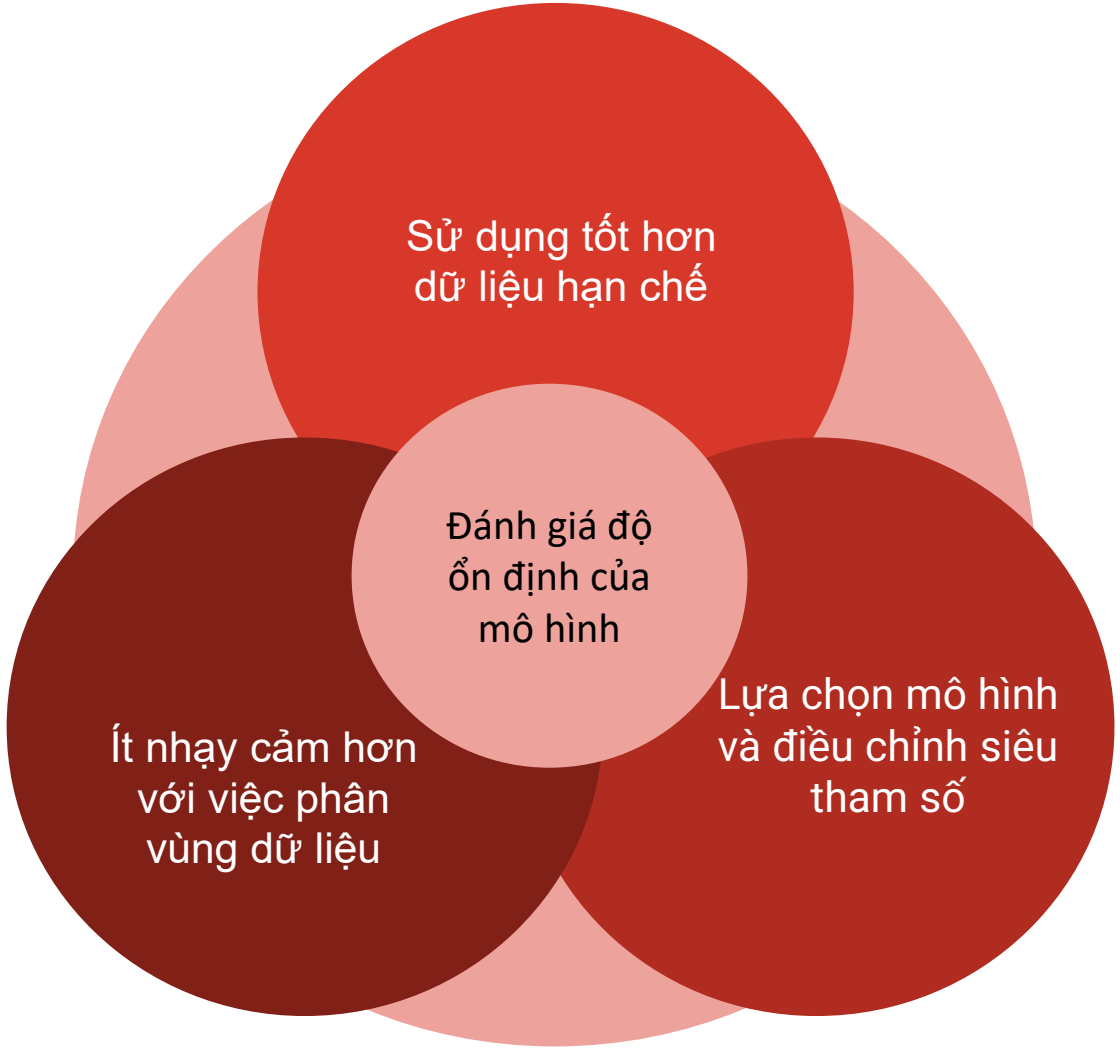


https://scikit-learn.org/stable/modules/cross_validation.html

Tại sao chúng ta cần Cross Validation?



Tại sao Cross Validation?



CV: Advantages & Disadvantages

Advantages

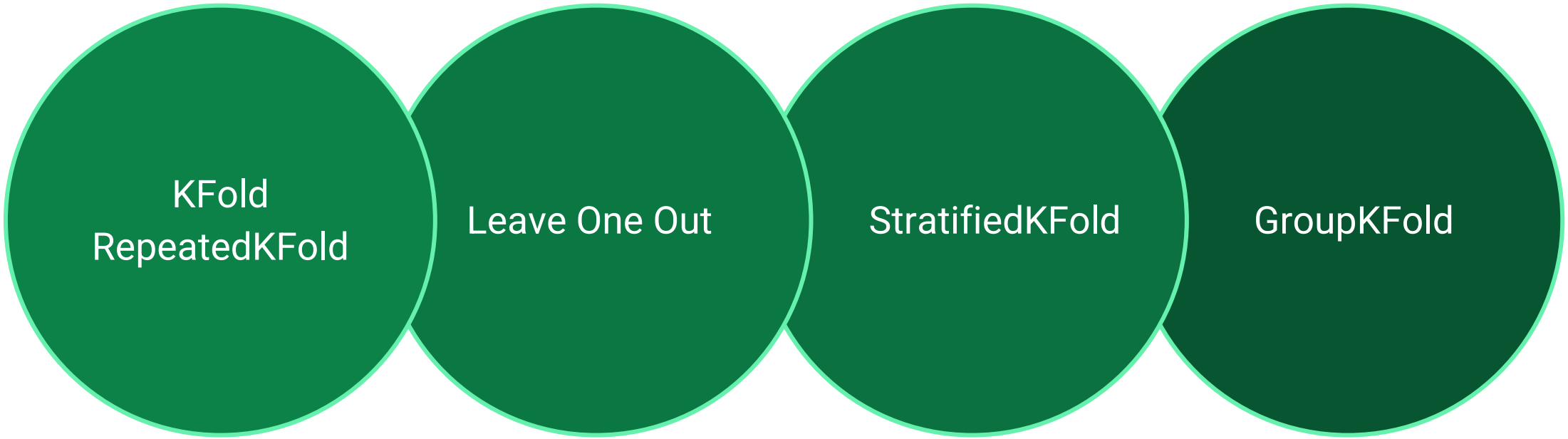
Thường được ưu tiên khi cần ước tính chính xác và đáng tin cậy hơn về hiệu suất mô hình, đặc biệt là để lựa chọn mô hình và điều chỉnh siêu tham số

Disadvantages

Tăng chi phí tính toán do huấn luyện và đánh giá mô hình nhiều lần

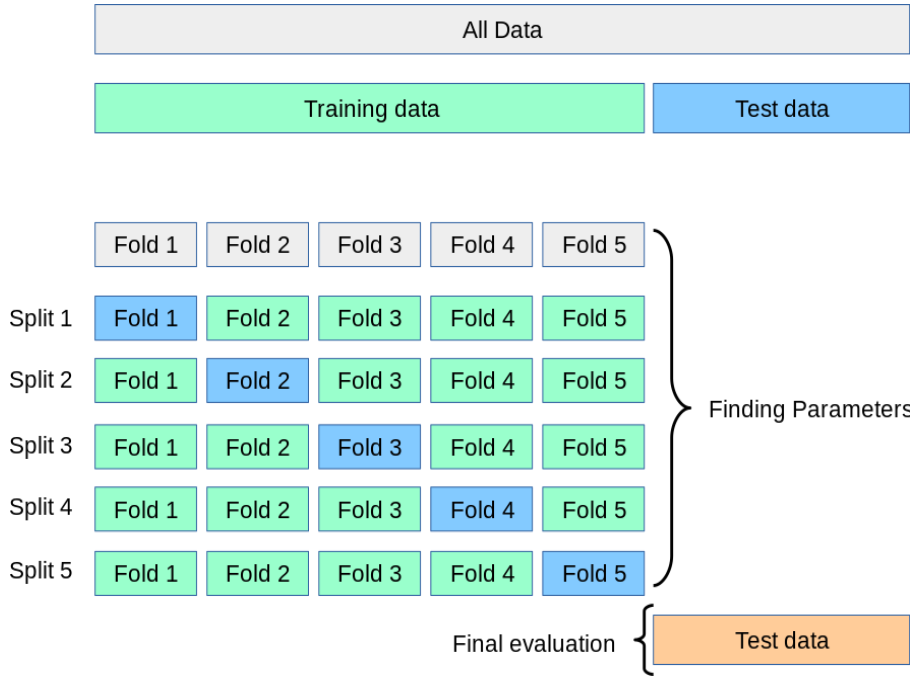
Tập dữ liệu lớn → **simple validation strategy** có thể thực tế hơn

Các loại Cross Validation



KFold

- Một mô hình được huấn luyện bằng cách sử dụng (k - 1) số fold làm dữ liệu huấn luyện, một fold gấp để xác thực
- Kết quả mô hình được đánh giá trên phần dữ liệu còn lại (được sử dụng làm bộ kiểm tra để tính toán độ đo hiệu suất, chẳng hạn như độ chính xác)




```
>>> import numpy as np
>>> from sklearn.model_selection import KFold

>>> X = ["a", "b", "c", "d"]
>>> kf = KFold(n_splits=2)
>>> for train, test in kf.split(X):
...     print("%s %s" % (train, test))
[2 3] [0 1]
[0 1] [2 3]
```

Leave One Out (LOO)

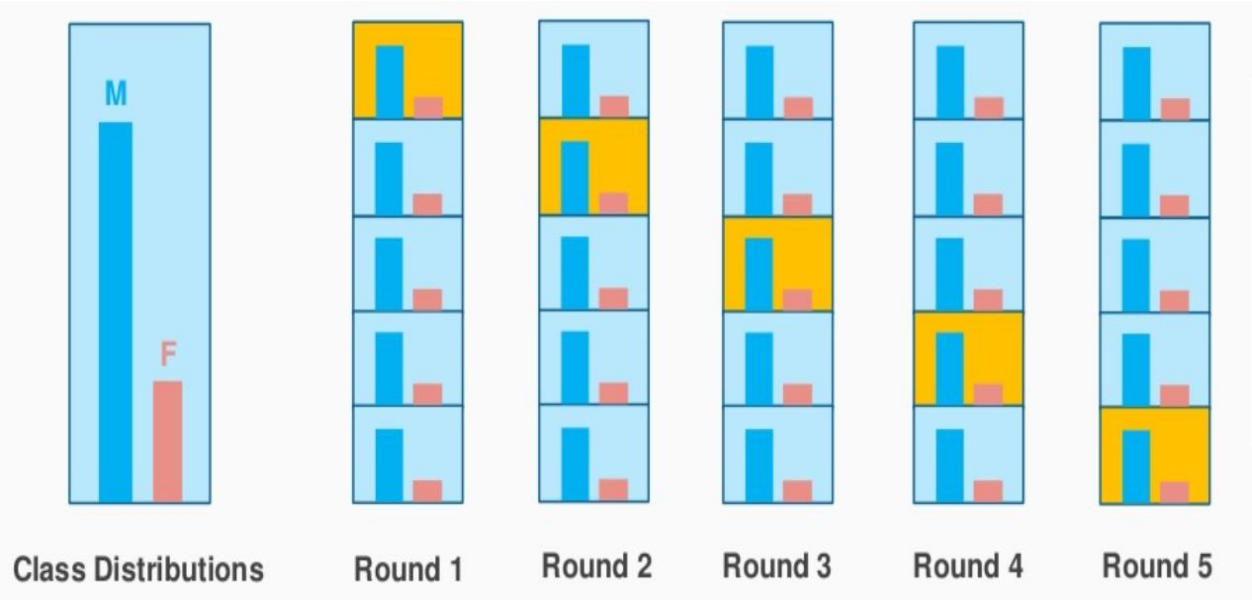
- Với **n mẫu**, có n tập huấn luyện và n tập kiểm tra khác nhau (KFold với k = n) → **chi phí tính toán đắt, phương sai cao**
- Theo nguyên tắc chung, hầu hết các nghiên cứu đều cho rằng nên ưu tiên 5 hoặc 10-CV so với LOO (có thể **đánh giá quá cao lỗi khái quát hóa**)


```
>>> from sklearn.model_selection import LeaveOneOut

>>> X = [1, 2, 3, 4]
>>> loo = LeaveOneOut()
>>> for train, test in loo.split(X):
...     print("%s %s" % (train, test))
[1 2 3] [0]
[0 2 3] [1]
[0 1 3] [2]
[0 1 2] [3]
```

StratifiedKFold

Biến thể của KFold trả về các nếp gấp phân tầng: mỗi bộ chứa khoảng phần trăm mẫu của từng lớp mục tiêu dưới dạng bộ hoàn chỉnh

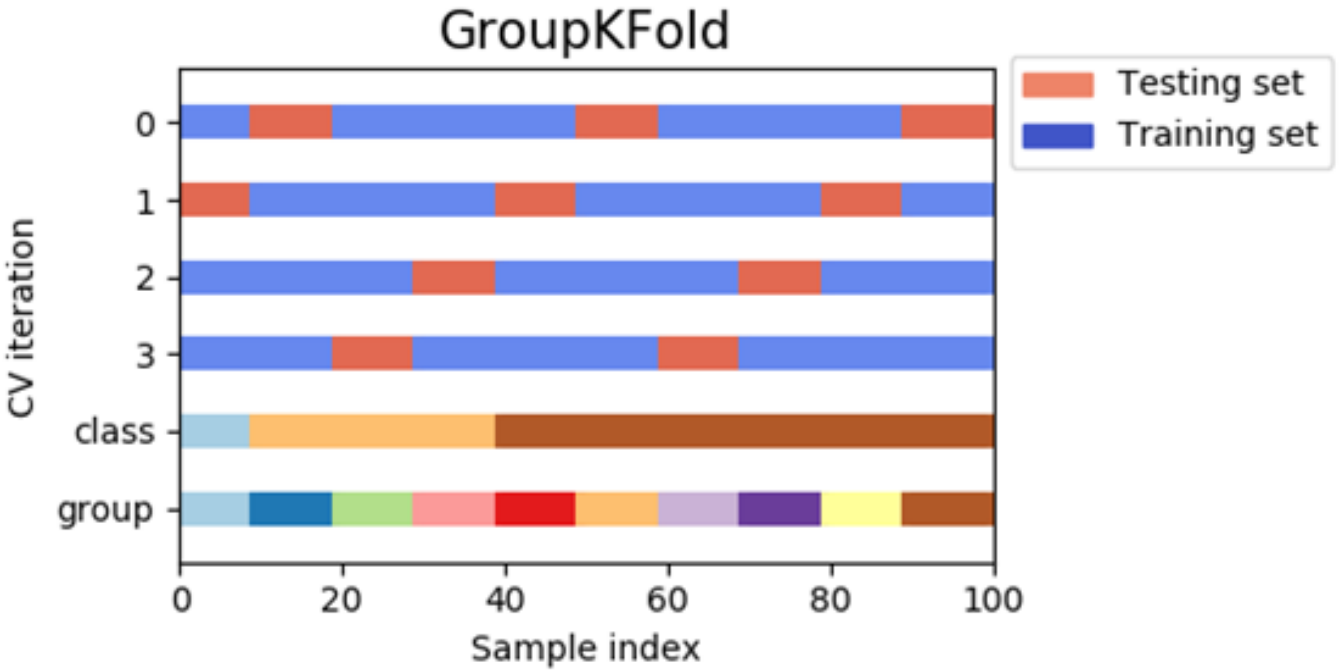



```
kf = StratifiedKFold(n_splits=5,shuffle=True,random_state=42)
for fold, (train_index, val_index) in enumerate(kf.split(X,y)):

    X_train, X_val = X.loc[train_index], X.loc[val_index]
    y_train, y_val = y.loc[train_index], y.loc[val_index]
```

GroupKFold

- Biến thể của KFold đảm bảo rằng cùng một nhóm không có mặt trong cả tập huấn luyện và tập kiểm tra



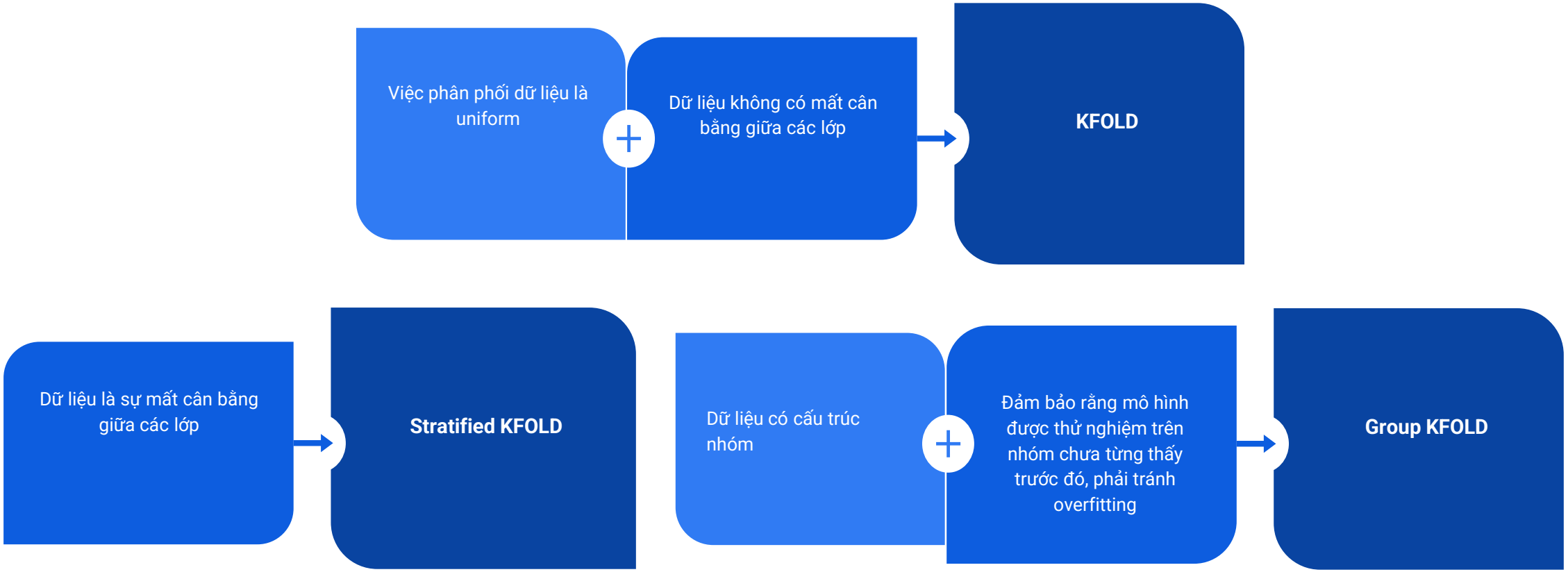

```
>>> from sklearn.model_selection import GroupKFold

>>> X = [0.1, 0.2, 2.2, 2.4, 2.3, 4.55, 5.8, 8.8, 9, 10]
>>> y = ["a", "b", "b", "b", "c", "c", "c", "d", "d", "d"]
>>> groups = [1, 1, 1, 2, 2, 2, 3, 3, 3, 3]

>>> gkf = GroupKFold(n_splits=3)
>>> for train, test in gkf.split(X, y, groups=groups):
...     print("%s %s" % (train, test))
[0 1 2 3 4 5] [6 7 8 9]
[0 1 2 6 7 8 9] [3 4 5]
[3 4 5 6 7 8 9] [0 1 2]
```

Tổng hợp các phương pháp chia CV

Khi nào nên sử dụng các phương pháp?



Sklearn: Splitter Class

sklearn.model_selection

Splitter Classes	
model_selection.GroupKFold([n_splits])	K-fold iterator variant with non-overlapping groups.
model_selection.GroupShuffleSplit([...])	Shuffle-Group(s)-Out cross-validation iterator
model_selection.KFold([n_splits, shuffle, ...])	K-Folds cross-validator
model_selection.LeaveOneGroupOut()	Leave One Group Out cross-validator
model_selection.LeavePGroupsOut(n_groups)	Leave P Group(s) Out cross-validator
model_selection.LeaveOneOut()	Leave-One-Out cross-validator
model_selection.LeavePOut(p)	Leave-P-Out cross-validator
model_selection.PredefinedSplit(test_fold)	Predefined split cross-validator
model_selection.RepeatedKFold(*[, n_splits, ...])	Repeated K-Fold cross validator.
model_selection.RepeatedStratifiedKFold(*[, ...])	Repeated Stratified K-Fold cross validator.
model_selection.ShuffleSplit([n_splits, ...])	Random permutation cross-validator
model_selection.StratifiedKFold([n_splits, ...])	Stratified K-Folds cross-validator.
model_selection.StratifiedShuffleSplit([...])	Stratified ShuffleSplit cross-validator
model_selection.StratifiedGroupKFold([...])	Stratified K-Folds iterator variant with non-overlapping groups.
model_selection.TimeSeriesSplit([n_splits, ...])	Time Series cross-validator

Lựa chọn mô hình

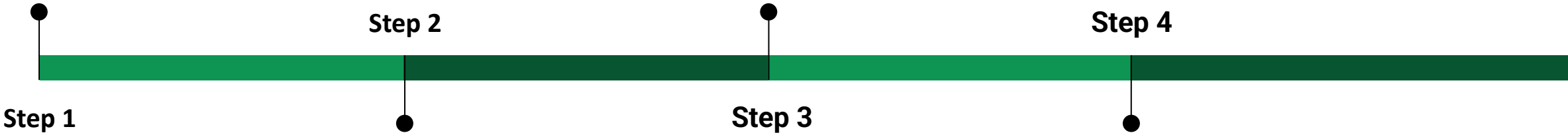
- Lựa chọn mô hình bao gồm việc chọn mô hình dự đoán dự kiến sẽ cung cấp hiệu suất tốt nhất trên dữ liệu trong tương lai.

Xác định tập hợp các mô hình ứng viên cần kiểm tra

Một thực tế phổ biến là xác định một tập hợp các mô hình ngày càng phức tạp

Đánh giá hiệu suất của từng mô hình bằng một phương pháp cross-validation (CV)

CV score cung cấp ước tính về hiệu suất khái quát hóa

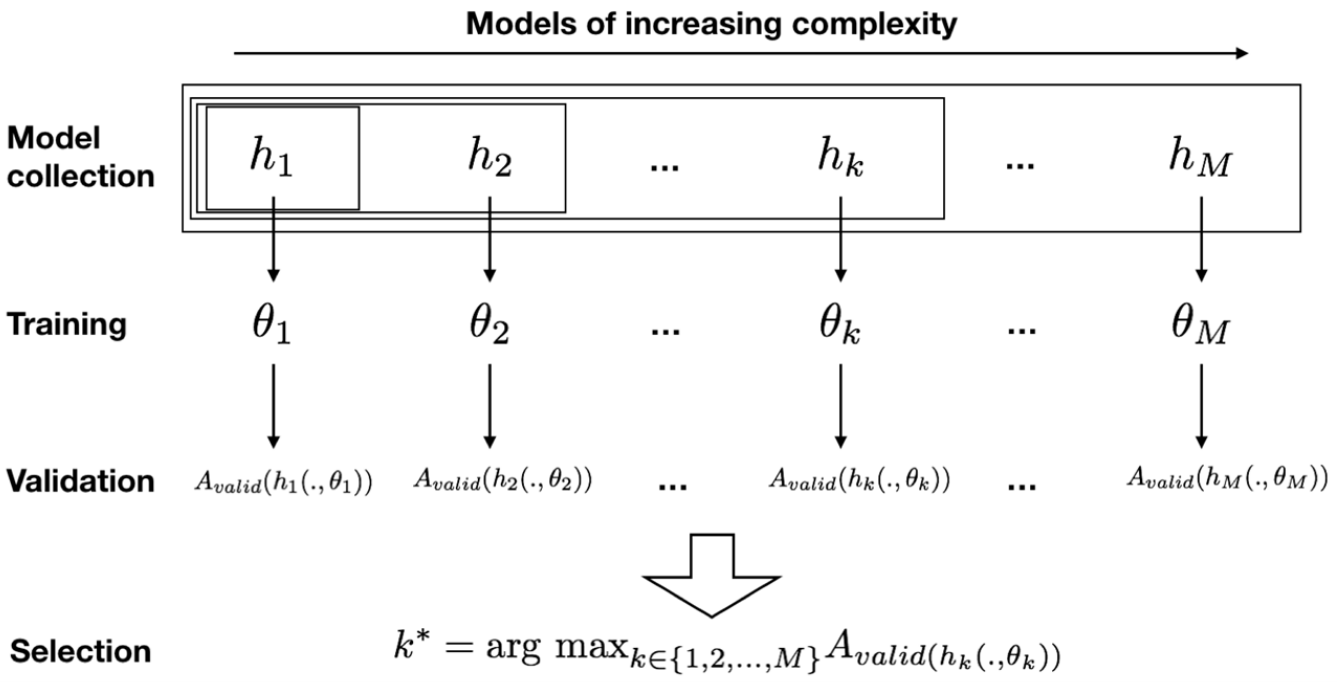


Huấn luyện từng mô hình ứng viên bằng cách sử dụng dữ liệu huấn luyện

Chọn mô hình có hiệu suất xác thực cao nhất (CV score)

Quy trình lựa chọn mô hình

- Phương pháp tiêu chuẩn bao gồm bốn bước chính:



source:https://fraud-detection-handbook.github.io

Hyper-parameters Optimization

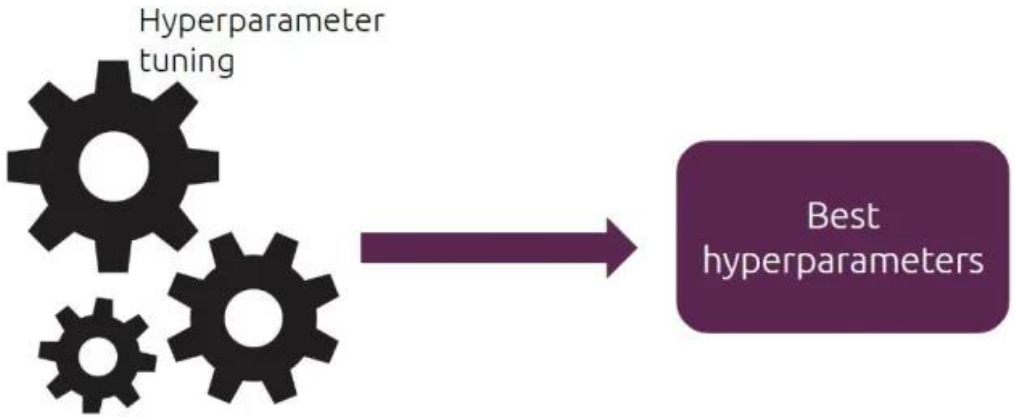
- Siêu tham số là cài đặt hoặc cấu hình của mô hình được đặt trước khi quá trình huấn luyện bắt đầu và không thể học trực tiếp từ dữ liệu.
- Tối ưu hóa siêu tham số là quá trình tìm ra sự kết hợp tốt nhất của các siêu tham số cho một mô hình học máy nhằm tối ưu hóa hiệu suất của nó trên một tác vụ nhất định.

```
Hyperparameter_tuning (training_data, validation_data, hp_list):
  hp_perf = []
  foreach hp_setting in hp_list:
    m = train_model(training_data, hp_setting)
    validation_results = eval_model(m, validation_data)
    hp_perf.append(validation_results)
  best_hp_setting = hp_list[max_index(hp_perf)]
  best_m = train_model(training_data.append(validation_data), best_hp_setting)
  return (best_hp_setting, best_m)
```

pseudo code for Hyper-parameters Optimization

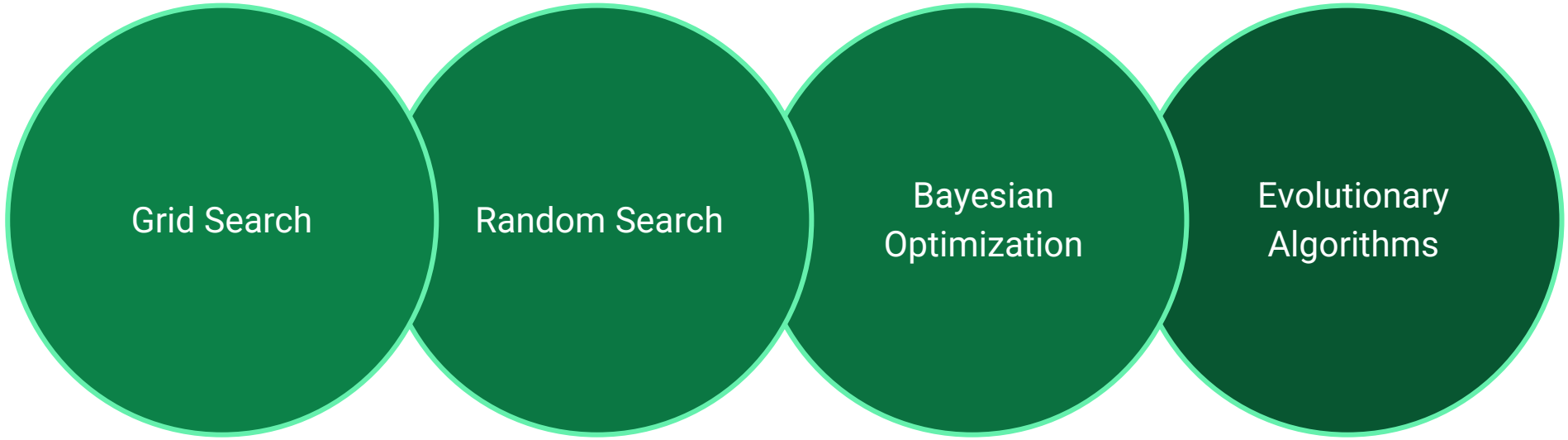
Cách tìm kiếm siêu tham số?

- Tìm kiếm bao gồm:
  - mô hình ước tính/ (hồi quy, phân loại,..)
  - Không gian tham số
  - Phương pháp tìm kiếm hoặc lấy mẫu ứng viên
  - Xác thực chéo
  - Hàm tính điểm



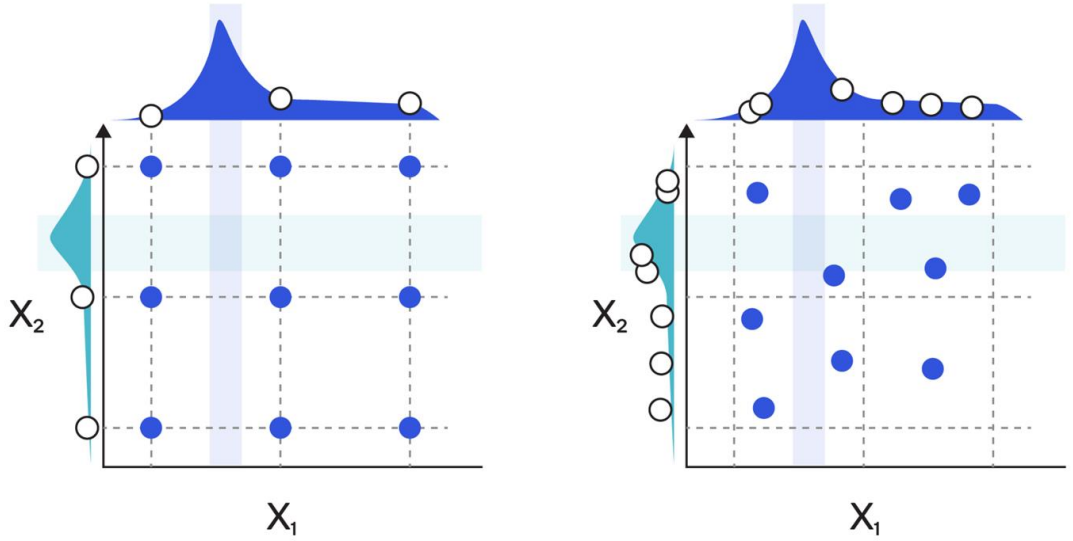
Phương pháp

- Dưới đây là một số phương pháp tối ưu hóa siêu tham số thường được sử dụng trong học máy:



Grid Search & Random Search

- Tìm kiếm lưới: chọn ra một lưới các giá trị siêu tham số, đánh giá từng giá trị trong số chúng và trả về giá trị có hiệu suất cao nhất.
- Tìm kiếm ngẫu nhiên là một biến thể nhỏ trên tìm kiếm lưới. Thay vì tìm kiếm trên toàn bộ lưới, tìm kiếm ngẫu nhiên chỉ đánh giá một mẫu điểm ngẫu nhiên trên lưới



(a) Standard Grid Search (b) Random Search

Grid Search: Ví dụ

- Default hyperparams & hyperparams optimize with GridSearchCV

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	wors smoothnes
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.162
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.123
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.144
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.209
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.137

5 rows × 30 columns

```python
#import all necessary libraries
import sklearn
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.datasets import load_breast_cancer
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split

#load the dataset and split it into training and testing sets
dataset = load_breast_cancer()
X=dataset.data
Y=dataset.target
X_train, X_test, y_train, y_test = train_test_split(
                        X,Y,test_size = 0.30, random_state = 101)
# train the model on train set without using GridSearchCV
model = SVC()
model.fit(X_train, y_train)

# print prediction results
predictions = model.predict(X_test)
print(classification_report(y_test, predictions))
```

```
OUTPUT:
 precision    recall  f1-score   support

           0       0.95      0.85      0.90        66
           1       0.91      0.97      0.94       105

    accuracy                           0.92       171
   macro avg       0.93      0.91      0.92       171
weighted avg       0.93      0.92      0.92       171
```

```python
# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'gamma':['scale', 'auto'],
              'kernel': ['linear']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3,n_jobs=-1)

# fitting the model for grid search
grid.fit(X_train, y_train)

# print best parameter after tuning
print(grid.best_params_)
grid_predictions = grid.predict(X_test)

# print classification report
print(classification_report(y_test, grid_predictions))
```

```
Output:
 {'C': 100, 'gamma': 'scale', 'kernel': 'linear'}
              precision    recall  f1-score   support

           0       0.97      0.91      0.94        66
           1       0.94      0.98      0.96       105

    accuracy                           0.95       171
   macro avg       0.96      0.95      0.95       171
weighted avg       0.95      0.95      0.95       171
```

Bayesian Optimization

- Tools
  - Bayesian Optimization: github
  - Hyperopt: github
  - Optuna: github
- Installation

• PyPI (pip):

```
$ pip install bayesian-optimization
```

• Conda from conda-forge channel:

```
$ conda install -c conda-forge bayesian-optimization
```

Install hyperopt from PyPI

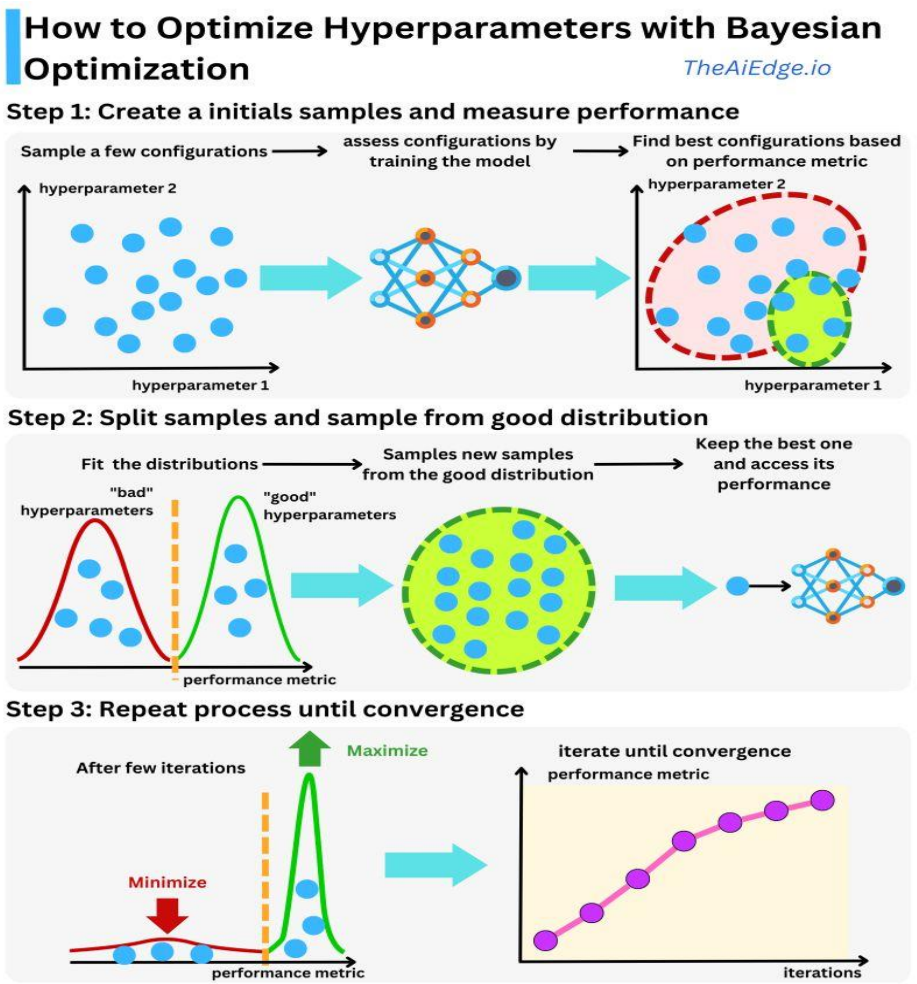
```
pip install hyperopt
```

Optuna is available at the Python Package Index and on Anaconda Cloud.

```
# PyPI
$ pip install optuna
```

```
# Anaconda Cloud
$ conda install -c conda-forge optuna
```

Optuna supports Python 3.7 or newer.



https://newsletter.theaiedge.io

Ví dụ: Bước 1

- Các thông số cần điều chỉnh
- Lưu ý: các giá trị cho siêu tham số phải có ý nghĩa:
  - ‘num_leaves’ & ‘max_depth' cần phải là một số nguyên
  - ‘feature_fraction' & ‘bagging_fraction' nên từ 0 đến 1

```
def lgb_eval(num_leaves, feature_fraction, bagging_fraction, max_depth, lambda_l1, lambda_l2, min_split_gain, min_child_weight):
    params = {'application':'binary','num_iterations':4000, 'learning_rate':0.05, 'early_stopping_round':100, 'metric':'auc'}
    params["num_leaves"] = round(num_leaves)
    params['feature_fraction'] = max(min(feature_fraction, 1), 0)
    params['bagging_fraction'] = max(min(bagging_fraction, 1), 0)
    params['max_depth'] = round(max_depth)
    params['lambda_l1'] = max(lambda_l1, 0)
    params['lambda_l2'] = max(lambda_l2, 0)
    params['min_split_gain'] = min_split_gain
    params['min_child_weight'] = min_child_weight
    cv_result = lgb.cv(params, train_data, nfold=n_folds, seed=random_seed, stratified=True, verbose_eval =200, metrics=['auc'])
    return max(cv_result['auc-mean'])
```

Ví dụ: Bước 2

- Đặt phạm vi cho mỗi siêu tham số
- Cố gắng làm cho phạm vi càng hẹp càng tốt

```
lgbBO = BayesianOptimization(lgb_eval, {'num_leaves': (24, 45),
                                        'feature_fraction': (0.1, 0.9),
                                        'bagging_fraction': (0.8, 1),
                                        'max_depth': (5, 8.99),
                                        'lambda_l1': (0, 5),
                                        'lambda_l2': (0, 3),
                                        'min_split_gain': (0.001, 0.1),
                                        'min_child_weight': (5, 50)}, random_state=0)
```

Ví dụ: Bước 3

- Có rất nhiều thông số bạn có thể dùng để tối đa hóa, tuy nhiên, những thông số quan trọng nhất là:
  - n_iter: Bạn muốn thực hiện bao nhiêu bước tối ưu hóa bayesian. Càng nhiều bước, bạn càng có nhiều khả năng tìm thấy mức tối đa tốt.
  - init_points: Bạn muốn thực hiện bao nhiêu bước khám phá ngẫu nhiên. Khám phá ngẫu nhiên có thể giúp bằng cách đa dạng hóa không gian thăm dò.

```
    # optimize
    lgbBO.maximize(init_points=init_round, n_iter=opt_round)

    # output optimization process
    if output_process==True: lgbBO.points_to_csv("bayes_opt_result.csv")

    # return best parameters
    return lgbBO.res['max']['max_params']

opt_params = bayes_parameter_opt_lgb(X, y, init_round=5, opt_round=10, n_folds=3, random_seed=6, n_estimators=100, learning_rate=0.0
5)
```

Ví dụ: Kết quả

- Result

```
Initialization
-------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------
 Step |   Time |      Value |   bagging_fraction |   feature_fraction |   lambda_l1 |   lambda_l2 |   max_depth |   min_child_wei
ght |   min_split_gain |   num_leaves |
    1 | 00m59s |    0.74761 |             0.9583 |             0.6167 |      4.8931 |      1.9198 |      5.3476 |            32.7
936 |           0.0272 |      35.5251 |
    2 | 00m59s |    0.74885 |             0.9058 |             0.4501 |      3.9958 |      0.4301 |      5.0807 |            32.5
443 |           0.0776 |      39.0190 |
    3 | 01m09s |    0.74990 |             0.9136 |             0.8134 |      2.3074 |      2.8340 |      8.3222 |            32.7
620 |           0.0462 |      36.6580 |
    4 | 00m51s |    0.75006 |             0.9851 |             0.8709 |      3.9026 |      1.5655 |      8.1048 |            47.4
687 |           0.0573 |      35.4425 |
    5 | 00m35s |    0.75089 |             0.8142 |             0.4068 |      0.5914 |      1.2440 |      8.4713 |            35.6
819 |           0.0029 |      32.8968 |
Bayesian Optimization
-------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------
 Step |   Time |      Value |   bagging_fraction |   feature_fraction |   lambda_l1 |   lambda_l2 |   max_depth |   min_child_wei
ght |   min_split_gain |   num_leaves |
    6 | 00m45s |    0.75122 |             0.8824 |             0.2785 |      0.0636 |      2.2423 |      5.8154 |            49.1
417 |           0.0100 |      44.1038 |
    7 | 00m58s |    0.75131 |             0.9785 |             0.8560 |      0.2812 |      0.0292 |      8.9339 |            49.9
035 |           0.0821 |      43.8205 |
    8 | 00m55s |    0.75166 |             0.9212 |             0.2732 |      0.2660 |      1.9191 |      8.8121 |             5.3
852 |           0.0012 |      44.5129 |
    9 | 00m47s |    0.73409 |             0.8018 |             0.1068 |      4.9587 |      2.9412 |      8.6253 |            39.9
238 |           0.0773 |      44.9553 |
   10 | 00m46s |    0.74762 |             0.8871 |             0.7749 |      0.3189 |      0.4824 |      5.3296 |             5.3
105 |           0.0562 |      25.0965 |
   11 | 00m45s |    0.74755 |             0.9459 |             0.8495 |      0.8824 |      2.9314 |      5.1176 |            49.9
633 |           0.0341 |      24.0223 |
```

Phương pháp so sánh: Ví dụ

- Ví dụ để so sánh: Grid Search, Random Search & HyperOpt

```
# Generate dataset with 1000 samples, 100 features and 2 classes

def gen_dataset(n_samples=1000, n_features=100, n_classes=2, random_state=123):
    X, y = datasets.make_classification(
        n_features=n_features,
        n_samples=n_samples,
        n_informative=int(0.6 * n_features),    # the number of informative features
        n_redundant=int(0.1 * n_features),      # the number of redundant features
        n_classes=n_classes,
        random_state=random_state)
    return (X, y)

X, y = gen_dataset(n_samples=1000, n_features=100, n_classes=2)

# Train / test split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
if optimizer == 'grid_search':
    grid_obj = GridSearchCV(estimator=pipeline,
                            param_grid=parameters,
                            cv=5,
                            refit=True,
                            return_train_score=False,
                            scoring = 'accuracy',
                           )
    grid_obj.fit(X_train, y_train,)

elif optimizer == 'random_search':
    grid_obj = RandomizedSearchCV(estimator=pipeline,
                          param_distributions=parameters
                          cv=5,
                          n_iter=n_iter,
                          refit=True,
                          return_train_score=False,
                          scoring = 'accuracy',
                          random_state=1)
    grid_obj.fit(X_train, y_train,)
```

```
param_gridsearch = {
    'clf__learning_rate' : [0.01, 0.1, 1],
    'clf__max_depth' : [5, 10, 15],
    'clf__n_estimators' : [5, 20, 35],
    'clf__num_leaves' : [5, 25, 50],
    'clf__boosting_type': ['gbdt', 'dart'],
    'clf__colsample_bytree' : [0.6, 0.75, 1],
    'clf__reg_lambda': [0.01, 0.1, 1],
}

param_random = {
    'clf__learning_rate': list(np.logspace(np.log(0.01), np.log(1), num = 500, base=3)),
    'clf__max_depth': list(range(5, 15)),
    'clf__n_estimators': list(range(5, 35)),
    'clf__num_leaves': list(range(5, 50)),
    'clf__boosting_type': ['gbdt', 'dart'],
    'clf__colsample_bytree': list(np.linspace(0.6, 1, 500)),
    'clf__reg_lambda': list(np.linspace(0, 1, 500)),
 param_hyperopt= {
     'learning_rate': hp.loguniform('learning_rate', np.log(0.01), np.log(1)),
     'max_depth': scope.int(hp.quniform('max_depth', 5, 15, 1)),
     'n_estimators': scope.int(hp.quniform('n_estimators', 5, 35, 1)),
     'num_leaves': scope.int(hp.quniform('num_leaves', 5, 50, 1)),
     'boosting_type': hp.choice('boosting_type', ['gbdt', 'dart']),
     'colsample_bytree': hp.uniform('colsample_by_tree', 0.6, 1.0),
     'reg_lambda': hp.uniform('reg_lambda', 0.0, 1.0),
 }
```

Phương pháp so sánh: Kết quả

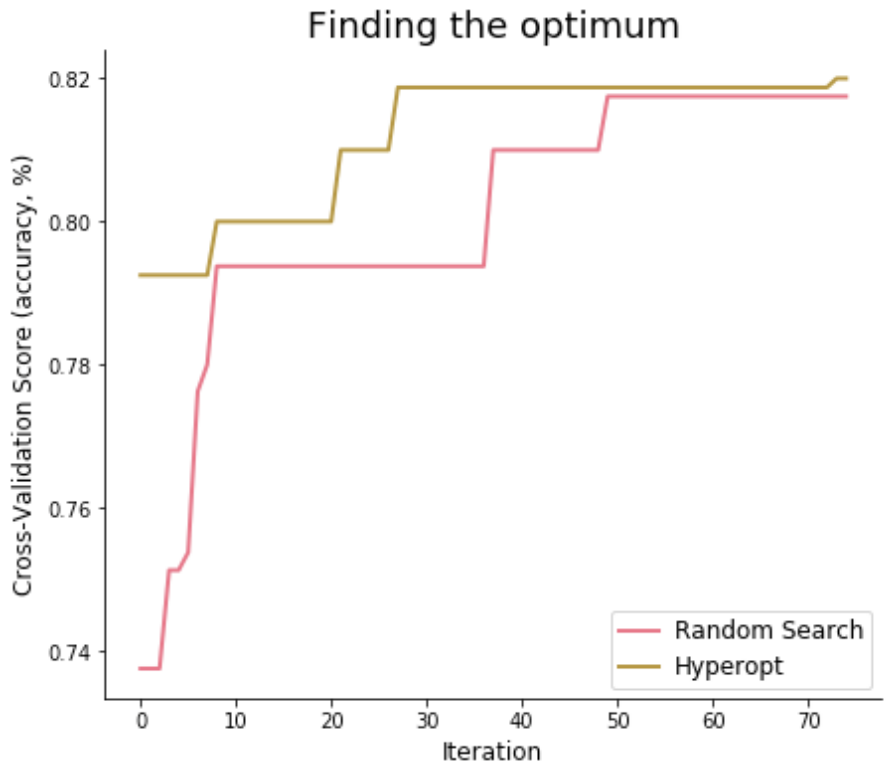
- Ví dụ để so sánh: Grid Search, Random Search & HyperOpt

Technique	Parameters Evaluated	# of iterations to get to optimum	Overall time taken (sec)*	Cross-Validation Score	Test score
Grid Search	1451	942	237	0.81	0.79
Random Search	75	48	11	0.82	0.8
Hyperopt	75	23	15	0.82	0.84

In case of Grid Search: time taken to iterate over all parameter combinations

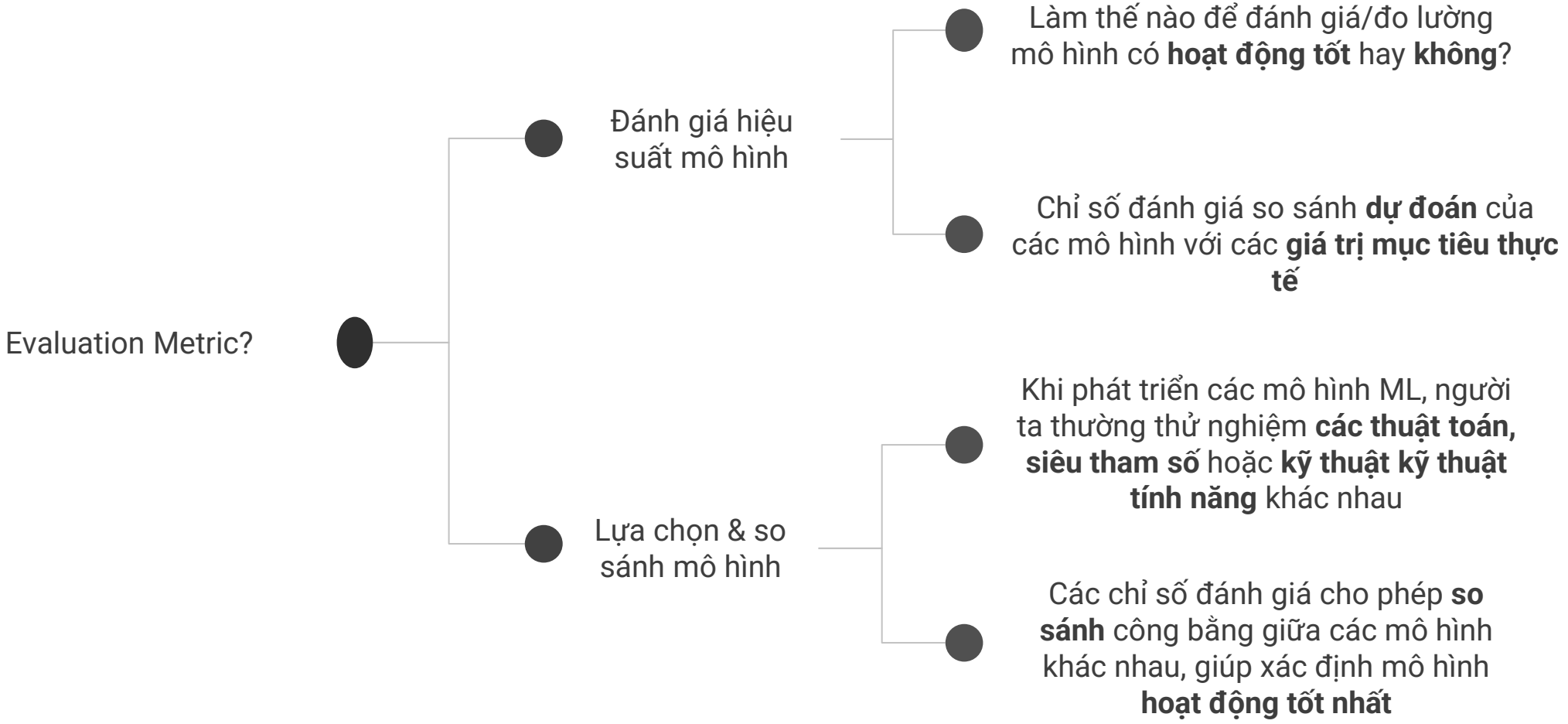
In case of Random Search / Hyperopt: time taken to go over the predefined number of iterations (75)

https://www.vantage-ai.com

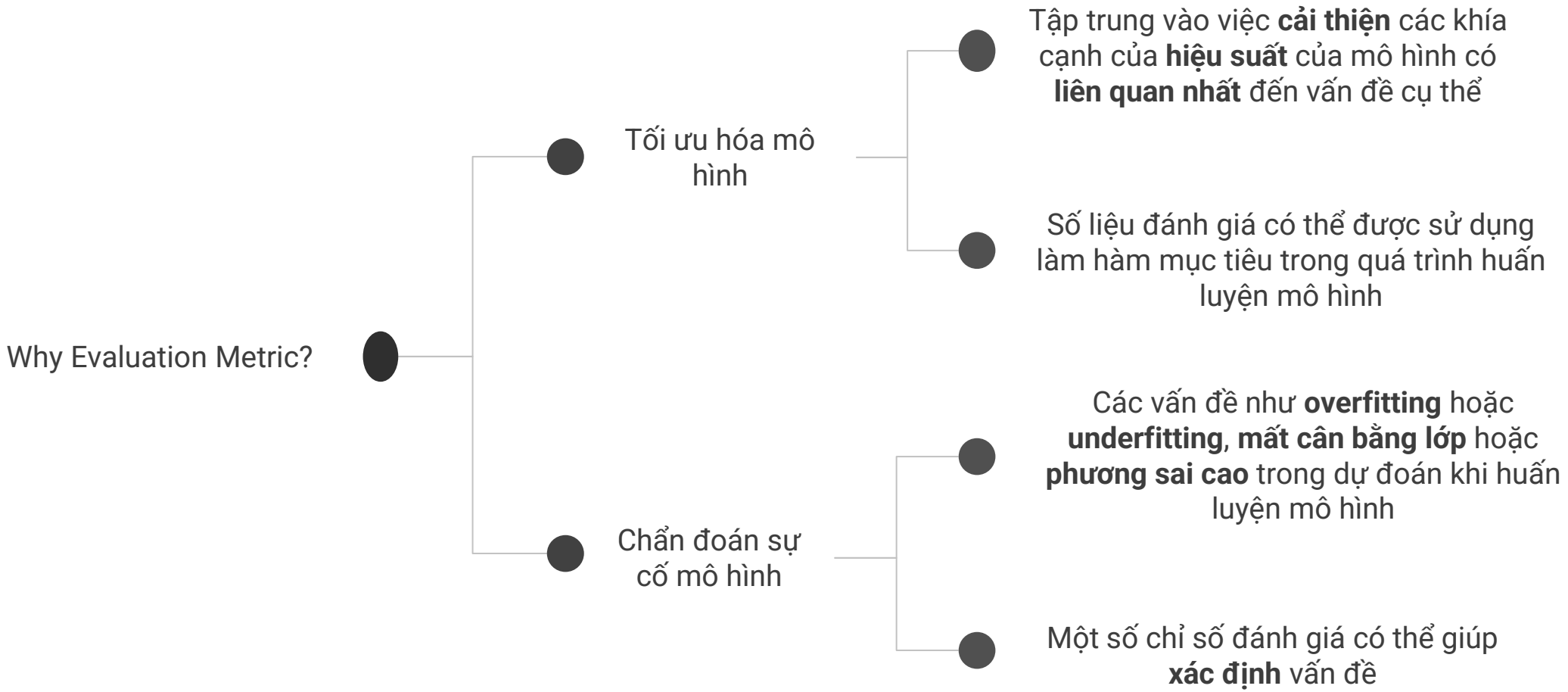


https://www.vantage-ai.com

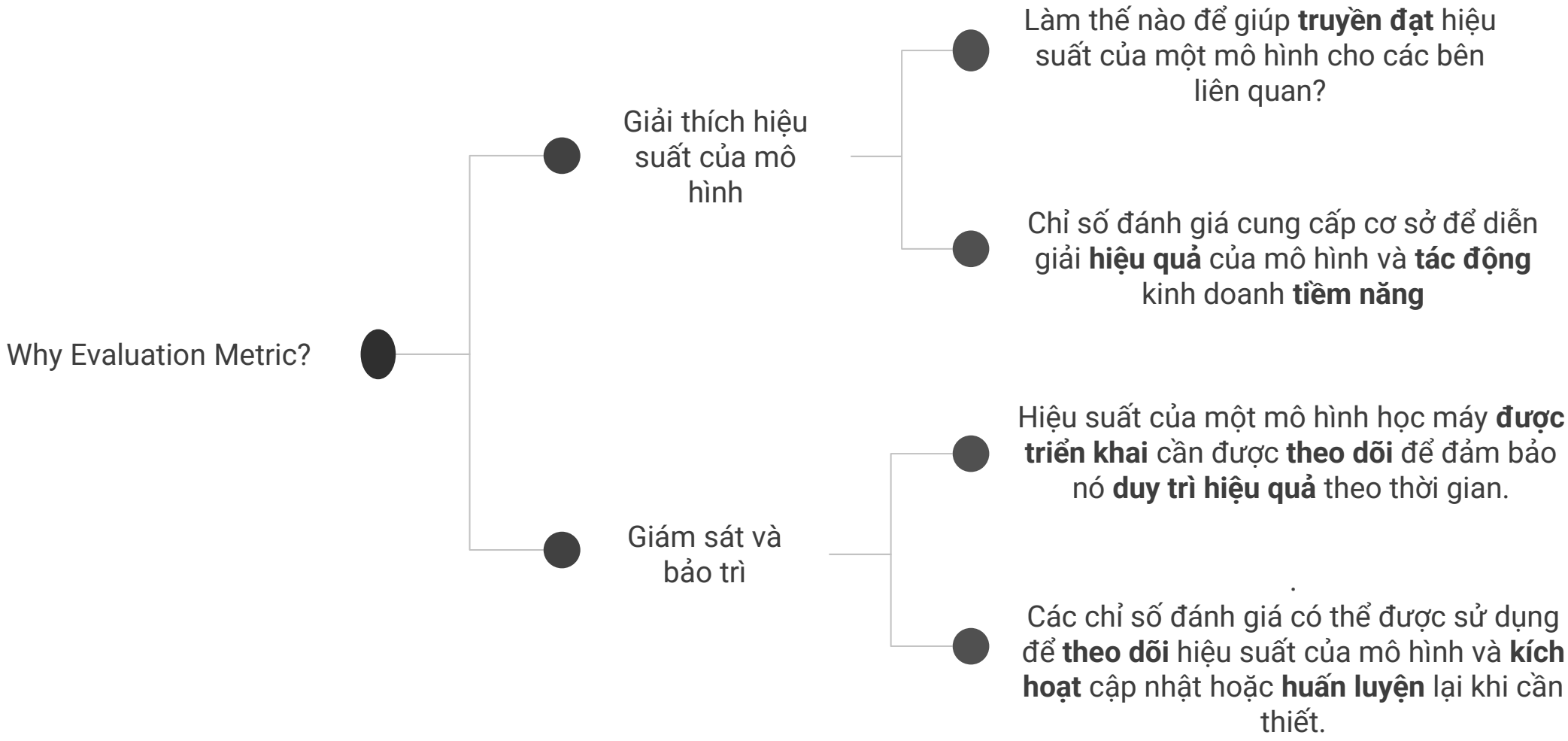
Tại sao lại là số liệu đánh giá?



Tại sao Evaluation Metric?

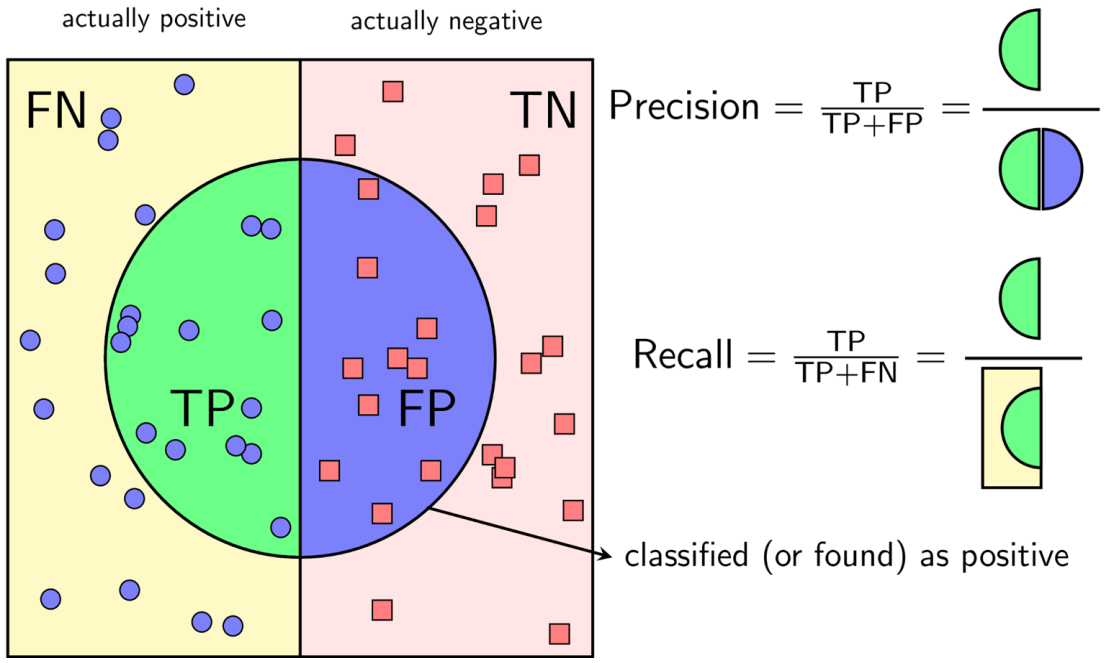


Tại sao Evaluation Metric?



Chỉ số đánh giá phân loại

- Một số chỉ số đánh giá phân loại



$$\text{Precision} = \frac{TP}{TP+FP} =$$

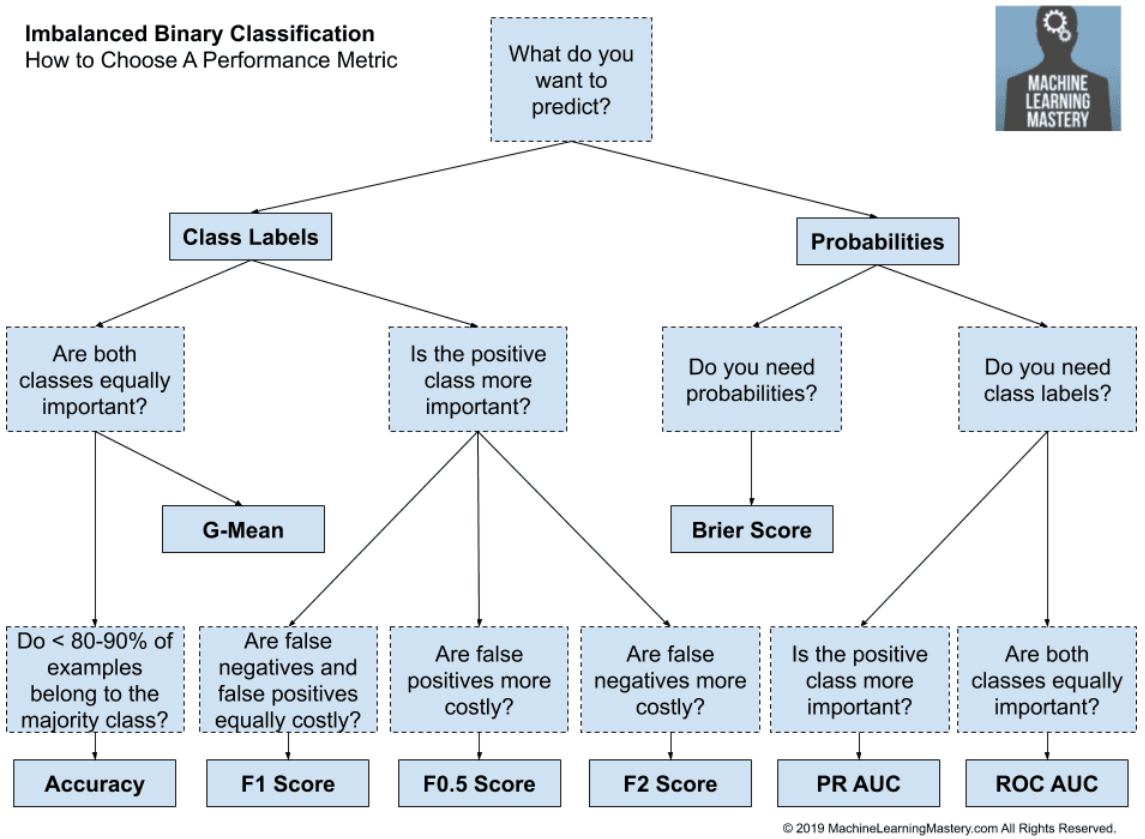
$$\text{Recall} = \frac{TP}{TP+FN} =$$

Metric	Formula	Evaluation Focus
Accuracy	$ACC = \frac{TP+TN}{TP+TN+FP+FN}$	Overall effectiveness of a classifier
Error rate	$ERR = \frac{FP+FN}{TP+TN+FP+FN}$	Classification error
Precision	$PRC = \frac{TP}{TP+FP}$	Class agreement of the data labels with the positive labels given by the classifier
Sensitivity	$SNS = \frac{TP}{TP+FN}$	Effectiveness of a classifier to identify positive labels
Specificity	$SPC = \frac{TN}{TN+FP}$	How effectively a classifier identifies negative labels
ROC	$ROC = \frac{\sqrt{SNS^2+SPC^2}}{\sqrt{2}}$	Combined metric based on the Receiver Operating Characteristic (ROC) space [53]
F_1 score	$F_1 = 2\frac{PRC \cdot SNS}{PRC+SNS}$	Combination of precision (PRC) and sensitivity (SNS) in a single metric
Geometric Mean	$GM = \sqrt{SNS \cdot SPC}$	Combination of sensitivity (SNS) and specificity (SPC) in a single metric

https://machinelearningcoban.com/2017/08/31/evaluation/

Số liệu đánh giá cho Imbalanced Classification

- Imbalanced Classification: Chọn chỉ số phù hợp với trường hợp



https://machinelearningmastery.com/tour-of-evaluation-metrics-for-imbalanced-classification/

Chỉ số cho bài toán hồi quy

- Một số chỉ số hồi quy:

Example Use Case	Error Metric	Formula	Notes
Which model best captures the rapid changes in the volatile stock market?	**1.1** Mean squared error (MSE)	$\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$	Weights big differences more
	1.2 Root Mean squared error (RMSE)	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2}$	
Which model best estimates the energy consumption in the long term?	**2.** Mean absolute error (MAE)	$\frac{1}{n}\sum_{i=1}^{n}\lvert y_i - f(x_i)\rvert$	Equal weights to all distances
Are the sales forecasting models for different products equally accurate?	**3.** Mean absolute percentage error (MAPE)	$\frac{1}{n}\sum_{i=1}^{n}\frac{\lvert y_i - f(x_i)\rvert}{\lvert y_i\rvert}$	Requires non-zero target column values
Does a running app provide unrealistic expectations?	**4.** Mean signed difference	$\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))$	Only informative about the direction of the error
How much of our years of education can be explained through access to literature?	**5.** R-squared	$1 - \frac{\sum_{i=1}^{n}(y_i - f(x_i))^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$	Universal range: the closer to 1 the better

Sklearn: Metrics

- sklearn.metrics

Scoring	Function
Classification	
'accuracy'	metrics.accuracy_score
'balanced_accuracy'	metrics.balanced_accuracy_score
'top_k_accuracy'	metrics.top_k_accuracy_score
'average_precision'	metrics.average_precision_score
'neg_brier_score'	metrics.brier_score_loss
'f1'	metrics.f1_score
'f1_micro'	metrics.f1_score
'f1_macro'	metrics.f1_score
'f1_weighted'	metrics.f1_score
'f1_samples'	metrics.f1_score
'neg_log_loss'	metrics.log_loss
'precision' etc.	metrics.precision_score
'recall' etc.	metrics.recall_score
'jaccard' etc.	metrics.jaccard_score
'roc_auc'	metrics.roc_auc_score
'roc_auc_ovr'	metrics.roc_auc_score
'roc_auc_ovo'	metrics.roc_auc_score
'roc_auc_ovr_weighted'	metrics.roc_auc_score
'roc_auc_ovo_weighted'	metrics.roc_auc_score

Regression	
'explained_variance'	metrics.explained_variance_score
'max_error'	metrics.max_error
'neg_mean_absolute_error'	metrics.mean_absolute_error
'neg_mean_squared_error'	metrics.mean_squared_error
'neg_root_mean_squared_error'	metrics.mean_squared_error
'neg_mean_squared_log_error'	metrics.mean_squared_log_error
'neg_median_absolute_error'	metrics.median_absolute_error
'r2'	metrics.r2_score
'neg_mean_poisson_deviance'	metrics.mean_poisson_deviance
'neg_mean_gamma_deviance'	metrics.mean_gamma_deviance
'neg_mean_absolute_percentage_error'	metrics.mean_absolute_percentage_error
'd2_absolute_error_score'	metrics.d2_absolute_error_score
'd2_pinball_score'	metrics.d2_pinball_score
'd2_tweedie_score'	metrics.d2_tweedie_score

Tài liệu tham khảo

- Pattern Recognition and Machine Learning
- https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/

HỎI ĐÁP