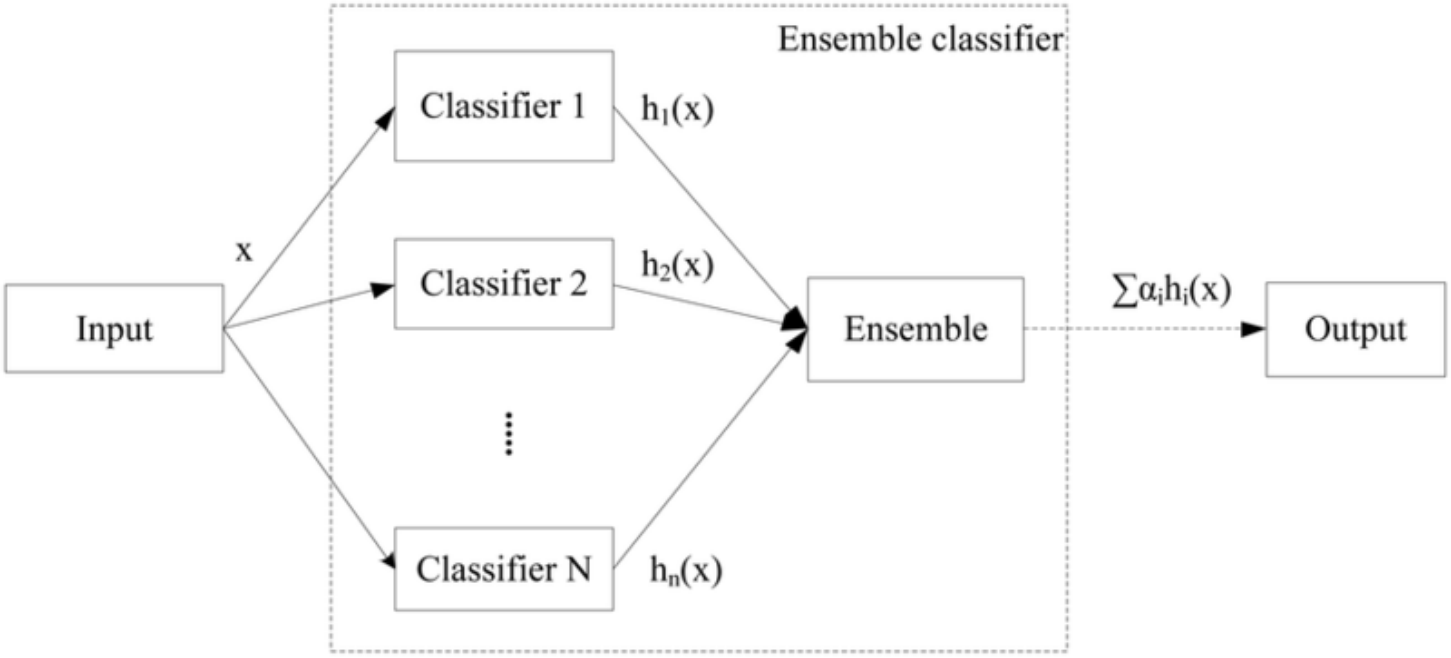
09. 앙상블과 랜덤포레스트 (Ensemble & Random Forest)

- 앙상블 학습(Ensemble Learning)
- 배깅(Bagging) & 랜덤 포레스트(Random Forest)
- 부스팅(Boosting)
- 스태킹(Stacking)

앙상블 학습(Ensemble Learning)

- 앙상블 학습(Ensemble Learning)
  - 여러 개의 분류기를 생성하고, 그 예측을 결합함으로써 보다 정확한 예측을 도출하는 기법
  - 강력한 하나의 모델을 사용하는 대신 보다 약한 모델 여러 개를 조합하여 더 정확한 예측에 도움을 주는 방식



유형

- 보팅(Voting)
  - 여러 개의 분류기가 투표를 통해 최종 예측 결과를 결정하는 방식
- 배깅(Bagging: Bootstrap AGGregatING)
  - 데이터 샘플링(Bootstrap)을 통해 모델을 학습시키고 결과를 집계(Aggregating) 하는 방법 (예) Random Forest 등
- 부스팅(Boosting)
  - 여러 개의 분류기가 순차적으로 학습을 수행(다음 분류기에게 가중치를 부스팅)
- 스태킹(Stacking: stacked generalization)
  - 여러 개의 분류기 결과를 취합하는 마지막 예측기(블렌더 또는 메타학습기)를 학습하는 방법

특징

- 가능한한 모든 분류기가 완전히 독립적이며, 오차의 상관관계가 없을 것
- 즉, 가능한한 서로 다른 알고리즘으로 분류기를 학습시킴으로써 서로 많이 다른 종류의 오차를 생성할 가능성이 높아지도록 해야 높은 정확도를 얻을 수 있음
- 성능을 분산시키므로 과적합(overfitting) 감소 효과가 있음

보팅(Voting)

- 여러 개의 분류기가 투표를 통해 최종 예측 결과를 결정하는 방식
- 보통 서로 다른 알고리즘을 여러 개 결합하여 사용
- 보팅 방식
  - 하드보팅(Hard Voting)
    - 다수의 분류기가 예측한 결과값을 최종 결과로 선정
    - 직접투표에 해당
  - 소프트보팅(Soft Voting)
    - 모든 분류기가 예측한 레이블 값의 결정 확률 평균을 계산
    - 가장 확률이 높은 레이블 값을 최종 결과로 선정
    - 간접투표에 해당
    - 하드보팅에 비해 성능이 좋은 편임

sklearn.ensemble.**VotingClassifier**

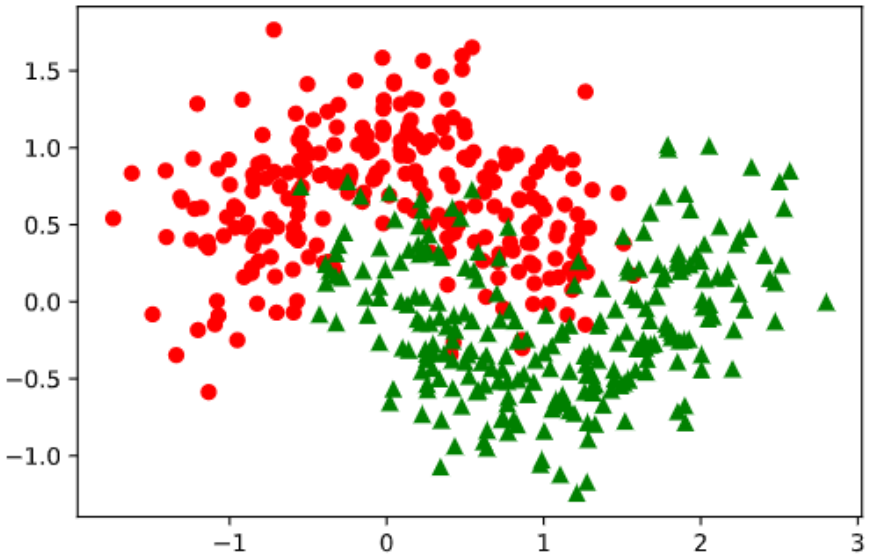
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier

models = [
    ('knn', KNeighborsClassifier()),
    ('svc', SVC(probability=True)),
    ('dtc', DecisionTreeClassifier())
]
# hard vote
hard_vote  = VotingClassifier(models, voting='hard')
hard_vote.fit(X_train, y_train)
# soft vote
soft_vote  = VotingClassifier(models, voting='soft')
soft_vote.fit(X_train, y_train)
```

결합할 분류기들

voting 방법

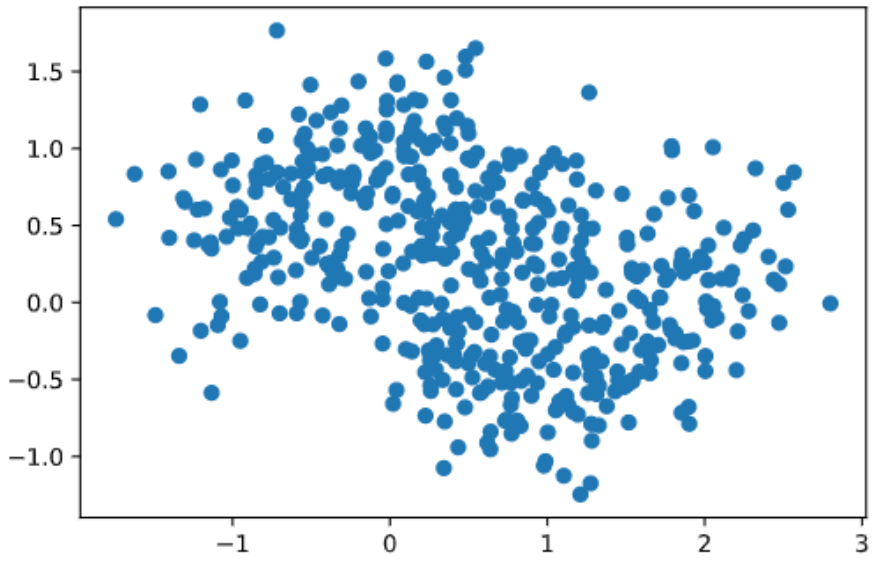
데이터 준비

```
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt

X, y = make_moons(n_samples=500, noise=0.30, random_state=42)
plt.plot(X[:, 0][y==0], X[:, 1][y==0], "ro")
plt.plot(X[:, 0][y==1], X[:, 1][y==1], "g^")
plt.show()
```

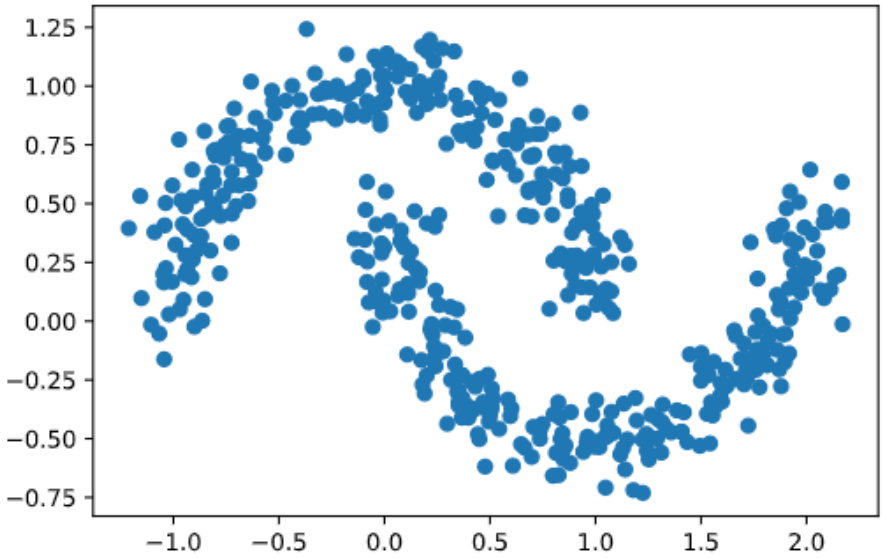


참고:

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

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)
```



noise=0.30



noise=0.10

Hard Voting

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import accuracy_score

log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier(n_estimators=10)
svm_clf = SVC()

models = [('lr', log_clf), ('rf', rnd_clf), ('svm', svm_clf)]
voting_clf = VotingClassifier(estimators=models, voting='hard')
voting_clf.fit(X_train, y_train)

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 0.872
SVC 0.896
VotingClassifier 0.904
```

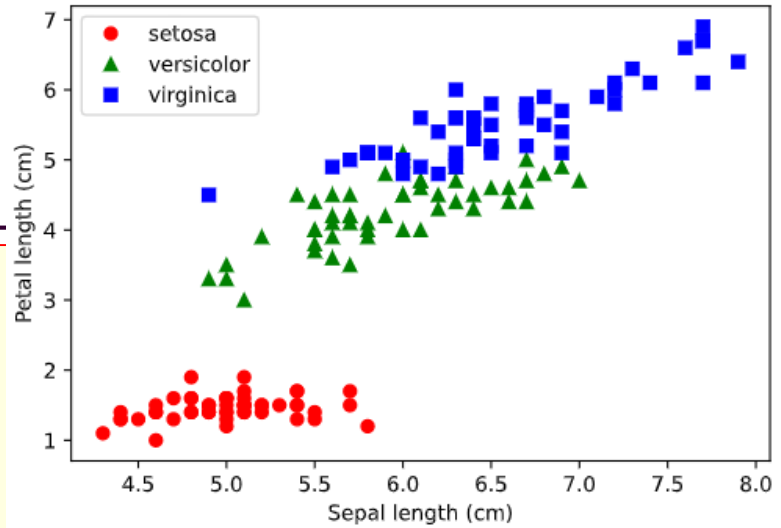
Soft Voting

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import accuracy_score

log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier(n_estimators=10)
svm_clf = SVC(probability=True)

models = [('lr', log_clf), ('rf', rnd_clf), ('svm', svm_clf)]
voting_clf = VotingClassifier(estimators=models, voting='soft')
voting_clf.fit(X_train, y_train)

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))
```

SVC는 각 레이블에 대한 확률을 제공하지 않으므로 probability=True로 지정해야 함

```
LogisticRegression 0.864
RandomForestClassifier 0.92
SVC 0.896
VotingClassifier 0.928
```

데이터 준비




```
rom sklearn.datasets import load_iris
import matplotlib.pyplot as plt

iris = load_iris()
X = iris.data[:, [0, 2]] # 꽃받침의 길이와 꽃잎의 길이
y = iris.target

plt.plot(X[:,0][y==0], X[:,1][y==0], "ro", label='setosa')
plt.plot(X[:,0][y==1], X[:,1][y==1], "g^", label='versicolor')
plt.plot(X[:,0][y==2], X[:,1][y==2], "bs", label='virginica')
plt.xlabel('Sepal length (cm)')
plt.ylabel('Petal length (cm)')
plt.legend()
plt.show()
```

Ensemble(Voting)이 반드시 좋을까??

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y)

dt_clf = DecisionTreeClassifier(max_depth=4)
knn_clf = KNeighborsClassifier(n_neighbors=7)
svm_clf = SVC(gamma=0.1, probability=True)
models = [('dt', dt_clf), ('knn', knn_clf), ('svm', svm_clf)]
voting_clf = VotingClassifier(models, voting='soft', weights=[2, 1, 2])
voting_clf.fit(X_train, y_train)

for clf in (dt_clf, knn_clf, svm_clf, voting_clf):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(f"{clf.__class__.__name__}: {accuracy_score(y_test,y_pred):.4f}")
```

```
DecisionTreeClassifier: 0.9211
KNeighborsClassifier: 0.9737
SVC: 0.9737
VotingClassifier: 0.9211
```

배깅(Bagging)

- 데이터 샘플링(Bootstrap:중복허용)을 통해 모델을 학습시키고 결과를 집계(Aggregating) 하는 방법
- 모두 같은 유형의 알고리즘 기반의 분류기를 사용
- 데이터 분할 시 중복을 허용하는 방식임
  - 중복을 허용하지 않는 방식을 페이스팅(Pasting)이라고 함
- 집계방법
  - Categorical Data : 다수결 투표 방식으로 결과 집계
  - Continuous Data : 평균값 집계
- 대표적인 배깅 방식
  - 랜덤 포레스트(Random Forest) 알고리즘

sklearn.ensemble.**BaggingClassifier**

```
BaggingClassifier(base_estimator=None,
                  n_estimators=10,
                  bootstrap=True, oob_score=False,
                  n_jobs=None, random_state=None)
```

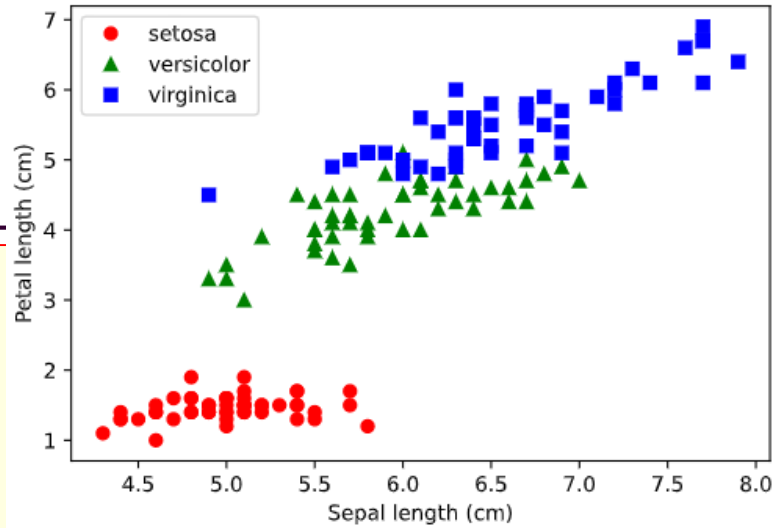
- base_estimator: 기본 분류기
- n_estimators: 기본 분류기의 개수
- bootstrap: 샘플이 무작위로 샘플링될지의 여부
- oob_score: out-of-bag 샘플을 사용할지 여부
  - oob(out-of-bag): 선택되지 않은 학습샘플의 나머지
- n_jobs: 병렬로 실행할 작업의 개수

sklearn.ensemble.**BaggingRegressor**

```
BaggingRegressor(base_estimator=None,
                 n_estimators=10,
                 bootstrap=True, oob_score=False,
                 n_jobs=None, random_state=None)
```

- base_estimator: 기본 분류기
- n_estimators: 기본 분류기의 개수
- bootstrap: 샘플이 무작위로 샘플링될지의 여부
- oob_score: out-of-bag 샘플을 사용할지 여부
  - oob(out-of-bag): 선택되지 않은 학습샘플의 나머지
- n_jobs: 병렬로 실행할 작업의 개수

데이터 준비




```
rom sklearn.datasets import load_iris
import matplotlib.pyplot as plt

iris = load_iris()
X = iris.data[:, [0, 2]] # 꽃받침의 길이와 꽃잎의 길이
y = iris.target

plt.plot(X[:,0][y==0], X[:,1][y==0], "ro", label='setosa')
plt.plot(X[:,0][y==1], X[:,1][y==1], "g^", label='versicolor')
plt.plot(X[:,0][y==2], X[:,1][y==2], "bs", label='virginica')
plt.xlabel('Sepal length (cm)')
plt.ylabel('Petal length (cm)')
plt.legend()
plt.show()
```

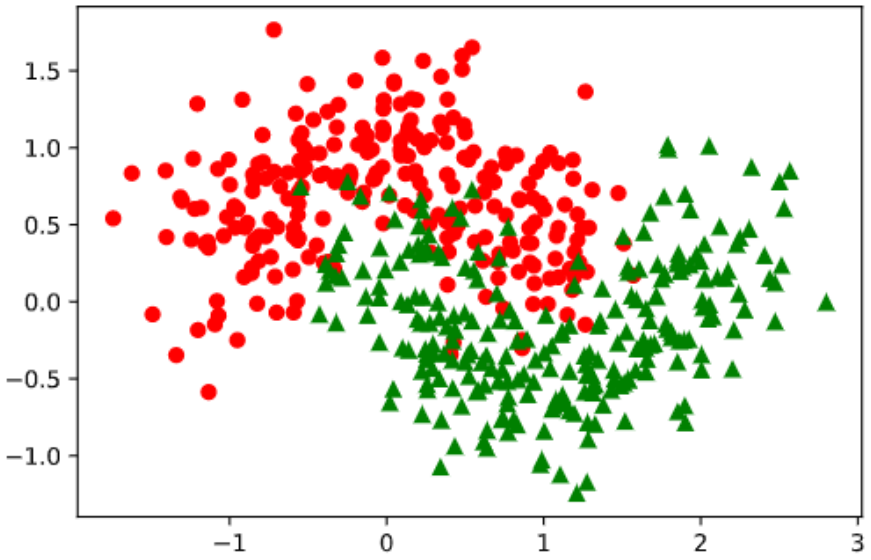
배깅(Bagging)

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

dt_clf = DecisionTreeClassifier()
bag_clf = BaggingClassifier(dt_clf, n_estimators=500)
bag_clf.fit(X_train, y_train)

for clf in (dt_clf, bag_clf):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(f"{clf.__class__.__name__}: \
        {accuracy_score(y_test,y_pred):.4f}")
```

```
DecisionTreeClassifier: 0.8947
BaggingClassifier: 0.9474
```

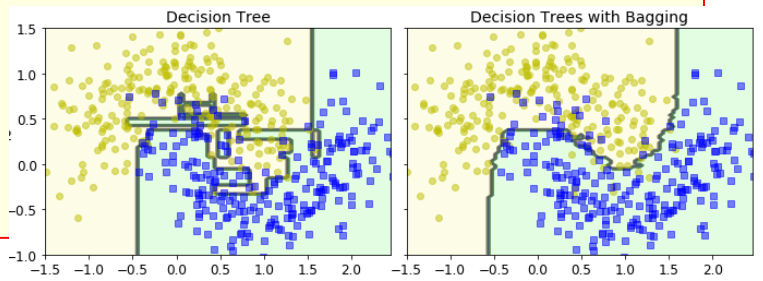
데이터 준비

```
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt

X, y = make_moons(n_samples=500, noise=0.30, random_state=42)
plt.plot(X[:, 0][y==0], X[:, 1][y==0], "ro")
plt.plot(X[:, 0][y==1], X[:, 1][y==1], "g^")
plt.show()
```



```python
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import accuracy_score

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)

dt_clf = DecisionTreeClassifier()
bag_clf = BaggingClassifier(dt_clf, n_estimators=500)
bag_clf.fit(X_train, y_train)

for clf in (dt_clf, bag_clf):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(f"{clf.__class__.__name__}: \
        {accuracy_score(y_test,y_pred):.4f}")
```

```
DecisionTreeClassifier: 0.8960
BaggingClassifier: 0.9200
```



랜덤 포레스트(Random Forest)

- Decision Tree를 개별 기본분류기로 사용하는 분류기 결합 방법

```
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

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)

clf = RandomForestClassifier(n_estimators=500)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
print(f"{clf.__class__.__name__}: {accuracy_score(y_test,y_pred):.4f}")
```

```
RandomForestClassifier: 0.9360
```

부스팅(Boosting)

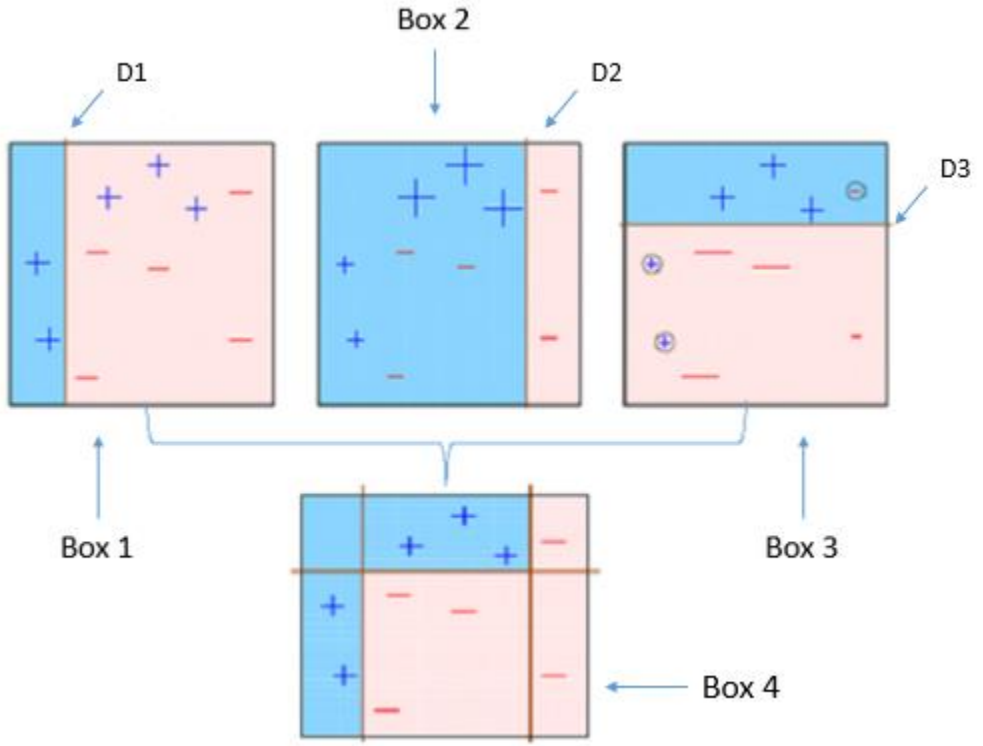
- 여러 개의 분류기가 순차적으로 학습을 수행
  - 이전 분류기가 예측이 틀린 데이터에 대해서 올바르게 예측할 수 있도록 다음 분류기에게 가중치(weight)를 부여하면서 학습과 예측을 진행
  - 계속하여 분류기에게 가중치를 부스팅하며 학습을 진행하기에 부스팅 방식이라고 불림
  - 예측 성능이 뛰어나 앙상블 학습을 주도
- 대표적인 부스팅 모듈
  - AdaBoost, GBM(Gradient Boosting Machine)
  - XGBoost, LightBoost
- 주의
  - 배깅에 비해 성능이 좋지만, 속도가 느리고 과적합 가능성 있음

- **AdaBoost(Adaptive Boosting)**
  - 이전까지의 오차를 보정하도록 예측기를 순차적(Sequential)으로 추가
  - 오차보정을 위해 데이터들에 가중치를 부여하면서 동작
    - 분류하기 어려운 Instances : 가중치 증가
    - 이미 잘 분류된 Instances : 가중치 감소
  - 약한 학습기(weak learner)의 오류에 가중치를 더하면서 부스팅을 수행하는 알고리즘
  - 약한 학습기(weak learner)로 결정트리(Decision Tree)를 사용
    - binary classifier

$$H(x) = \alpha_1 h_1(x) + \alpha_2 h_2(x) + \cdots + \alpha_t h_t(x) = \sum_{i=1}^{T} \alpha_i h_i(x)$$

작동방식 설명

- 첫 번째 약분류기가 첫번째 분류기준(D1)으로 + 와 - 를 분류
- 잘못 분류된 데이터에 대해 가중치를 부여
  - 두 번째 그림에서 커진 + 표시
- 두 번째 약분류기가 두번째 분류기준(D2)으로 +와 - 를 다시 분류
- 잘못 분류된 데이터에 대해 가중치를 부여
  - 세 번째 그림에서 커진 - 표시
- 세 번째 약분류기가 세번째 분류기준으로(D3) +와 -를 다시 분류
- ...
- 마지막으로 분류기들을 결합하여 최종 예측 수행



출처: https://injo.tistory.com/31

약분류기(weak classifier)의 예

$$h(x, f, \theta) = \begin{cases} 1 & if\ f(x) < \theta \\ 0 & otherwise \end{cases}$$

- 정말 간단히(대충...) 입력 샘플 x의 특정 특징(f)에 대해 주어진 분류기준 임계치(θ)를 가지고 class label을 분류
- (예)
  - x1 = {0.4, 0.8, 0.2}, y1 = 1
  - x2 = {0.3, 0.5, 0.1}, y1 = 0
  - 첫번째 특징(f)에 대해 분류기준 임계치(θ)를 0.5로 한다면...
  - 분류결과: $h(x1) = 1,\ h(x2) = 1$

알고리즘

$samples: (x_1, y_1), \ldots (x_n, y_n), y = 1 \; or \; 0 \; (binary \; classification)$

가중치 초기화: $w_{1i} = \frac{1}{n}$

$for \;\; t = 1 \;\; \ldots T$

$w_{ti} = \frac{w_{ti}}{\sum_j^n w_{tj}}$ # 가중치 정규화

$\epsilon_t = \min_{f,\theta} \sum_i^n w_{ti} |h(x_i, f_t, \theta_t) - y_i|$ # 가장 성능이 좋은 약분류기 h를 찾고, 그 에러를 ϵ_t로

$h_t(x) = h(x, f_t, \theta_t)$ # 가장 성능이 좋은 약분류기

$w_{t+1,i} = w_i \cdot \beta_t^{1-e_i}$ (# 가중치 변경), $\beta_t = \frac{\epsilon_t}{1 - \epsilon_t}$, $e_i = \begin{cases} 0 & h_t(x_i) == y_i \\ 1 & otherwise \end{cases}$

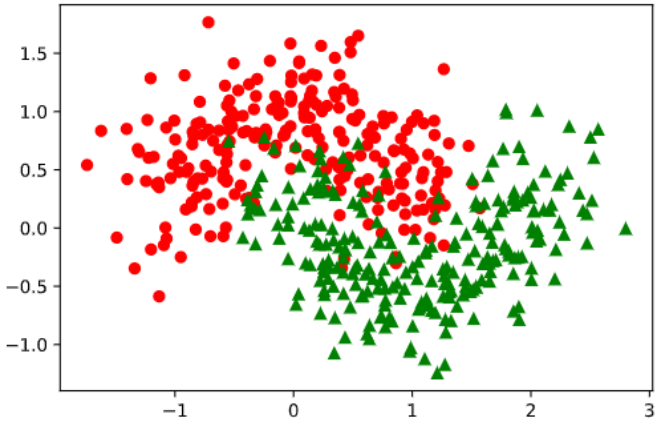
end

$\hat{y} = H(x) = \sum_{i=1}^{T} \alpha_i h_i(x)$ # 약분류기들을 다수결투표 방식으로 연결하여 최종 예측

sklearn.ensemble.**AdaBoostClassifier**

```
AdaBoostClassifier(base_estimator=None,
                   n_estimators=50,
                   learning_rate=1.0,
                   random_state=None)
```

- **base_estimator: 약분류기**
  - None이면 DecisionTreeClassifier(max_depth=1)
- **n_estimators: 생성할 약분류기의 개수**
- **learning_rate: 학습률(0~1)**
  - 약분류기(Weak learner)가 순차적으로 오류 값을 보정해 나갈 때 적용하는 계수




```
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_s
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score

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)

clf = AdaBoostClassifier(n_estimators=200, learning_rate=0.1)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
print(f"{clf.__class__.__name__}:
            {accuracy_score(y_test,y_pred):.4f}")
```

```
AdaBoostClassifier: 0.9520
```

Gradienet Boosting Machine (GBM)

- AdaBoost처럼 매 반복 마다 샘플의 가중치를 조정하는 대신에 이전 예측기가 만든 잔여 오차(Resudial Error: 정답과 예측의 차이)를 예측하는 새로운 예측기를 만들고 학습시킴
  - 이전 모델의 Residual를 가지고 약분류기(weak learner)를 강화함. 즉, Residual를 예측하는 형태의 모델
- 가중치 업데이트
  - AdaBoost: 가중치를 단순히 증가 또는 감소
  - GBM : 경사하강법(Gradient Descent)
- 과적합(Overfitting) 및 속도의 이슈가 있음

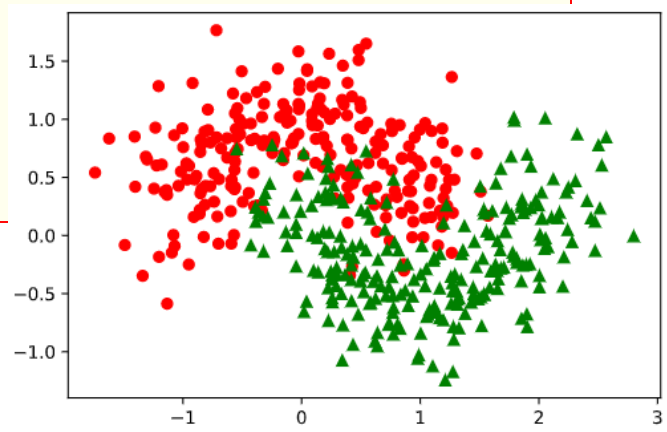
sklearn.ensemble.**GradientBoostingClassifier**

```
GradientBoostingClassifier(loss='deviance’,
                           learning_rate=0.1,
                           n_estimators=100,
                           subsample=1.0)
```

- **loss: 경사하강법에서 사용할 비용함수**
- **learning_rate: 학습률(0~1)**
  - 약분류기(Weak learner)가 순차적으로 오류 값을 보정해 나갈 때 적용하는 계수
- **n_estimators: 생성할 약분류기의 개수**
- **subsample: 학습에 사용할 샘플의 비율**

```python
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score

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)

clf = GradientBoostingClassifier(n_estimators=200,
                                 learning_rate=0.1)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
print(f"{clf.__class__.__name__}:
            {accuracy_score(y_test,y_pred):.4f}")
```

```
GradientBoostingClassifier: 0.9440
```



- **XGBoost**
  - GBM의 과적합(Overfitting) 문제, 속도의 문제와 같은 여러 단점을 보완하기 위한 모델
    - GBM과 마찬가지로 가중치 업데이트를 경사하강법(Gradient Descent) 기법을 사용
  - 특징
    - 과적합(Overfitting) 방지를 위한 규제(Regularization)
    - GBM 보다 빠른 속도(분산환경에서도 실행 가능)
    - CART(Classification & Regression Tree) 기반
      - 분류(Classification)와 회귀(Regression) 둘 다 가능
    - 조기종료(early sttoping) 제공
- **설치**
  - pip install xgboost

class xgboost.**XGBClassifier**

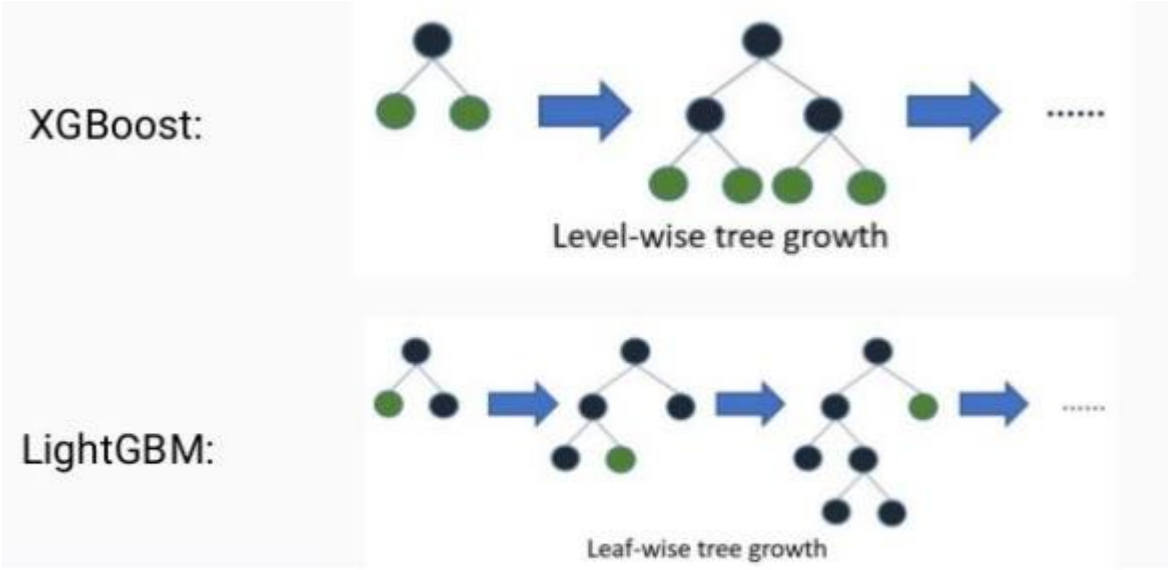
```
class xgboost.XGBClassifier(*,
                            objective='binary:logistic’,
                            use_label_encoder=True, **kwargs)
```

- **n_estimators: 생성할 약분류기의 개수**
- **max_depth: 트리의 깊이**
- **learning_rate: 학습률(0~1)**
  - 약분류기(Weak learner)가 순차적으로 오류 값을 보정해 나갈 때 적용하는 계수
- **subsample: 학습에 사용할 샘플의 비율**

```python
import warnings
warnings.filterwarnings(action='ignore')

from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score

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)

clf = XGBClassifier(n_estimators=200, learning_rate=0.1)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
print(f"{clf.__class__.__name__}: {accuracy_score(y_test,y_pred):.4f}")
```

```
[13:44:07] WARNING: C:/Users/Administrator/workspace/xgboost-
...
like to restore the old behavior.
XGBClassifier: 0.9440
```

- **LightGBM**
  - XGBoost의 느린 학습시간을 보완하기 위한 모델
    - 대용량 처리 가능, 적은 메모리 사용, GPU 지원 등
  - LightGBM은 리프 중심 트리 분할(Leaf Wise) 방식을 사용
    - XGBoost : 균형 트리 분할(Level Wise) 방식을 사용
  - 설치
    - pip install lightgbm



출처 : https://www.slideshare.net/GabrielCyprianoSaca/xgboost-lightgbm

class lightgbm.**LGBMClassifier**

```
LGBMClassifier(max_depth=-1, learning_rate=0.1,
               n_estimators=100, min_child_samples=20,
               subsample=1.0, n_jobs=-1)
```

- max_depth: 트리의 깊이
- learning_rate: 학습률(0~1)
- n_estimators: 생성할 약분류기의 개수
- min_child_samples: leaf node가 되기위한 최소샘플개수
- subsample: 학습에 사용할 샘플의 비율
- n_jobs: 쓰레드(thread)의 개수

```
import warnings
warnings.filterwarnings(action='ignore')

from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from lightgbm import LGBMClassifier
from sklearn.metrics import accuracy_score

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)

clf = LGBMClassifier(n_estimators=200, learning_rate=0.1)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
print(f"{clf.__class__.__name__}:
                    {accuracy_score(y_test,y_pred):.4f}")
```

```
LGBMClassifier: 0.9360
```

스태킹(Stacking)

- 여러 개의 분류기 결과를 취합하는 마지막 예측기(블렌더 또는 메타학습기)를 학습하는 방법
  - 여러 예측기에서 각각 다른 값을 예측하면, 마지막 예측기(블렌더, blender 또는 메타 학습기, meta learner)가 이 예측을 입력으로 받아 최종 예측
  - 블렌더의 학습
    - 블렌더를 학습시키는 일반적인 방법은 홀드 아웃(hold-out) 세트를 사용하는 것
    - 훈련 세트를 두 개의 서브셋으로 나누고, 첫 번째 서브셋은 첫번째 레이어의 예측을 훈련 시키기 위해 사용
    - 첫번째 예측기를 사용해 두 번째 세트(홀드 아웃)에 대한 예측
- 과적합 문제 해결방법
  - 스태킹 학습 과정에서 교차 검증(Cross Validation)을 추가