

▼ XỬ LÝ SỐ LIỆU THỐNG KÊ - BÁO CÁO CUỐI HỌC KÌ

Danh sách nhóm 4:

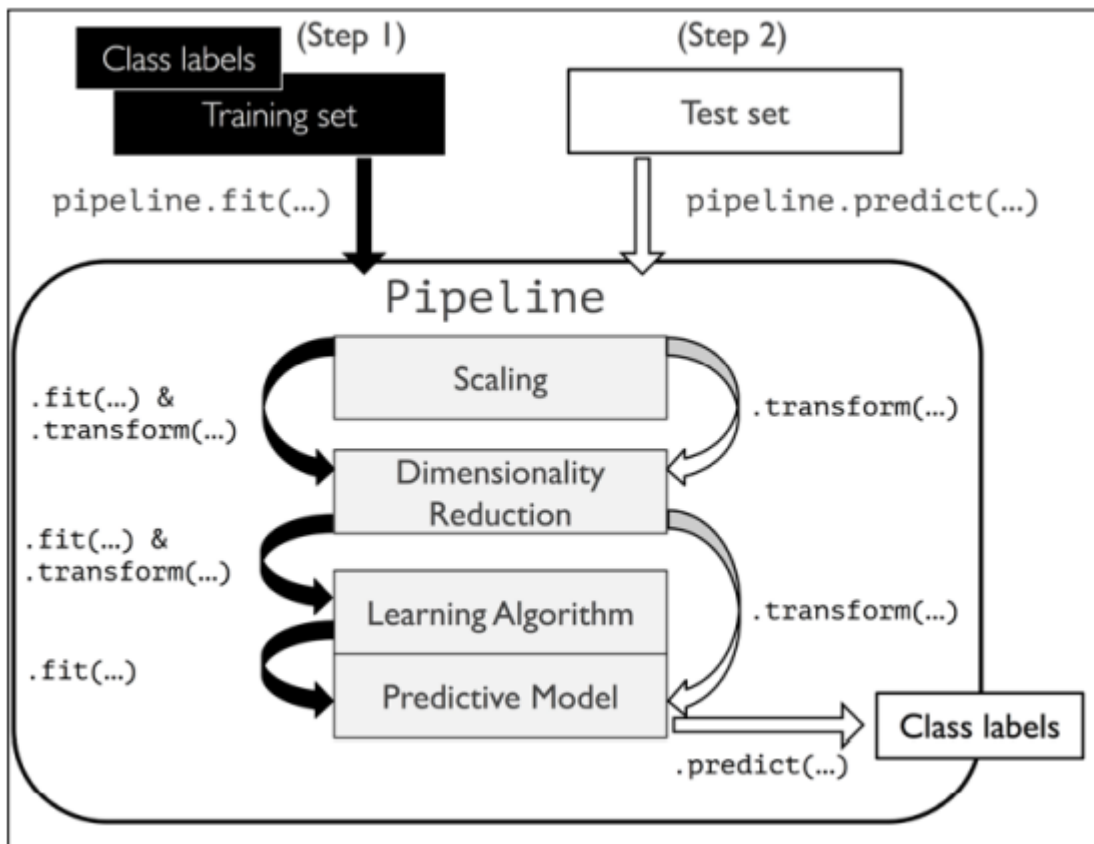
Họ và tên	Mã số SV	Phần đảm nhận
Nguyễn Lê Công Duy	2011016	6.1. Streamlining workflows with pipelines 6.2. Using k-fold cross-validation to assess model performance
Trần Thị Kỳ Phương	20110287	6.3. Debugging algorithms with learning and validation curves 6.4. Fine-tuning machine learning models via grid search
Bùi Cao Kim Long	20110229	6.5. Looking at different performance evaluation metrics 6.6. Dealing with class imbalance
Hà Thành Long	20280061	7.1. Learning with ensembles
Nguyễn Mạnh Khiêm	19110093	7.2. Combining classifiers via majority vote
Đinh Anh Tú	19110497	7.3. Bagging – building an ensemble of classifiers from bootstrap samples
Võ Đức Trọng	19110494	7.4 Leveraging weak learners via adaptive boosting

6. Learning Best Practices for Model Evaluation and Hyperparameter Tuning

Tìm hiểu các phương pháp hay nhất để đánh giá mô hình và điều chỉnh các siêu tham số

▼ 6.1. Pipeline

Phần này, chúng ta sẽ tìm hiểu về một công cụ cực kỳ tiện dụng, class Pipeline trong scikit_learn. Nó cho phép chúng ta điều chỉnh một mô hình bao gồm một số phép biến đổi tùy ý các bước và áp dụng nó để đưa ra dự đoán về dữ liệu mới.



```

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import LabelEncoder
import pandas as pd
import numpy as np

```

```

df = pd.read_csv('https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master

```

```

df.head()

```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothr
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	

```
# load Breast Cancer Wisconsin dataset
```

```
import pandas as pd
```

```
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.model_selection import train_test_split
```

```
df = pd.read_csv('https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master
```

```
X = df.loc[:, 2:].values
```

```
y = df.loc[:, 1].values
```

```
le = LabelEncoder()
```

```
y = le.fit_transform(y)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=1)
```

```
print(le.transform(['M', 'B']))
```

```
[1 0]
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.20, stratify=y,random_sta
```

```
from os import pipe
```

```
pipe_lr = make_pipeline(StandardScaler(), PCA(n_components = 2), LogisticRegression(random_st
```

```
pipe_lr.fit(X_train, y_train)
```

```
y_pred = pipe_lr.predict(X_test)
```

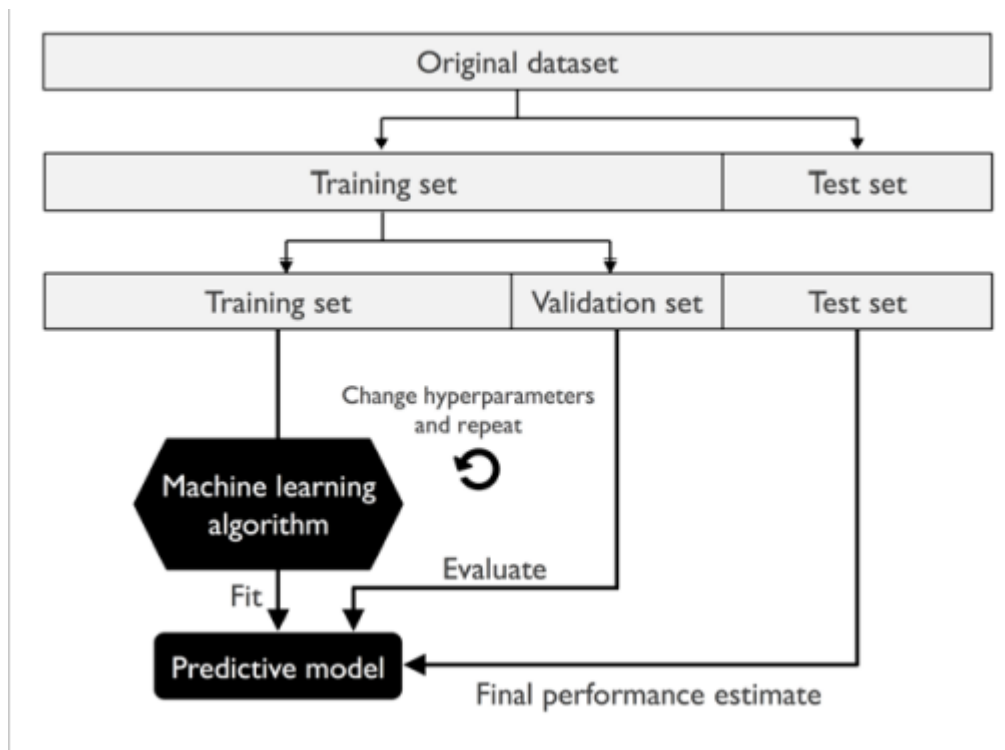
```
print('Test Acurracy: %.3f' % pipe_lr.score(X_test,y_test))
```

```
Test Acurracy: 0.956
```

▼ The holdout method

- Chia tập dữ liệu ban đầu của mình thành hai tập dữ liệu: test_dataset và training_dataset. Tập dữ liệu đầu được sử dụng để training mô hình và tập dữ liệu sau được sử dụng để ước tính hiệu suất tổng quát hóa của nó.

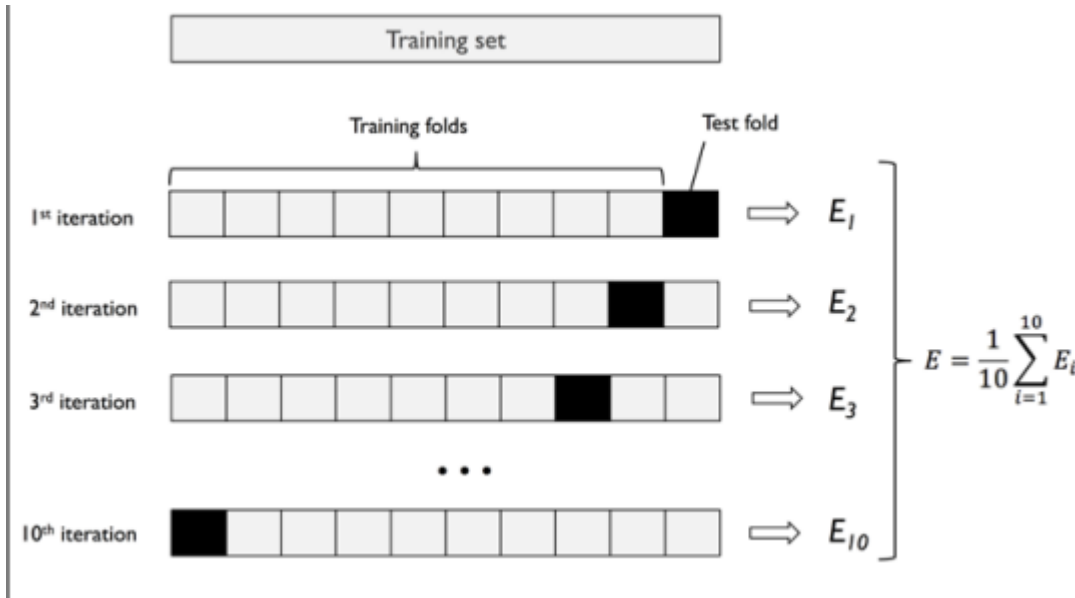
- Tuy nhiên, nếu chúng ta sử dụng đi sử dụng lại cùng một tập dữ liệu thử nghiệm trong quá trình lựa chọn mô hình, nó sẽ trở thành một phần dữ liệu huấn luyện của chúng ta và do đó, mô hình sẽ có nhiều khả năng bị overfit hơn. Bất chấp vấn đề này, nhiều người vẫn sử dụng bộ kiểm tra cho mô hình lựa chọn, đây không phải là một phương pháp tốt.
- Vì vậy, chúng ta tách dữ liệu thành ba phần: training set, validation set và test set
- Tập huấn luyện được sử dụng để phù hợp với các mô hình khác nhau và hiệu suất trên bộ xác thực sau đó được sử dụng để lựa chọn mô hình.
- Ưu điểm của việc có một bộ kiểm tra mà mô hình chưa từng thấy trước đây trong các bước đào tạo và lựa chọn mô hình là chúng ta có thể có được ước tính ít sai lệch hơn về khả năng khái quát hóa dữ liệu mới của nó



▼ 6.2. K-fold cross-validation

- Trong K-fold cross-validation, chúng ta chia ngẫu nhiên training_dataset thành k tập con nhỏ, trong đó $k - 1$ tập con được sử dụng để training model và một nếp gấp được sử dụng để đánh giá hiệu suất. Quy trình này được lặp lại k lần để chúng tôi có được k model và ước tính hiệu suất.
- Sau đó, tính toán hiệu suất trung bình của các mô hình dựa trên các tập dữ liệu con khác nhau để có được ước tính hiệu suất ít nhạy cảm hơn với phân vùng phụ của dữ liệu huấn luyện so với phương pháp holdout.

- Thông thường, chúng tôi sử dụng xác thực chéo k-fold để điều chỉnh mô hình, nghĩa là tìm các giá trị siêu tham số tối ưu mang lại hiệu suất tổng quát hóa thỏa mãn. Khi chúng tôi đã tìm thấy các giá trị siêu tham số thỏa đáng, chúng tôi có thể đào tạo lại mô hình trên tập huấn luyện hoàn chỉnh và có được ước tính hiệu suất cuối cùng bằng cách sử dụng tập kiểm tra độc lập.



```
import numpy as np
from sklearn.model_selection import StratifiedKFold

# Lựa chọn số k = 10
kfold = StratifiedKFold(n_splits=10,
                        random_state = None).split(X_train, y_train)

scores = []
for k, (train, test) in enumerate(kfold):
    pipe_lr.fit(X_train[train], y_train[train])
    score = pipe_lr.score(X_train[test], y_train[test])
    scores.append(score)
    print('Fold: %2d, Class dist.: %s, Acc: %.3f'
          % (k+1, np.bincount(y_train[train]), score))

Fold:  1, Class dist.: [256 153], Acc: 0.935
Fold:  2, Class dist.: [256 153], Acc: 0.935
Fold:  3, Class dist.: [256 153], Acc: 0.957
Fold:  4, Class dist.: [256 153], Acc: 0.957
Fold:  5, Class dist.: [256 153], Acc: 0.935
Fold:  6, Class dist.: [257 153], Acc: 0.956
Fold:  7, Class dist.: [257 153], Acc: 0.978
Fold:  8, Class dist.: [257 153], Acc: 0.933
Fold:  9, Class dist.: [257 153], Acc: 0.956
Fold: 10, Class dist.: [257 153], Acc: 0.956
```

```
print('\nCV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

```
CV accuracy: 0.950 +/- 0.014
```

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(estimator=pipe_lr,X=X_train,y=y_train,cv=10,n_jobs=1)
print('CV accuracy scores: %s' % scores)
```

```
CV accuracy scores: [0.93478261 0.93478261 0.95652174 0.95652174 0.93478261 0.95555556
0.97777778 0.93333333 0.95555556 0.95555556]
```

```
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))
```

```
CV accuracy: 0.950 +/- 0.014
```

6.3. Debugging algorithms with learning and validation curves

Chỉnh sửa thuật toán dựa trên learning và validation curves

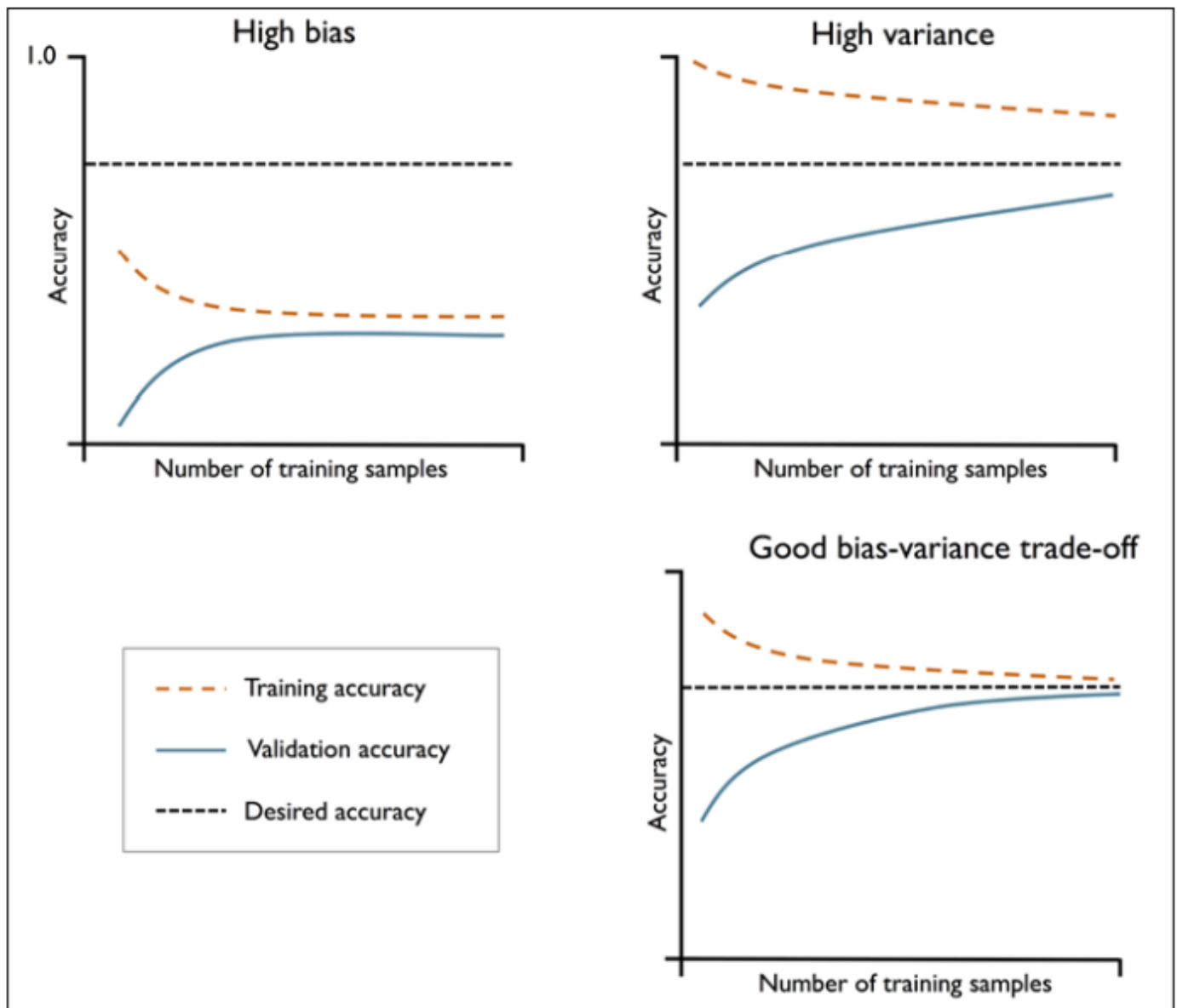
Ở phần này, nội dung xoay quanh với 2 biểu đồ quan trọng là *learning curves* và *validation curves* để đánh giá các thuật toán liệu có các vấn đề sau:

- Underfit
- Overfit
- Goodfit

Diagnosing bias and variance problems with learning curves

Phát hiện các vấn đề về độ lệch và phương sai với learning curves

Learning curves: Sơ đồ bên dưới cho biết độ chính xác (trục y) cho tập train và valid của mô hình dưới dạng các hàm số dựa trên kích thước của tập huấn luyện (trục x). Ta có thể dễ dàng phát hiện xem mô hình có phương sai cao hay độ lệch cao hay không và liệu việc thu thập thêm dữ liệu có thể giúp giải quyết vấn đề này?



- High Bias (Underfit Learning Curves) : có thể thấy rằng model không thể học tập các data_train, đường cong *Training accuracy* liên tục giảm cho đến khi kết thúc training. Giải quyết vấn đề bằng cách tăng độ phức tạp của model.
- High Variance (Overfit Learning Curves): có thể thấy rằng model học tập các data_train rất tốt nhưng đối với tập data_test không hiệu quả như mong đợi. Giải quyết vấn đề bằng cách thu thập thêm data, có thể áp dụng phương pháp *regularization* (một loại ràng buộc mềm - soft constrain).
- Good bias-variance trade-off: Model tốt nhất để training các dữ liệu.

Sử dụng function `scikit-learn` để đánh giá Model:

```
import numpy as np
```

```

import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Đọc bộ dữ liệu ung thư
df = pd.read_csv('https://archive.ics.uci.edu/ml/'
                 'machine-learning-databases'
                 '/breast-cancer-wisconsin/wdbc.data', header=None)

X = df.loc[:, 2:].values
y = df.loc[:, 1].values
le = LabelEncoder()
y = le.fit_transform(y)

# Chia tập train, test tỉ lệ 80 - 20
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y, random_s

import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

# Khởi tạo pipeline với 2 bước chuẩn hóa và bộ phân loại Logistic
pipe_lr = make_pipeline(StandardScaler(),
                        LogisticRegression(penalty='l2', random_state=1))

# Đánh giá score dựa bằng learning curves
train_sizes, train_scores, test_scores = \
    learning_curve(estimator=pipe_lr, X=X_train, y=y_train,
                  train_sizes=np.linspace(0.1, 1.0, 10), cv=10, n_jobs=1)

# Tính trung bình các giá trị
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)

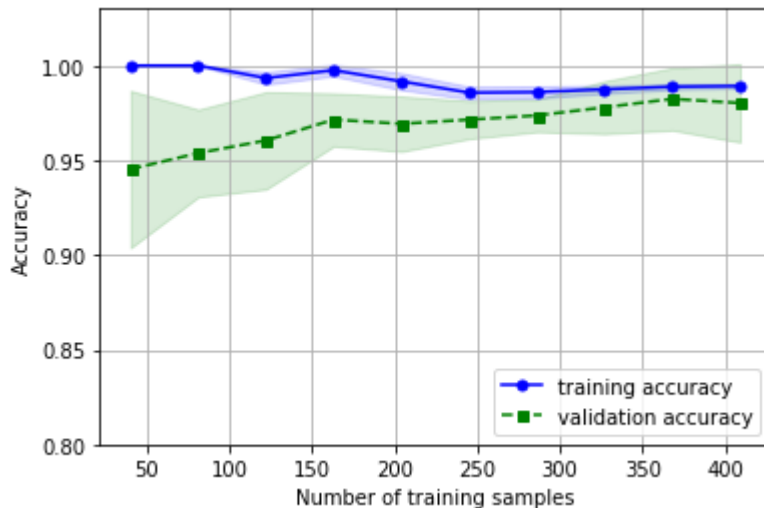
# Độ lệch chuẩn tập test
test_std = np.std(test_scores, axis=1)

# Vẽ hình
plt.plot(train_sizes, train_mean, color='blue', marker='o',
         markersize=5, label='training accuracy')
plt.fill_between(train_sizes, train_mean + train_std,
                 train_mean - train_std,
                 alpha=0.15, color='blue')
plt.plot(train_sizes, test_mean, color='green', linestyle='--',
         marker='s', markersize=5, label='validation accuracy')
plt.fill_between(train_sizes, test_mean + test_std,
                 test_mean - test_std, alpha=0.15, color='green')

```



```
plt.grid()
plt.xlabel('Number of training samples')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.ylim([0.8, 1.03])
plt.show()
```



▼ Addressing over- and underfitting with validation curves

Giải quyết các vấn đề over và underfitting với validation curves

Validation curves: là đường chuẩn đoán quan trọng để thấy mức độ nhạy cảm giữa những thay đổi về độ chính xác của model với thay đổi tham số trong model. Đưa ra nhiều lựa chọn và chọn lọc các vùng tối ưu nhất.

```
# Hàm tránh in ra các dòng cảnh báo lỗi
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import validation_curve

# Khởi tạo vùng giá trị tham số
param_range = [0.001, 0.01, 0.1, 1.0, 10.0, 100.0]

# Khởi tạo validation curves
train_scores, test_scores = validation_curve(estimator=pipe_lr, X=X_train, y=y_train,
                                             param_name='logisticregression__C',
                                             param_range=param_range, cv=10)

# Tính trung bình và độ lệch chuẩn dùng để vẽ hình
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
```

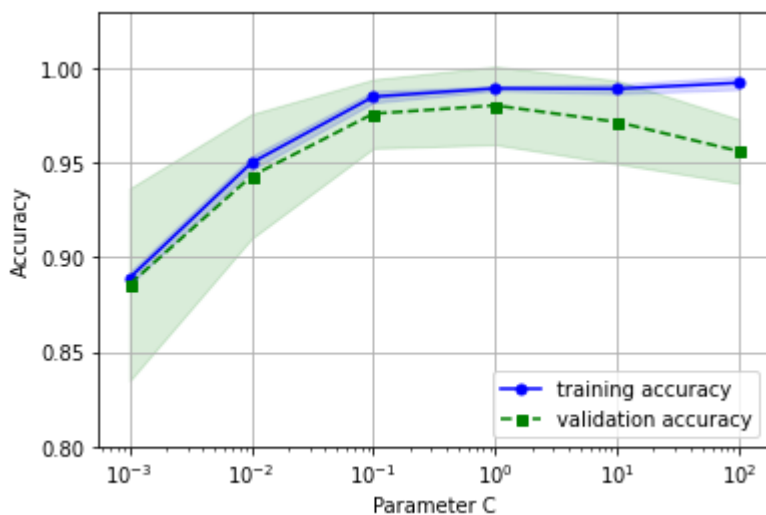
```

test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Vẽ hình
plt.plot(param_range, train_mean,color='blue', marker='o',
         markersize=5, label='training accuracy')
plt.fill_between(param_range, train_mean + train_std,
                 train_mean - train_std, alpha=0.15,color='blue')
plt.plot(param_range, test_mean,color='green', linestyle='--',
         marker='s', markersize=5,label='validation accuracy')
plt.fill_between(param_range,test_mean + test_std,
                 test_mean - test_std,alpha=0.15, color='green')

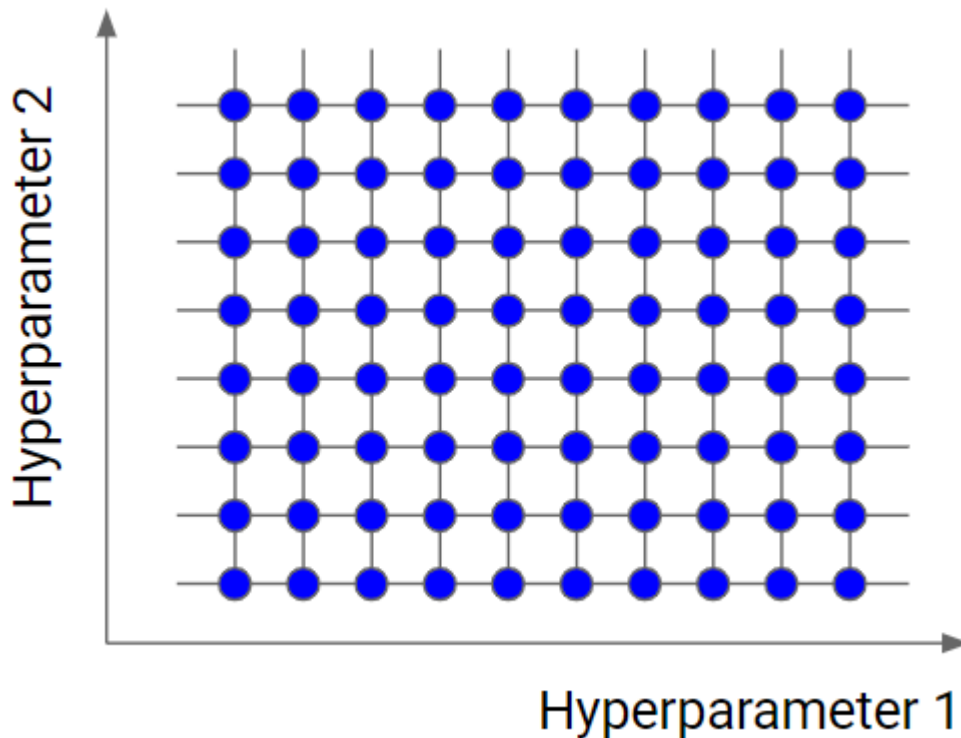
plt.grid()
plt.xscale('log')
plt.legend(loc='lower right')
plt.xlabel('Parameter C')
plt.ylabel('Accuracy')
plt.ylim([0.8, 1.03])
plt.show()

```



▼ 6.4. Fine-tuning machine learning models via grid search

- **Hyperparameters** (siêu tham số) là các tham số không hoặc ít bị thay đổi qua các lần lặp và không phụ thuộc vào tập training.
- **Grid search** là thuật toán đơn giản để điều chỉnh các *hyperparameter*, chia miền tham số thành một grid (lưới) rời rạc, thử kết hợp giá trị của lưới này và tính toán một số chỉ số hiệu suất bằng cách sử dụng xác thực chéo. Điểm của lưới sẽ là điểm tối đa của các giá trị trung bình trong k-fold cross-valid, là sự kết hợp tối ưu của các giá trị cho *hyperparameter*.



▼ Tuning hyperparameters via grid search

Điều chỉnh các siêu tham số bằng phương pháp lưới

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
pipe_svc = make_pipeline(StandardScaler(),SVC(random_state=1))
param_range = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
param_grid = [{'svc__C': param_range,
                'svc__kernel': ['linear']},
               {'svc__C': param_range,
                'svc__gamma': param_range,
                'svc__kernel': ['rbf']}]
gs = GridSearchCV(estimator=pipe_svc, param_grid=param_grid,scoring='accuracy',cv=10, n_jobs=
gs = gs.fit(X_train, y_train)
print(gs.best_score_)
print(gs.best_params_)

0.9846859903381642
{'svc__C': 100.0, 'svc__gamma': 0.001, 'svc__kernel': 'rbf'}

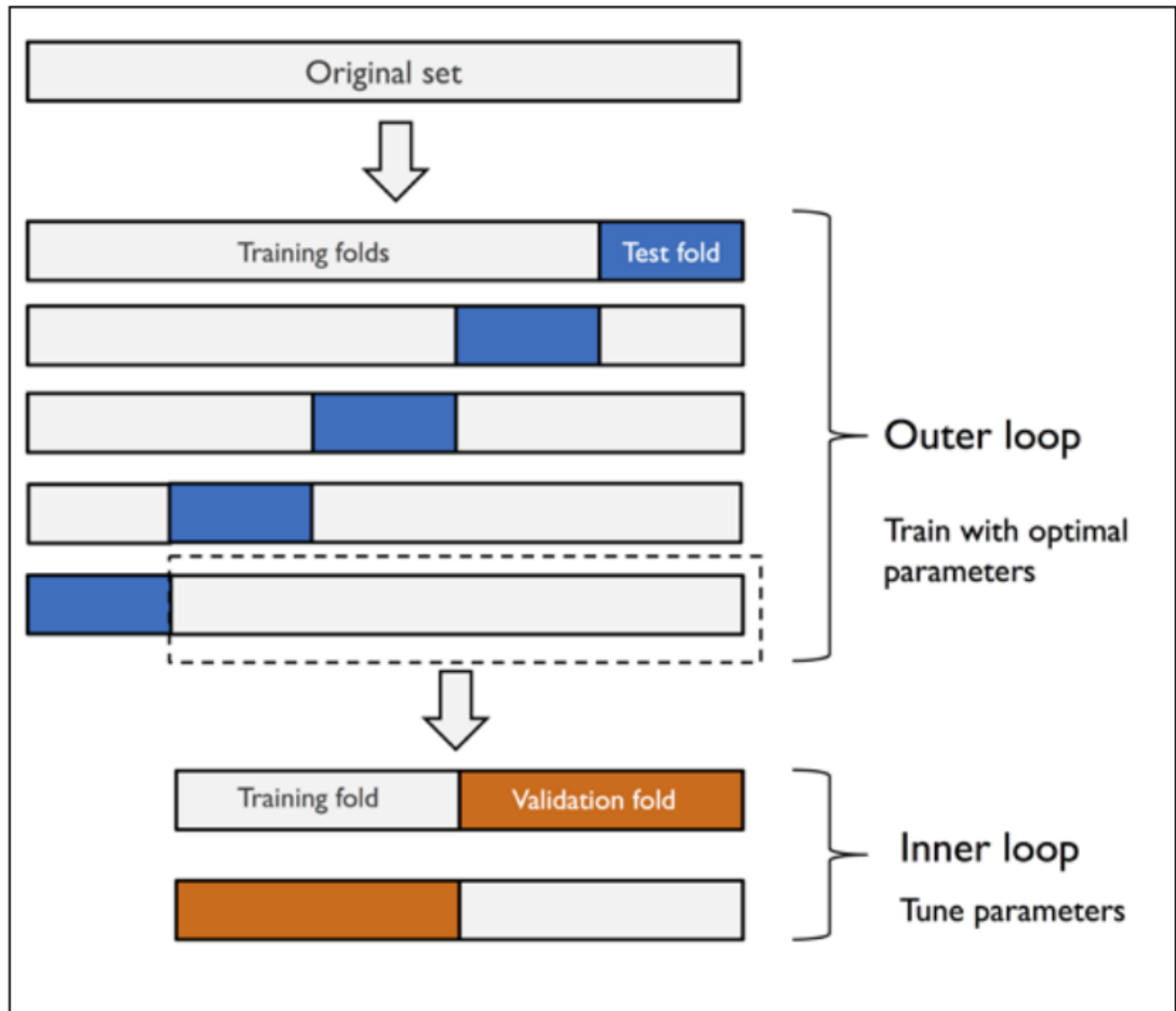
clf = gs.best_estimator_
clf.fit(X_train, y_train)
print('Test accuracy: %.3f' % clf.score(X_test, y_test))
```

Test accuracy: 0.974

▼ Algorithm selection with nested cross-validation

Lựa chọn thuật toán với phương pháp Nested cross-validation

Nested cross-validation là một cách tiếp cận để tối ưu hóa model *hyperparameter* và lựa chọn mô hình nhằm khắc phục vấn đề overfit trong tập train. Quy trình này liên quan đến việc tối ưu hóa *hyperparameter* như một phần của chính model và đánh giá nó trong quy trình *k-fold cross-validation* rộng hơn để đánh giá các model sau đó so sánh và lựa chọn.



```
zfrom sklearn.model_selection import cross_val_score
```

```

from sklearn.tree import DecisionTreeClassifier

gs = GridSearchCV(estimator=pipe_svc, param_grid=param_grid, scoring='accuracy',cv=2) #inner
scores = cross_val_score(gs, X_train, y_train, scoring='accuracy', cv=5)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))

gs = GridSearchCV(estimator=DecisionTreeClassifier(random_state=0),
                  param_grid=[{'max_depth': [1, 2, 3,4, 5, 6, 7, None]}],scoring='accuracy',c
scores = cross_val_score(gs, X_train, y_train,scoring='accuracy', cv=5)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))

CV accuracy: 0.974 +/- 0.015
CV accuracy: 0.934 +/- 0.016

```

▼ Looking at different performance evaluation metrics

Xem xét các số liệu đánh giá hiệu suất khác nhau

- Trong các phần và chương trước, chúng ta đã đánh giá các mô hình của mình bằng cách sử dụng độ chính xác của mô hình, đây là thước đo hữu ích để định lượng hiệu suất của một mô hình nói chung.
- Tuy nhiên, có một số chỉ số hiệu suất khác có thể được sử dụng để đo lường mức độ phù hợp của mô hình.

▼ 6.5. Reading a confusion matrix

Đọc ma trận nhầm lẫn

- Confusion Matrix (ma trận nhầm lẫn) là một bố cục bảng cụ thể cho phép hình dung hiệu suất của một thuật toán.
- Là một trong những kỹ thuật đo lường hiệu suất phổ biến nhất và được sử dụng rộng rãi cho các mô hình phân loại.

		Predicted class	
		P	N
Actual class	P	True positives (TP)	False negatives (FN)
	N	False positives (FP)	True negatives (TN)

Trong đó:

1. True Positive (TP): Dương tính thật.
2. False Negative (FN): Âm tính giả.
3. False Positive (FP): Dương tính giả.
4. True Negative (TN): Âm tính thật.

```
# load Breast Cancer Wisconsin dataset
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

df = pd.read_csv('https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master')
X = df.loc[:, 2:].values
y = df.loc[:, 1].values
le = LabelEncoder()
y = le.fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=1)

from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
```

```

from sklearn.pipeline import Pipeline

# construct pipeline for SVC
pipe_svc = Pipeline([('scl', StandardScaler()),
                      ('clf', SVC(random_state=1))])

from sklearn.metrics import confusion_matrix

# train the model
pipe_svc.fit(X_train, y_train)

# obtain predicted label
y_pred = pipe_svc.predict(X_test)

# print confusion matrix
confmat = confusion_matrix(y_true=y_test, y_pred=y_pred)
print(confmat)

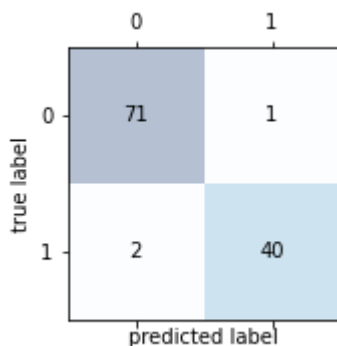
[[71  1]
 [ 2 40]]

# plot the confusion matrix illustration using matplotlib's matshow function
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(2.5, 2.5))
ax.matshow(confmat, cmap=plt.cm.Blues, alpha=0.3)
for i in range(confmat.shape[0]):
    for j in range(confmat.shape[1]):
        ax.text(x=j, y=i, s=confmat[i, j], va='center', ha='center')

plt.xlabel('predicted label')
plt.ylabel('true label')
plt.show()

```



▼ Optimizing the precision and recall of a classification model

Tối ưu hóa độ chính xác và độ nhạy của một mô hình phân loại

- Độ chính xác toàn thể - Accuracy (Acc)

$$ACC = \frac{TP + TN}{TP + FP + FN + TN}$$

- Sai số toàn thể - Error (Err)

$$ERR = \frac{FP + FN}{TP + FP + FN + TN} = 1 - ACC$$

- Độ nhạy - Recall (Rec) = Tỷ lệ dương tính thật - True positive rate (TPR): trong số các bệnh nhân chuẩn đoán có bệnh có bao nhiêu người được dự đoán là mắc bệnh.

$$REC = TPR = \frac{TP}{TP + FN}$$

- Precision (PRE): trong số những người được chuẩn đoán mắc bệnh, bao nhiêu phần trăm là mắc bệnh thật

$$PRE = \frac{TP}{TP + FP}$$

- Độ đặc hiệu: trong số bệnh nhân không mắc bệnh, bao nhiêu phần trăm được dự báo không mắc bệnh

$$Specificity = \frac{TN}{TN + FP}$$

- Tỷ lệ dương tính giả (FPR): trong số bệnh nhân không mắc bệnh, bao nhiêu phần trăm được dự báo mắc bệnh (báo nhầm)

$$FPR = \frac{FP}{FP + TN}$$

- F1-score:

$$F1 = 2 \frac{PRE \times REC}{PRE + REC}$$

- Độ chính xác toàn thể không phải là thước đo tốt trong trường hợp mẫu không cân xứng.

xét ví dụ: 100 bệnh nhân chuẩn đoán không mắc bệnh, trong đó 90 bệnh nhân không mắc bệnh và 10 bệnh nhân mắc bệnh. Ta có ma trận sau

Actual Class	Predicted Class		
		TRUE	FALSE
	TRUE	0	10
	FALSE	0	90

Độ chính xác toàn thể:

$$ACC = \frac{TP + TN}{TP + FP + FN + TN} = \frac{90}{100} = 0.9$$

Độ nhạy:

$$REC = \frac{TP}{TP + FN} = \frac{0}{10} = 0$$

- Độ đặc hiệu:

$$Spec = \frac{FP}{TN + NP} = \frac{90}{90} = 1$$

Nhận xét:

- Mô hình có độ chính xác cao
- Độ nhạy thấp: không chuẩn đoán được bệnh nhân thực tế mắc bệnh.

```
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin4/breast-cancer-wisconsin4.data')
X = df.loc[:, 2:].values
y = df.loc[:, 1].values
le = LabelEncoder()
y = le.fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=1)
```

```
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score, f1_score
print('Precision: %.3f' % precision_score(
    y_true=y_test, y_pred=y_pred))
```

Precision: 0.976

```
print('Recall: %.3f' % recall_score(y_true=y_test, y_pred=y_pred))
```

Recall: 0.952

```
print('F1: %.3f' % f1_score(y_true=y_test, y_pred=y_pred))
```

F1: 0.964

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
pipe_lr = make_pipeline(StandardScaler(),
                        PCA(n_components=2),
                        LogisticRegression(random_state=1))
pipe_lr.fit(X_train, y_train)
y_pred = pipe_lr.predict(X_test)
print('Test Accuracy: %.3f' % pipe_lr.score(X_test, y_test))
```

Test Accuracy: 0.947

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
```

```
scores = cross_val_score(estimator=pipe_lr,
                          X=X_train,
                          y=y_train,
                          cv=10,
                          n_jobs=1)
print('CV accuracy scores: %s' % scores)
```

Giống với SVM

```
from sklearn.svm import SVC
import sklearn
from sklearn.preprocessing import StandardScaler
```

```
pipe_svc = sklearn.pipeline.make_pipeline(StandardScaler(),
                                           SVC(random_state=1))
```

Valid curve

```
param_range = [0.0001, 0.001, 0.01, 0.1,
               1.0, 10.0, 100.0, 1000.0]
```

Cài đặt các tham số cho Grid search

```
param_grid = [{'svc__C': param_range,
               'svc__kernel': ['linear']}, # Cho trường hợp Linear SVM
               {'svc__C': param_range,
               'svc__gamma': param_range,
               'svc__kernel': ['rbf']}] # Cho trường hợp Kernel
```

```
gs = GridSearchCV(estimator=pipe_svc,
                  param_grid=param_grid,
```

```

        scoring='accuracy',
        cv=10,
        n_jobs=-1)

gs = gs.fit(X_train, y_train)

gs = GridSearchCV(estimator=pipe_svc,
                  param_grid=param_grid,
                  scoring='accuracy',
                  cv=2)
scores = cross_val_score(gs, X_train, y_train,
                          scoring='accuracy', cv=5)

gs = GridSearchCV(estimator=
                  DecisionTreeClassifier(random_state=0),
                  param_grid=[{'max_depth':
                               [1, 2, 3, 4, 5, 6, 7, None]}],
                  scoring='accuracy', cv=2)
scores = cross_val_score(gs, X_train, y_train,
                          scoring='accuracy', cv=5)

CV accuracy scores: [0.91304348 0.97826087 0.97826087 0.91304348 0.93478261 0.97777778
0.93333333 0.95555556 0.97777778 0.95555556]

from sklearn.metrics import make_scorer, f1_score
scorer = make_scorer(f1_score, pos_label=0)
gs = GridSearchCV(estimator=pipe_svc,
                  param_grid=param_grid,
                  scoring=scorer,
                  cv=10)
gs = gs.fit(X_train, y_train)
print(gs.best_score_)

0.98287253786131

```

Receiver operating characteristic (ROC) and Area under the curve (AUC)

- Receiver operating characteristic (ROC) là Đồ thị sẽ biểu diễn với mỗi điểm cutpoint ứng với nó sẽ có tỷ lệ TPR (báo đúng) và tỷ lệ FPR (báo nhầm) là bao nhiêu:

Dựa trên ROC curve, ta có thể chỉ ra rằng một mô hình có hiệu quả hay không. Một mô hình hiệu quả khi có FPR (báo nhầm) thấp và TPR (báo đúng) cao, tức tồn tại một điểm trên ROC curve gần với điểm có tọa độ (0, 1) trên đồ thị (góc trên bên trái). Curve càng gần thì mô hình càng hiệu quả.

- Area under the curve (AUC):

1. Là diện tích phần nằm dưới ROC.
2. Thể hiện khả năng dự báo của mô hình: đúng hơn trong trường hợp mẫu mất cân bằng.

```
from sklearn.metrics import roc_curve, auc
from scipy import interp
```

```
pipe_lr = make_pipeline(StandardScaler(),PCA(n_components=2),
                        LogisticRegression(penalty='l2',
                                           random_state=1,C=100.0))
```

```
# Split dataset, 20% test, 80% train
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = \
    train_test_split(X, y,
                    test_size=0.20,
                    stratify=y,
                    random_state=1)
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
pipe_lr = make_pipeline(StandardScaler(),
                        PCA(n_components=2),
                        LogisticRegression(random_state=1))
pipe_lr.fit(X_train, y_train)
y_pred = pipe_lr.predict(X_test)
print('Test Accuracy: %.3f' % pipe_lr.score(X_test, y_test))
```

Test Accuracy: 0.956

```
import numpy as np
from sklearn.model_selection import StratifiedKFold
```

```
# Lựa chọn số k = 10
kfold = StratifiedKFold(n_splits=10,
                        random_state = None).split(X_train, y_train)
```

```
from sklearn.metrics import roc_curve, auc
from scipy import interp
import warnings
warnings.filterwarnings("ignore")
```

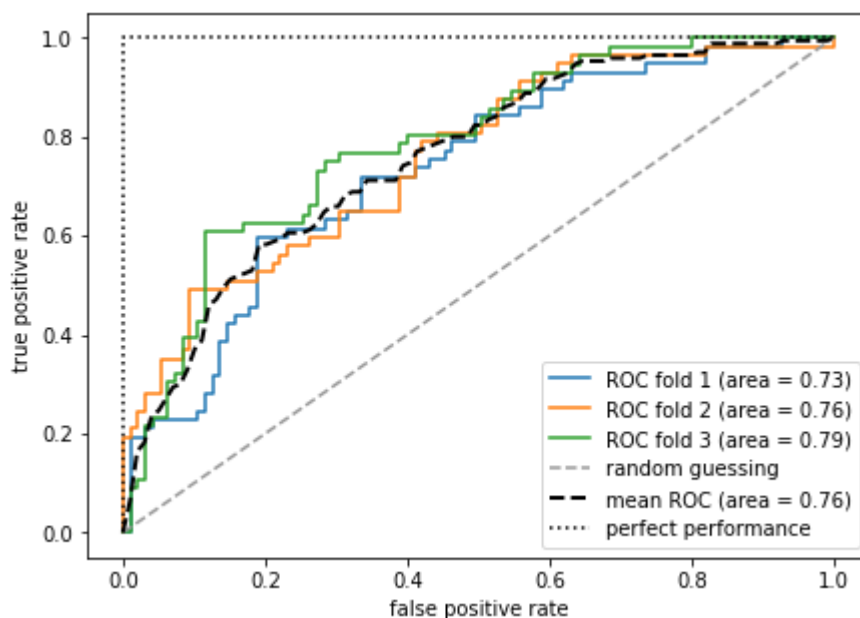
```
pipe_lr = make_pipeline(StandardScaler(),
                        PCA(n_components=2),
                        LogisticRegression(penalty='l2', random_state=1, C=100.0))
```

```

X_train2 = X_train[:, [4, 14]]
cv = list(StratifiedKFold(n_splits=3, random_state=None).split(X_train, y_train))
fig = plt.figure(figsize=(7, 5))
mean_tpr = 0.0
mean_fpr = np.linspace(0, 1, 100)
all_tpr = []
for i, (train, test) in enumerate(cv):
    probas = pipe_lr.fit(X_train2[train],
                        y_train[train]).predict_proba(X_train2[test])
    fpr, tpr, thresholds = roc_curve(y_train[test],
                                    probas[:, 1],
                                    pos_label=1)

    mean_tpr += interp(mean_fpr, fpr, tpr)
    mean_tpr[0] = 0.0
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr,
             label='ROC fold %d (area = %0.2f)'
             % (i+1, roc_auc))
plt.plot([0, 1], [0, 1], linestyle='--', color=(0.6, 0.6, 0.6), label='random guessing')
mean_tpr /= len(cv)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
plt.plot(mean_fpr, mean_tpr, 'k--', label='mean ROC (area = %0.2f)' % mean_auc, lw=2)
plt.plot([0, 0, 1], [0, 1, 1], linestyle=':',
        color='black', label='perfect performance')
plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.legend(loc="lower right")
plt.show()

```



▼ Scoring metrics for multiclass classification

Đánh giá bộ đa phân loại

- ROC curve, precision-recall curve ban đầu được định nghĩa cho bài toán phân lớp nhị phân. Để có thể áp dụng các phép đo này cho bài toán multi-class classification (nhiều lớp), các đại lượng đầu ra cần được đưa về dạng nhị phân
- Ta có thể đưa bài toán phân lớp nhiều lớp về bài toán phân lớp nhị phân bằng cách xem xét từng lớp. Với mỗi lớp, ta coi dữ liệu thuộc lớp đó có label là positive, tất cả các dữ liệu còn lại có label là negative. Sau đó, giá trị Precision, Recall, và PR curve được áp dụng lên từng lớp. Với mỗi lớp, ta sẽ nhận được một cặp giá trị precision và recall.
- Macro-average có trọng số được tính bằng cách tính trọng số của từng lớp, sau đó tính trung bình của các trọng số này.

VD: Tính trung bình của Precision:

$$PRE_{macro} = \frac{PRE_1 + \dots + PRE_k}{k}$$

Ưu điểm của Macro-average:

1. Hữu ích nếu chúng ta muốn tính trọng số tất cả các lớp như nhau để đánh giá hiệu suất tổng thể của classifier.
2. Có trọng số rất hữu ích nếu chúng ta đang xử lý sự mất cân bằng của lớp.

- Micro-average được tính từ các TP, TN, FP và FN riêng lẻ của từng lớp.

VD: Tính trung bình của Precision:

$$PRE_{micro} = \frac{TP_1 + \dots + TP_k}{TP_1 + \dots + TP_k + FP_1 + \dots + FP_k}$$

Ưu điểm của Micro-average: hữu ích nếu chúng ta muốn cân bằng từng trường hợp dự đoán như nhau

Ta có thể dùng hàm `make_scorer` hoặc hàm `precision_score` trong thư viện `scikitlearn` của Python để tính

```
pre_scorer = make_scorer(score_func=precision_score,
```

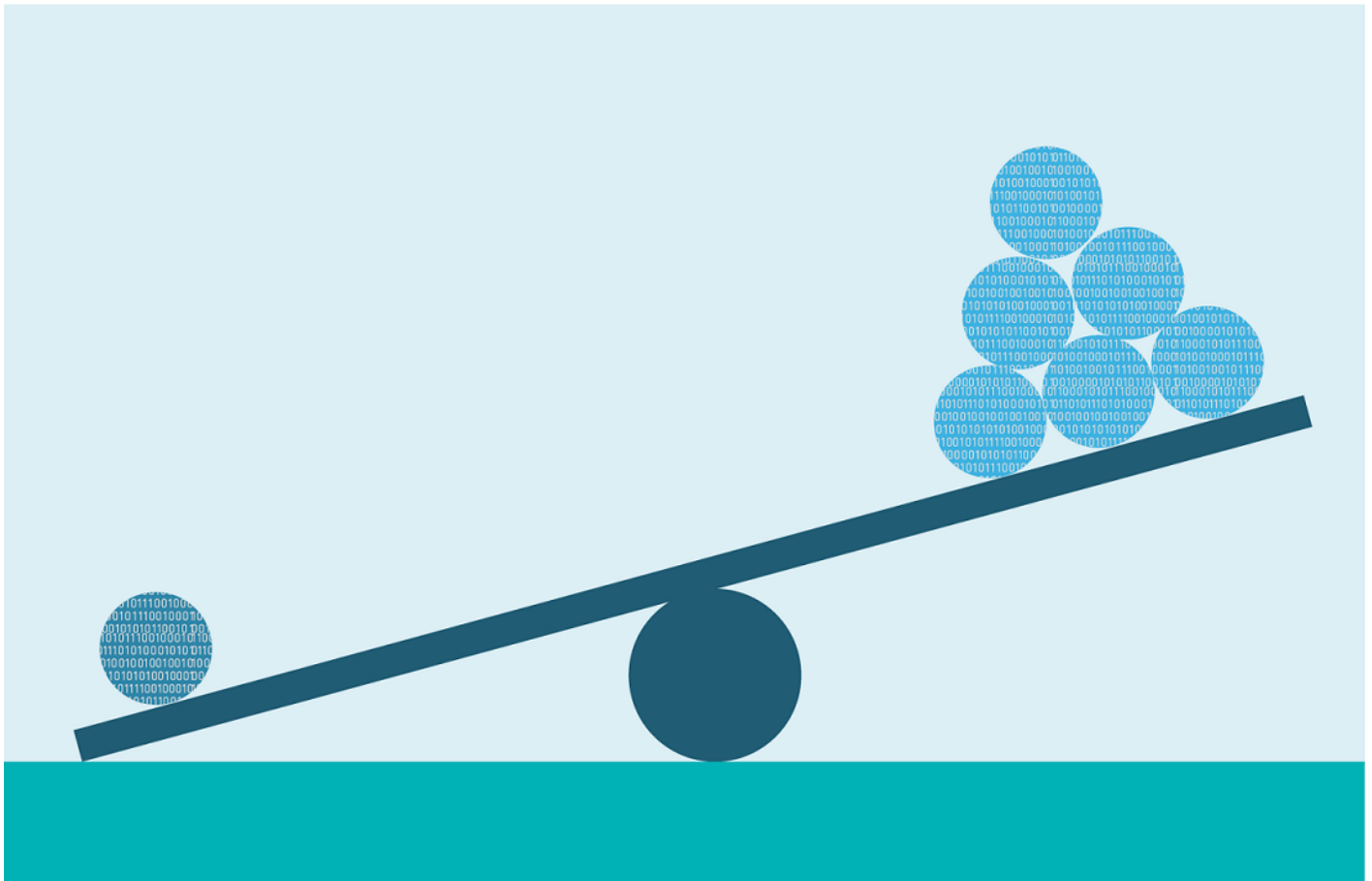
```
pos_label=1,  
greater_is_better=True,  
average='micro')
```

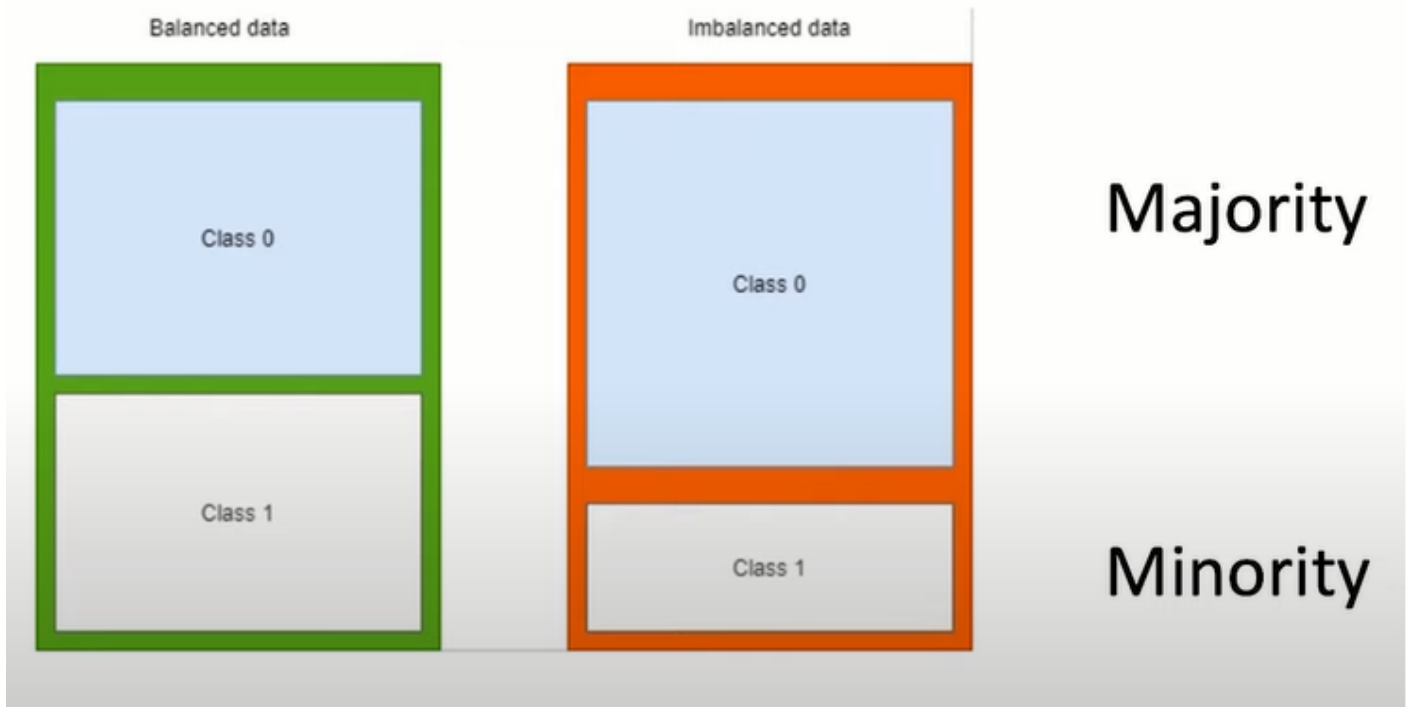
6.6. Dealing with class imbalance

Xử lý mất cân bằng lớp

Trong các phần trước chúng ta đề cập đến việc mất cân bằng lớp, ở phần này chúng ta sẽ tìm cách để xử lý việc mất cân bằng lớp này

- Bài toán phân loại (classification) trong máy học là quá trình dự đoán nhãn (label) của các điểm dữ liệu đã cho.
- Mất cân bằng lớp là một lớp hoặc nhiều lớp được đại diện quá mức trong tập dữ liệu





Tác hại của việc mất cân bằng lớp:

1. Model có thiên hướng dự đoán ra lớp Majority
2. Accuracy không còn tác dụng để đánh giá

VD: Trong tập dữ liệu về ung thư vú mà chúng ta đang làm việc trong chương này bao gồm 90% bệnh nhân khỏe mạnh.

- Nếu ta sử dụng một model luôn trả về không ung thư thì accuracy của model đó đạt tới 90%.
- Khi ta train model sẽ có xu hướng trả về không ung thư để đạt được accuracy cao.

⇒ Mô hình không tốt vì không dự đoán được người có bệnh ung thư.

- Từ bộ dữ liệu ung thư vú, ban đầu bao gồm 357 khối u lành tính (loại 0) và 212 khối u ác tính (loại 1)
- Chúng ta đã lấy tất cả 357 mẫu lành tính và xếp chồng chúng với 40 mẫu ác tính đầu tiên để tạo ra sự mất cân bằng lớp rõ rệt. Nếu chúng ta tính toán độ chính xác của một mô hình luôn dự đoán lớp đa số (lành tính, lớp 0), thì chúng ta sẽ đạt được độ chính xác dự đoán xấp xỉ 90%

```
X_imb = np.vstack((X[y == 0], X[y == 1][:40]))
y_imb = np.hstack((y[y == 0], y[y == 1][:40]))
```



```
y_pred = np.zeros(y_imb.shape[0])
np.mean(y_pred == y_imb) * 100
```

89.92443324937027

Các phương pháp xử lý

1. Thay đổi metric đánh giá Model.
2. Undersampling.
3. Oversampling.
4. Class Weighted
5. Ensemble & Boosting

Thay đổi metric đánh giá Model

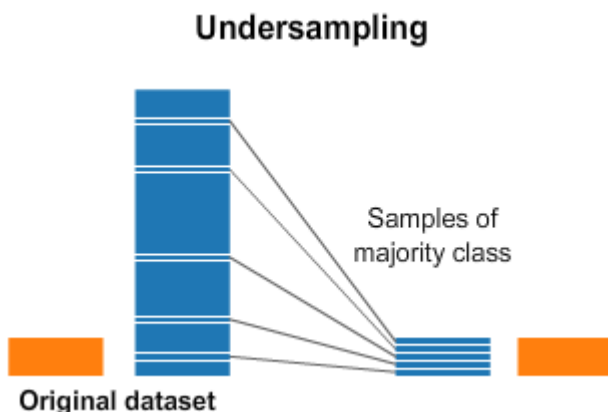
Khi xử lý bộ phân loại trên các bộ dữ liệu như vậy, sẽ hợp lý khi tập trung vào các số liệu khác hơn là độ chính xác khi so sánh các mô hình khác nhau, chẳng hạn như độ chính xác (Precision), khả năng thu hồi (Recall), đường cong ROC hoặc bất cứ tham số cần quan tâm nhất trong bài toán.

vd: Trong bài toán tìm bệnh nhân ung thư phổi trên cần quan tâm đến giá trị Recall để không bỏ sót bất kì bệnh nhân bị bệnh nào.

Undersampling

Under sampling là việc ta giảm số lượng các sample của nhóm đa số để nó trở nên cân bằng với số sample của nhóm thiểu số.

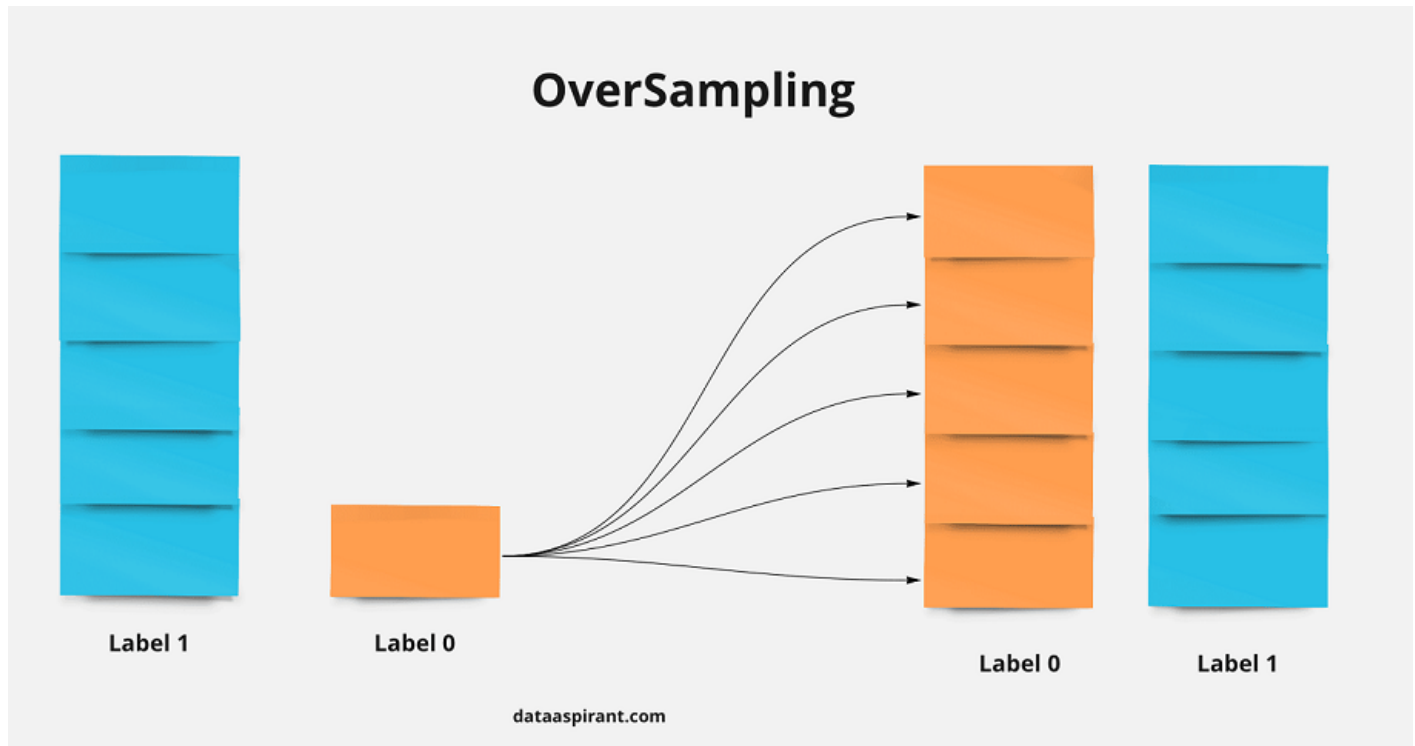
Random Undersampling



1. Ưu điểm: Tạo ra bộ dataset balance để train được.

2. Nhược điểm: Có thể bỏ sót các sample quan trọng trong lớp Majority

Oversampling



random Over sampling là phương pháp tái chọn mẫu là dữ liệu mẫu giả lập mới sẽ giống dữ liệu sẵn có. Do đó ta sẽ cân bằng mẫu bằng cách lựa chọn ngẫu nhiên có lặp lại các sample thuộc nhóm thiểu số.

1. Ưu điểm: có bộ dataset balanced

2. Nhược điểm:

- Phá vỡ sự phân phối của Label
- Dễ bị overfit

```
from sklearn.utils import resample
print('Number of class 1 samples before:',
      X_imb[y_imb == 1].shape[0])
```

Number of class 1 samples before: 40

```
X_upsampled, y_upsampled = resample(X_imb[y_imb == 1],
                                     y_imb[y_imb == 1], replace=True, n_samples=X_imb[y_imb == 0].shape[0], random_state=123)
```

```
print('Number of class 1 samples after:', X_upsampled.shape[0])
```

```
Number of class 1 samples after: 357
```

```
X_bal = np.vstack((X[y == 0], X_upsampled))
y_bal = np.hstack((y[y == 0], y_upsampled))
```

```
y_pred = np.zeros(y_bal.shape[0])
np.mean(y_pred == y_bal) * 100
```

```
50.0
```

Class_Weight

- Việc dự báo sai một quan sát thuộc mẫu đa số sẽ ít nghiêm trọng hơn so với dự báo sai một quan sát thuộc mẫu thiểu số. Xuất phát từ ý tưởng đó chúng ta sẽ phạt nặng hơn đối với sai số dự báo thuộc nhóm thiểu bằng cách gán cho nó một trọng số lớn hơn trong công thức sau.

$$W_j = \frac{n_{sample}}{n_{classes} * n_{sample_j}}$$

Trong đó:

- W_j là trọng số của mỗi class (j là class)
- n_{sample} là tổng số sample trong dataset
- $n_{classes}$ là tổng số lớp trong data.
- n_{sample_j} là tổng số sample trong class j.

Nhận xét: trong công thức này n_{sample} và $n_{classes}$ là số cố định, còn n_{sample_j} là tùy thuộc vào từng class nếu sample nhiều thì trọng số sẽ giảm còn nếu ít sample thì trọng số sẽ lớn.

7. Combining Different Models for Ensemble Learning

Kết hợp các mô hình khác nhau cho việc học nhóm

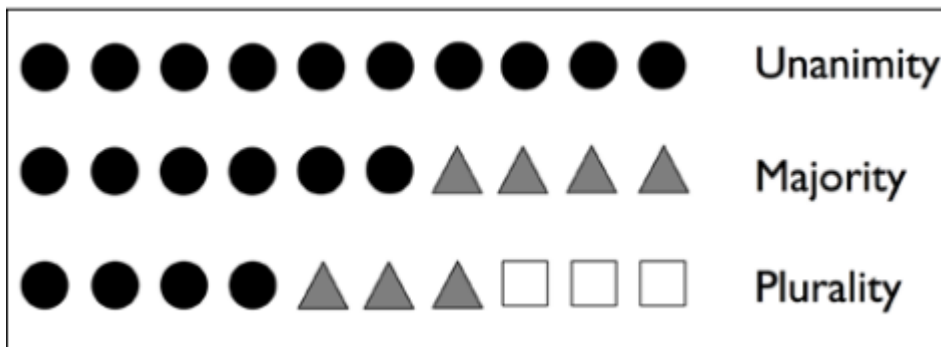
▼ 7.1. Learning with ensembles

Học kết hợp

- Mục tiêu của phương pháp này là kết hợp các bộ phân loại khác nhau thành một bộ phân loại được gọi là bộ đa phân loại (meta-classifier)
- Bộ đa phân loại này sẽ có hiệu suất tổng quát hiệu quả hơn so với các bộ phân loại cấu thành nên nó.

Các phương pháp Ensemble Learning phổ biến nhất sử dụng nguyên tắc bầu chọn đa số. Nghĩa là chúng ta sẽ chọn các nhãn được dự đoán bởi phần lớn các nhà phân loại. Chiếm hơn 50% tỉ lệ chọn. Thuật ngữ đa số phiếu (*majority vote*) chỉ đề cập đến cài đặt lớp nhị phân.

- VD 1:

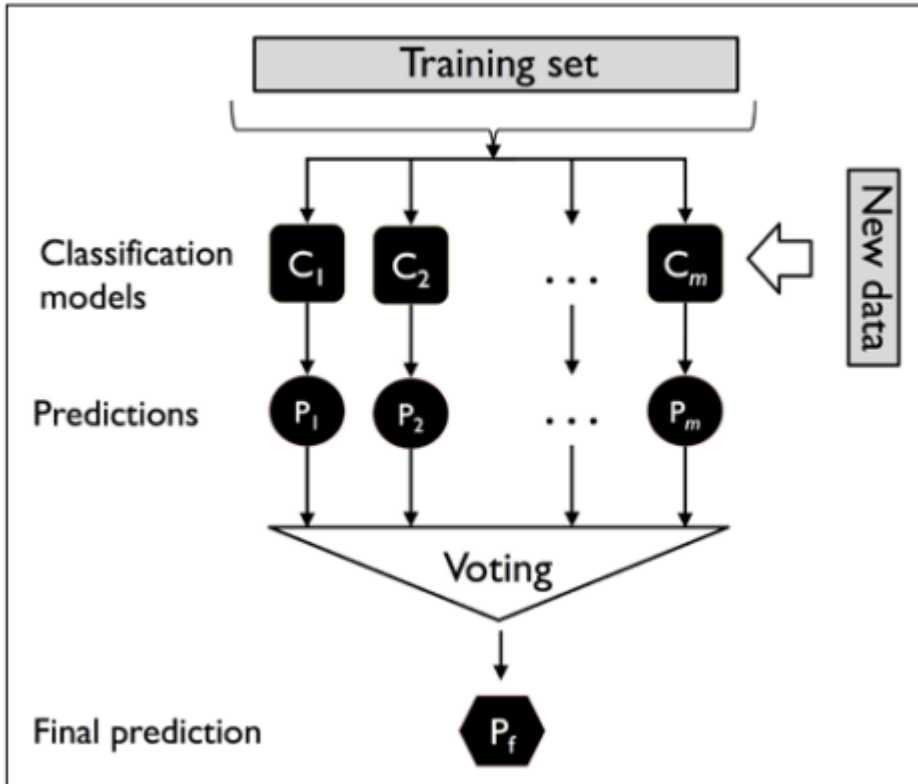


Sử dụng các tập huấn luyện khác nhau, chúng ta bắt đầu bằng cách huấn luyện các lớp phân loại khác nhau. Dựa trên kĩ thuật này, một ensemble có thể được xây dựng dựa trên các thuật toán phân loại khác nhau. VD:

- Cây quyết định (Decision Tree) là một mô hình thuộc nhóm thuật toán giám sát
- SVM (Support vector machines) là một thuật toán giám sát, nó có thể sử dụng cho cả việc phân loại hoặc đệ quy. Tuy nhiên nó được sử dụng chủ yếu cho việc phân loại
- Logistic regression classifiers là 1 thuật toán phân loại được dùng để gán các đối tượng cho 1 tập hợp giá trị rời rạc

Ngoài ra, chúng ta có thể sử dụng các thuật toán phân loại dựa trên cùng cơ sở, tập hợp các khóa huấn luyện phù hợp khác nhau. Một ví dụ nổi bật là thuật toán Random Forest (Rừng ngẫu nhiên), là sự kết hợp nhiều cây quyết định khác nhau.

- VD 2:



Để dự đoán một lớp nhãn hiệu thông qua đa số hoặc bầu cử đa phần, chúng tôi kết hợp các nhãn dự đoán của mỗi bộ phân loại, C_j , và chọn nhãn lớp, \hat{y} , nhận được nhiều phiếu bầu.

$$\hat{y} = \text{mode}\{C_1(x), C_2(x), \dots, C_m(x)\}$$

Ví dụ, trong một công việc phân loại nhị phân nơi $class_1 = -1$ và $class_2 = +1$, chúng ta có thể viết một dự đoán bỏ phiếu đa số như sau:

$$C(x) = \text{sign}\left[\sum_j^m C_j(x)\right] = \begin{cases} 1 & \text{nếu } \sum_i C_j(x) \geq 0 \\ -1 & \text{khác} \end{cases}$$

Để minh họa tại sao phương pháp tập hợp có thể hoạt động tốt hơn so với việc phân loại riêng lẻ, chúng ta hãy áp dụng những khái niệm đơn giản về tổ hợp.

$$P(y \geq k) = \sum_k^n \binom{n}{k} \epsilon^k (1 - \epsilon)^{n-k} = \epsilon_{\text{tổ hợp}}$$

$\binom{n}{k}$ là các nhị thức của hệ số n chọn k . Nói cách khác chúng ta tính xác suất rằng dự đoán của toàn bộ nhóm là sai.

Hệ số nhị thức: là số cách mà chúng ta có thể chọn tập con của k phần tử không có thứ tự nhất định từ tập có kích thước n . Thường được gọi là **n chọn k** . Đôi khi nó cũng được gọi là tổ hợp. Được viết dưới dạng:

$$\frac{n!}{(n-k)!k!}$$

```
import numpy as np
import matplotlib.pyplot as plt
import math

# from scipy.misc import comb # lỗi do scipy.misc.comb
# không được sử dụng nữa, thay thế bằng scipy.special
from scipy.special import comb

# Hàm tính tỉ lệ lỗi của ensemble, mục tiêu là tính xác suất toàn bộ mô hình sai
def ensemble_error(n_classifier, error):
    k_start = int(math.ceil(n_classifier / 2.)) # chỉ cần tính tỉ lệ hơn 50% bộ phân loại ch
    probs = [comb(n_classifier, k) *
              error**k *
              (1-error)**(n_classifier - k)
              for k in range(k_start, n_classifier + 1)] # xác suất hơn 50% bộ phân loại cho k
    return sum(probs)

ensemble_error(n_classifier=11, error=0.25)
```

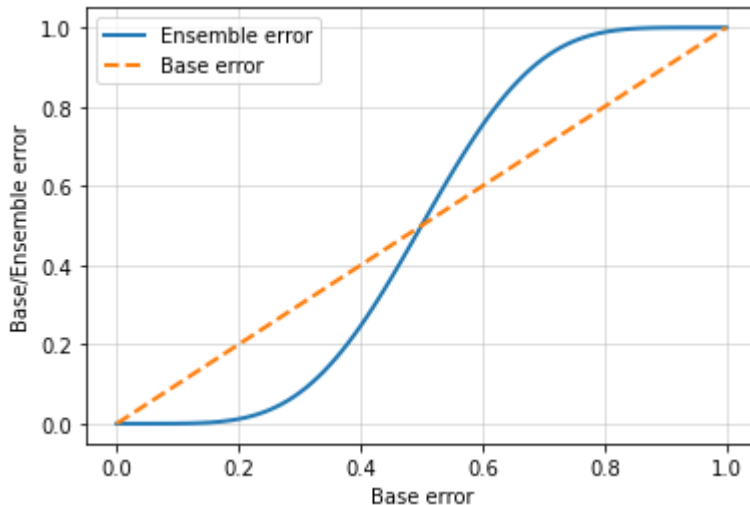
Như chúng ta có thể thấy, tỷ lệ sai lầm của ensemble (0,034) thấp hơn tỷ lệ lỗi của mỗi bộ phân loại riêng (0,25) nếu tất cả các giả định đều được đáp ứng.

Sau khi thực hiện hàm ensemble_error, chúng ta có thể tính toán tỷ lệ lỗi ensemble cho một loạt các lỗi cơ sở khác nhau từ 0,0 đến 1,0 để hình dung mối quan hệ giữa lỗi tập hợp và lỗi cơ sở trong biểu đồ đường:

```
error_range = np.arange(0.0, 1.01, 0.01)
ens_errors = [ensemble_error(n_classifier=11, error=error)
               for error in error_range]
```

Vẽ hình, không có gì liên quan đến phương pháp

```
plt.plot(error_range, ens_errors,
         label='Ensemble error',
         linewidth=2)
plt.plot(error_range, error_range,
         linestyle='--', label='Base error',
         linewidth=2)
plt.xlabel('Base error')
plt.ylabel('Base/Ensemble error')
plt.legend(loc='upper left')
plt.grid(alpha=0.5)
plt.show()
```



Xác suất lỗi của một nhóm luôn tốt hơn của một bộ phân loại cơ sở riêng lẻ, mỗi bộ phân loại cơ sở riêng lẻ phải tốt hơn đoán ngẫu nhiên ($\epsilon < 0.5$)

▼ 7.2. Combining classifiers via majority vote

Kết hợp các bộ phân loại thông qua bỏ phiếu đa số

▼ Implementing a simple majority vote classifier

Thực hiện phân loại theo bầu cử đa số đơn giản

Thuật toán cho phép kết hợp các thuật toán phân loại khác nhau được liên kết với các trọng số riêng lẻ của từng bộ phân loại để tạo độ tin cậy. Mục tiêu đặt ra là xây dựng một meta-classifier mạnh hơn để cân bằng các điểm yếu của các phân loại riêng lẻ trên một tập dữ liệu cụ thể.

$$\hat{y} = \underset{i}{\operatorname{argmax}} \sum_{j=1}^m w_j \chi_A(C_j(x) = i)$$

Để cân bằng trọng lượng, ta có thể đơn giản hóa phương trình như sau:

$$\hat{y} = \operatorname{mode}\{C_1(x), C_2(x), \dots, C_m(x)\}$$

▼ Trọng số

- Tùy từng mục đích, domain knowledge khác nhau mà người dùng có thể cài đặt trọng số khác nhau

Ví dụ, một tập hợp gồm ba bộ phân loại cơ sở, $C_j, j = 1, \dots, n$ và muốn dự đoán nhãn các mẫu quan sát x đã cho với hai trong số ba bộ có dự đoán nhãn lớp 0 và một có dự đoán rằng mẫu lớp 1

$$C_1(x) \rightarrow 0, C_2(x) \rightarrow 0, C_3(x) \rightarrow 1$$

$$\hat{y} = \operatorname{mode}\{0, 0, 1\} = 0$$

Ta có thể gán trọng số 0,6 cho C_3 và 0.2 cho C_1 và C_2 :

$$\hat{y} = \underset{i}{\operatorname{argmax}} \sum_{j=1}^m w_j \chi_A(C_j(x) = i)$$

$$= \underset{i}{\operatorname{argmax}} [0.2 \times i_0 + 0.2 \times i_1 + 0.6 \times i_1] = 1$$

Hoặc có thể viết lại:

$$\hat{y} = \operatorname{mode}\{0, 0, 1, 1, 1\} = 1$$

- Code:

```
import numpy as np
np.argmax(np.bincount([0, 0, 1], weights=[0.2, 0.2, 0.6]))

1
```

Nhắc lại phần hồi quy logistic trong Chương 3, một số bộ phân loại nhất định trong scikit-learning cũng có thể trả về xác suất của nhãn lớp dự đoán thông qua hàm `predict_proba`.

- Việc sử dụng xác suất lớp được dự đoán thay vì nhãn lớp để biểu quyết theo đa số tốt hơn nếu các bộ phân loại trong tập hợp của chúng ta được hiệu chỉnh đúng.

Vậy ta viết lại dưới dạng xác suất:

$$\hat{y} = \underset{j=1}{\operatorname{argmax}} \sum_{j=1}^m w_j p_{ij}$$

Với p_{ij} là xác suất dự đoán của bộ phân loại thứ j cho nhãn i

Ví dụ 2: Giả sử rằng chúng ta có một vấn đề phân loại nhị phân với các nhãn lớp $i \in \{0,1\}$ và một tập hợp gồm ba bộ phân loại C_j ($j \in \{1,2,3\}$).

Các bộ phân loại C_j trả về xác suất thành phần với một điểm cụ thể x như sau:

$$C_1(x) \rightarrow [0.9, 0.1], C_2(x) \rightarrow [0.8, 0.2], C_3(x) \rightarrow [0.4, 0.6]$$

Ta tính xác suất các lớp thành phần:

$$p(0|x) = 0.2 \times 0.9 + 0.2 \times 0.8 + 0.6 \times 0.4 = 0.58$$

$$p(1|x) = 0.2 \times 0.1 + 0.2 \times 0.2 + 0.6 \times 0.6 = 0.42$$

$$\hat{y} = \underset{i}{\operatorname{argmax}} [p(i_0|x), p(i_1|x)] = 0$$

Code:

```
ex = np.array([[0.9, 0.1],
               [0.8, 0.2],
               [0.4, 0.6]])
# Tính trung bình tất cả trọng số * xác suất
p = np.average(ex, axis=0, weights=[0.2, 0.2, 0.6])
print(p)
np.argmax(p)

[0.58 0.42]
0
```

Viết Class cài đặt Majority Vote:

```
from sklearn.base import BaseEstimator
from sklearn.base import ClassifierMixin
from sklearn.preprocessing import LabelEncoder
import six
from sklearn.base import clone
from sklearn.pipeline import _name_estimators
import numpy as np
import operator

class MajorityVoteClassifier(BaseEstimator, ClassifierMixin):
    # Khởi tạo các tham số cần thiết bao gồm: bộ phân loại, tên nhãn, trọng số
    def __init__(self, classifiers,
                 vote='classlabel', weights=None):
```

```

self.classifiers = classifiers
self.named_classifiers = {key: value
                           for key, value in _name_estimators(classifiers)} # truy cập
self.vote = vote # số dự đoán
self.weights = weights # trọng số

```

Training

```

def fit(self, X, y):
    self.lablenc_ = LabelEncoder() # mã hóa các label
    self.lablenc_.fit(y) # training
    self.classes_ = self.lablenc_.classes_ # truy cập đến các lớp sau khi Encoder
    self.classifiers_ = [] # khởi tạo bộ phân loại
    for clf in self.classifiers:
        fitted_clf = clone(clf).fit(X,
                                     self.lablenc_.transform(y)) # fit X với các label y đã
    self.classifiers_.append(fitted_clf) # trả về kết quả label
    return self

```

Trả về kết quả majority voting dự đoán

```

def predict(self, X):

    # Nếu dự đoán dựa trên xác suất
    if self.vote == 'probability':
        maj_vote = np.argmax(self.predict_proba(X),
                              axis=1) # Trả về label có xác suất dự đoán cao nhất

    # Nếu dự đoán dựa trên mode
    else:
        predictions = np.asarray([clf.predict(X)
                                   for clf in self.classifiers_]).T
        maj_vote = np.apply_along_axis(lambda x:
                                         np.argmax(np.bincount(x,
                                                                  weights=self.weights)),
                                         axis=1,
                                         arr=predictions)
    maj_vote = self.lablenc_.inverse_transform(maj_vote) # mã hóa ngược lại label y đã encode
    return maj_vote

```

Trả về các xác suất trung bình

```

def predict_proba(self, X):
    probas = np.asarray([clf.predict_proba(X) for clf in self.classifiers_]) # tính xác suất
    avg_proba = np.average(probas, axis=0, weights=self.weights) # trung bình xác suất theo trọng số
    return avg_proba

```

Trả về tên và giá của các tham số để tính toán độ chính xác của dự đoán dựa trên GridSearch

```

def get_params(self, deep=True):
    # Nếu không muốn trả về
    if not deep:
        return super(MajorityVoteClassifier, self).get_params(deep=False)
    # Nếu trả về
    else:
        out = self.named_classifiers.copy()

```

```

for name, step in six.iteritems(self.named_classifiers): # định dạng lại named_cl
    for key, value in six.iteritems(
        step.get_params(deep=True)):
        out['%s__%s' % (name, key)] = value
return out

```

▼ Sử dụng Majority Voting để đưa ra dự đoán

Sử dụng tập dữ liệu Iris từ scikit-learn. Chọn hai tính năng, chiều rộng đài hoa và chiều dài cánh hoa

```

from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
iris = datasets.load_iris()
X, y = iris.data[50:, [1, 2]], iris.target[50:] # Chỉ lấy 100 mẫu, có label là 1 và 2
print(y)
le = LabelEncoder() # Khởi tạo hàm mã hóa
y = le.fit_transform(y)

```

```

[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]

```

y

```

array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])

```

Chia các mẫu Iris thành 50% dữ liệu huấn luyện và 50% dữ liệu kiểm tra

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5,
                                                    random_state = 2021, # giữ nguyên dữ liệu
                                                    stratify=y) # giữ nguyên tỉ lệ số lượng r

```

Sử dụng ba bộ phân loại khác nhau:

- Phân loại hồi quy logistic
- Phân loại cây quyết định
- Phân loại k-láng giềng gần nhất

- Đánh giá hiệu suất mô hình của từng bộ phân loại thông qua cross-validation 10 lần trên tập dữ liệu huấn luyện trước khi kết hợp thành một nhóm:

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
import numpy as np

# Khởi tạo các bộ phân loại
clf1 = LogisticRegression(penalty='l2', C=0.001, random_state=1)
clf2 = DecisionTreeClassifier(max_depth=1, criterion='entropy', random_state=0)
clf3 = KNeighborsClassifier(n_neighbors=1, p=2, metric='minkowski')

# Khởi tạo pipeline, DecisionTree không cần có bước chuẩn hóa
pipe1 = Pipeline([['sc', StandardScaler()], ['clf', clf1]])
pipe3 = Pipeline([['sc', StandardScaler()], ['clf', clf3]])
clf_labels = ['Logistic regression', 'Decision tree', 'KNN']
print('10-fold cross validation:\n')
for clf, label in zip([pipe1, clf2, pipe3], clf_labels):
    scores = cross_val_score(estimator=clf, X=X_train,
                              y=y_train,
                              cv=10, # 10 fold
                              scoring='roc_auc') # Đánh giá hiệu suất dựa trên đường cong ROC
    print("ROC AUC: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))

10-fold cross validation:

ROC AUC: 0.92 (+/- 0.15) [Logistic regression]
ROC AUC: 0.87 (+/- 0.18) [Decision tree]
ROC AUC: 0.85 (+/- 0.13) [KNN]
```

- Kết hợp các bộ phân loại

```
mv_clf = MajorityVoteClassifier(classifiers=[pipe1, clf2, pipe3]) # Kết hợp cả 3 bộ phân loại
clf_labels += ['Majority voting'] # Đặt tên
all_clf = [pipe1, clf2, pipe3, mv_clf] # Kiểm tra cả 4 phương pháp

for clf, label in zip(all_clf, clf_labels):
    scores = cross_val_score(estimator=clf,
                              X=X_train,
                              y=y_train,
                              cv=10, scoring='roc_auc')
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))

Accuracy: 0.92 (+/- 0.15) [Logistic regression]
Accuracy: 0.87 (+/- 0.18) [Decision tree]
```

Accuracy: 0.85 (+/- 0.13) [KNN]

Accuracy: 0.98 (+/- 0.05) [Majority voting]

- Vậy hiệu suất của majorityVotingClassifier đã được cải thiện so với các bộ phân loại riêng lẻ trong 10 lần đánh giá cross-validation

▼ Đánh giá và điều chỉnh bộ phân loại kết hợp

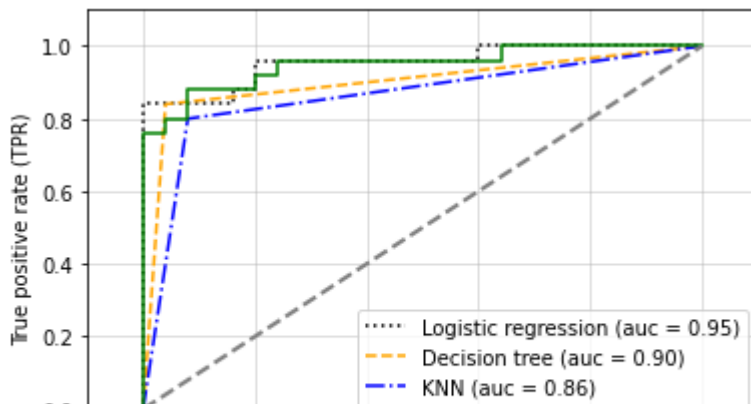
- Tính toán các đường cong ROC từ bộ test để kiểm tra xem MajorityVoteClassifier có tổng quát hóa tốt với dữ liệu chưa từng gặp hay không.
- Bộ kiểm tra không được sử dụng để lựa chọn mô hình; mục đích của nó chỉ đơn thuần là báo cáo ước tính khách quan về hiệu suất tổng quát hóa của hệ thống phân loại:

```
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
colors = ['black', 'orange', 'blue', 'green']
linestyles = [':', '--', '-.', '-']

# Mục đích kiểm tra xem bộ phân loại kết hợp có bị overfit không
for clf, label, clr, ls in zip(all_clf,
                              clf_labels,
                              colors, linestyles):
    # Giả sử tất cả label đều là 1, đánh giá dựa trên ROC
    y_pred = clf.fit(X_train, y_train).predict_proba(X_test)[:, 1]
    fpr, tpr, thresholds = roc_curve(y_true=y_test,
                                     y_score=y_pred)

    roc_auc = auc(x=fpr, y=tpr)
    # Vẽ hình qua mỗi phương pháp thành phần
    plt.plot(fpr, tpr, color=clr, linestyle=ls,
             label='%s (auc = %0.2f)' % (label, roc_auc))

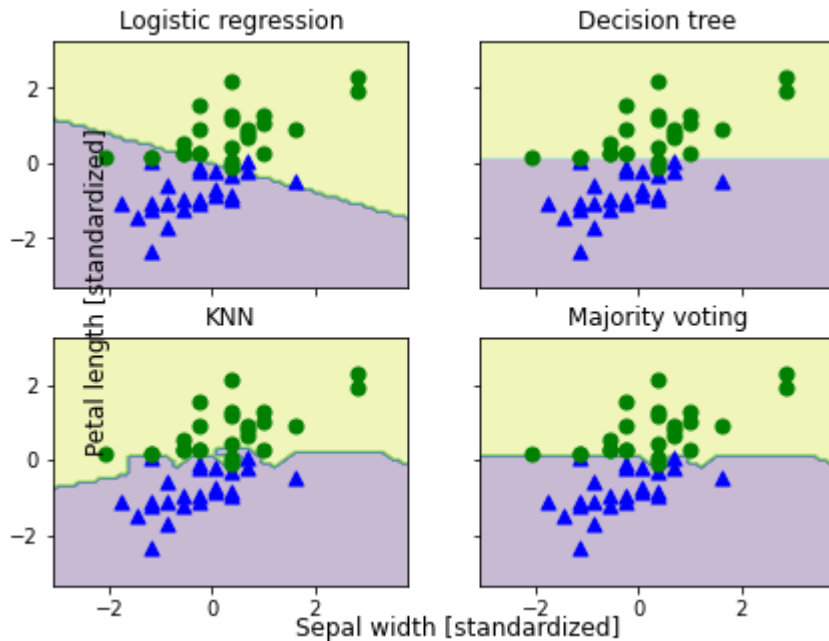
# Vẽ hình tổng
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1],
         linestyle='--',
         color='gray', linewidth=2)
plt.xlim([-0.1, 1.1])
plt.ylim([-0.1, 1.1])
plt.grid(alpha=0.5)
plt.xlabel('False positive rate (FPR)')
plt.ylabel('True positive rate (TPR)')
plt.show()
```



- Kết quả ROC cho biết trình phân loại tập hợp cũng hoạt động tốt trên tập kiểm tra (ROC AUC = 0,95).

Code chuẩn hóa:

```
# Nhằm giúp các nhánh của cây quyết định cùng một tỷ lệ với Logistic và KNN cho các mục đích
sc = StandardScaler() # chuẩn hóa
X_train_std = sc.fit_transform(X_train)
from itertools import product
x_min = X_train_std[:, 0].min() - 1
x_max = X_train_std[:, 0].max() + 1
y_min = X_train_std[:, 1].min() - 1
y_max = X_train_std[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
np.arange(y_min, y_max, 0.1))
f, axarr = plt.subplots(nrows=2, ncols=2,
                        sharex='col',
                        sharey='row', figsize=(7, 5))
for idx, clf, tt in zip(product([0, 1], [0, 1]),
                        all_clf, clf_labels):
    clf.fit(X_train_std, y_train)
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    axarr[idx[0], idx[1]].contourf(xx, yy, Z, alpha=0.3)
    axarr[idx[0], idx[1]].scatter(X_train_std[y_train==0, 0],
                                X_train_std[y_train==0, 1],
                                c='blue',
                                marker='^', s=50)
    axarr[idx[0], idx[1]].scatter(X_train_std[y_train==1, 0],
                                X_train_std[y_train==1, 1], c='green',
                                marker='o', s=50)
    axarr[idx[0], idx[1]].set_title(tt)
plt.text(-3.5, -4.5, s='Sepal width [standardized]',
        ha='center', va='center', fontsize=12)
plt.text(-10.5, 4.5, s='Petal length [standardized]',
        ha='center', va='center',
        fontsize=12, rotation=90)
plt.show()
```



- Biên của Majority Vote đa số trông rất giống với cây quyết định, trực giao với trục y đối và chiều rộng biên ≥ 1 .
- Tuy nhiên, có thêm sự phi tuyến tính từ bộ phân loại KNN
- Cách truy cập các thuộc tính của bộ phân loại là riêng lẻ.
- Ta có thể điều chỉnh tham số C của bộ phân loại hồi quy logistic và độ sâu của cây quyết định thông qua **GridSearch**:

```
from sklearn.model_selection import GridSearchCV
params = {'decisiontreeclassifier__max_depth': [1, 2],
          'pipeline-1__clf__C': [0.001, 0.1, 100.0]}
grid = GridSearchCV(estimator=mv_clf,
                    param_grid=params, cv=10,
                    scoring='roc_auc')
grid.fit(X_train, y_train)

GridSearchCV(cv=10,
             estimator=MajorityVoteClassifier(classifiers=[Pipeline(steps=[('sc',
StandardScaler()),
                                                                           ('clf',
LogisticRegression(C=0.001,
random_state=1))]),
DecisionTreeClassifier(criterion='entropy',
max_depth=1,
```

```

random_state=0),

Pipeline(steps=[('sc',

StandardScaler()),

['clf',

KNeighborsClassifier(n_neighbors=1)]))],
    param_grid={'decisiontreeclassifier__max_depth': [1, 2],
                'pipeline-1__clf__C': [0.001, 0.1, 100.0]},
    scoring='roc_auc')

```

- Sau khi tìm kiếm lưới, ta có kết hợp giá trị siêu tham số khác nhau và điểm ROC AUC trung bình được tính thông qua 10 lần cross-validation như sau:

```

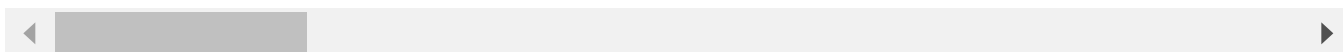
for i in ['mean_test_score', 'std_test_score', 'params']:
    print(i, " : ", grid.cv_results_[i])

print('Best parameters: %s' % grid.best_params_)

print('Accuracy: %.2f' % grid.best_score_)

mean_test_score : [0.98333333 0.98333333 0.96666667 0.98333333 0.98333333 0.96666667]
std_test_score  : [0.05 0.05 0.1 0.05 0.05 0.1 ]
params         : [{'decisiontreeclassifier__max_depth': 1, 'pipeline-1__clf__C': 0.001}, {'de
Best parameters: {'decisiontreeclassifier__max_depth': 1, 'pipeline-1__clf__C': 0.001}
Accuracy: 0.98

```



Mở rộng

- Để nhằm lẫn phương pháp **majority vote** và phương pháp xếp chồng - **stacking**.

Thuật toán xếp chồng có thể được hiểu là một tập hợp gồm hai lớp:

- Lớp đầu tiên gồm các bộ phân loại thành phần nhằm đưa dự đoán của chúng lên lớp thứ hai
- Lớp thứ 2 gồm một bộ phân loại khác (thường là hồi quy logistic) phù hợp với các dự đoán của bộ phân loại thứ nhất để đưa ra dự đoán cuối cùng

Thuật toán xếp chồng đã được mô tả chi tiết hơn bởi David H. Wolpert trong Stacked generalization, Neural Networks, 5(2):241–259, 1992.

- Tuy nhiên thuật toán này chưa được triển khai trong scikit-learning tại thời điểm viết bài;

Tham khảo:

http://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/

và

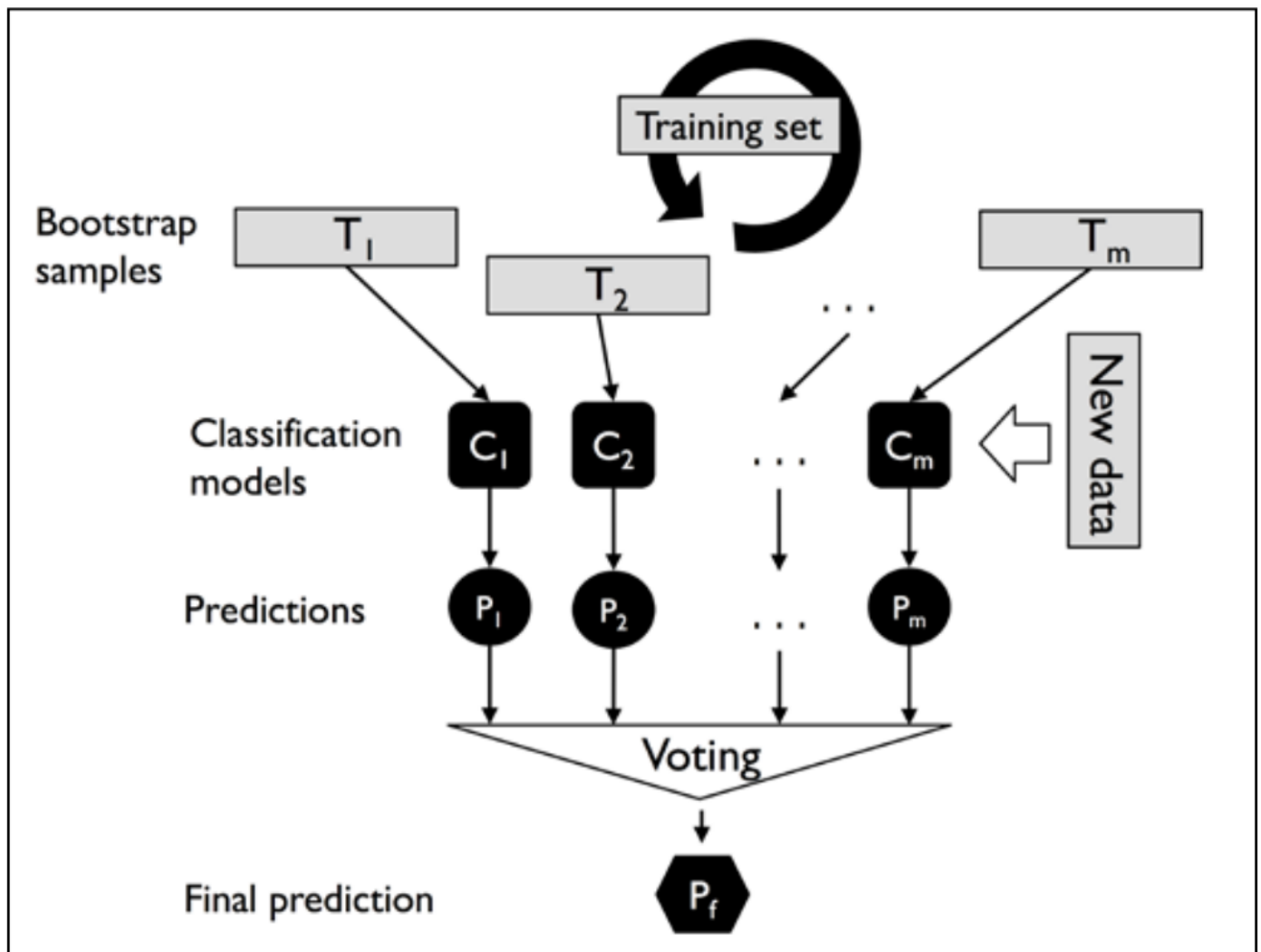
http://rasbt.github.io/mlxtend/user_guide/classifier/StackingCVClassifier/

7.3. Bagging – building an ensemble of classifiers from bootstrap samples

Bagging - xây dựng một mô hình phân loại kết hợp dựa trên phương pháp lấy mẫu bootstrap

Bagging còn được gọi là Bootstrap aggregating, một kỹ thuật ensemble learning giúp cải thiện hiệu suất và độ chính xác của machine learning algorithms. Nó được sử dụng để giải quyết với sự đánh đổi sai lệch phương sai và giảm phương sai của một mô hình dự đoán. Tính năng bagging giúp tránh **overfitting** và được sử dụng cho cả mô hình hồi quy và mô hình phân loại, đặc biệt cho decision tree algorithms.

- Là sự kết hợp của các weak-learner đồng nhất với nhau, chúng học hỏi lẫn nhau một cách độc lập, song song và sau đó kết hợp các weak-learner để xác định mức trung bình của mô hình.

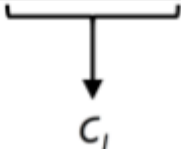


Các bước thực hiện Bagging

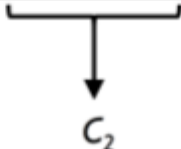
1. (Bootstrap samples): Các bộ data con được tạo từ bộ data gốc với các bộ dữ liệu bằng nhau.
2. Mỗi weak-learner được tạo nên ứng với mỗi bộ data con
3. Các weak-learner đó sẽ học độc lập với nhau và song song với mỗi tập data train.
4. Dự đoán cuối cùng bằng cách kết hợp các dự đoán của model

Bootstrapping là 1 kĩ thuật tạo các bộ data con từ data gốc với cơ chế lấy ngẫu nhiên và có lặp lại

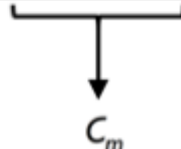
Sample indices	Bagging round 1	Bagging round 2	...
1	2	7	...
2	2	3	...
3	1	2	...
4	3	1	...
5	7	1	...
6	2	7	...
7	4	7	...



C_1



C_2



C_m

Ví dụ:

- Random forest

▼ Applying bagging to classify samples in the Wine dataset

Áp dụng bagging để phân loại bộ dữ liệu về rượu

```
import numpy as np
import matplotlib.pyplot as plt
```

Xét 2 tính năng để phân loại rượu: Nồng độ cồn và OD280/OD315 của độ pha loãng rượu

```
import pandas as pd
df_wine = pd.read_csv('https://archive.ics.uci.edu/ml/'
                      'machine-learning-databases/wine/wine.data', header=None)
df_wine.columns = ['Class label', 'Alcohol', 'Malic acid',
                  'Ash', 'Alcalinity of ash', 'Magnesium',
                  'Total phenols', 'Flavanoids', 'Nonflavanoid phenols',
                  'Proanthocyanins', 'Color intensity', 'Hue',
                  'OD280/OD315 of diluted wines', 'Proline']

# drop 1 class
df_wine = df_wine[df_wine['Class label'] != 1]
```

```

y = df_wine['Class label'].values
X = df_wine[['Alcohol','OD280/OD315 of diluted wines']].values

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
le = LabelEncoder()
y = le.fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1, stratify

```

Mã hóa class labels thành định dạng nhị phân và chia tập dữ liệu thành 80% tập huấn luyện và 20% tập kiểm tra:

```

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import BaggingClassifier
tree = DecisionTreeClassifier(criterion='entropy', random_state=1, max_depth=None)
bag = BaggingClassifier(base_estimator=tree, n_estimators=500, max_samples=1.0, max_features=1.0

```

Tính toán điểm chính xác của dự đoán trên tập dữ liệu training và test để so sánh hiệu suất của bagging classifier với hiệu suất của một decision tree:

```

from sklearn.metrics import accuracy_score
tree = tree.fit(X_train, y_train)
y_train_pred = tree.predict(X_train)
y_test_pred = tree.predict(X_test)
tree_train = accuracy_score(y_train, y_train_pred)
tree_test = accuracy_score(y_test, y_test_pred)
print('Decision tree train/test accuracies %.3f/%.3f' % (tree_train, tree_test))

```

Decision tree train/test accuracies 1.000/0.833

Dựa trên các giá trị độ chính xác mà chúng tôi đã in ở đây, decision tree dự đoán chính xác tất cả the class labels của các mẫu đào tạo; tuy nhiên, độ chính xác kiểm tra thấp hơn đáng kể cho thấy phương sai cao (overfitting) của mô hình:

```

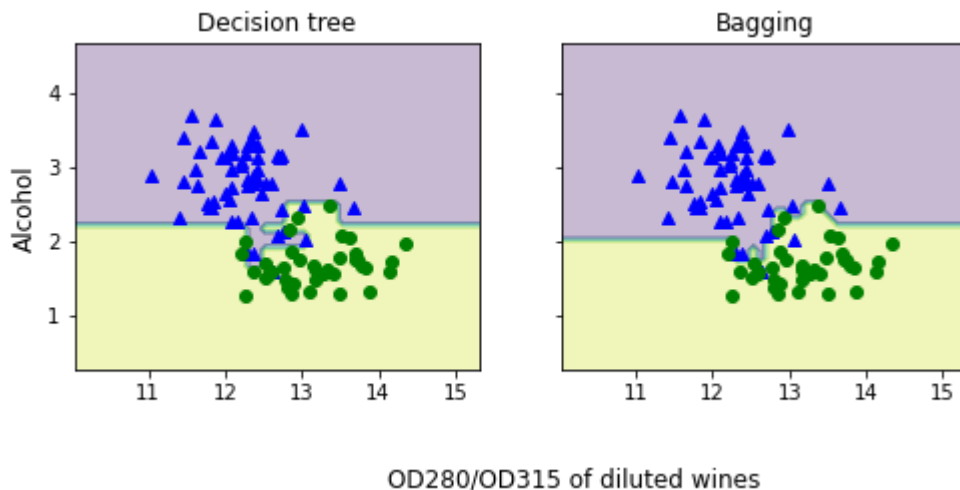
bag = bag.fit(X_train, y_train)
y_train_pred = bag.predict(X_train)
y_test_pred = bag.predict(X_test)
bag_train = accuracy_score(y_train, y_train_pred)
bag_test = accuracy_score(y_test, y_test_pred)
print('Bagging train/test accuracies %.3f/%.3f' % (bag_train, bag_test))

```

Bagging train/test accuracies 1.000/0.917

Mặc dù độ chính xác đào tạo của the decision tree và bagging classifier là tương tự nhau trên training set (cả 100%), chúng ta có thể thấy rằng the bagging classifier có hiệu suất tổng quát hóa tốt hơn một chút, như ước tính trên test set. Tiếp theo, chúng ta hãy so sánh các decision regions giữa decision tree và bagging classifier:

```
x_min = X_train[:, 0].min() - 1
x_max = X_train[:, 0].max() + 1
y_min = X_train[:, 1].min() - 1
y_max = X_train[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
np.arange(y_min, y_max, 0.1))
f, axarr = plt.subplots(nrows=1, ncols=2,
                        sharex='col',
                        sharey='row',
                        figsize=(8, 3))
for idx, clf, tt in zip([0, 1], [tree, bag], ['Decision tree', 'Bagging']):
    clf.fit(X_train, y_train)
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    axarr[idx].contourf(xx, yy, Z, alpha=0.3)
    axarr[idx].scatter(X_train[y_train==0, 0],
                       X_train[y_train==0, 1],
                       c='blue', marker='^')
    axarr[idx].scatter(X_train[y_train==1, 0],
                       X_train[y_train==1, 1],
                       c='green', marker='o')
    axarr[idx].set_title(tt)
axarr[0].set_ylabel('Alcohol', fontsize=12)
plt.text(10.2, -1.2,
        s='OD280/OD315 of diluted wines',
        ha='center', va='center', fontsize=12)
plt.show()
```



▼ 7.4. Boosting

Như ta có thể thấy các model trong bagging hoạt động một cách riêng lẻ, không ảnh hưởng với nhau. Chúng ta mong đợi các weak-learner có thể hỗ trợ lẫn nhau, học từ nhau để tránh những sai lầm trước đó.

Ý tưởng chính của Boosting là thay vì đào tạo các model (weak learner) song song, ta có thể đào tạo chúng một cách tuần tự. Và các model sau sẽ tập trung vào những phần phân loại kém của model trước đó

Boosting cố gắng xây dựng một model phân loại mạnh mẽ từ các weak-learner.

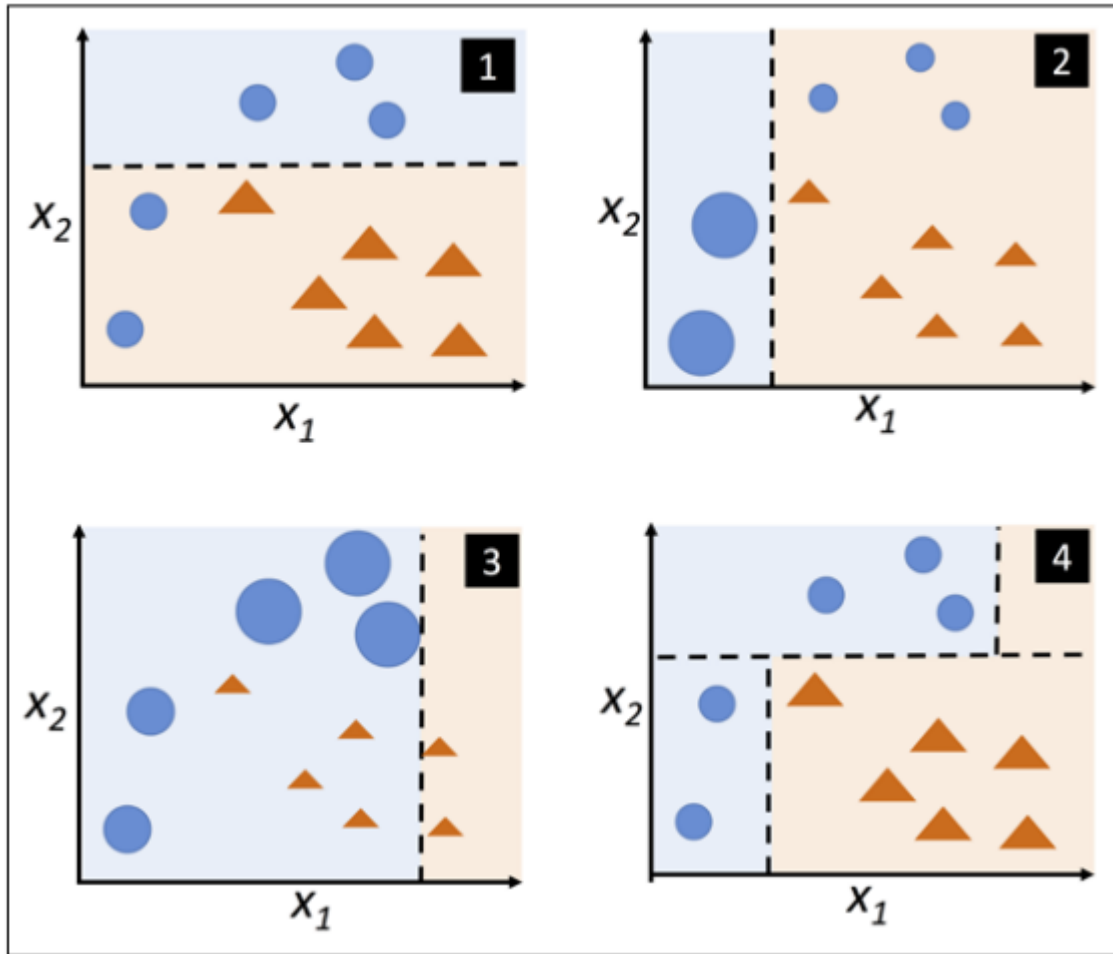
- Đầu tiên, weak-learner được xây dựng từ data train.
- Sau đó, weak-learner thứ 2 được xây dựng để cố gắng sửa các lỗi có trong model thứ nhất.
- Quy trình này được tiếp tục và các weak-learner được thêm vào cho đến khi tập data train hoàn chỉnh được dự đoán chính xác hoặc số lượng weak-learner tối đa được thêm vào.

Boosting tiến hành đánh trọng số cho các mô hình mới được thêm vào dựa trên cách đánh tối ưu khác nhau. Tùy theo cách đánh và tổng hợp lại các model, từ đó hình thành nên 2 loại Boosting:

- AdaBoost
- Gradient Boosting

▼ AdaBoost

AdaBoost hoạt động bằng cách đánh trọng số cho các điểm dữ liệu và các weak-learner, cái weak-learner sẽ học một cách tuần từ, weak-learner sau sẽ tập trung vào những điểm dữ liệu sai của weak-learner trước dự đoán. Bằng cách tăng trọng số cho các điểm dữ liệu dự đoán sai và giảm trọng số cho các điểm dữ liệu dự đoán chính xác. Vào sau cùng kết hợp các weak-learner lại với nhau.



- Tập dữ liệu $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$, với $y_i \in \{-1, 1\}, i \in \{1, 2, \dots, n\}$
- Trọng số các điểm dữ liệu tại weak learner thứ t : $w_1^t, w_2^t, \dots, w_n^t, i \in \{1, 2, \dots, n\}$
- weak-learners $h : x \rightarrow \{-1, 1\}$
- Error function: $E(\hat{y}_i, y, i) = e^{-y_i \hat{y}_i}$, trong đó $\hat{y} = \alpha h_t(x_i)$
- Output: $H_t(x), H_t(x)$ còn được gọi là strong learner tại thời điểm t

Các bước thực hiện:

B1: Khởi tạo weights cho từng input: $w_i^1 = \frac{1}{n}, i \in \{1, 2, \dots, n\} \rightarrow \sum_i w_i^1 = 1$

B2: Với mỗi vòng lặp $t \in \{1, 2, \dots, T\}$:

- Tìm weak-learners $h_t(x)$ để tối thiểu hóa tổng error của các điểm bị phân loại sai,

$$E = \sum_{y_i \neq h_t(x_i)} w_i^t$$
- Tỷ lệ lỗi của weak learners: $\varepsilon_t = \frac{\sum_{y_i \neq h_t(x_i)} w_i^t}{\sum_{i=1}^n w_i^t} = \sum_{y_i \neq h_t(x_i)} w_i^t$, vì $\sum_{i=1}^n w_i^t = 1$
- Gán trọng số cho weak-learners giá trị $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
- Cập nhật strong learner: $H_t(x) = H_{t-1}(x) + \alpha_t h_t(x)$
- Cập nhật lại weights: $w_i^{t+1} = w_i^t e^{-y_i \alpha_t h_t(x_i)}, i \in \{1, 2, \dots, n\}$

- Chuẩn hóa lại weights: $w_i^{t+1} \leftarrow \frac{w_i^{t+1}}{\sum_i w_i^{t+1}}$, có nghĩa là $\sum_i w_i^{t+1} = 1$

B3: Ouput chính là dấu của biểu thức tổng các weak-learners nhân với trọng số của chúng, hay

$$H(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right)$$

Sample indices	x	y	Weights	$\hat{y}(x \leq 3.0)?$	Correct?	Updated weights
1	1.0	1	0.1	1	Yes	0.072
2	2.0	1	0.1	1	Yes	0.072
3	3.0	1	0.1	1	Yes	0.072
4	4.0	-1	0.1	-1	Yes	0.072
5	5.0	-1	0.1	-1	Yes	0.072
6	6.0	-1	0.1	-1	Yes	0.072
7	7.0	1	0.1	-1	No	0.167
8	8.0	1	0.1	-1	No	0.167
9	9.0	1	0.1	-1	No	0.167
10	10.0	-1	0.1	-1	Yes	0.072

- Tính tỉ lệ lỗi của weak-learner:

$$\varepsilon_t = 0.1 \times 0 + 0.1 \times 0 + 0.1 \times 0 + 0.1 \times 0 + 0.1 \times 0 + 0.1 \times 0 + 0.1 \times 1 + 0.1 \times 1$$

- Tính trọng số của weak-learner:

$$\alpha_t = 0.5 \log \left(\frac{1 - \varepsilon}{\varepsilon} \right) \approx 0.424$$

Cập nhật trọng số cho các điểm dữ liệu:

- Những điểm được phân lớp đúng trọng số được cập nhật lại:

$$0.1 \times \exp(-0.424 \times 1 \times 1) \approx 0.065$$

- Những điểm được phân lớp sai trọng số được cập nhật lại:

$$0.1 \times \exp(-0.424 \times (-1) \times (1)) \approx 0.153$$

Sau cùng chúng ta sẽ chuẩn hóa vecto trọng số của bộ dữ liệu và ta được kết quả như bản trên:

▼ Applying AdaBoost using scikit-learn

```
from sklearn.ensemble import AdaBoostClassifier
tree = DecisionTreeClassifier(criterion='entropy',
                             random_state=1,
                             max_depth=1)
ada = AdaBoostClassifier(base_estimator=tree,
                         n_estimators=500,
                         learning_rate=0.1,
                         random_state=1)

tree = tree.fit(X_train, y_train)
y_train_pred = tree.predict(X_train)
y_test_pred = tree.predict(X_test)
tree_train = accuracy_score(y_train, y_train_pred)
tree_test = accuracy_score(y_test, y_test_pred)
print('Decision tree train/test accuracies %.3f/%.3f' % (tree_train, tree_test))
```

Decision tree train/test accuracies 0.916/0.875

Như chúng ta có thể thấy, gốc decision tree không dự đoán tốt như decision tree (decision tree được train với độ sâu của cây sâu hơn) được trình bày ở trong phần trình bày trước (bagging).

```
ada = ada.fit(X_train, y_train)
y_train_pred = ada.predict(X_train)
y_test_pred = ada.predict(X_test)
ada_train = accuracy_score(y_train, y_train_pred)
ada_test = accuracy_score(y_test, y_test_pred)
print('AdaBoost train/test accuracies %.3f/%.3f' % (ada_train, ada_test))
```

AdaBoost train/test accuracies 1.000/0.917

Model Adaboost dự đoán tốt trên tập train và cho hiệu suất trên test cũng tăng so với gốc decision tree. Cho thấy được việc kết hợp các weak learner và cải thiện những lỗi của weak learner trước đó. Kết hợp cho ra model có hiệu suất tốt hơn.

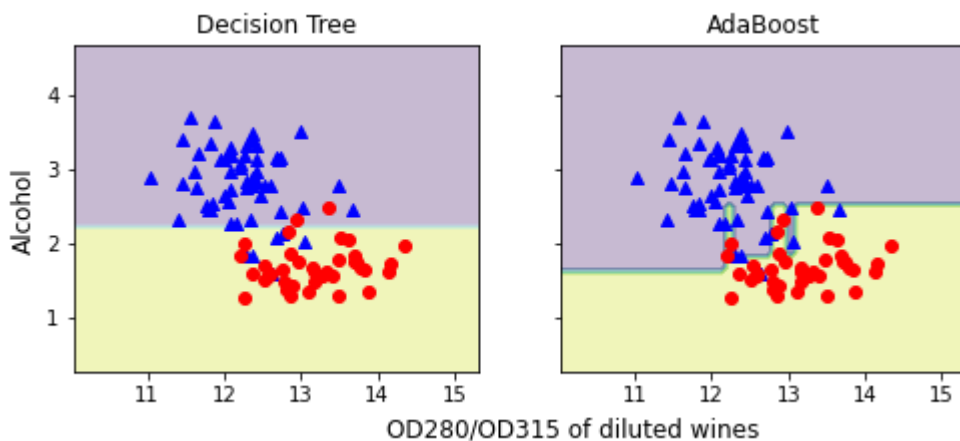
```
x_min = X_train[:, 0].min() - 1
x_max = X_train[:, 0].max() + 1
y_min = X_train[:, 1].min() - 1
y_max = X_train[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))
f, axarr = plt.subplots(1, 2,
                       sharex='col',
                       sharey='row',
                       figsize=(8, 3))

for idx, clf, tt in zip([0, 1],
```

```

[tree, ada],
['Decision Tree', 'AdaBoost']):
clf.fit(X_train, y_train)
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
axarr[idx].contourf(xx, yy, Z, alpha=0.3)
axarr[idx].scatter(X_train[y_train==0, 0],
                  X_train[y_train==0, 1],
                  c='blue',
                  marker='^')
axarr[idx].scatter(X_train[y_train==1, 0],
                  X_train[y_train==1, 1],
                  c='red',
                  marker='o')
axarr[idx].set_title(tt)
axarr[0].set_ylabel('Alcohol', fontsize=12)
plt.text(10.2, -0.5,
        s='OD280/OD315 of diluted wines',
        ha='center',
        va='center',
        fontsize=12)
plt.show()

```



Chúng ta có thể thấy được ranh giới giữa các vùng được tạo bởi Adaboost phức tạp hơn đáng kể so với ranh giới của gốc Decision Tree

▼ References

- <https://www.geeksforgeeks.org/bagging-vs-boosting-in-machine-learning/>
- <https://www.geeksforgeeks.org/ensemble-classifier-data-mining/>
- https://cse.buffalo.edu/~jcorso/t/CSE555/files/lecture_boosting.pdf
- <https://towardsdatascience.com/boosting-and-adaboost-clearly-explained-856e21152d3e>

- Adaboost: <https://viblo.asia/p/adaboost-buoc-di-dau-cua-boosting-gAm5yrGwKdb>

Các sản phẩm có tính phí của Colab - Huỷ hợp đồng tại đây

