

6강: 램덤포레스트, XGBoost,GBM의 이해

인공지능 일반강좌 : 기계학습의 이해(L2-1)

Contents

양상을 학습

램덤 포레스트 : 유방암 데이터

램덤 포레스트 : 사용자 행동 인식

AdaBoost 개요

그레디언트부스트 (GBM) 개요

XGBoost 소개

양상블 학습 개요(1)

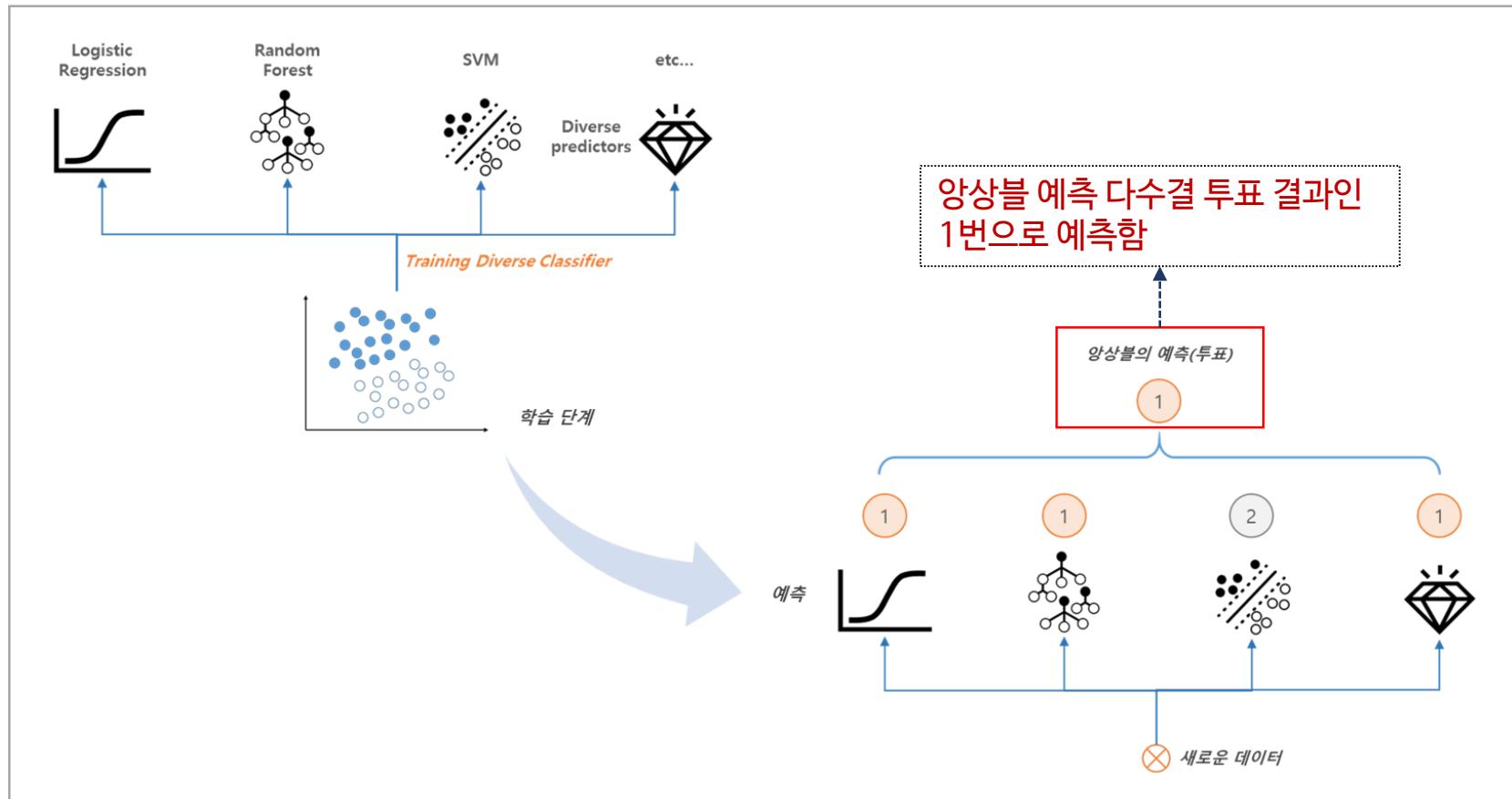
- 양상블 학습 개요
 - ✓ 여러 개의 분류기를 생성하고 그 예측을 결합. 보다 정확한 최종 예측을 도달함
 - 대중의 지혜(wisdom of the crowd)
 - 단일 분류기 보다 신뢰성이 높은 예측 값을 얻는 것이 핵심
 - ✓ 딥러닝이 뛰어난 분야는 비정형 데이터인 이미지, 영상, 음성 분야
 - ✓ 양상블이 뛰어난 분야는 정형 데이터 분류, 램덤포레스트와, 그레디언트 부스팅
 - ✓ 대표 알고리즘
 - XGboost, 훨씬 빠른 LightGBM, 메타 모델을 수립하는 스태킹(Stacking)
 - ✓ 양상블 학습은 일련의 예측기(분류, 회귀)로부터 예측을 수집

양상블 학습 개요(2)

- **양상블 학습 유형**
 - ✓ **보팅(Voting) : 여러 개의 분류기가 투표를 통해 최종 예측**
 - 서로 다른 알고리즘(knn, svm, kmeans)을 가진 분류기를 결합하는 방식
 - ✓ **배깅(Bagging) : 여러 개의 분류기가 투표를 통해 최종 예측**
 - 분류기는 같은 유형의 알고리즘이지만, 데이터 샘플링을 서로 다르게 학습
 - 대표적인 배깅 방식은 램덤 포레스트 알고리즘이다.
 - 부트스트래핑(Bootstrapping) 분할 방식을 사용, 즉, 원본 학습 데이터를 샘플링해서 추출하는 방식
 - 배깅은 교차 검증과 달리 데이터 셋트 간의 중첩을 허용하는 것이 차이점이다.
 - ✓ **부스팅(Boosting)**
 - 앞에서 학습한 분류기가 예측이 틀린 데이터에 대해서 올바른 예측을 할 수 있도록 다음 분류기에는 가중치(weight)를 부여하면서 학습과 예측을 진행하는 것
 - ✓ **스태킹(Stacking)**
 - 여러 가지 다른 모델의 예측 결과 값을 다시 학습 데이터로 만들어서 다른 모델(메타모델)로 재학습시켜 결과를 예측하는 방법

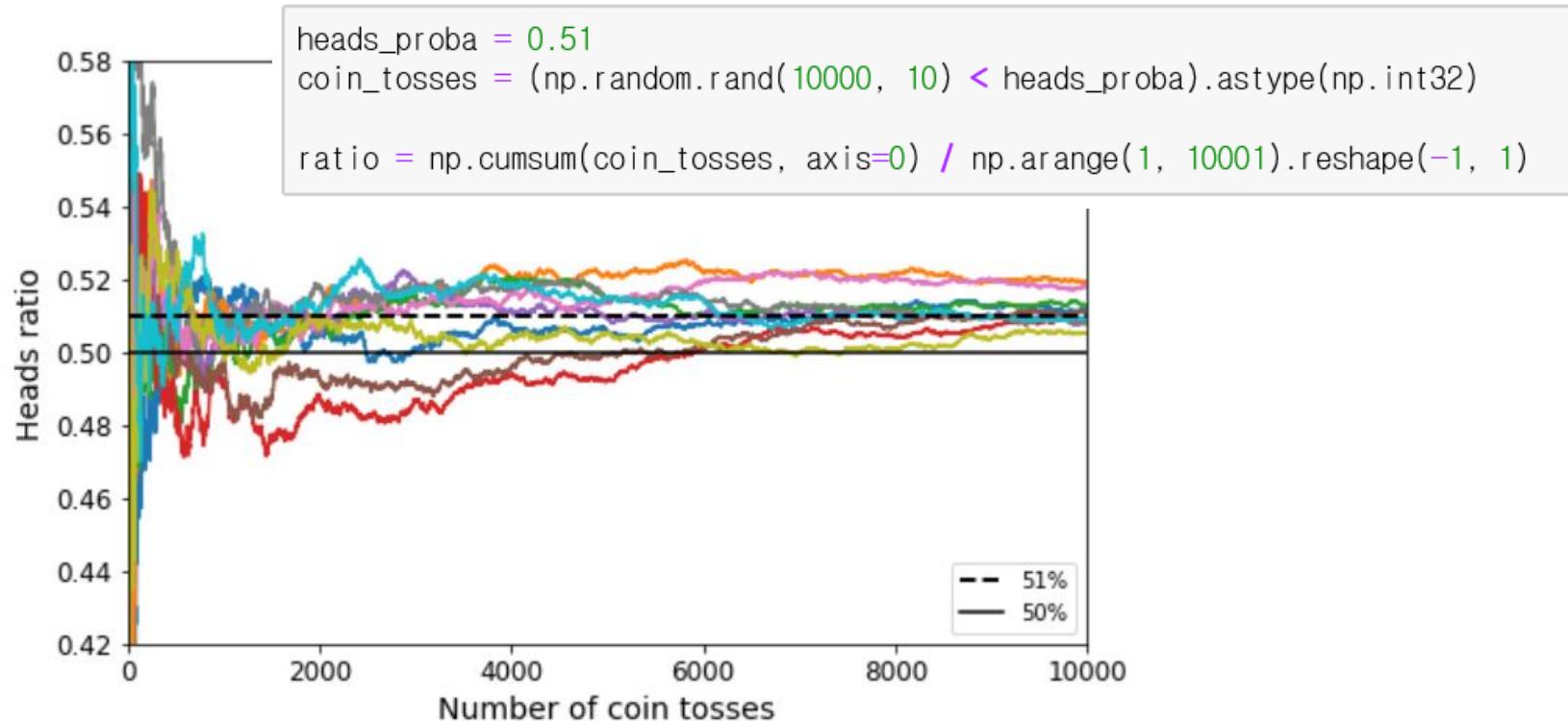
양상블 학습 개요(4)

다수결 투표 결과로 정해지는 직접 투표(Hard voting) 분류기



동전 확률 51% 데이터 분석 (1)

- 가정하자. 바이어스 된 동전
 - ✓ 앞면이 나올 확률 51%이고, 뒷면이 나올 확률은 49%
 - ✓ 1000번 던졌을때, 510은 앞면, 490은 뒷면을 얻을 것임.



(예제) 하드 보팅 (1)

```
from sklearn.model_selection import train_test_split  
from sklearn.datasets import make_moons
```

moons 데이터셋에 앙상블 학습 적용하기

```
X, y = make_moons(n_samples=500, noise=0.30, random_state=42)  
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.ensemble import VotingClassifier
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.svm import SVC
```

```
log_clf = LogisticRegression(solver="lbfgs", random_state=42)
```

```
rnd_clf = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
svm_clf = SVC(gamma="scale", random_state=42)
```

```
voting_clf = VotingClassifier(  
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svm', svm_clf)],  
    voting='hard')
```

디폴트

보팅 선택: '하드'

(예제) 하드 보팅 (2)

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

```
VotingClassifier(estimators=[('lr',
                             LogisticRegression(C=1.0, class_weight=None,
                                                dual=False, fit_intercept=True,
                                                intercept_scaling=1,
                                                l1_ratio=None, max_iter=100,
                                                multi_class='auto',
                                                n_jobs=None, penalty='l2',
                                                random_state=42,
                                                solver='lbfgs', tol=0.0001,
                                                verbose=0, warm_start=False)),
                           ('rf',
                             RandomForestClassifier(bootstrap=True,
                                                   ccp_alpha=0.0,
                                                   class_weight=None,
                                                   crit...
                                                   oob_score=False,
                                                   random_state=42, verbose=0,
                                                   warm_start=False)),
                           ('svm',
                             SVC(C=1.0, break_ties=False, cache_size=200,
                                 class_weight=None, coef0=0.0,
                                 decision_function_shape='ovr', degree=3,
                                 gamma='scale', kernel='rbf', max_iter=-1,
                                 probability=False, random_state=42,
                                 shrinking=True, tol=0.001, verbose=False))],
                           flatten_transform=True, n_jobs=None, voting='hard',
                           weights=None)
```

(예제) 하드 보팅 (3)

```
from sklearn.metrics import accuracy_score

for clf in (log_clf, rnd_clf, svm_clf, voting_clf):

    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_test)

    print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
```

LogisticRegression 0.864
RandomForestClassifier 0.896
SVC 0.896
VotingClassifier 0.912

개별 분류기 성능 정확도

하드 보팅 정확도가 가장 좋다. 91.2%

(예제) 소프트 보팅 (4)

소프트 보팅 선택

```
log_clf = LogisticRegression(solver="lbfgs", random_state=42)
rnd_clf = RandomForestClassifier(n_estimators=100, random_state=42)
svm_clf = SVC(gamma="scale", probability=True, random_state=42)

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svm', svm_clf)], voting='soft')

voting_clf.fit(X_train, y_train)
```

```
from sklearn.metrics import accuracy_score

for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
```

LogisticRegression 0.864

RandomForestClassifier 0.896

SVC 0.896

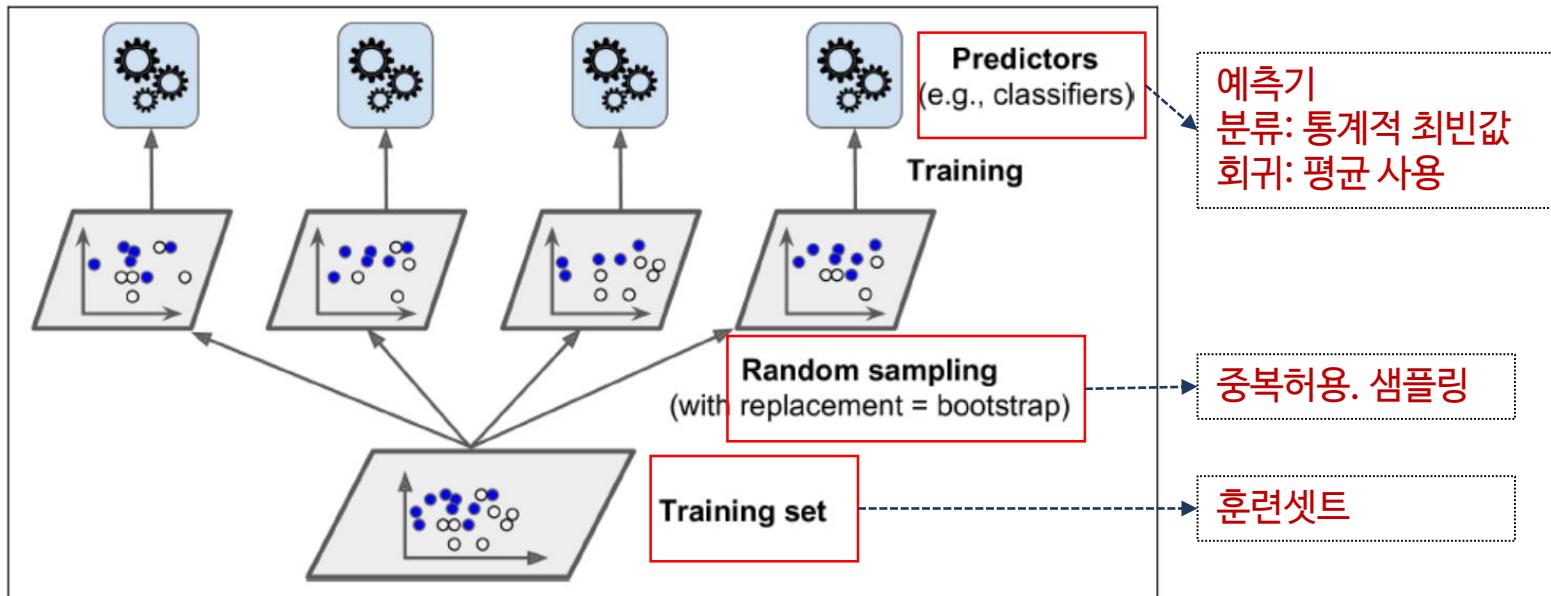
VotingClassifier 0.92

소프트 보팅의 정확도는 92%!

배깅과 페이스팅 (1)

- 같은 알고리즘을 사용하지만 훈련 세트의 서브셋을 무작위로
 - ✓ 배깅(bagging) : 훈련 세트에서 중복을 허용하는 샘플링
 - Bootstrap aggregating의 줄임말로, 부트스트래핑은 중복 허용 리샘플링 방식.
 - ✓ 페이스팅(pasting) : 훈련 세트에서 중복을 허용하지 않음

양상블의 결과는 원본 데이터셋으로 하나의 예측기보다 분산은 줄어듬.



배깅과 페이스팅 (2)

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(random_state=42), n_estimators=500,
    max_samples=100, bootstrap=True, random_state=42)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

분류기
양상별 개수
배깅. 중복허용 무작위 100개 샘플 선택함.
페이스팅은 bootstrap=False로 선택한다.

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

0.904

```
tree_clf = DecisionTreeClassifier(random_state=42)
tree_clf.fit(X_train, y_train)
y_pred_tree = tree_clf.predict(X_test)
print(accuracy_score(y_test, y_pred_tree))
```

0.856

배경과 페이스팅 (3)

```
from matplotlib.colors import ListedColormap

def plot_decision_boundary(clf, X, y, axes=[-1.5, 2.45, -1, 1.5], alpha=0.5, contour=True):
    x1s = np.linspace(axes[0], axes[1], 100)
    x2s = np.linspace(axes[2], axes[3], 100)
    x1, x2 = np.meshgrid(x1s, x2s)
    X_new = np.c_[x1.ravel(), x2.ravel()]
    y_pred = clf.predict(X_new).reshape(x1.shape)
    custom_cmap = ListedColormap(['#fafab0', '#9898ff', '#a0faa0'])
    plt.contourf(x1, x2, y_pred, alpha=0.12, cmap=custom_cmap)
    if contour:
        custom_cmap2 = ListedColormap(['#7d7d58', '#4c4c7f', '#507d50'])
        plt.contour(x1, x2, y_pred, cmap=custom_cmap2, alpha=0.8)
    plt.plot(X[:, 0][y==0], X[:, 1][y==0], "yo", alpha=alpha)
    plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs", alpha=alpha)
    plt.axis(axes)
    plt.xlabel(r"$x_1$", fontsize=18)
    plt.ylabel(r"$x_2$", fontsize=18, rotation=0)
```

배깅과 페이스팅 (4)

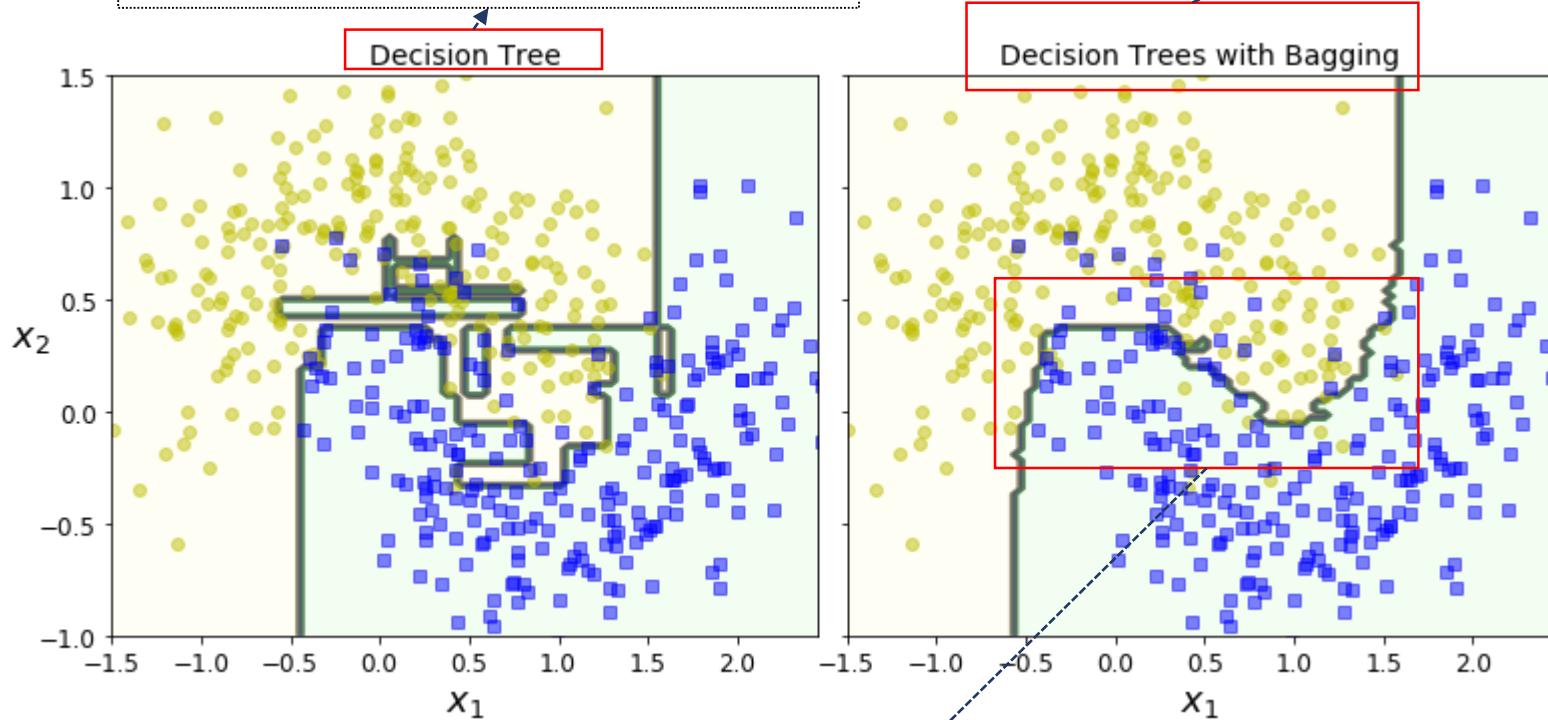
```
fix, axes = plt.subplots(ncols=2, figsize=(11,5), sharey=True)
plt.sca(axes[0])
plot_decision_boundary(tree_clf, X, y)
plt.title("Decision Tree", fontsize=14)
plt.sca(axes[1])
plot_decision_boundary(bag_clf, X, y)
plt.title("Decision Trees with Bagging", fontsize=14)
plt.ylabel("")

plt.show()
```

배깅과 페이스팅 (5)

단일 결정트리 경계. 500개 moos 데이터
85.6% 정확도

500개의 트리를 사용한 배깅 양상블. 90.4%



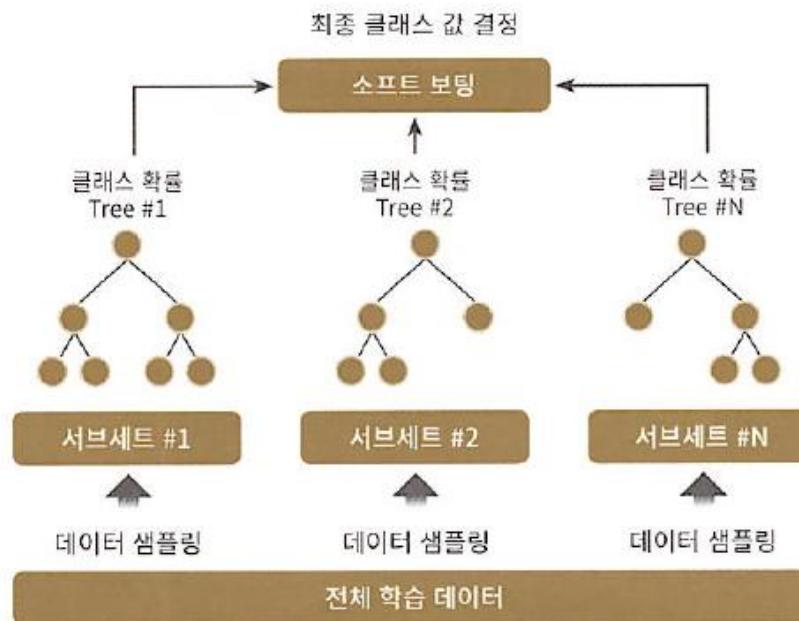
양상블 예측이 결정트리 하나의 예측보다 일반화가 훨씬 잘되었음. 양상블은 비슷한 평향에서 더 작은
분산을 만듬. (즉, 훈련 셋트의 오차 수가 거의 비슷하지만 결정 경계는 덜 불규칙하다)

배깅과 페이스팅 (6)

- 평가 oob (out of bag)
 - ✓ 베깅 분류기가 훈련 세트 크기인 m 개를 샘플할 경우, 보통 63%만 샘플링됨
 - ✓ 나머지 37% 데이터는 훈련에 사용이 안됨
 - ✓ 그래서 나머지 데이터를 검증/테스트에 사용하면 됨.
 - ✓ 즉, `oob_score=True`로 설정하여 oob를 평가함
- 램덤 패치와 램덤 서브스페이스
 - ✓ 고차원의 이미지 데이터 셋 다룰 때, `BaggingClassifier`는
 - ✓ `Max_features`, `bootstrap_feature`를 제공.
 - ✓ 특성에 대한 샘플링 방식.

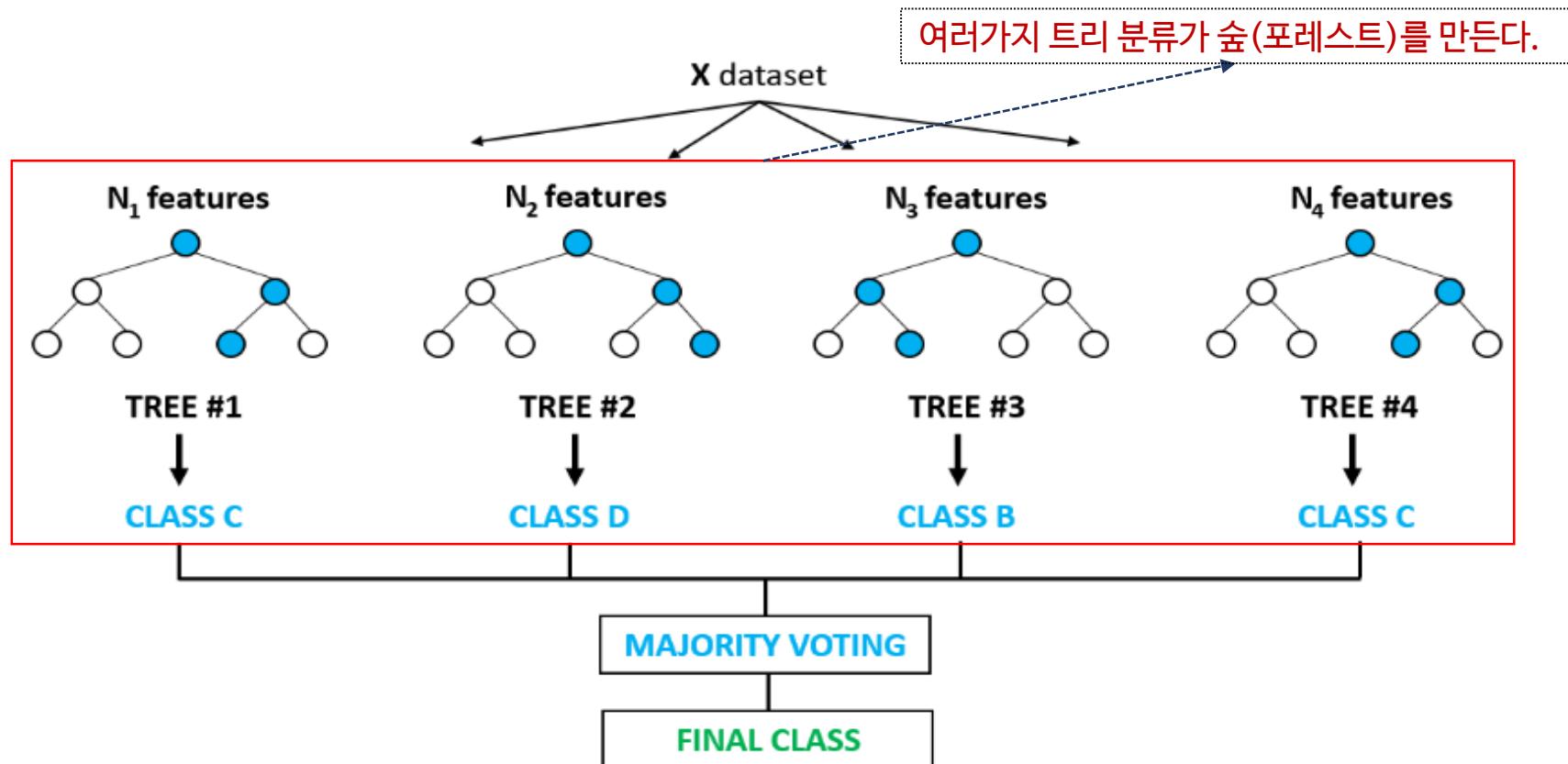
랜덤 포레스트 (1)

- 랜덤 포레스트 개요
 - ✓ 배깅의 대표적 알고리즘은 랜덤 포레스트
 - ✓ 결정 트리는 대표적인 랜덤 포레스트의 기반 알고리즘
 - ✓ 여러 개의 결정 트리 분류기가 전체 데이터에서 배깅 방식으로 보팅 함



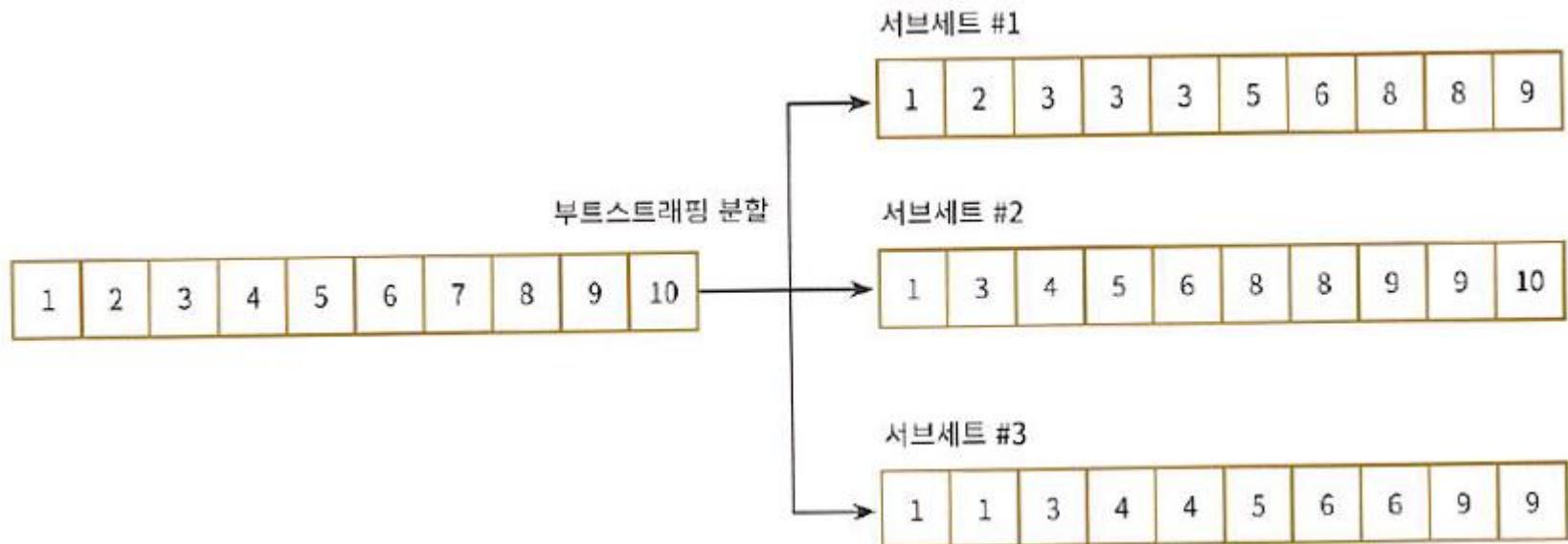
랜덤 포레스트 (2)

- ✓ 결정트리의 양상을
- ✓ 훈련 세트로부터 무작위로 각기 다른 서브셋을 만들어 일련의 결정 트리 분류기를 훈련시킴.
- ✓ 예측을 하려면 모든 개별 트리의 예측을 구하면 됨



램덤 포레스트 (3)

- 부트스트래핑(bootstrapping) 분할 방식
 - 여러 개의 데이터 세트를 중첩되게 분리하는 것



랜덤 포레스트 분류 : 붓꽃 (1)

```
bag_clf = BaggingClassifier(  
    DecisionTreeClassifier(splitter="random", max_leaf_nodes=16, random_state=42),  
    n_estimators=500, max_samples=1.0, bootstrap=True, random_state=42)
```

배깅 적용한 결정트리

리프노드는 16개

```
bag_clf.fit(X_train, y_train)  
y_pred = bag_clf.predict(X_test)
```

트리를 다양하게 만들고, 편향을 손해보는
대신 분산을 낮추어 훌륭한 모델 만듬

```
from sklearn.ensemble import RandomForestClassifier  
  
rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, random_state=42)  
rnd_clf.fit(X_train, y_train)  
  
y_pred_rf = rnd_clf.predict(X_test)
```

랜덤 포레스트는 트리의 노드를 분할할때 전체 특성 중에서 최선의
특성을 찾는 대신 무작위로 선택한 후보에서 최적의 특성을 찾는 방식

```
np.sum(y_pred == y_pred_rf) / len(y_pred) # almost identical predictions
```

0.976

랜덤 포레스트 분류 : 붓꽃 (2)

```
from sklearn.datasets import load_iris
iris = load_iris()
rnd_clf = RandomForestClassifier(n_estimators=500, random_state=42)
rnd_clf.fit(iris["data"], iris["target"])
for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
    print(name, score)
```

sepal length (cm) 0.11249225099876375
sepal width (cm) 0.02311928828251033
petal length (cm) 0.4410304643639577
petal width (cm) 0.4233579963547682

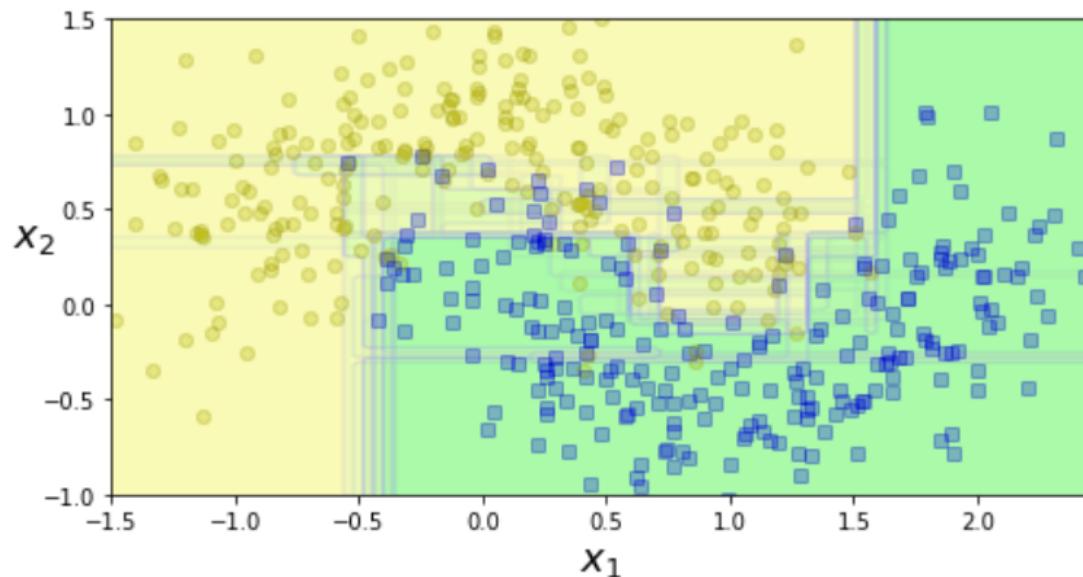
```
rnd_clf.feature_importances_
```

array([0.11249225, 0.02311929, 0.44103046, 0.423358])

랜덤 포레스트 분류 : 붓꽃 (3)

```
plt.figure(figsize=(8, 4))
for i in range(15):
    tree_clf = DecisionTreeClassifier(max_leaf_nodes=16, random_state=42 + i)
    indices_with_replacement = np.random.randint(0, len(X_train), len(X_train))
    tree_clf.fit(X[indices_with_replacement], y[indices_with_replacement])
    plot_decision_boundary(tree_clf, X, y, axes=[-1.5, 2.45, -1, 1.5], alpha=0.02, contour=False)

plt.show()
```



랜덤 포레스트 회귀 (1)

랜덤포레스트 회귀 IRIS

```
df=pd.read_csv('./input/iris.csv')
```

```
df.head(3)
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa

```
df.drop(["Id"],axis=1,inplace=True)
```

```
x=df.SepalLengthCm.values.reshape(-1,1)
y=df.PetalLengthCm.values.reshape(-1,1)
```

랜덤 포레스트 회귀 (2)

```
from sklearn.ensemble import RandomForestRegressor  
rf=RandomForestRegressor(n_estimators=100,random_state=42)  
rf.fit(x,y)
```

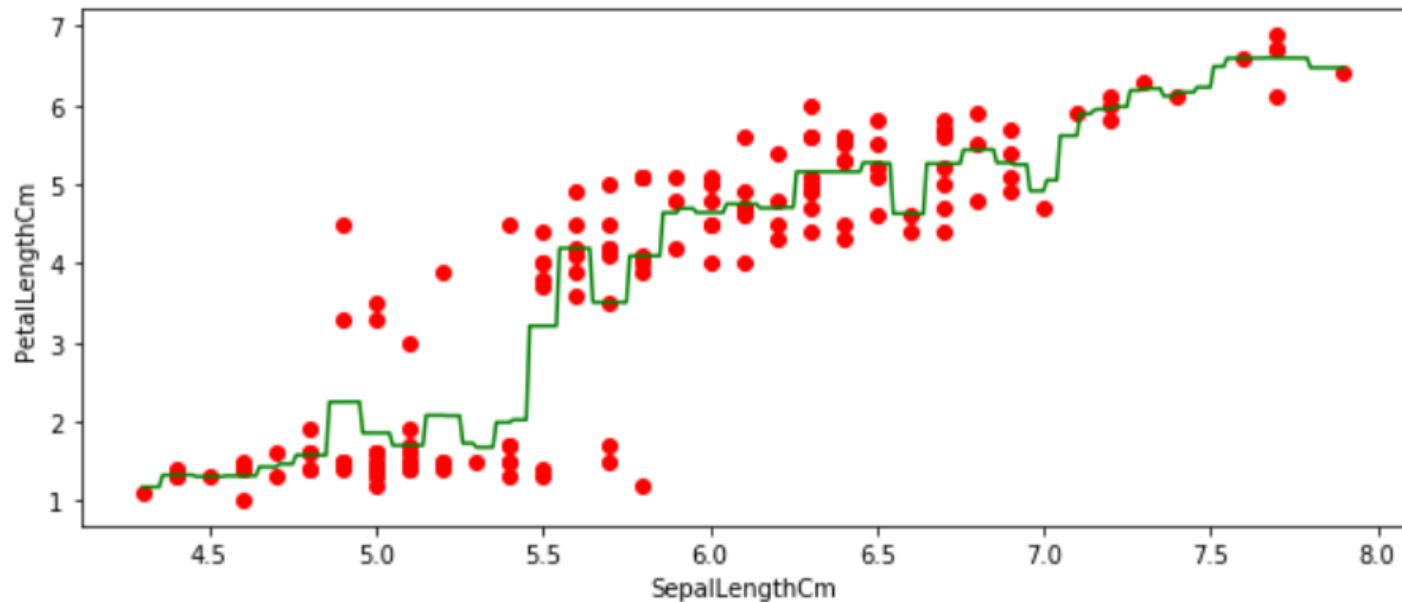
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',  
                      max_depth=None, max_features='auto', max_leaf_nodes=None,  
                      max_samples=None, min_impurity_decrease=0.0,  
                      min_impurity_split=None, min_samples_leaf=1,  
                      min_samples_split=2, min_weight_fraction_leaf=0.0,  
                      n_estimators=100, n_jobs=None, oob_score=False,  
                      random_state=42, verbose=0, warm_start=False)
```

```
x_=np.arange(min(x),max(x),0.01).reshape(-1,1)
```

```
y_head=rf.predict(x_)
```

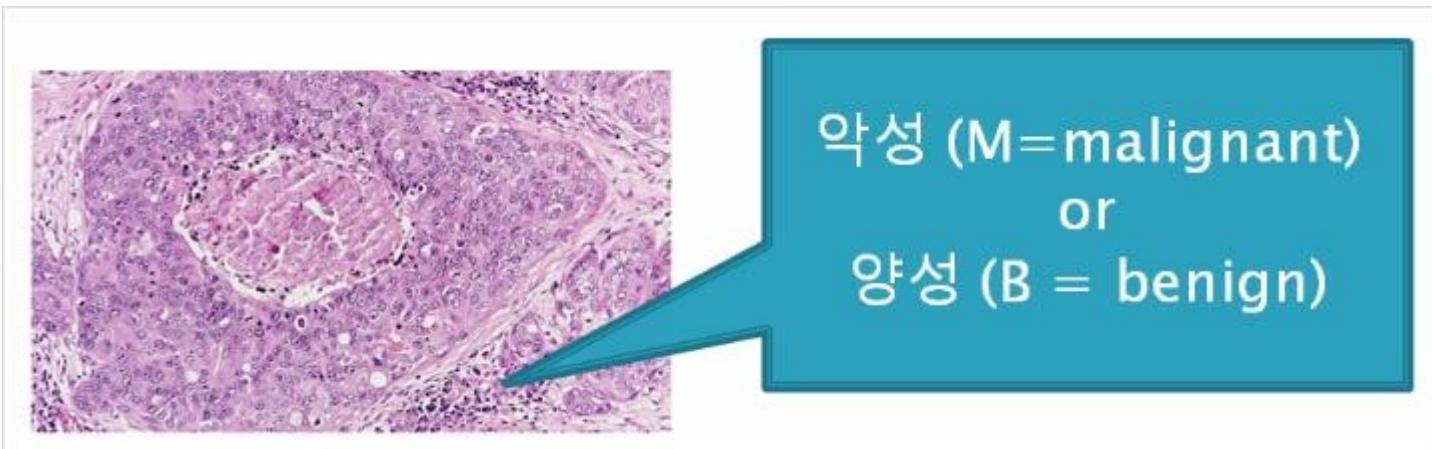
램덤 포레스트 회귀 (3)

```
plt.figure(figsize=(10,4))
plt.scatter(x,y,color="r")
plt.plot(x_,y_head,color="g")
plt.xlabel("SepalLengthCm")
plt.ylabel("PetalLengthCm")
plt.show()
```



위스콘신 유방암 데이터(1)

- WDBC(Wisconsin Diagnostic Breast Cancer) Dataset
 - ✓ 미국 위스콘신 주 Wisconsin 대학병원에서 1995년에 공개한 자료
 - ✓ 가는 주사바늘을 사용하여 낭종(주모니 혹은 내용물이나 멍울)의 세포나 조직을 떼어내서 조직검사한 이미지를 디지털로 변환
 - ✓ 측정한 30개의 설명변수와, 환자 ID, 진단 결과(악성 M=malignant, 양성 B=benign)의 변수를 포함하여
 - ✓ 총 32개의 변수로 구성되어 있고, 569명의 환자에 대해 조사한 데이터셋



악성 (M=malignant)
or
양성 (B = benign)

위스콘신 유방암 데이터(2)

- 할 일
 - ✓ 사이킷런에는 `load_breast_cancer()` 제공
 - ✓ 로지스틱 회귀(Regression)와 KNN으로 보팅 분류기를 만들자

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity
0	17.99	10.38	122.8	1001.0	0.11840	0.27760	0.3001
1	20.57	17.77	132.9	1326.0	0.08474	0.07864	0.0869
2	19.69	21.25	130.0	1203.0	0.10960	0.15990	0.1974

3 rows × 30 columns

위스콘신 유방암 데이터(3)

```
import pandas as pd
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

cancer = load_breast_cancer()

df_cancer = pd.DataFrame(cancer.data, columns=cancer.feature_names)
df_cancer.head(3)
```

위스콘신 유방암 데이터(4)

sklearn.ensemble.VotingClassifier

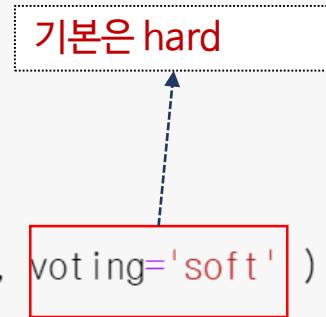
```
class sklearn.ensemble.VotingClassifier(estimators, voting='hard', weights=None,  
n_jobs=None, flatten_transform=True)
```

Soft Voting/Majority Rule classifier for unfitted estimators.

- `estimators` : 개별 모형 목록, 리스트나 named parameter 형식으로 입력
- `voting` : 문자열 { `hard` , `soft` } hard voting 과 soft voting 선택. 디폴트는 `hard`
- `weights` : 사용자 가중치 리스트

위스콘신 유방암 데이터(5)

```
# 개별 모델은 로지스틱 회귀와 KNN 입.  
lr_clf = LogisticRegression()  
  
knn_clf = KNeighborsClassifier(n_neighbors=8)  
  
# 개별 모델을 소프트 보팅 기반의 앙상블 모델로 구현한 분류기  
vo_clf = VotingClassifier( estimators=[('LR',lr_clf),('KNN',knn_clf)] , voting='soft' )  
  
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,  
test_size=0.2 , random_state= 156)  
  
# VotingClassifier 학습/예측/평가.  
vo_clf.fit(X_train , y_train)  
  
pred = vo_clf.predict(X_test)  
  
print('Voting 분류기 정확도: {:.4f}'.format(accuracy_score(y_test , pred)))
```



위스콘신 유방암 데이터(6)

```
print('Voting 분류기 정확도: {:.4f}'.format(accuracy_score(y_test , pred)))  
  
# 개별 모델의 학습/예측/평가.  
classifiers = [lr_clf, knn_clf]  
  
for classifier in classifiers:  
    classifier.fit(X_train , y_train)  
    pred = classifier.predict(X_test)  
    class_name= classifier.__class__.__name__  
    print ('{} 정확도: {:.4f}'.format(class_name, accuracy_score(y_test , pred)))
```

Voting 분류기 정확도: 0.9386
LogisticRegression 정확도: 0.9386
KNeighborsClassifier 정확도: 0.9386

개별 모델의 학습/예측/평가.
classifiers = [lr_clf, knn_clf]

2개 별도의 모델분류기로 하드 투표. 혹은 소프트 투표를 확인해라.

하드 보팅의 경우에 해당.
소프트 보팅이면 분류기의 정확도는 얼마인가?

사용자 행동 인식 데이터 셋트(1)

<https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones>



Human Activity Recognition Using Smartphones Data Set

Download: [Data Folder](#), [Data Set Description](#)

데이터 구조
.input/human

Abstract: Human Activity Recognition database built from the recordings of 30 subjects performing activities of daily living (ADL) while c

Data Set Characteristics:	Multivariate, Time-Series	Number of Instances:	10299	Area:	Computer
Attribute Characteristics:	N/A	Number of Attributes:	561	Date Donated	2012-12-10
Associated Tasks:	Classification, Clustering	Missing Values?	N/A	Number of Web Hits:	933198

The image displays a screenshot of a web browser showing the contents of the 'UCI HAR Dataset' folder. The URL in the address bar is 'PC > 다운로드 > UCI HAR Dataset > UCI HAR'. The page lists several files and folders:

- Top-level folder: _MACOSX (highlighted with a blue selection bar)
- Sub-folders: test, train
- Text files: .DS_Store, activity_labels.txt, features.txt, features_info.txt, README.txt
- Text files under 'Inertial Signals': subject_train.txt, X_train.txt, y_train.txt

사용자 행동 인식 데이터 셋트(2)

- 데이터 특성
 - ✓ Human Activity Recognition Using Smartphones Dataset
 - ✓ 30명 참가자 (19~48세)
 - ✓ 삼성 스마트폰(Galaxy S)를 차고 6가지 행동
 - WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, LAYING
 - 3축 선형 가속도와 3축으로 각속도를 50Hz 주기로 측정
 - 비디오로 녹화하고, 70%는 훈련용 30%는 테스트 용으로 사용
 - ✓ 피쳐는 561개이며, 공백으로 구분

- 'features_info.txt': Shows information about the variables used on the feature vector.
- 'features.txt': List of all features.
- 'activity_labels.txt': Links the class labels with their activity name.
- 'train/X_train.txt': Training set.
- 'train/y_train.txt': Training labels.
- 'test/X_test.txt': Test set.
- 'test/y_test.txt': Test labels.

사용자 행동 인식 데이터 셋트(3)

1. 램덤 포레스트를 적용하여 분류를 해보자. (물론 결정 트리도 가능. 숙제)

```
import pandas as pd

def get_human_dataset( ):

    # 각 데이터 파일들은 공백으로 분리되어 있으므로 read_csv에서 공백 문자를 sep으로 할당.

    feature_name_df = pd.read_csv('./input/human/features.txt',sep='\t',
                                  header=None,names=['column_index','column_name'])
    |
    # DataFrame에 피처명을 컬럼으로 부여하기 위해 리스트 객체로 다시 변환
    feature_name = feature_name_df.iloc[:, 1].values.tolist()

    # 학습 피처 데이터 셋과 테스트 피처 데이터를 DataFrame으로 로딩. 컬럼명은 feature_name 적용
    X_train = pd.read_csv('./input/human/train/X_train.txt',sep='\t', names=feature_name)
    X_test = pd.read_csv('./input/human/test/X_test.txt',sep='\t', names=feature_name)

    # 학습 레이블과 테스트 레이블 데이터를 DataFrame으로 로딩하고 컬럼명은 action으로 부여
    y_train = pd.read_csv('./input/human/train/y_train.txt',sep='\t',header=None,names=['action'])
    y_test = pd.read_csv('./input/human/test/y_test.txt',sep='\t',header=None,names=['action'])

    # 로드된 학습/테스트용 DataFrame을 모두 반환
    return X_train, X_test, y_train, y_test

X_train, X_test, y_train, y_test = get_human_dataset()
```

사용자 행동 인식 데이터 셋트(4)

```
from sklearn.ensemble import RandomForestClassifier  
  
from sklearn.metrics import accuracy_score  
import pandas as pd  
  
# 결정 트리에서 사용한 get_human_dataset()을 이용해 학습/테스트용 DataFrame 반환  
X_train, X_test, y_train, y_test = get_human_dataset()  
  
# 랜덤 포레스트 학습 및 별도의 테스트 셋으로 예측 성능 평가  
  
rf_clf = RandomForestClassifier(random_state=0)  
  
rf_clf.fit(X_train, y_train)  
  
pred = rf_clf.predict(X_test)  
  
accuracy = accuracy_score(y_test, pred)  
  
print('랜덤 포레스트 정확도: {:.4f}'.format(accuracy))
```

랜덤 포레스트 정확도: 0.9253

사용자 행동 인식 데이터 셋트(5)

```
from sklearn.model_selection import GridSearchCV

params = {
    'n_estimators': [100],          ← 랜덤 포레스트에서 결정 트리의 개수. 디폴트는 10개
    'max_depth' : [6, 8, 10, 12],
    'min_samples_leaf' : [8, 12, 18 ],
    'min_samples_split' : [8, 16, 20]
}

# RandomForestClassifier 객체 생성 후 GridSearchCV 수행
rf_clf = RandomForestClassifier(random_state=0, n_jobs=-1)          ← 모든 CPU를 이용하여 병렬처리
grid_cv = GridSearchCV(rf_clf , param_grid=params , cv=2, n_jobs=-1 )
grid_cv.fit(X_train , y_train)
print('최적 하이퍼 파라미터:\n', grid_cv.best_params_)
print('최고 예측 정확도: {:.4f}'.format(grid_cv.best_score_))
```

최적 하이퍼 파라미터:

```
{'max_depth': 10, 'min_samples_leaf': 8, 'min_samples_split': 8, 'n_estimators': 100}
최고 예측 정확도: 0.9180
```

얻은 최적의 하이퍼 파라미터, 예측 정확도는 91.8%

사용자 행동 인식 데이터 셋트(6)

```
rf_clf1 = RandomForestClassifier(n_estimators=300,  
                                 max_depth=10,  
                                 min_samples_leaf=8,  
                                 min_samples_split=8,  
                                 random_state=0)  
  
rf_clf1.fit(X_train, y_train)  
  
pred = rf_clf1.predict(X_test)  
  
print('예측 정확도: {:.4f}'.format(accuracy_score(y_test, pred)))
```

예측 정확도: 0.9165

최적화된 하이퍼 파라미터를 사용할 경우의 예측 정확도는 91.65%

사용자 행동 인식 데이터 셋트(7)

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

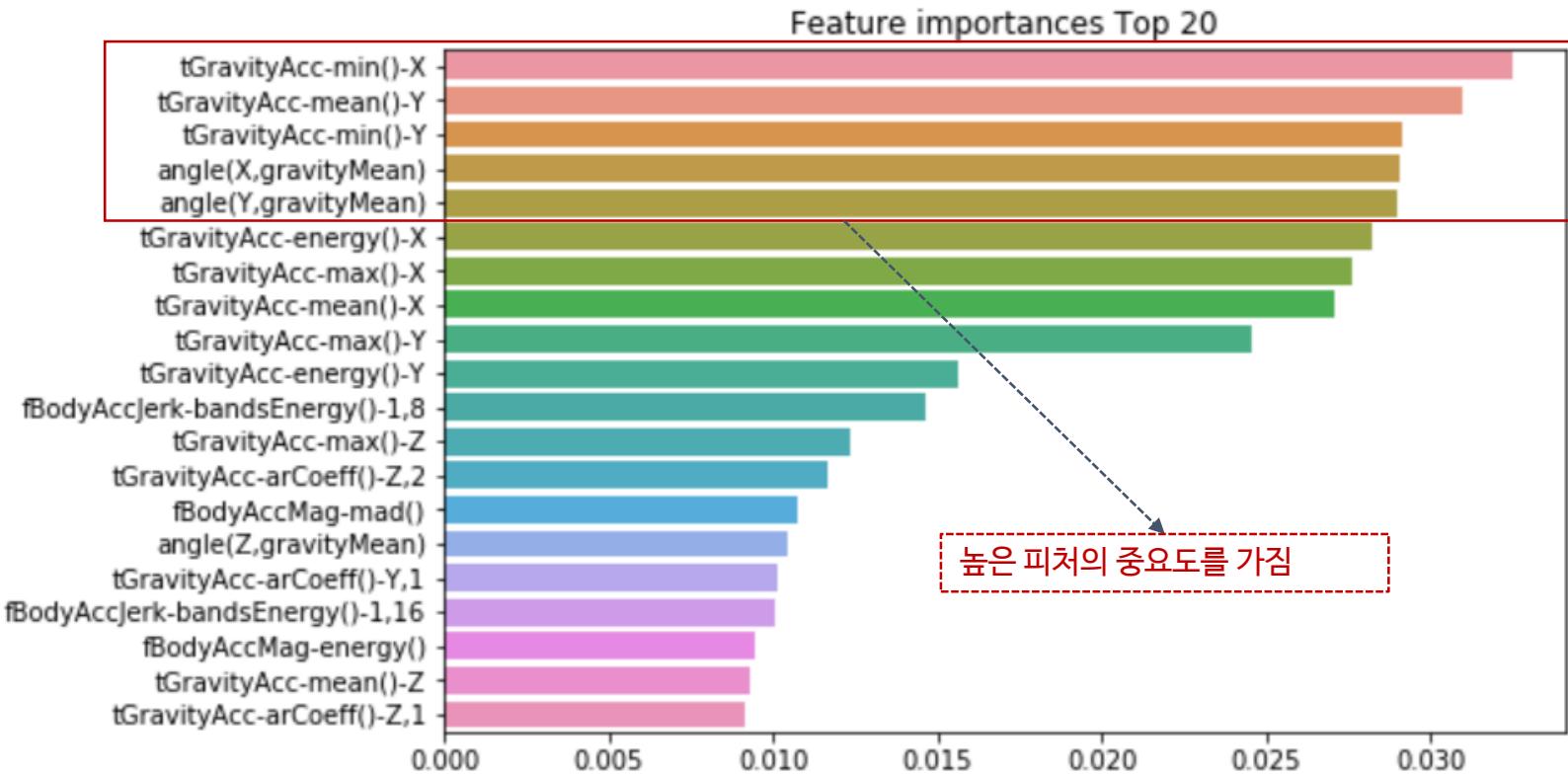
ftr_importances_values = rf_clf1.feature_importances_

ftr_importances = pd.Series(ftr_importances_values, index=X_train.columns )
|
ftr_top20 = ftr_importances.sort_values(ascending=False)[:20]

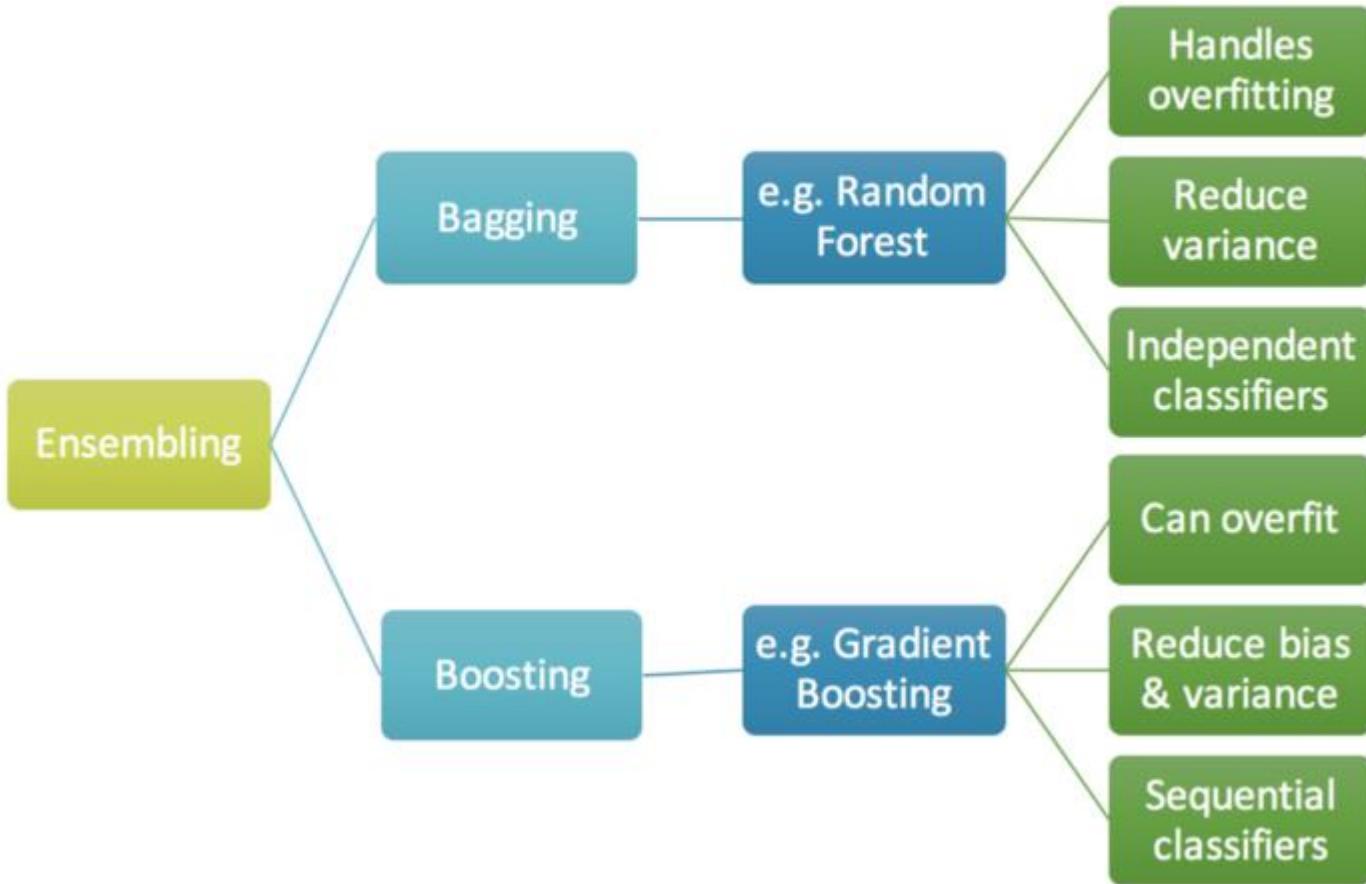
plt.figure(figsize=(12,8))
plt.title('Feature importances Top 20')

sns.barplot(x=ftr_top20 , y = ftr_top20.index)
plt.show()
```

사용자 행동 인식 데이터 셋트(8)

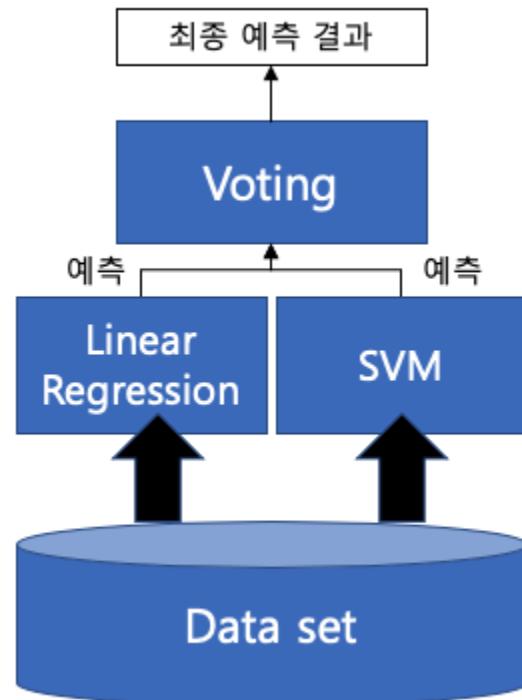


양상별 학습



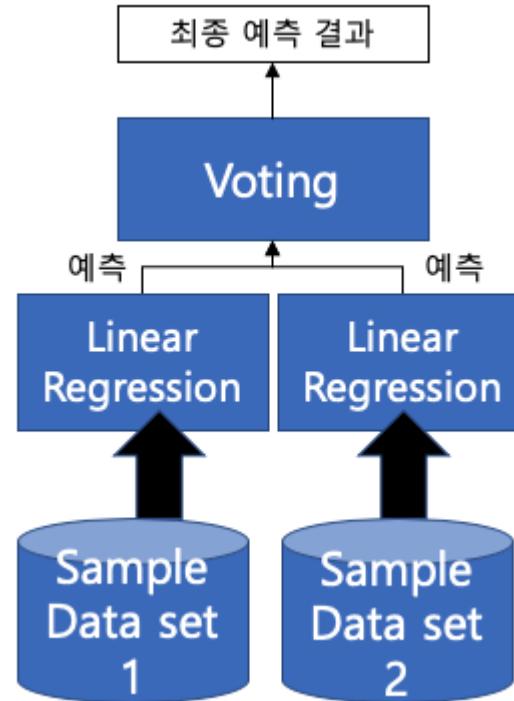
양상별 학습 -보팅

Voting: 다른 ML 알고리즘이 같은 데이터 세트에 대해 학습하고 예측한 결과를 가지고 보팅을 통해 최종 예측 결과 선정



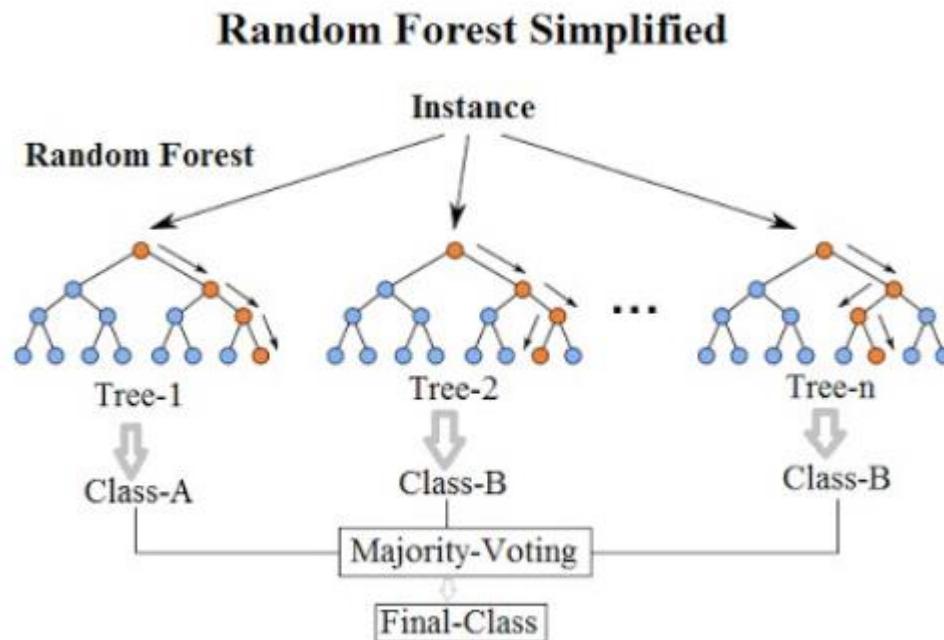
베깅

Bagging: 단일 ML 알고리즘이 Bootstrapping 방식으로 샘플링된 데이터 세트에 대해서 학습을 통해 개별적인 예측을 수행한 결과를 보팅을 통해 최종 예측 결과 선정



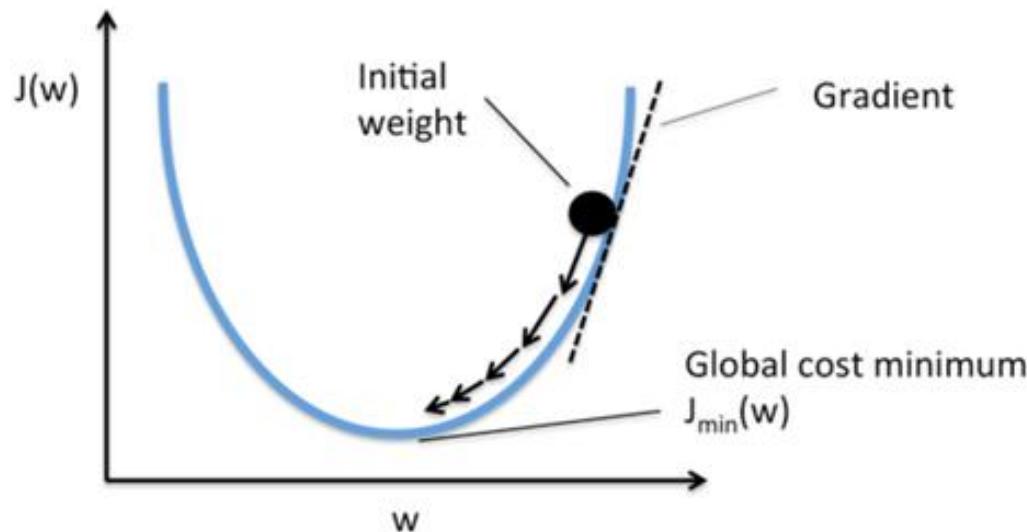
RandomForest

- Bagging의 대표적인 알고리즘 Random Forest;
- 비교적 빠른 수행 속도 + 높은 예측 성능



부스팅 알고리즘

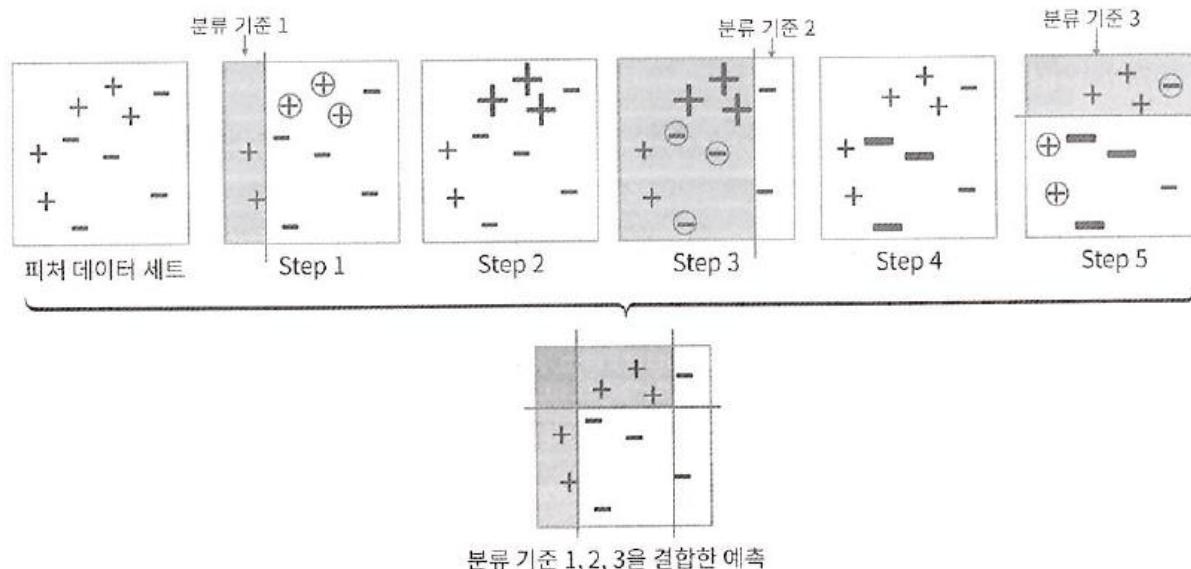
- AdaBoost와 GBM(Gradient Boost Machine)이 있다.
 - ✓ AdaBoost: 오류 데이터에 가중치를 부여하면서 부스팅을 수행하는 알고리즘
 - ✓ GBM : 가중치 업데이트를 경사 하강법을 이용하는 것이 큰 차이다.
 - 에이다 부스트, GBM 과의 차이: GBM은 가중치 업데이트를 경사하강법을 이용한다.



에이다부스팅(AdaBoosting) (1)

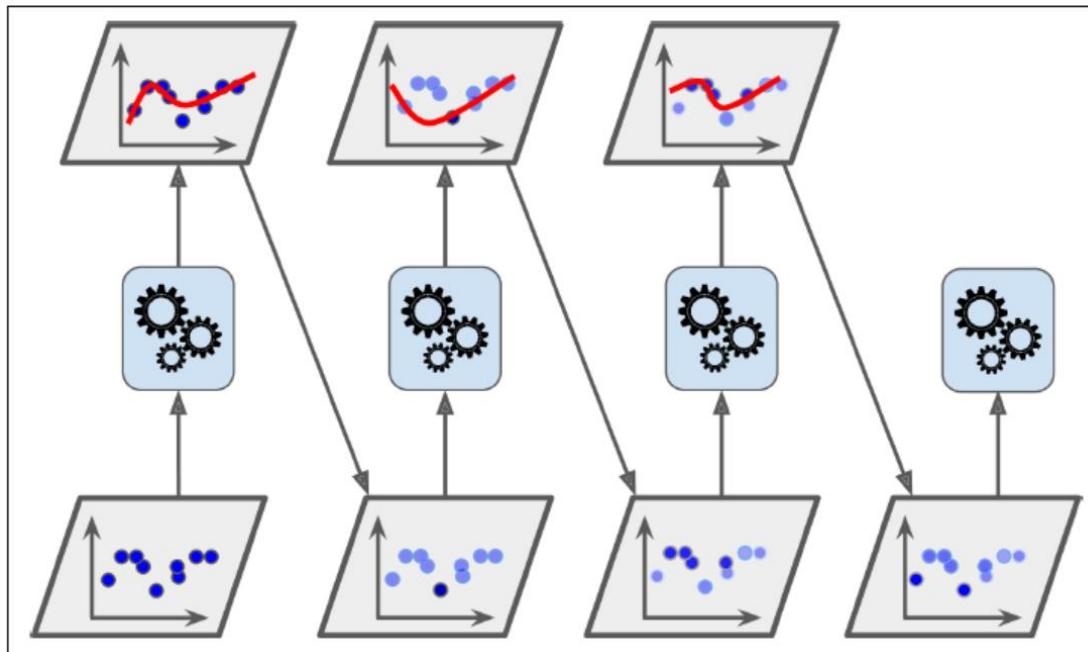
- 부스팅 알고리즘

- 여러 개의 약한 학습기(learner)를 순차적으로 학습-예측하면서 잘못 예측한 데 이터에 가중치 부여를 통해 오류를 개선하는 방식
- 에이다부스팅(Adaboosting)은 아래 그림처럼 오류 데이터에 가중치 부여 방식
- 그레디언트 부스팅(GBM)은 가중치 업데이트시 경사 하강법을 사용함



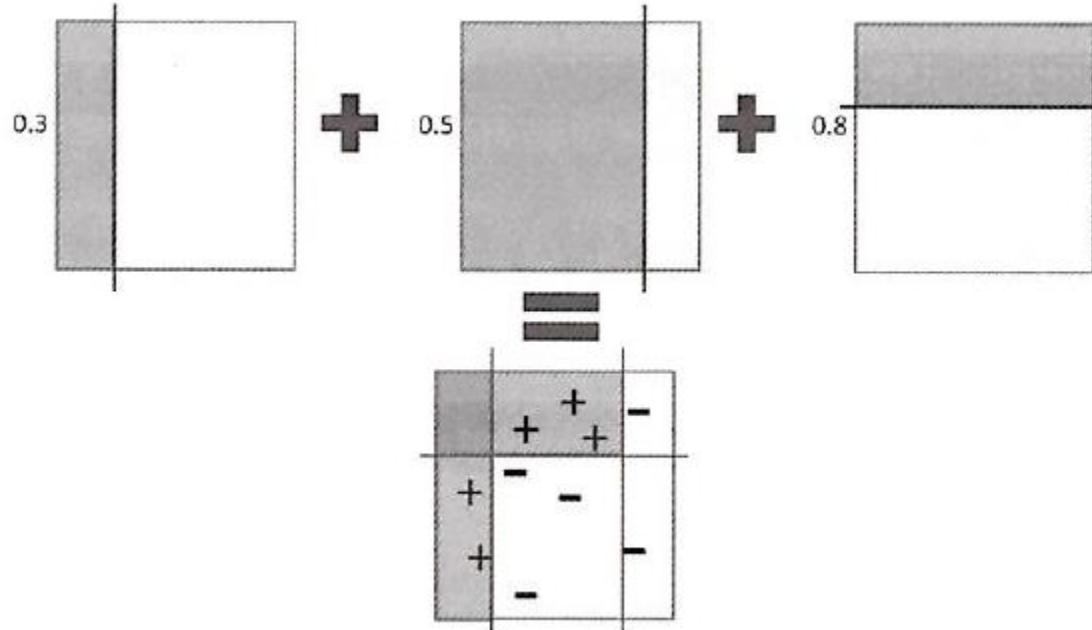
에이다부스팅(AdaBoosting) (2)

- 약한 학습기를 여러 개 연결하여 강한 학습기를 만드는 양상을
 - ✓ 에이다부스팅(Adaptive Boosting)
 - ✓ 이전 모델이 과소적합했던 훈련 샘플의 가중치를 더 높이는 것



에이다부스팅(AdaBoosting) (3)

- 에이다부스팅에서 가중치 부여 방식
 - ✓ 개별 약한 학습기는 그림과 같이 가중치를 부여함



에이다부스팅(AdaBoosting) (4)

- ✓ 양상블에 이전까지의 오차를 보정하도록 예측기를 순차적으로 추가함
- ✓ 하지만, 반복마다 샘플의 가중치를 수정하는 대신 이전 예측기라 만든 잔여오차(residual error)에 새로운 예측기를 학습시킴

```
np.random.seed(42)          ← 잡음이 섞인 2차 곡선 형태의 훈련 세트
X = np.random.rand(100, 1) - 0.5
y = 3*X[:, 0]**2 + 0.05 * np.random.randn(100)
```

```
from sklearn.tree import DecisionTreeRegressor

tree_reg1 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg1.fit(X, y)
```

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=2,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=42, splitter='best')
```

에이다부스팅(AdaBoosting) (5)

```
y2 = y - tree_reg1.predict(X)
```

첫 번째 예측기에서 생긴 잔여 오차

```
tree_reg2 = DecisionTreeRegressor(max_depth=2, random_state=42)
```

```
tree_reg2.fit(X, y2)
```

잔여오차를 두번째 결정트리 회귀로 훈련

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=2,  
                      max_features=None, max_leaf_nodes=None,  
                      min_impurity_decrease=0.0, min_impurity_split=None,  
                      min_samples_leaf=1, min_samples_split=2,  
                      min_weight_fraction_leaf=0.0, presort='deprecated',  
                      random_state=42, splitter='best')
```

```
y3 = y2 - tree_reg2.predict(X)
```

두 번째 예측기에서 생긴 잔여 오차에 세번째 회귀 모델 훈련

```
tree_reg3 = DecisionTreeRegressor(max_depth=2, random_state=42)
```

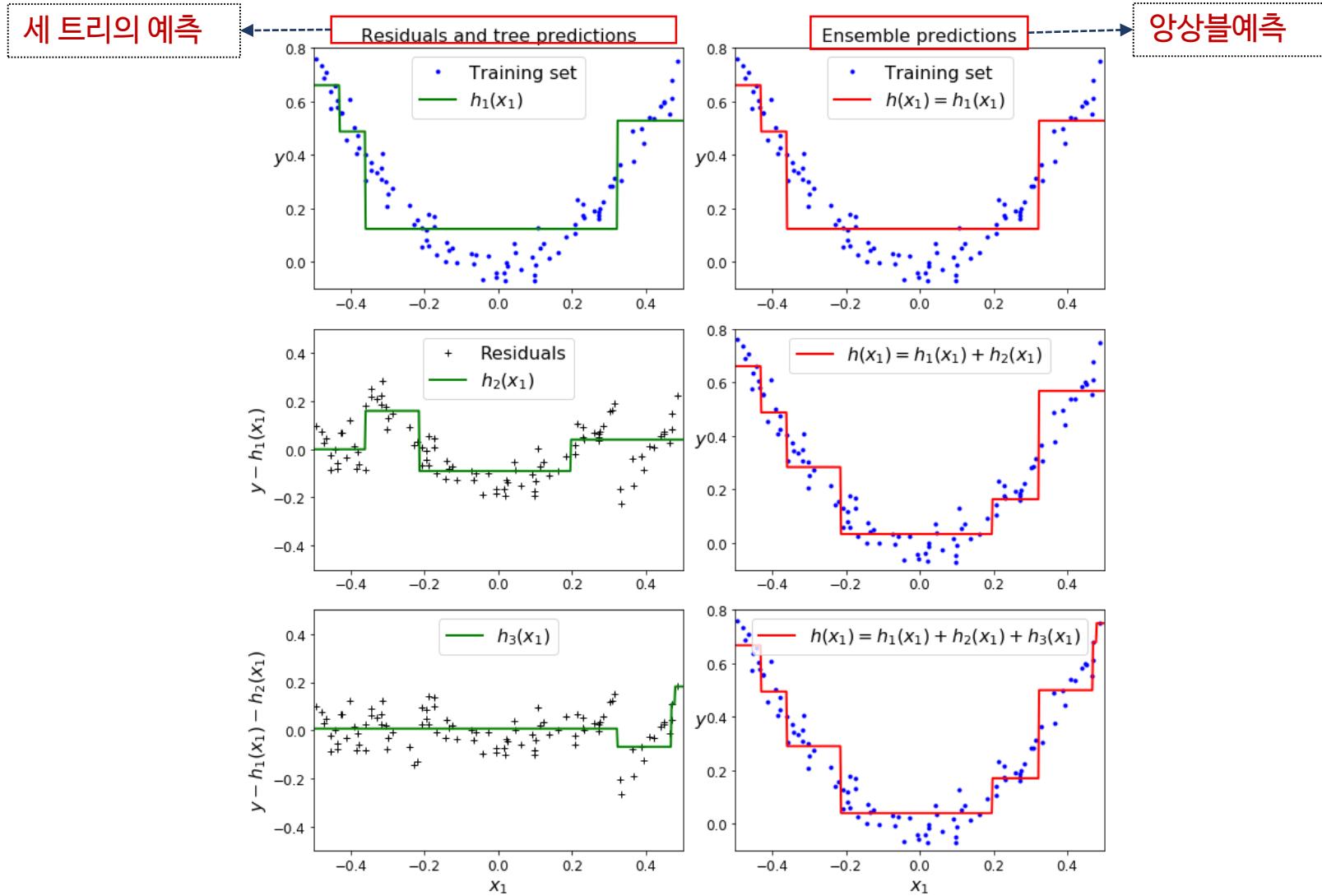
```
tree_reg3.fit(X, y3)
```

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=2,  
                      max_features=None, max_leaf_nodes=None,
```

```
y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
```

새로운 샘플에 대한 예측을 만들려면 모든 트리의 예측을 더하면 됨

에이다부스팅(AdaBoosting) (6)



AdaBoost 연습 (1)

AdaBoost

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=500, noise=0.30, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
```

AdaBoost 연습 (2)

```
from sklearn.ensemble import AdaBoostClassifier  
  
ada_clf = AdaBoostClassifier(  
    DecisionTreeClassifier(max_depth=1), n_estimators=200,  
    algorithm="SAMME.R", learning_rate=0.5, random_state=42)  
ada_clf.fit(X_train, y_train)
```

```
AdaBoostClassifier(algorithm='SAMME.R',  
                  base_estimator=DecisionTreeClassifier(ccp_alpha=0.0,  
                                                          class_weight=None,  
                                                          criterion='gini',  
                                                          max_depth=1,  
                                                          max_features=None,  
                                                          max_leaf_nodes=None,  
                                                          min_impurity_decrease=0.0,  
                                                          min_impurity_split=None,  
                                                          min_samples_leaf=1,  
                                                          min_samples_split=2,  
                                                          min_weight_fraction_leaf=0.0,  
                                                          presort='deprecated',  
                                                          random_state=None,  
                                                          splitter='best'),  
                  learning_rate=0.5, n_estimators=200, random_state=42)
```

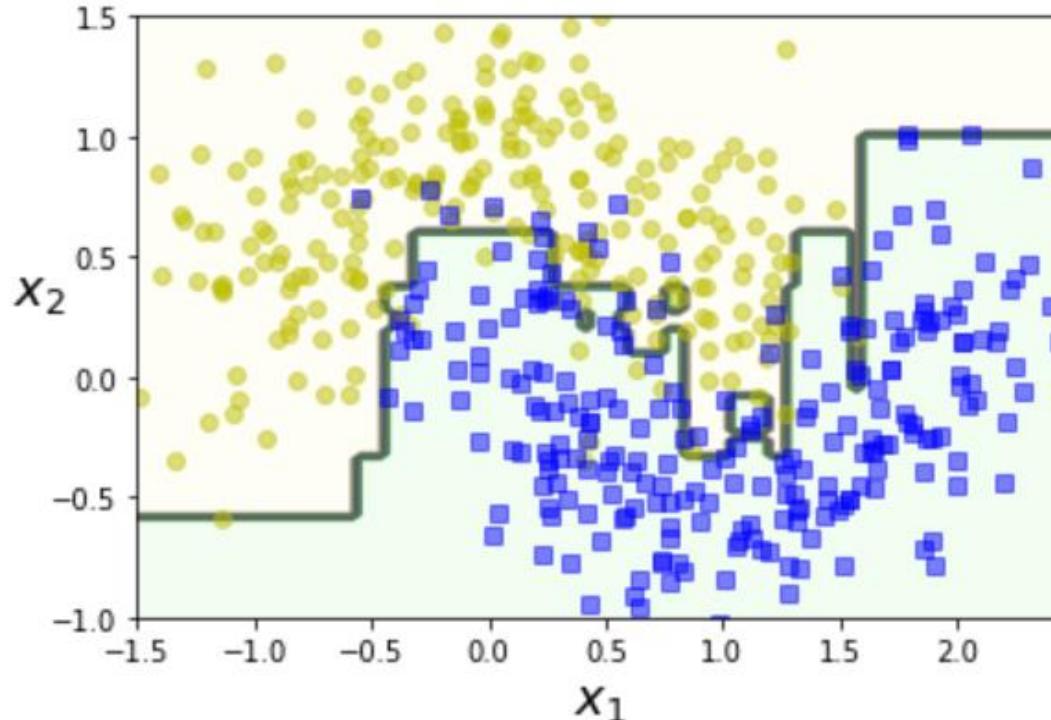
AdaBoost 연습 (3)

```
from matplotlib.colors import ListedColormap

def plot_decision_boundary(clf, X, y, axes=[-1.5, 2.45, -1, 1.5], alpha=0.5, contour=True):
    x1s = np.linspace(axes[0], axes[1], 100)
    x2s = np.linspace(axes[2], axes[3], 100)
    x1, x2 = np.meshgrid(x1s, x2s)
    X_new = np.c_[x1.ravel(), x2.ravel()]
    y_pred = clf.predict(X_new).reshape(x1.shape)
    custom_cmap = ListedColormap(['#fafab0','#9898ff','#a0faa0'])
    plt.contourf(x1, x2, y_pred, alpha=0.12, cmap=custom_cmap)
    if contour:
        custom_cmap2 = ListedColormap(['#7d7d58','#4c4c7f','#507d50'])
        plt.contour(x1, x2, y_pred, cmap=custom_cmap2, alpha=0.8)
    plt.plot(X[:, 0][y==0], X[:, 1][y==0], "yo", alpha=alpha)
    plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs", alpha=alpha)
    plt.axis(axes)
    plt.xlabel(r"$x_1$", fontsize=18)
    plt.ylabel(r"$x_2$", fontsize=18, rotation=0)
```

AdaBoost 연습 (4)

```
plot_decision_boundary(ada_clf, X, y)
```



AdaBoost 연습 (5)

```
m = len(X_train)

fix, axes = plt.subplots(ncols=2, figsize=(10,4), sharey=True)
for subplot, learning_rate in ((0, 1), (1, 0.5)):
    sample_weights = np.ones(m)
    plt.sca(axes[subplot])
    for i in range(5):
        svm_clf = SVC(kernel="rbf", C=0.05, gamma="scale", random_state=42)
        svm_clf.fit(X_train, y_train, sample_weight=sample_weights)
        y_pred = svm_clf.predict(X_train)
        sample_weights[y_pred != y_train] *= (1 + learning_rate)
        plot_decision_boundary(svm_clf, X, y, alpha=0.2)
        plt.title("learning_rate = {}".format(learning_rate), fontsize=16)
    if subplot == 0:
        plt.text(-0.7, -0.65, "1", fontsize=14)
        plt.text(-0.6, -0.10, "2", fontsize=14)
        plt.text(-0.5, 0.10, "3", fontsize=14)
        plt.text(-0.4, 0.55, "4", fontsize=14)
        plt.text(-0.3, 0.90, "5", fontsize=14)
    else:
        plt.ylabel("")

plt.show()
```

GBM 소개

- GBM 하이퍼 파라미터 튜팅

- ✓ loss

- 경사 하강법에서 사용하는 비용함수. 기본값인 ‘deviance’를 사용

- ✓ learning rate

- GBM 학습을 진행할 때마다 적용하는 학습률. weaker_learner가 순차적으로 오류 값을 보정해 가는데 적용함. 0~1 사이 값이며, 기본값은 0.1이다.

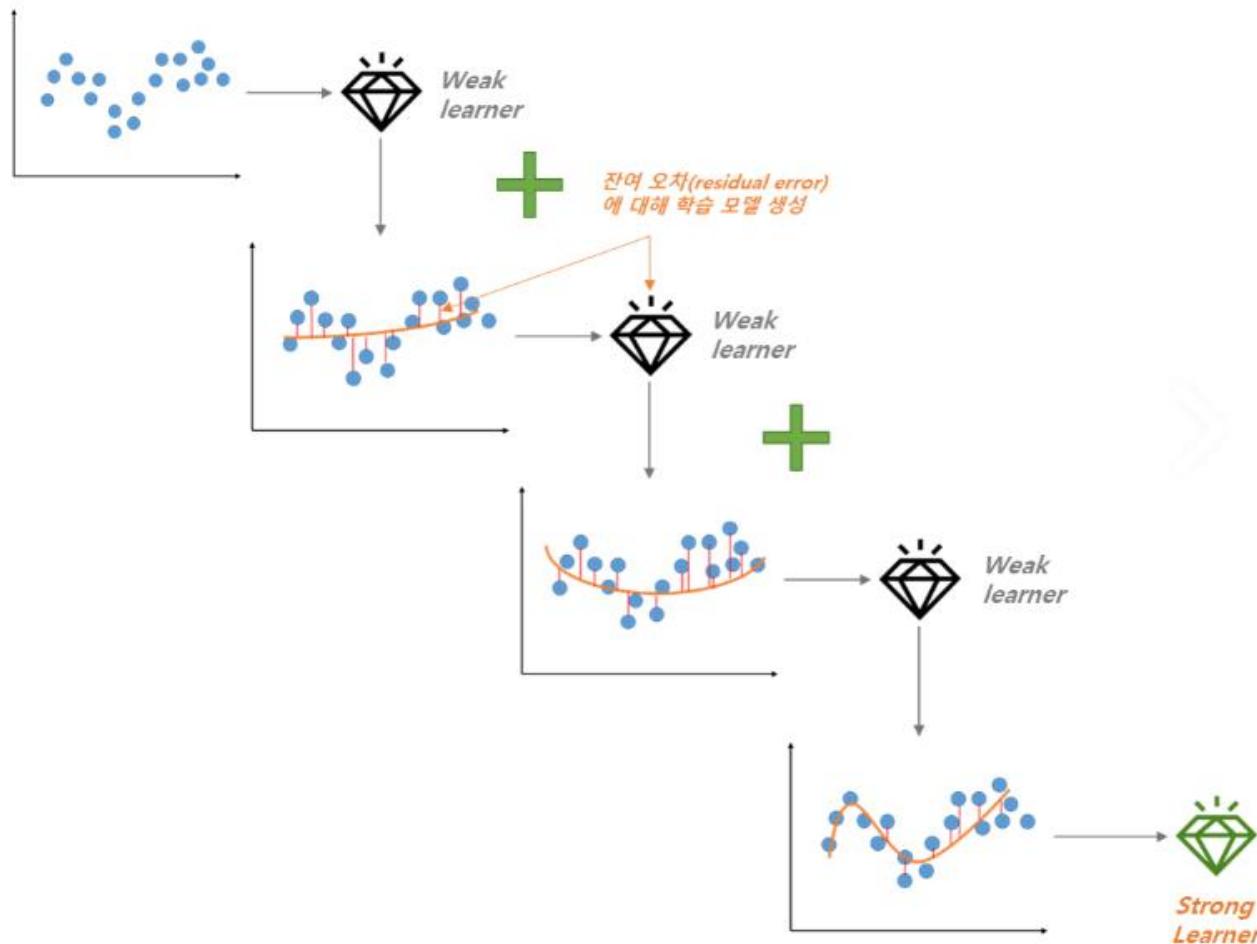
- ✓ n_estimators

- weak_learner 개수. 계수가 많을 수록 계산 시간이 오래 걸린다.

- ✓ subsample: weak_learner가 학습에 사용하는 데이터의 샘플링 비율

GBM 소개

Gradient Boosting



그레디언트 부스트 (1)

Gradient Boosting

```
np.random.seed(42)
X = np.random.rand(100, 1) - 0.5
y = 3*X[:, 0]**2 + 0.05 * np.random.randn(100)
```

```
from sklearn.tree import DecisionTreeRegressor

tree_reg1 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg1.fit(X, y)
```

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=2,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=42, splitter='best')
```

```
y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg2.fit(X, y2)
```

그레디언트 부스트 (2)

```
y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg3.fit(X, y3)
```

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=2,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=42, splitter='best')
```

```
X_new = np.array([[0.8]])
```

```
y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
```

```
y_pred
```

```
array([0.75026781])
```

그레디언트 부스트 (3)

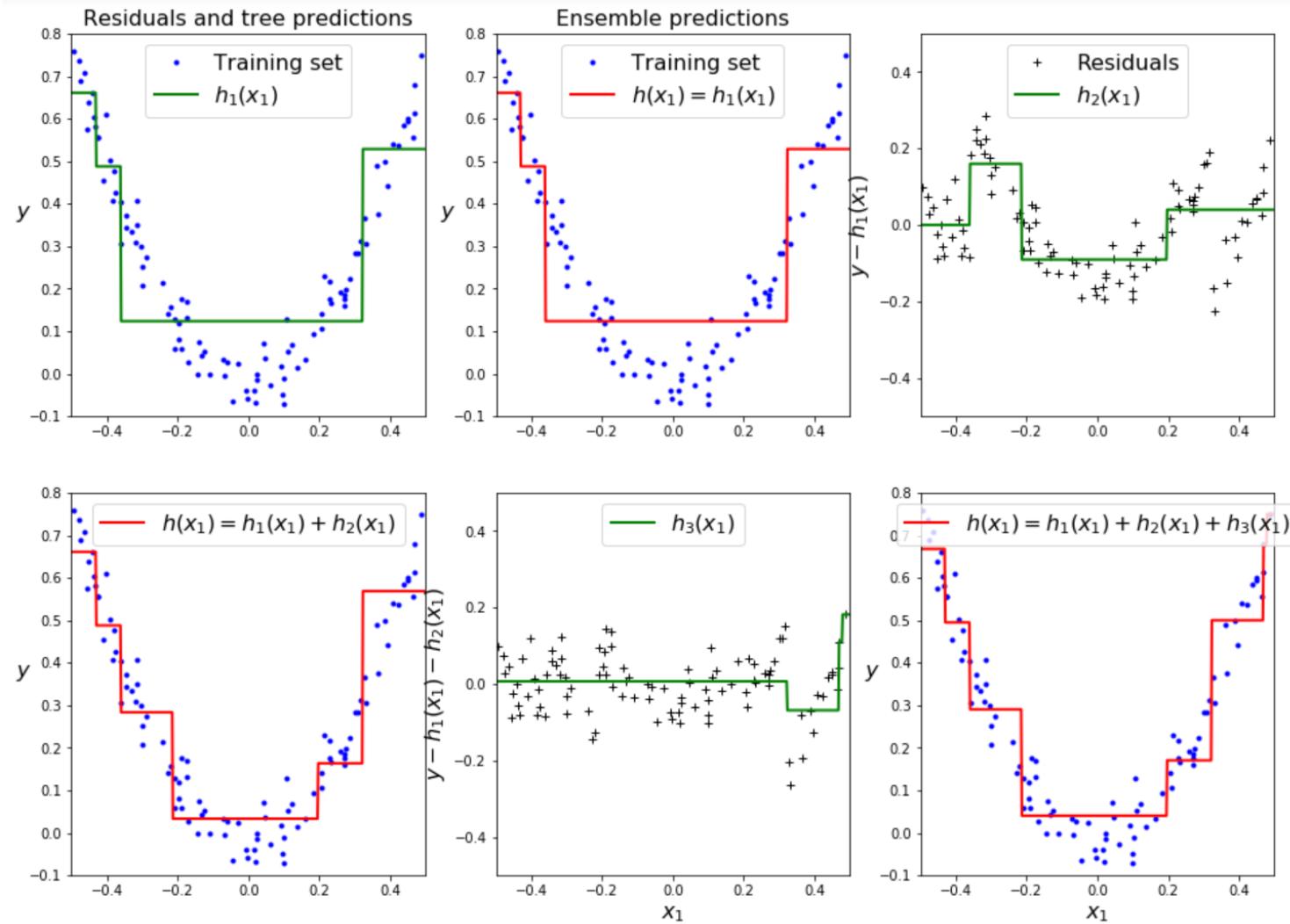
```
def plot_predictions(regressors, X, y, axes, label=None, style="r-", data_style="b.", data_label=None):
    x1 = np.linspace(axes[0], axes[1], 500)
    y_pred = sum(regressor.predict(x1.reshape(-1, 1)) for regressor in regressors)
    plt.plot(X[:, 0], y, data_style, label=data_label)
    plt.plot(x1, y_pred, style, linewidth=2, label=label)
    if label or data_label:
        plt.legend(loc="upper center", fontsize=16)
    plt.axis(axes)
```

```
plt.figure(figsize=(11,11))

plt.subplot(321)
plot_predictions([tree_reg1], X, y, axes=[-0.5, 0.5, -0.1, 0.8],
                 label="$h_1(x_1)$", style="g-", data_label="Training set")
plt.ylabel("$y$", fontsize=16, rotation=0)
plt.title("Residuals and tree predictions", fontsize=16)

plt.subplot(322)
plot_predictions([tree_reg1], X, y, axes=[-0.5, 0.5, -0.1, 0.8],
                 label="$h(x_1) = h_1(x_1)$", data_label="Training set")
plt.ylabel("$y$", fontsize=16, rotation=0)
plt.title("Ensemble predictions", fontsize=16)
```

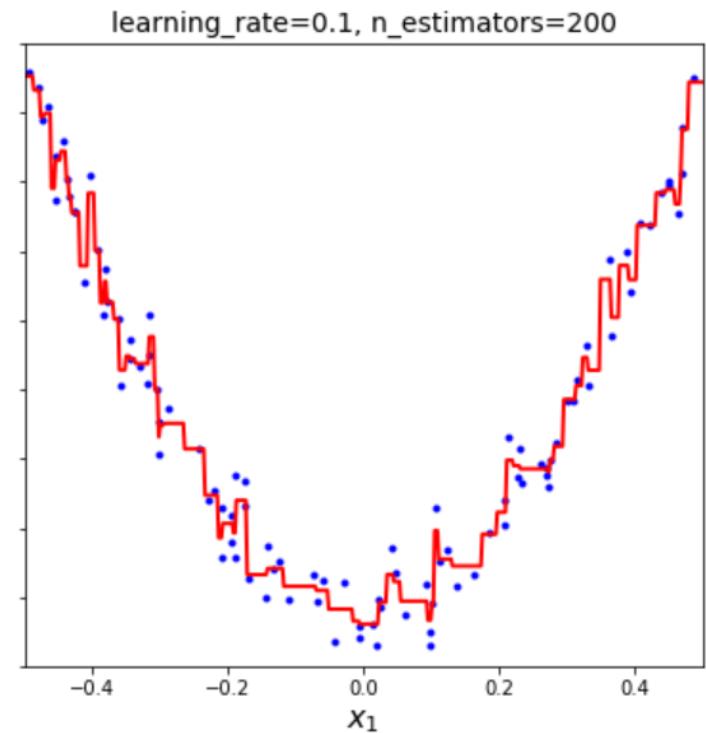
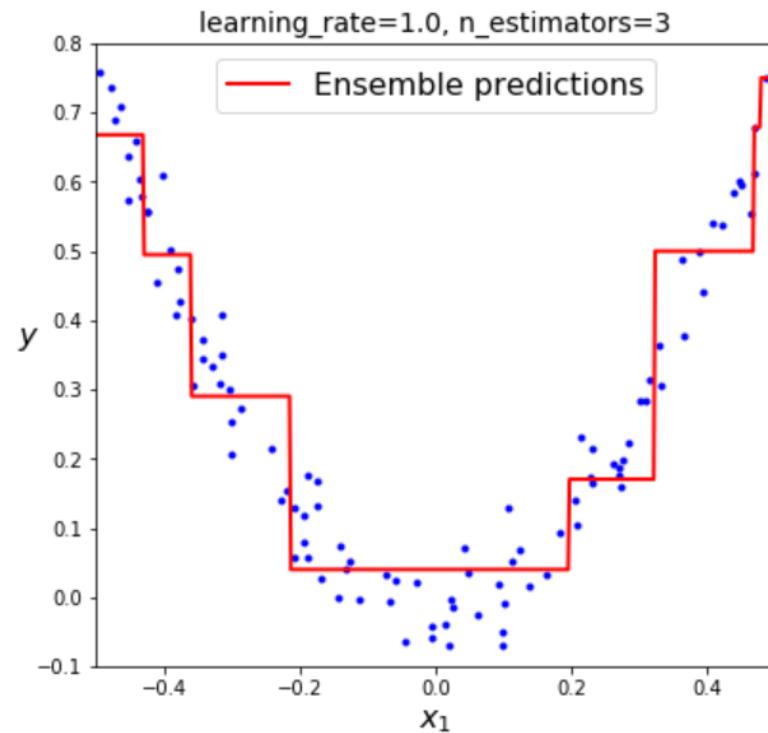
그레디언트 부스트 (4)



그레디언트 부스트 (5)

```
from sklearn.ensemble import GradientBoostingRegressor  
  
gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=3, learning_rate=1.0, random_state=42)  
gbrt.fit(X, y)  
  
gbrt_slow = GradientBoostingRegressor(max_depth=2, n_estimators=200, learning_rate=0.1, random_state=42)  
gbrt_slow.fit(X, y)  
  
fix, axes = plt.subplots(ncols=2, figsize=(14,6), sharey=True)  
  
plt.sca(axes[0])  
plot_predictions([gbrt], X, y, axes=[-0.5, 0.5, -0.1, 0.8], label="Ensemble predictions")  
plt.title("learning_rate={}, n_estimators={}".format(gbrt.learning_rate, gbrt.n_estimators), fontsize=14)  
plt.xlabel("$x_1$", fontsize=16)  
plt.ylabel("$y$", fontsize=16, rotation=0)  
  
plt.sca(axes[1])  
plot_predictions([gbrt_slow], X, y, axes=[-0.5, 0.5, -0.1, 0.8])  
plt.title("learning_rate={}, n_estimators={}".format(gbrt_slow.learning_rate, gbrt_slow.n_estimators), fontsize=14)  
plt.xlabel("$x_1$", fontsize=16)  
  
plt.show()
```

그레디언트 부스트 (6)



그레디언트 부스트 (7)

Gradient Boosting with Early stopping ¶

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X_train, X_val, y_train, y_val = train_test_split(X, y, random_state=49)

gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=120, random_state=42)
gbrt.fit(X_train, y_train)

errors = [mean_squared_error(y_val, y_pred)
          for y_pred in gbrt.staged_predict(X_val)]
bst_n_estimators = np.argmin(errors) + 1

gbrt_best = GradientBoostingRegressor(max_depth=2, n_estimators=bst_n_estimators, random_state=42)
gbrt_best.fit(X_train, y_train)
```

그레디언트 부스트 (8)

```
min_error = np.min(errors)

plt.figure(figsize=(10, 4))

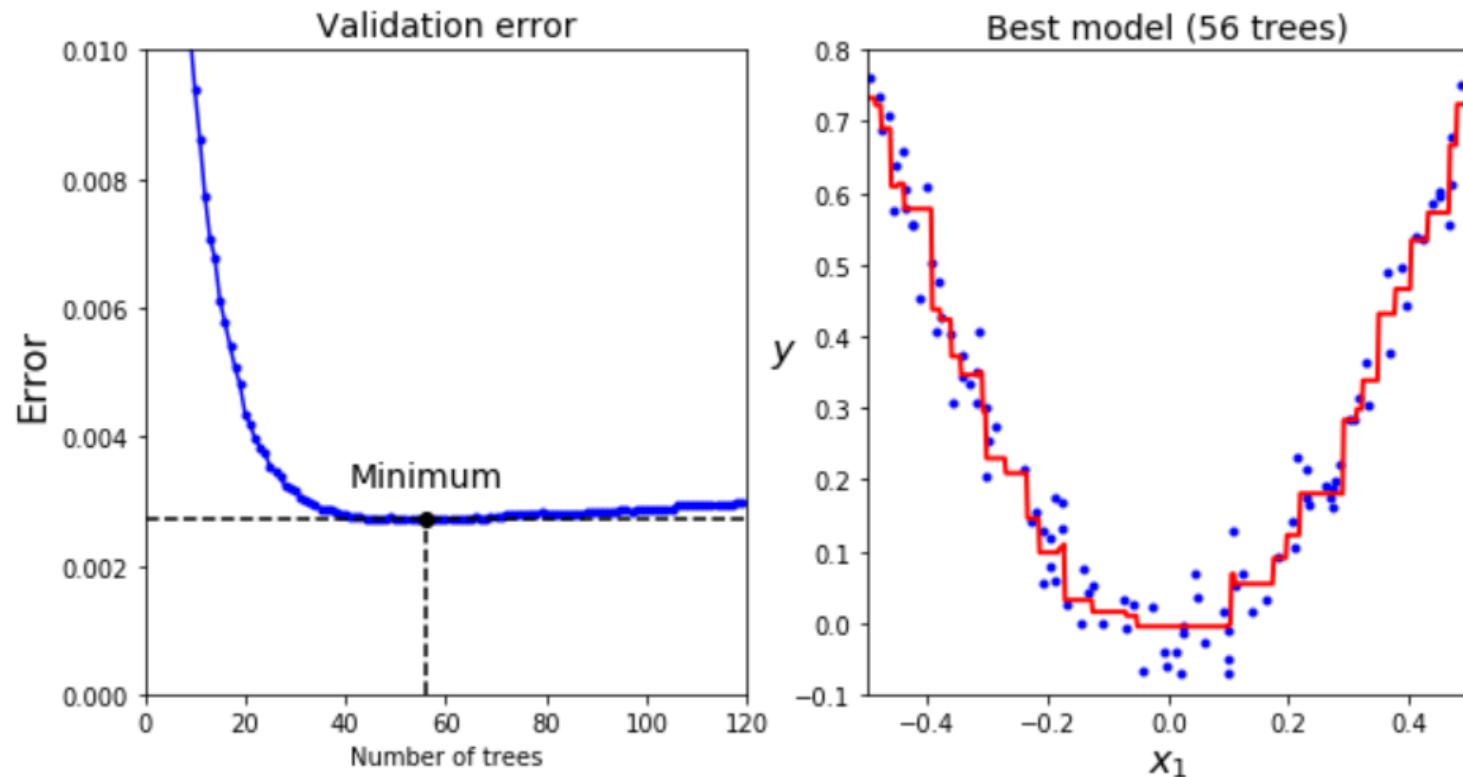
plt.subplot(121)
plt.plot(errors, "b.-")
plt.plot([bst_n_estimators, bst_n_estimators], [0, min_error], "k--")
plt.plot([0, 120], [min_error, min_error], "k--")
plt.plot(bst_n_estimators, min_error, "ko")
plt.text(bst_n_estimators, min_error*1.2, "Minimum", ha="center", fontsize=14)
plt.axis([0, 120, 0, 0.01])
plt.xlabel("Number of trees")
plt.ylabel("Error", fontsize=16)
plt.title("Validation error", fontsize=14)

plt.subplot(122)
plot_predictions([gbdt_best], X, y, axes=[-0.5, 0.5, -0.1, 0.8])
plt.title("Best model (%d trees)" % bst_n_estimators, fontsize=14)
plt.ylabel("$y$", fontsize=16, rotation=0)
plt.xlabel("$x_1$", fontsize=16)

plt.show()
```

그레디언트 부스트 (9)

그레디언트 브스팅과 조기종료 예제



그레디언트 부스트 (10)

```
gbdt = GradientBoostingRegressor(max_depth=2, warm_start=True, random_state=42)

min_val_error = float("inf")
error_going_up = 0
for n_estimators in range(1, 120):
    gbdt.n_estimators = n_estimators
    gbdt.fit(X_train, y_train)
    y_pred = gbdt.predict(X_val)
    val_error = mean_squared_error(y_val, y_pred)
    if val_error < min_val_error:
        min_val_error = val_error
        error_going_up = 0
    else:
        error_going_up += 1
        if error_going_up == 5:
            break # early stopping
```

```
print(gbdt.n_estimators)
```

61

```
print("Minimum validation MSE:", min_val_error)
```

Minimum validation MSE: 0.002712853325235463

XGBoost (eXtra Gradient Boost)

- XGBoost는 트리 기반의 양상을 학습의 최고의 알고리즘
 - ✓ GBM을 기반하고 있음.
 - ✓ GBM의 느린 단점인 수행시간 해결
 - ✓ GBM의 단점인 과적합 규제(Regularization) 부재를 해결 함.
 - ✓ 병렬 CPU 수행으로 GBM 보다 빠름
 - ✓ XGBoost는 C/C++로 개발되었으나, 사이킷런 래퍼를 제공.

XGBoost 사용하기 (1)

Using XGBoost ¶

```
if xgboost is not None: # not shown in the book
    xgb_reg = xgboost.XGBRegressor(random_state=42)
    xgb_reg.fit(X_train, y_train)
    y_pred = xgb_reg.predict(X_val)
    val_error = mean_squared_error(y_val, y_pred) # Not shown
    print("Validation MSE:", val_error) # Not shown
```

[16:35:35] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ol
vor of reg:squarederror.
Validation MSE: 0.0028512559726563943

```
if xgboost is not None: # not shown in the book
    xgb_reg.fit(X_train, y_train,
                eval_set=[(X_val, y_val)], early_stopping_rounds=2)
    y_pred = xgb_reg.predict(X_val)
    val_error = mean_squared_error(y_val, y_pred) # Not shown
    print("Validation MSE:", val_error) # Not shown
```

[16:35:06] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/ol
vor of reg:squarederror.
[0] validation_0-rmse:0.286719
Will train until validation_0-rmse hasn't improved in 2 rounds.
[1] validation_0-rmse:0.258221
[2] validation_0-rmse:0.232634

XGBoost 사용하기 (2)

```
: %timeit xgboost.XGBRegressor().fit(X_train, y_train) if xgboost is not None else None
[16:15:27] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152:
favor of reg:squarederror.
[16:15:27] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152:
favor of reg:squarederror.
[16:15:27] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152:
favor of reg:squarederror.
[16:15:28] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152:
favor of reg:squarederror.
10.9 ms ± 86 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
<
```



```
: %timeit GradientBoostingRegressor().fit(X_train, y_train)
20.3 ms ± 130 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

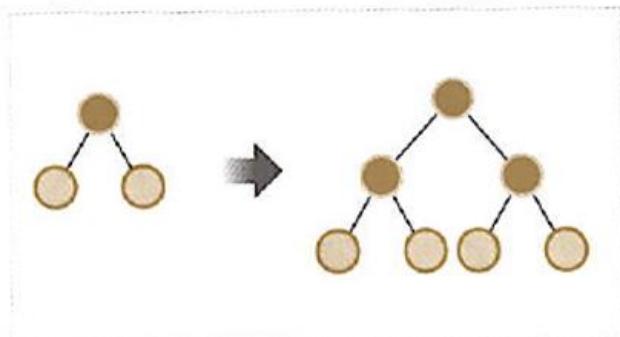
LightGBM 소개

- LightGBM은 XGBoost와 함께 부스팅 계열 알고리즘에서 각광
 - ✓ XGBoost는 GBM 보다 빠르지만 그래도 GridSearchCV 할때 느리다
 - ✓ LightGBM은 XGBoost 보다 빠르고, 메모리 용량도 적게 사용한단.
 - ✓ 단점
 - 적은 데이터 세트에 적용할 경우 과적합이 발생하기 쉽다. (10,000개 이하 데이터)

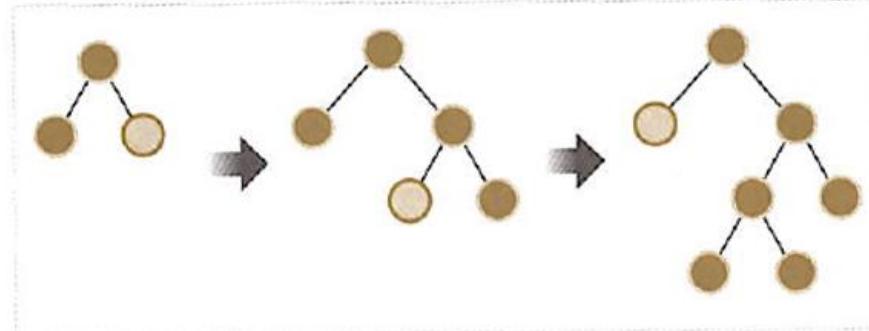
LightGBM 소개

- LightGBM은 GBM 계열의 트리 분할 방법이 다르다.
 - ✓ 기존은 균형 트리 분할
 - 최대한 균형 잡힌 트리를 유지하면서 분할하기에 트리의 깊이가 최소화
 - ✓ LightGBM은 리프 중심의 분할 방식 사용
 - 최대 손실 값을 가지는 리프 노드를 지속적으로 분할하면서 트리가 비대해짐

균형 트리 분할(Level Wise)



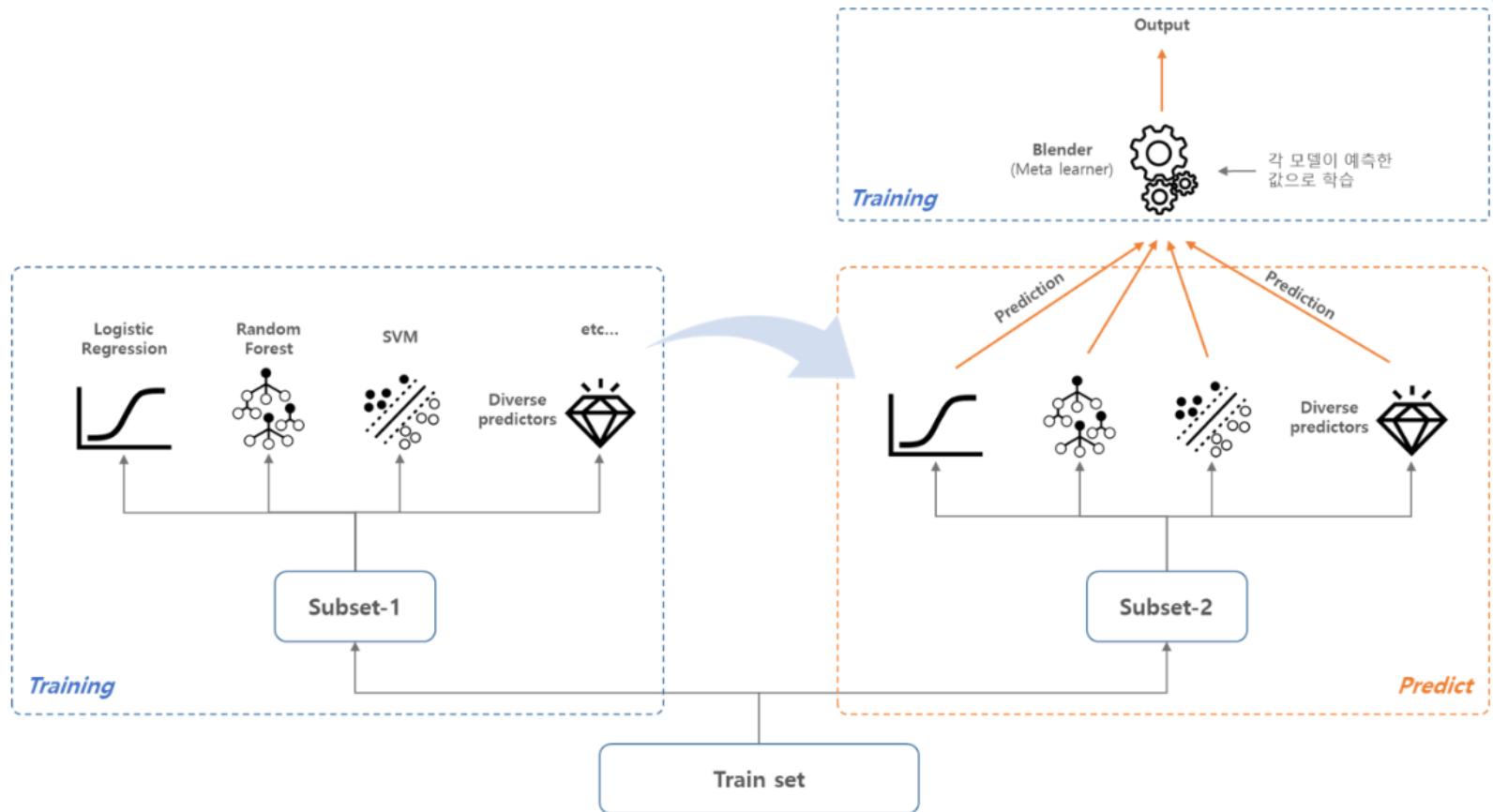
리프 중심 트리 분할(Leaf Wise)



스태킹 (Stacking)

- 스태킹(stacking, stacked generalization의 줄임)
 - ✓ 투표 기반 분류기에서 처럼 'hard voting', 'soft voting' 방법이 아니고,
 - ✓ 양상별 학습에서 각 모델의 예측 값을 가지고
 - ✓ 새로운 메타 모델(meta learner)을 학습시켜 최종 예측 모델을 만드는 방법
- 스태킹(stacking)의 과정
 - ✓ 학습 데이터셋에서 샘플링을 통해 서브셋1을 만들고, 각 모델을 학습시킴
 - ✓ 서브셋2 학습 시킨 모델을 이용해 각 모델의 예측 값을 출력하고 합함
 - ✓ 합쳐진 예측 값을 입력 피처로 사용하는 새로운 meta learner를 학습함

스태킹 개념 이해 : 메타러닝



Thank You!

www.ust.ac.kr