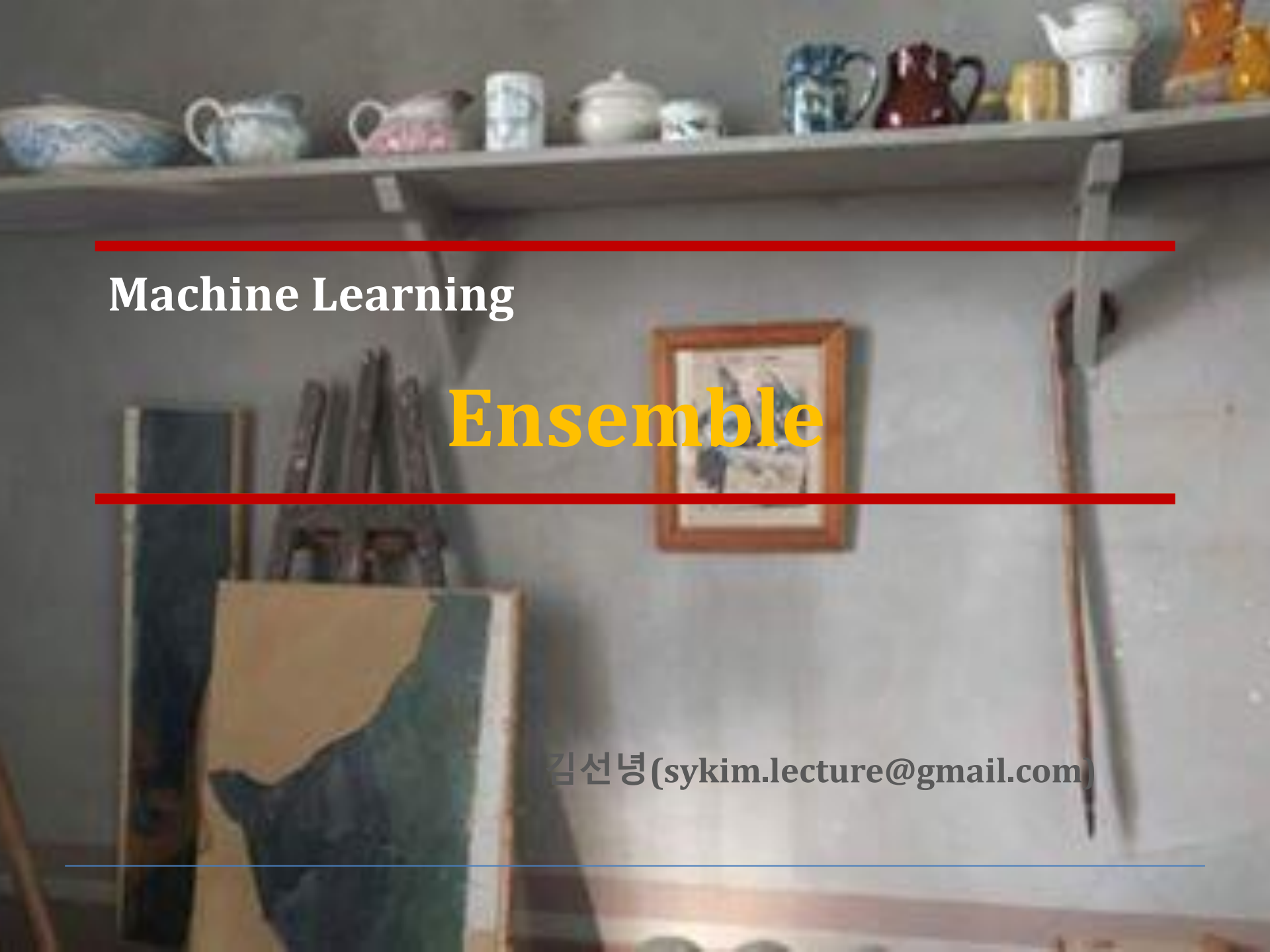


Machine Learning

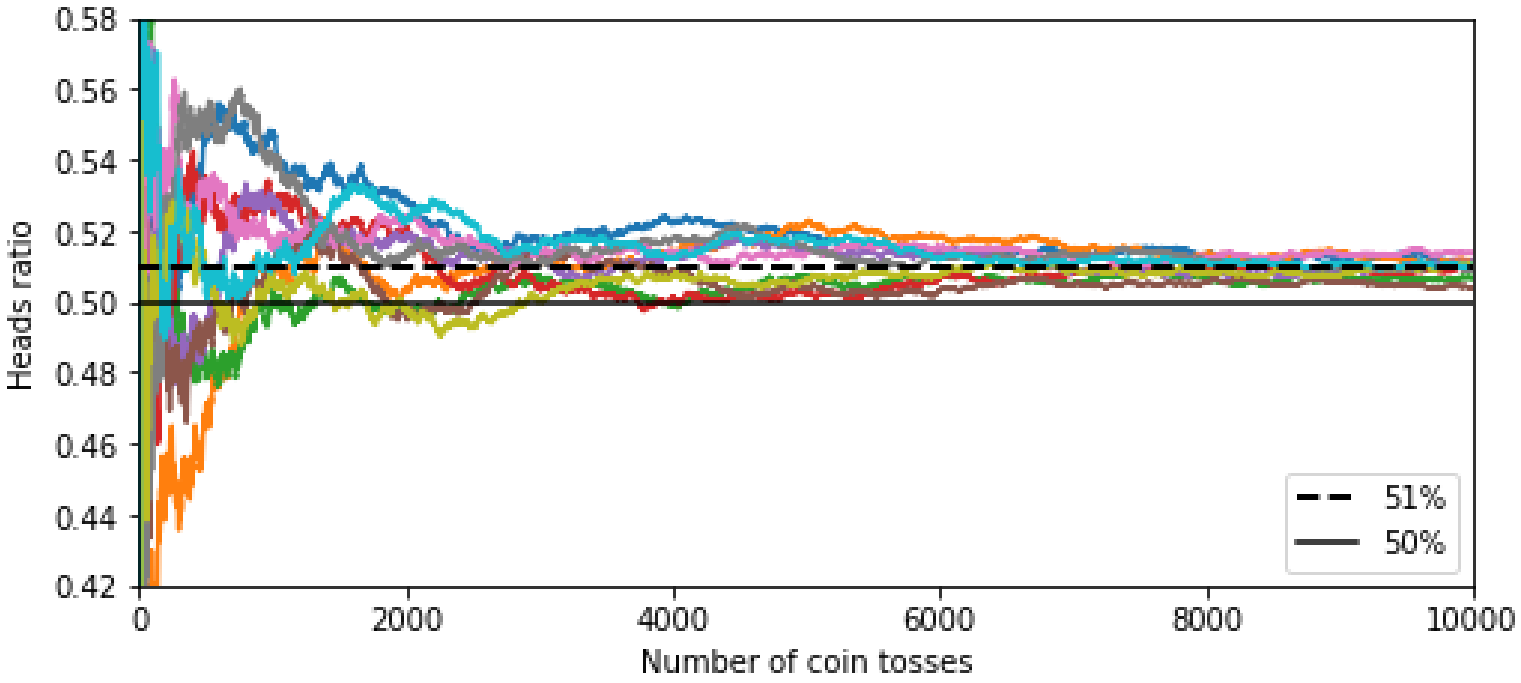
Ensemble

김선녕(sykim.lecture@gmail.com)

투표기반 분류기(Voting Classifiers)
배깅(Bagging)
부스팅(Boosting)
스태킹(Stacking)

강한 학습기

- 각 분류기가 **약한 학습기**(weak learner, 랜덤 추측보다 더 높은 성능을 내는 분류기)일지라도 충분히 많고 다양하다면 앙상블은 **강한 학습기**(strong learner, 높은 정확도를 내는 분류기)가 될 수 있다.
- 동전을 던졌을 때 앞면이 51%, 뒷면이 49%가 나오는 균형이 맞지 않는 동전이 있다고 가정하자.
- 동전을 자꾸 던질수록 앞면이 나올 확률은 51%에 가까워진다.
- 아래 그림은 1~10000번 던진 횟수를 10회 실행한 실험 결과이다.



Ref : Hands-on Machine Learning with Scikit-Learn,keras&tensorflow - O'REILLY

큰 수의 법칙(law of large numbers)

$\lim_{n\to\infty} P\left(\left|\frac{x}{n}-p\right|<\epsilon\right)=1$, $\frac{x}{n}$: 사건 A가 발생한 횟수(통계적확률), p : 사건 A가 발생할 수학적 확률

(수학적 확률과 통계적 확률의 차이가 임의의 양수 ϵ 보다 작을 확률은 n이 커짐에 따라 1에 가까워진다)

- 한 개의 주사위를 n회 던지는 시행에서 1의 눈이 나오는 횟수를 x라 하자. n이 $10, 30, 50$인 경우에 대해서 $P\left(\left|\frac{x}{n}-\frac{1}{6}\right|<0.1\right)$를 구해보자.

이항분포 $B(n,p)$에 따라 $B\left(10,\frac{1}{6}\right)$

${}_{10}C_1\left(\frac{1}{6}\right)^1\left(\frac{5}{6}\right)^9=0.323$ ${}_{10}C_2\left(\frac{1}{6}\right)^2\left(\frac{5}{6}\right)^8=0.291$

① $n=10$

$$\left|\frac{x}{10}-\frac{1}{6}\right|<0.1 \Leftrightarrow \frac{2}{3}<x<\frac{8}{3}$$

$$P\left(\left|\frac{x}{10}-\frac{1}{6}\right|<0.1\right)=P(x=1)+P(x=2)=0.323+0.291=\mathbf{0.614}$$

② $n=30$

$$\left|\frac{x}{30}-\frac{1}{6}\right|<0.1 \Leftrightarrow 2<x<8$$

$$P\left(\left|\frac{x}{30}-\frac{1}{6}\right|<0.1\right)=\sum_{k=3}^{7}P(x=k)=\mathbf{0.784}$$

③ $n=50$

$$\left|\frac{x}{50}-\frac{1}{6}\right|<0.1 \Leftrightarrow \frac{10}{3}<x<\frac{40}{3}$$

$$P\left(\left|\frac{x}{50}-\frac{1}{6}\right|<0.1\right)=\sum_{k=4}^{13}P(x=k)=\mathbf{0.946}$$

앙상블

- 여러 개의 분류기를 하나의 분류기로 연결하여 개별 분류기보다 더 좋은 일반화 성능을 달성
- 앙상블 방법은
  - 예측기가 가능한 한 서로 독립적일 때 최고의 성능 발휘
  - 다양한 분류기를 각기 **다른 알고리즘**으로 학습시키는 것
    - 다른 종류의 오차를 만들 가능성이 높기 때문에 앙상블 모델의 정확도 향상
    - 투표기반(voting) : 직접투표(hard voting), 간접투표(soft voting)
  - **같은 알고리즘**을 사용하고 훈련 세트의 서브셋을 무작위로 구성하여 분류기를 각기 다르게 학습시기는 것
    - 배깅(boostrap aggregating, bagging)과 페이스팅(pasting)
    - 부스팅(boosting)

투표기반 분류기(Voting Classifiers) : 직접투표(hard voting)

- 각 분류기의 예측을 모아서 가장 많이 선택된 클래스를 예측. 다수결 투표

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

w_j : 개별분류기 C_j 에 연관된 가중치

χ : 특성함수$[C_j(x) = i \in A]$,

A : 고유한 클래스 레이블 집합

가중치가 동일하면

$\hat{y} = mode\{C_1(x), C_2(x), \cdots, C_m(x)\}$, $mode$: 최빈값

- (예) 세 개의 분류기 $C_j (j \in \{0,1\})$

  샘플 x의 클래스 레이블을 예측 : $C_1(x) \rightarrow 0, C_2(x) \rightarrow 0, C_3(x) \rightarrow 1$

  – 가중치 동일한 경우 :

  $\hat{\boldsymbol{y}} = \boldsymbol{mode}\{\mathbf{0}, \mathbf{0}, \mathbf{1}\} = \mathbf{0}$

  – 가중치 동일하지 않은 경우 : $C_1 = 0.2, C_2 = 0.2, C_3 = 0.6$

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

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

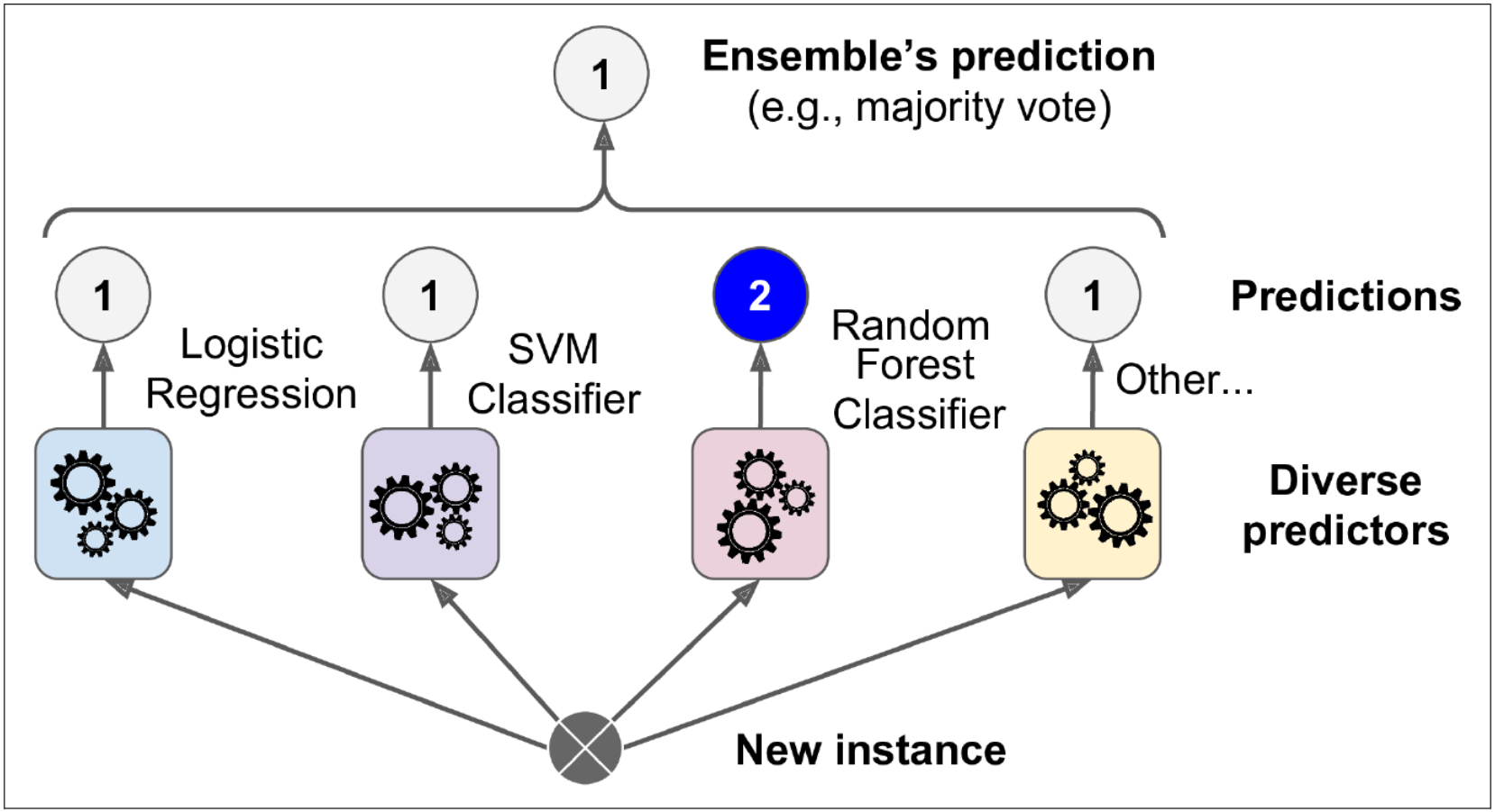
$$\hat{\boldsymbol{y}} = \boldsymbol{mode}\{\mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{1}, \mathbf{1}\} = \mathbf{1}$$

i_0 : 클래스 0일 때 1이고, 그 외에는 0

i_1 : 클래스 1일 때 1이고, 그 외에는 0

직접투표(Hard Voting)

- 다수결 투표에 따라 1선택



Ref : Hands-on Machine Learning with Scikit-Learn,keras&tensorflow - O'REILLY

투표기반 분류기(Voting Classifiers) : 간접 투표 (soft voting)

- 개별 분류기의 예측에서 각 클래스의 **확률**이 가장 높은 클래스를 예측

$$\hat{y} = \underset{i}{\operatorname{argmax}} \sum_{j=1}^{m} w_j p_{ij}$$ (p_{ij} : 클래스 레이블i에 대한 j번째 분류기 예측확률)

- (예) 레이블 $i \in \{0,1\}$, 세개의 분류기 $C_j (j \in \{1,2,3\})$

  가중치 : $C_1 = 0.2, C_2 = 0.2, C_3 = 0.6$

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

  $p(i_0|x) = 0.2 \times 0.9 + 0.2 \times 0.8 + 0.6 \times 0.4 = 0.58$

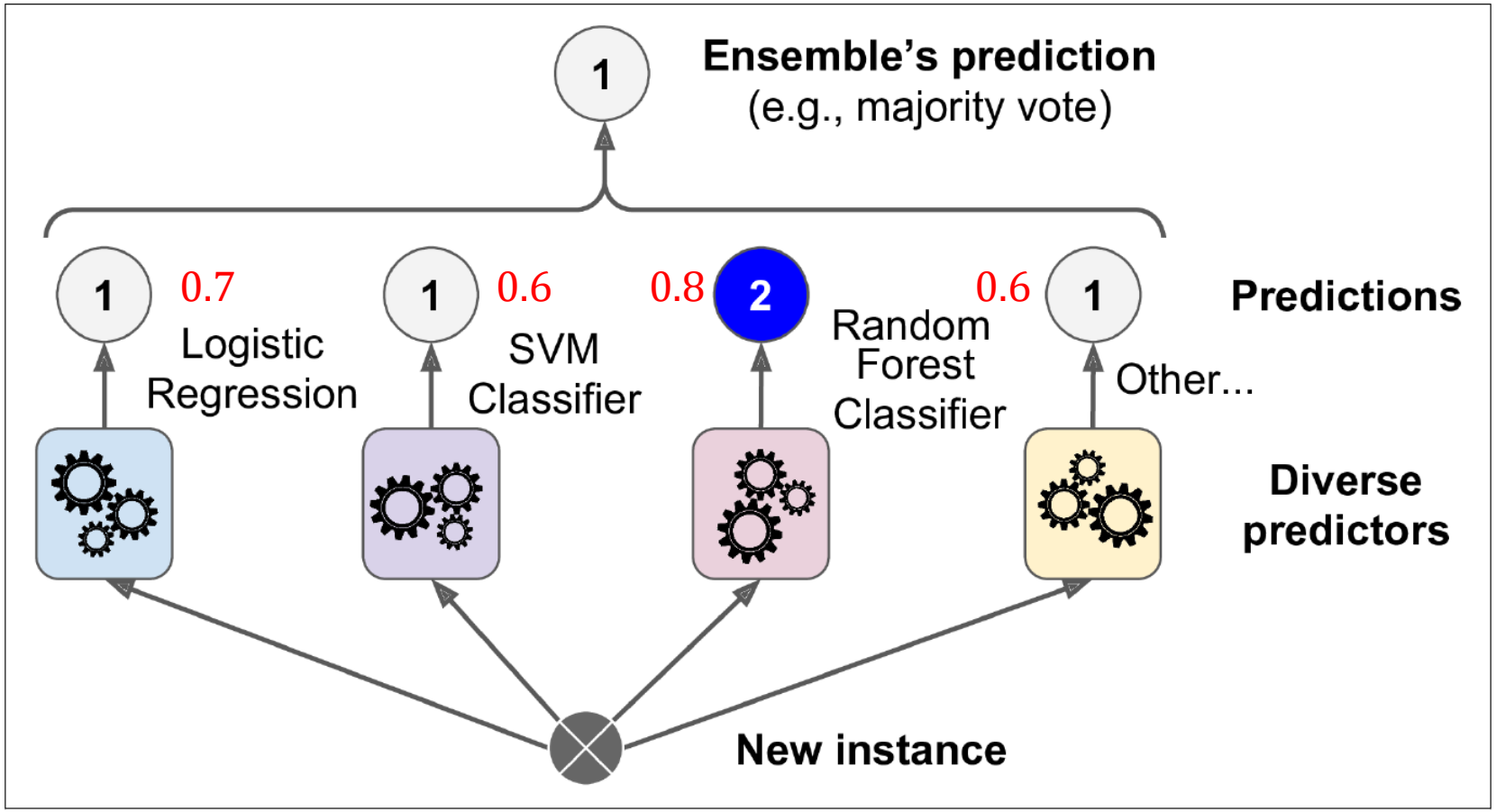
  $p(i_1|x) = 0.2 \times 0.1 + 0.2 \times 0.2 + 0.6 \times 0.6 = 0.42$

  $\hat{\boldsymbol{y}} = [\boldsymbol{p}(\boldsymbol{i_0}|\boldsymbol{x}), \boldsymbol{p}(\boldsymbol{i_1}|\boldsymbol{x})] = [\mathbf{0.58}, \mathbf{0.42}] = \mathbf{0}$

간접 투표 (Soft Voting)

$$p(1) = \frac{0.7 + 0.6 + 0.2 + 0.6}{4} = 0.525$$

$$p(2) = \frac{0.3 + 0.4 + 0.8 + 0.4}{4} = 0.475$$



Ref : Hands-on Machine Learning with Scikit-Learn,keras&tensorflow - O'REILLY

VotingClassfier


```
# 여러 분류기를 조합하여 사이킷런의 투표 기반 분류기를 만들고 훈련시키는 코드
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons
# moons 데이터셋
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)
# svm_clf = SVC(gamma="scale", random_state=42, probability=True) #soft경우(확률제공)
voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='hard') # 간접투표방식:voting='soft'

voting_clf.fit(X_train, y_train)
from sklearn.metrics import accuracy_score
# 각 분류기 정확도
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

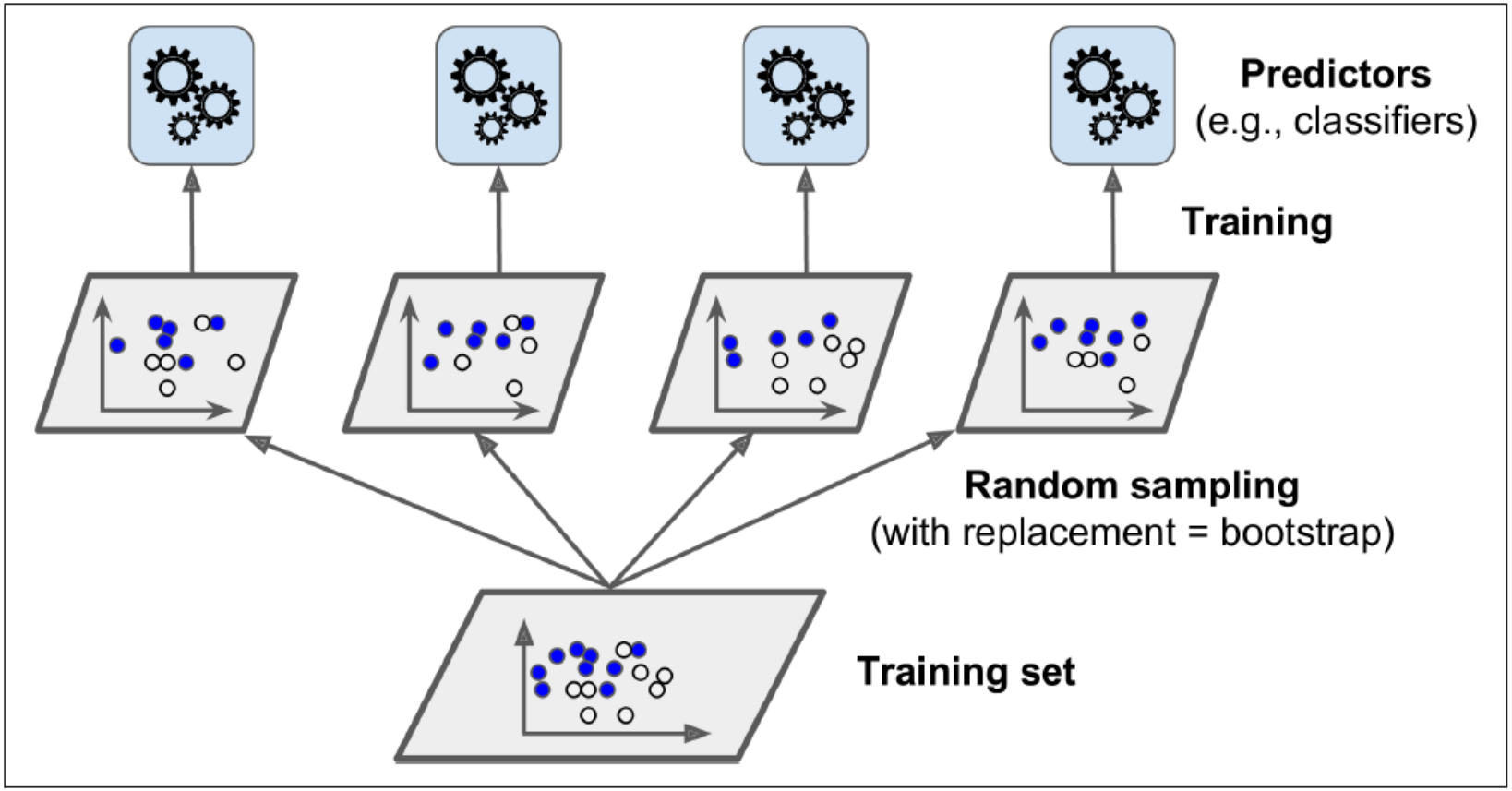
투표기반 분류기(Voting Classifiers)
배깅(Bagging)
부스팅(Boosting)
스태킹(Stacking)

배깅과 페이스팅

- 동일한 훈련 샘플을 여러 개의 예측기에서 사용
- 배깅(bootstrap aggregating, bagging)
  - 훈련세트에서 **중복을 허용**하여 샘플링하는 방식
- 페이스팅(pasting)
  - **중복을 허용하지 않고** 샘플링하는 방식
- 수집함수
  - 훈련된 모든 예측기의 예측을 수집
  - 전형적인 분류인 경우 : **통계적 최빈값**(다수결 투표)
  - 회귀인 경우 : **평균** 계산
  - 수집함수를 통과하면 편향과 분산이 감소
- 일반적인 앙상블의 결과는 원본 데이터셋으로 하나의 예측기를 훈련시킬 때와 비교해 편향은 비슷하지만 분산은 줄어든다.
- 배깅이 페이스팅보다 더 나은 모델을 만들기 때문에 일반적으로 **배깅을 선호**

배깅과 페이스팅

- 확장성 : 모두 동시에 다른 CPU코어나 서버에서 병렬로 학습



Ref : Hands-on Machine Learning with Scikit-Learn,keras&tensorflow - O'REILLY

사이킷런의 배깅과 페이스팅(1)


```
# 결정트리분류기 500개의 앙상블을 훈련시키는 코드
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

# 배깅 : 각 분류기는 훈련세트에서 중복을 허용하여 무작위로 선택된 100개의 샘플
# 페이스팅 : bootstrap=False
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)

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

0.904

```
# 단일결정트리
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))
```

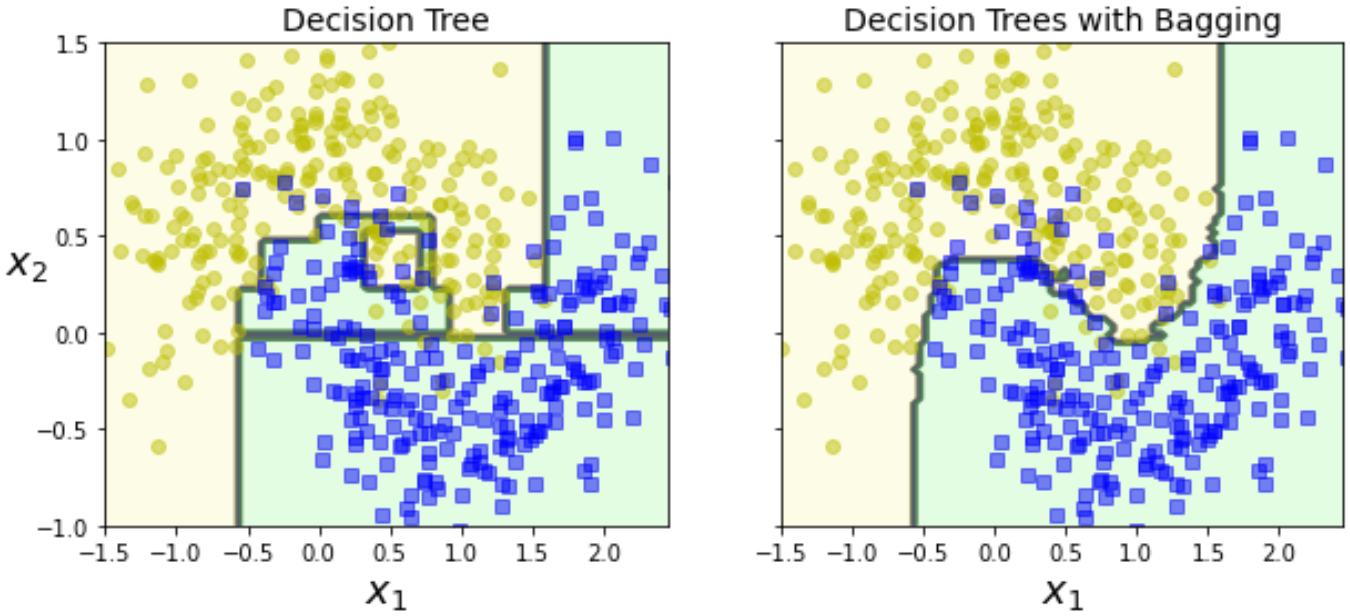
0.856

사이킷런의 배깅과 페이스팅(2)


```
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=0.3, 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)
```

사이킷런의 배깅과 페이스팅(3)


```
fix, axes = plt.subplots(ncols=2, figsize=(10,4), 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()
```



- 앙상블은 비슷한 편향에서 더 작은 분산 생성
- 즉, 오차 수가 거의 비슷하지만 결정경계는 더 일반화

Ref : Hands-on Machine Learning with Scikit-Learn,keras&tensorflow - O'REILLY

oob(out of bag) 평가

- 선택되지 않은 샘플을 사용하여 평가
- 평균적으로 각 예측기에 훈련 샘플의 63%정도만 샘플링
- oob(out of bag)
  - 선택되지 않은 샘플의 나머지 37%
  - 남겨진 샘플은 예측기마다 다름
- oob 샘플을 사용해 평가
  - 각 예측기는 oob평가를 평균하여 앙상블을 평가
  - oob_score=True 지정
- 장점
  - 추가적인 검증세트가 없어도 편향되지 않게 앙상블을 평가하도록 도와준다
  - 따라서 훈련에 더 많은 샘플을 사용할 수 있어서 앙상블의 성능은 조금 더 향상된다

포함되지 않을 확률

· n개의 샘플에서 하나를 무작위로 추출했을 때 선택되지 않을 확률 : $\mathbf{1-\frac{1}{n}}$

· n번 반복했을 때 한번도 선택되지 않을 확률 y : $y=\left(1-\frac{1}{n}\right)^n$

· 양변에 ln

$$ln(y)=ln\left(1-\frac{1}{n}\right)^n=n\cdot ln\left(1-\frac{1}{n}\right)$$

· $x=\frac{1}{n}$로 치환

$$ln(y)=\frac{1}{x}ln(1-x)=\frac{ln(1-x)}{x}$$

· $n\to\infty$, 즉$x\to 0$ 일 때 로피탈 정리(도함수를 통해 부정형의 극한을 구하는 정리) 적용

$$\boldsymbol{ln(y)}=\lim_{x\to 0}\frac{ln(1-x)}{x}=\lim_{x\to 0}\frac{\frac{d}{\delta x}ln(1-x)}{\frac{d}{\delta x}x}=\lim_{x\to 0}\frac{-1}{1-x}=-1$$

$$\boldsymbol{y=e^{-1}=0.3678794\cdots}$$

· n개의 샘플이 아주 클 때, 무작위로 n번의 선택에 한번도 포함되지 않을 확률 : 36.8%

(예) $n=3$이면

$$y=\left(1-\frac{1}{3}\right)^3=0.296296\cdots$$

OOB 평가


```
# OOB 평가
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(random_state=42), n_estimators=500,
    bootstrap=True, oob_score=True, random_state=40)
bag_clf.fit(X_train, y_train)
# 평가점수 결과는 oob_score_ 변수에 저장
bag_clf.oob_score_
```

```
0.9013333333333333
```

```
# 테스트 세트에서의 정확도
from sklearn.metrics import accuracy_score
y_pred = bag_clf.predict(X_test)
accuracy_score(y_test, y_pred)
```

```
0.912
```

```
# oob 샘플에 대한 결정 함수(각 훈련 샘플의 클래스 확률 반환)
#   - oob_decision_function_ 변수에서 확인
# 첫번째 훈련 샘플이 양성 클래스에 속할 확률:68.25%, 음성클래스 : 31.75%
bag_clf.oob_decision_function_
```

```
array([[0.31746032, 0.68253968],
       [0.34117647, 0.65882353],
       ...
       [0.03108808, 0.96891192],
       [0.57291667, 0.42708333]])
```

랜덤 포레스트

- 랜덤 포레스트는 배깅방법(또는 페이스팅)을 적용한 결정트리 앙상블
- `RandomForestClassifier` 클래스
  - `BaggingClassifier(DecisionTreeClassifier(...))`대신 결정트리에 최적화되어 사용하기 편리
  - `RandomForestRegressor` : 회귀문제

배깅방법(또는 페이스팅)을 적용한 결정트리 앙상블


```
# 랜덤 포레스트
from sklearn.ensemble import RandomForestClassifier
# 최대 16개의 리프노드를 갖는 500개 트리로 이루어진 랜덤 포레스트 분류기를
# 여러 CPU코어에서 훈련시키는 코드(n_jobs=-1이면 시스템의 CPU코어 모두 사용)
rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16
                                 n_jobs=-1, random_state=42)
rnd_clf.fit(X_train, y_train)
y_pred_rf = rnd_clf.predict(X_test)

# 배깅분류(BaggingClassifier)를 사용하여
# 랜덤 포레스트(RandomForestClassifier)와 유사하게 만든 것
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(max_leaf_nodes=16, random_state=42),
    n_estimators=500, max_samples=1.0, bootstrap=True, random_state=42)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)

np.sum(y_pred == y_pred_rf) / len(y_pred)  # 예측이 거의 동일.
```

특성중요도

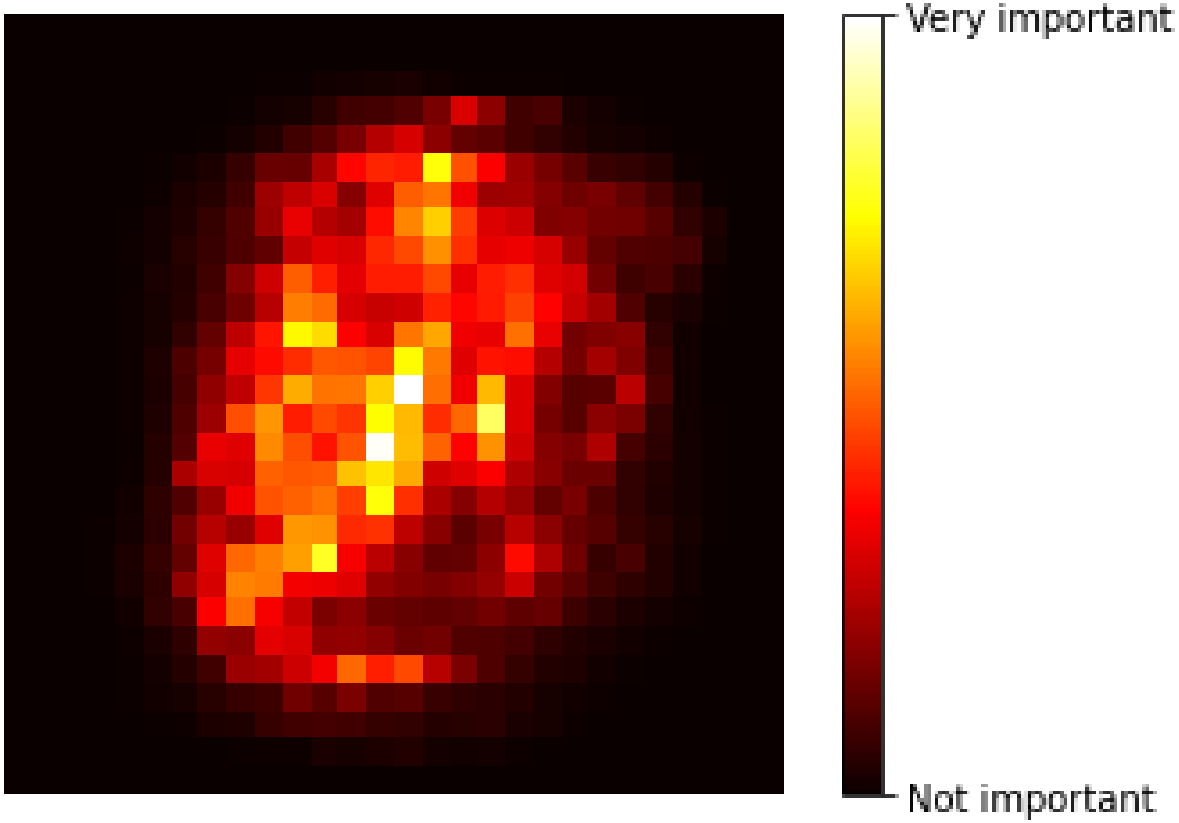
- 랜덤 포레스트의 또 다른 장점
  - 특성의 상대적 중요도를 측정하기 쉽다
  - 특성을 선택해야 할 때 어떤 특성이 중요한지 빠르게 확인할 수 있다
- 사이킷런은 훈련이 끝난 뒤 특성마다 자동으로 점수 계산
  - 중요도의 전체합이 1이 되도록 결과값을 정규화한다
  - feature_importances_ 변수에 정규화된 중요도 결과값 저장 (합:1)

```
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

가장 중요한 특성은 꽃잎의 길이(44%)와 너비(42%)
꽃받침의 길이와 너비는 비교적 덜 중요(11%, 2%)

MNIST 데이터셋에 랜덤 포레스트 분류기를 훈련시키고 각 픽셀의 중요도



Ref : Hands-on Machine Learning with Scikit-Learn,keras&tensorflow - O'REILLY

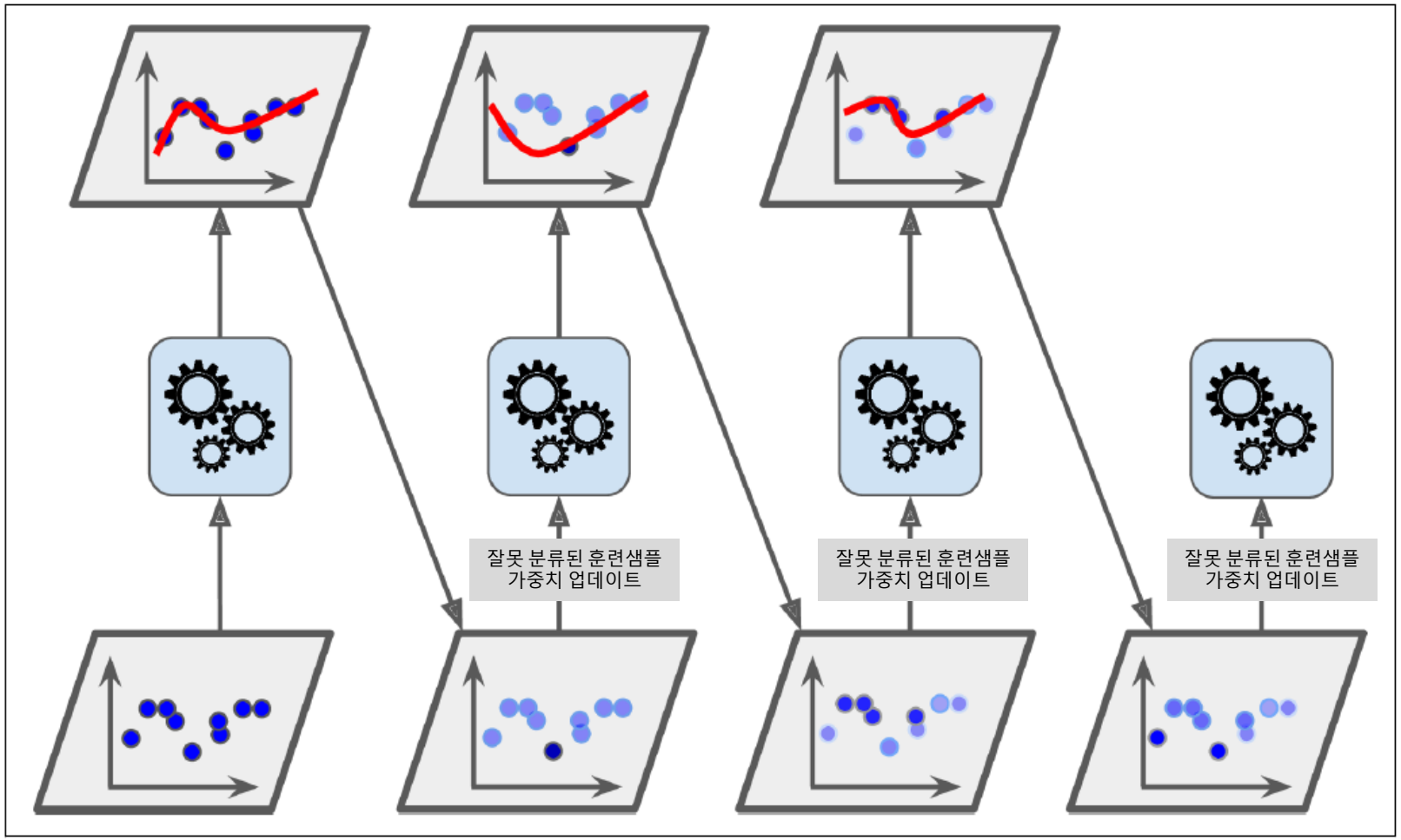

투표기반 분류기(Voting Classifiers)
배깅(Bagging)
부스팅(Boosting)
스태킹(Stacking)

부스팅(Boosting)

- 약한 학습기를 여러 개 연결하여 강한 학습기를 만드는 앙상블
  - 약한학습기의 전형적인 예 : 깊이가 1인 결정트리
- 앞의 모델을 보완해 나가면서 예측기를 학습시키는 것
- 부스팅 방법
  - 에이다부스트(Adaptive Boosting, AdaBoost) : 반복마다 샘플의 가중치 수정
  - 그레디언트 부스팅(Gradient Boosting) : 이전 예측기가 만든 잔여오차에 새로운 예측기를 학습

에이다부스트(AdaBoost)

- 이전 모델이 과소적합했던 훈련 샘플의 가중치를 더 높이는 것



Ref : Hands-on Machine Learning with Scikit-Learn,keras&tensorflow - O'REILLY

부스팅의 작동원리

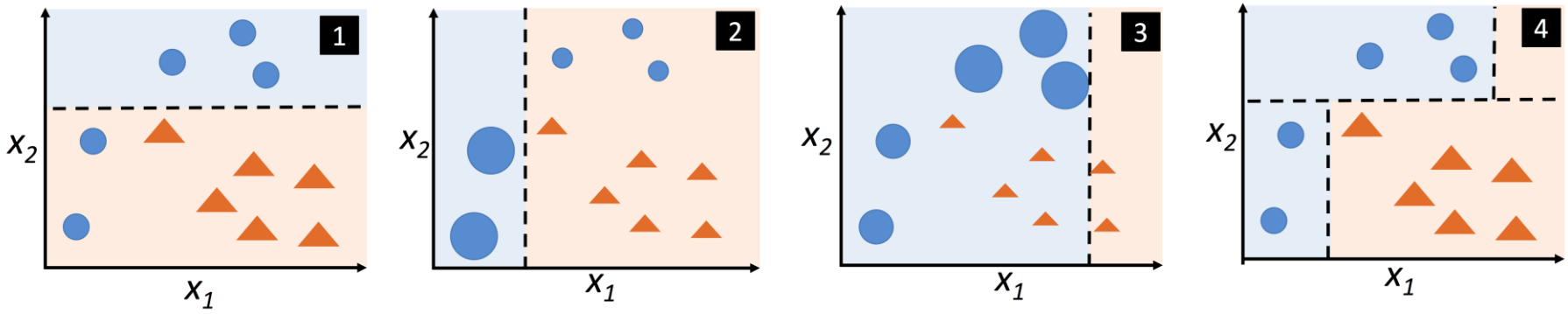
① 훈련세트 D에서 중복을 허용하지 않고 랜덤한 부분 집합 d_1을 뽑아 약한 학습기 C_1 을 훈련한다.

② 훈련세트에서 중복을 허용하지 않고 두 번째 랜덤한 훈련 부분 집합 d_2를 뽑고 이전에 분류된 샘플의 50%를 더해서 약한 학습기 C_2을 훈련한다.

③ 훈련세트 D에서 C_1와 C_2에서 잘못 분류된 훈련 샘플 d_2를 찾아 세 번째 약한 학습기인 C_3를 훈련한다

④ 약한학습기 C_1, C_2, C_3를 다수결 투표로 연결한다

에이다부스트의 기본 개념



- 세 번의 부스팅 단계만을 가진다고 가정
- 그림 1 : 이진분류를 위한 훈련세트
  - 모든 샘플은 동일한 가중치 설정
  - 깊이가 1인 결정트리(파선) 훈련
- 그림 2 : 이전에 잘못 분류된 샘플 두 개(원)에 **큰 가중치 부여**
  - 옳게 분류된 샘플의 가중치는 낮춘다
  - 다음 결정 트리는 가장 큰 가중치를 가진 훈련 샘플에 더 집중
- 그림 3 : 이전에 잘못 분류된 세 개의 원 모양 샘플에 **큰 가중치 부여**
- 그림 4 : 서로 다른 가중치가 부여된 훈련 세트에서 훈련된 세개의 약한 학습기를 다수결 투표 방식으로 합친다

에이다부스트 알고리즘

1. 가중치 벡터 w를 동일한 가중치로 설정. 즉 $\frac{1}{m}$로 초기화 : $\sum_{i=1}^{m} w_i = 1$

2. n번 부스팅 반복의 j번째에서 다음을 수행
   - 가중치가 부여된 약한 학습기 훈련 : $C_j = (X, y, w)$
   - 첫번째 클래스 레이블 예측 : $\hat{y} = predict(C_j, X)$
   - 가중치가 적용된 에러율 계산 : $\varepsilon = w \cdot (\hat{y} \neq y)$ $(\hat{y} \neq y)$:1또는 0으로 구성된 이진 벡터
   - 예측기의 가중치 계산 - η(에타, 학습률, 0.5)
     - 무작위로 예측하여 에러율(ε)이 0.5에 가까워지면 $\frac{1-\varepsilon}{\varepsilon} \approx 1$이 되어 가중치($\alpha$)가 0에 가까워지고,
     - 0.5보다 높으면 $\frac{1-\varepsilon}{\varepsilon} < 1$이 되어 음수가 된다

     $$\alpha_j := \eta log \frac{1-\varepsilon}{\varepsilon}$$
   - 가중치 업데이트 : $w := w \cdot exp(-\alpha_j \times \hat{y} \times y)$ $(\hat{y} \neq y)$
   - 합이 1이 되도록 가중치 정규화

   $$w = \frac{w}{\sum_{i=1}^{m} w_i}$$

3. 최종예측 : 가중치의 합이 가장 큰 클래스(N:예측기의 수)

$$\hat{y}(X) = \underset{k}{argmax} \sum_{\substack{j=1 \\ \hat{y}_j(X)=k}}^{N} \alpha_j$$

계산 샘플

- $label: y_i \in \{1, -1\}$
- $weight$: 가중치는 동일하게 초기화. 합이 1이 되도록 정규화

Sample indices	x	y	Weights	$\hat{y}$(x <= 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

$$\Sigma = 1$$

계산샘플

- 가중치 에러율

$$\varepsilon = 0.1 \times 0 + 0.1 \times 0 + 0.1 \times 0 + 0.1 \times 0 + 0.1 \times 0 + 0.1 \times 0 + 0.1 \times 1 + 0.1 \times 1 + 0.1 \times 1 + 0.1 \times 0 = \frac{3}{10} = 0.3$$

- 가중치 계산

$$\alpha_j = 0.5 \cdot log\left(\frac{1-\varepsilon}{\varepsilon}\right) = 0.5 \cdot log\left(\frac{1-0.3}{0.3}\right) \approx 0.424$$

- 가중치 업데이트

$$w := w \times exp(-\alpha_j \times \hat{y} \times y)$$

$0.1 \times exp(-0.424 \times 1 \times 1) \approx 0.065$ $(\hat{y} = y)$인 경우

$0.1 \times exp(-0.424 \times 1 \times (-1)) \approx 0.153$ $(\hat{y} \neq y)$인 경우

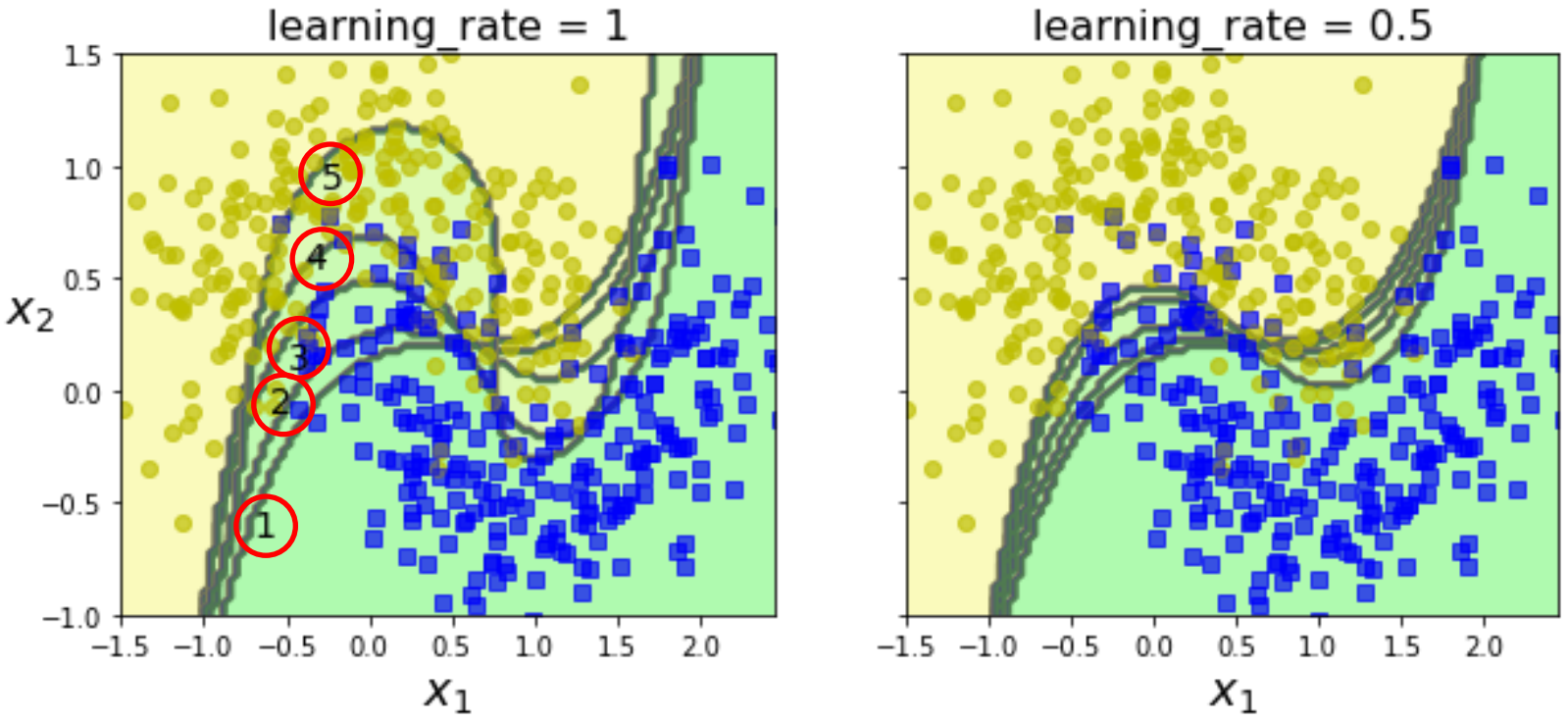
- 합이 1이 되도록 가중치 정규화 $\left(w := \frac{w}{\sum_i^m w_i}\right)$

$$\sum_i^m w_i = 7 \times 0.065 + 3 \times 0.153 = 0.914$$

$\frac{0.065}{0.914} \approx \mathbf{0.071}$(옳게 분류된 샘플에 대응하는 다음단계 가중치는 0.1보다 감소)

$\frac{0.153}{0.914} \approx \mathbf{0.167}$(잘못 분류된 샘플의다음단계 가중치는 0.1보다 증가)

moons 데이터셋 예측기 결정경계



- 첫번째 분류기(1번 붉은색원)는 많은 샘플을 잘못 분류해서 샘플들의 가중치가 높아졌다.
- 두번째 분류기(2번 붉은색원) 부터 점점 더 정확하게 예측
- 오른쪽 그래프 : 학습률만 변경
- 에이다부스트와 경사하강법의 차이
  - 에이다부스트 : 점차 좋아지도록 **앙상블에 예측기를 추가**
  - 경사하강법 : 비용함수를 최소화하기 위해 한 예측기의 모델 파라미터를 조정

```
# 에이다부스트
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)
        # 오류난 인덱스만 찾아서 weight값 조정
        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()
```

그레디언트 부스팅(Gradient Boosting)

- 에이다부스트처럼 반복마다 샘플의 가중치를 수정하는 대신 **이전 예측기가 만든 잔여오차에 새로운 예측기를 학습**시킨다.
- 그레디언트 트리 부스팅(gradient tree boosting)
  또는 그레디언트 부스티드 회귀트리(gradient boosted regression tree, GBRT)
  - 결정트리를 기반 예측기로 사용하는 회귀문제 : DecisionTreeRegressor

```
# 그레디언트 부스팅
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)
# 첫번째 예측기에서 생긴 잔여오차에 두번째 DecisionTreeRegressor를 훈련
y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg2.fit(X, y2)
# 두 번째 예측기에서 생긱 잔여오차에 세 번째 회귀모델을 훈련
y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg3.fit(X, y3)
```

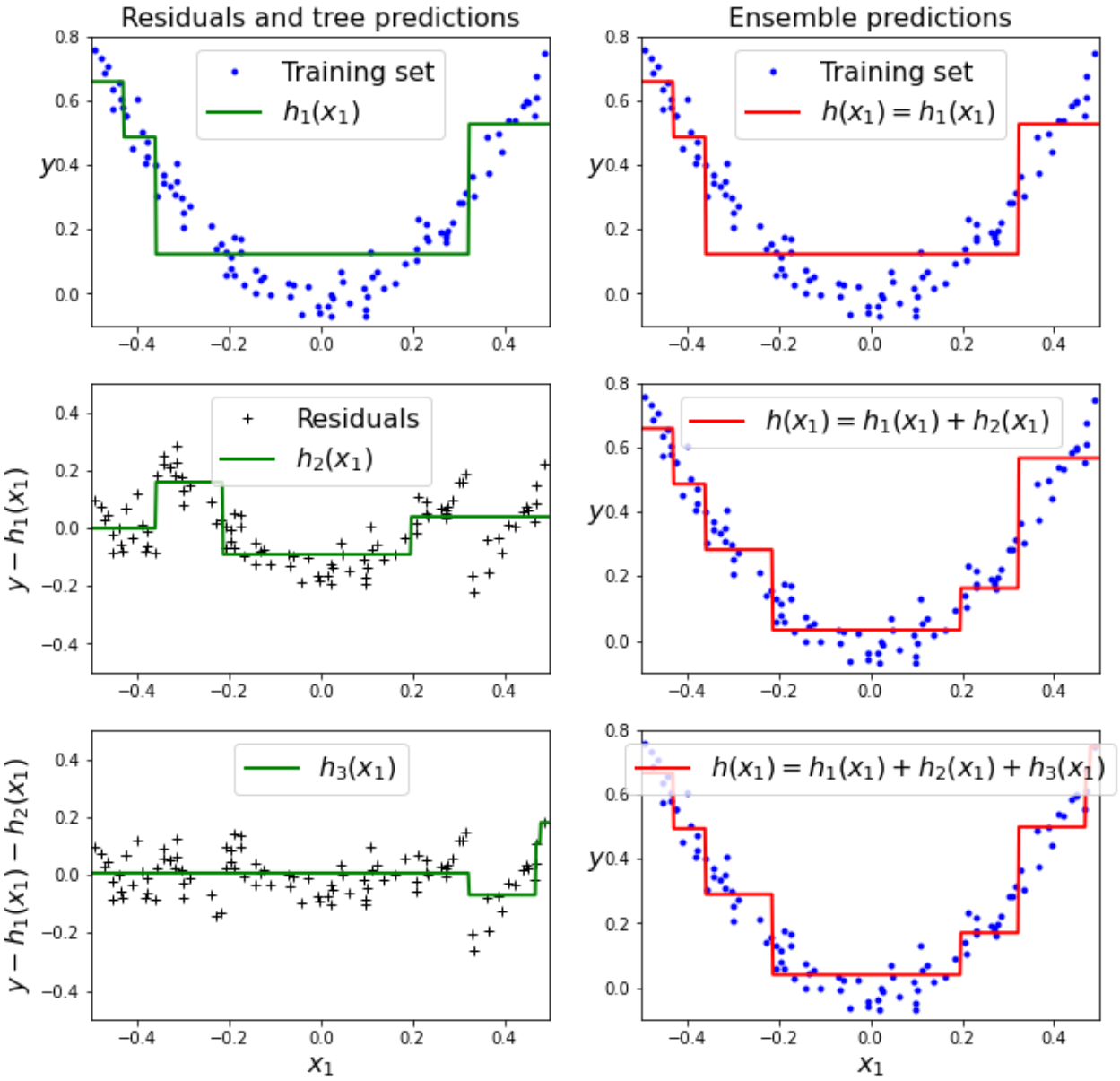
```
# 세 개의 트리를 포함하는 앙상블 모델
# 새로운 샘플에 대한 예측 : 모든 트리의 예측을 더한다
X_new = np.array([[0.8]])
y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
y_pred
```

output : array([0.75026781])

```
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)

plt.subplot(323)
plot_predictions([tree_reg2], X, y2, axes=[-0.5, 0.5, -0.5, 0.5], label="$h_2(x_1)$", style="g-
", data_style="k+", data_label="Residuals")
plt.ylabel("$y - h_1(x_1)$", fontsize=16)

plt.subplot(324)
plot_predictions([tree_reg1, tree_reg2], X, y, axes=[-0.5, 0.5, -
0.1, 0.8], label="$h(x_1) = h_1(x_1) + h_2(x_1)$")
plt.ylabel("$y$", fontsize=16, rotation=0)

plt.subplot(325)
plot_predictions([tree_reg3], X, y3, axes=[-0.5, 0.5, -0.5, 0.5], label="$h_3(x_1)$", style="g-
", data_style="k+")
plt.ylabel("$y - h_1(x_1) - h_2(x_1)$", fontsize=16)
plt.xlabel("$x_1$", fontsize=16)

plt.subplot(326)
plot_predictions([tree_reg1, tree_reg2, tree_reg3], X, y, axes=[-0.5, 0.5, -
0.1, 0.8], label="$h(x_1) = h_1(x_1) + h_2(x_1) + h_3(x_1)$")
plt.xlabel("$x_1$", fontsize=16)
plt.ylabel("$y$", fontsize=16, rotation=0)
plt.show()
```

잔여오차가 줄어들고 있다

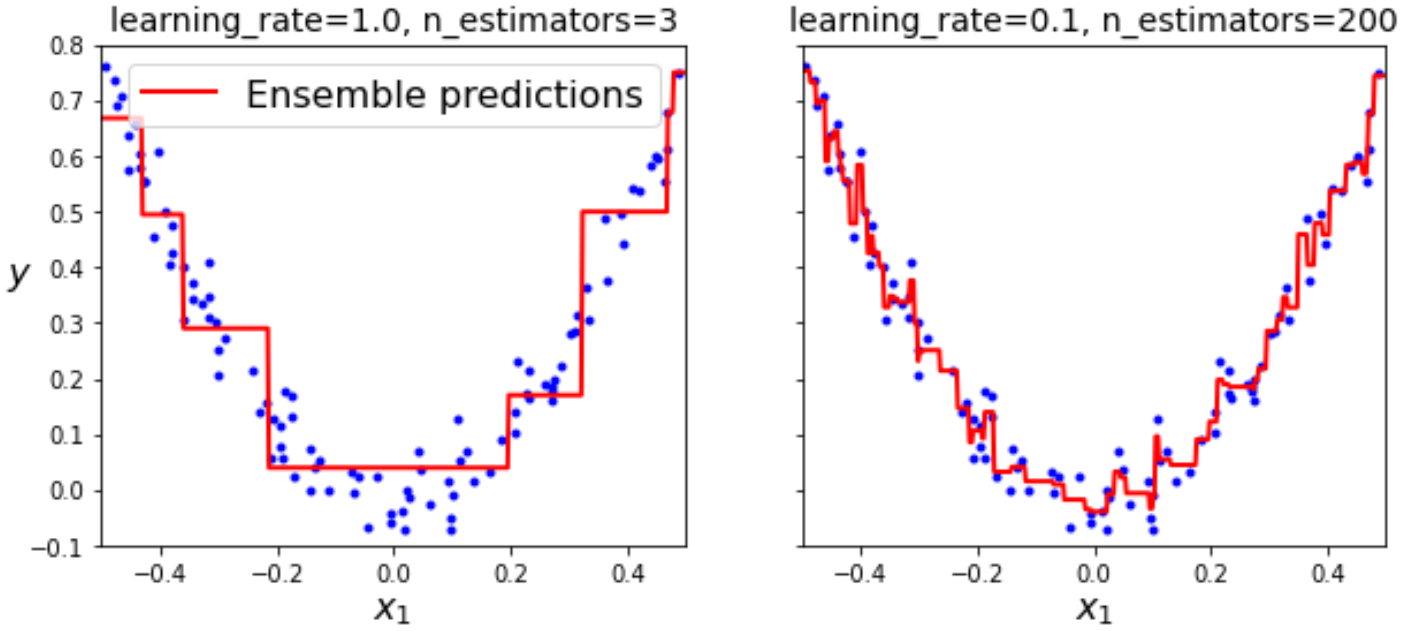


예측은 좋아지고 있다

Ref : Hands-on Machine Learning with Scikit-Learn,keras&tensorflow - O'REILLY

learning_rate 매개변수

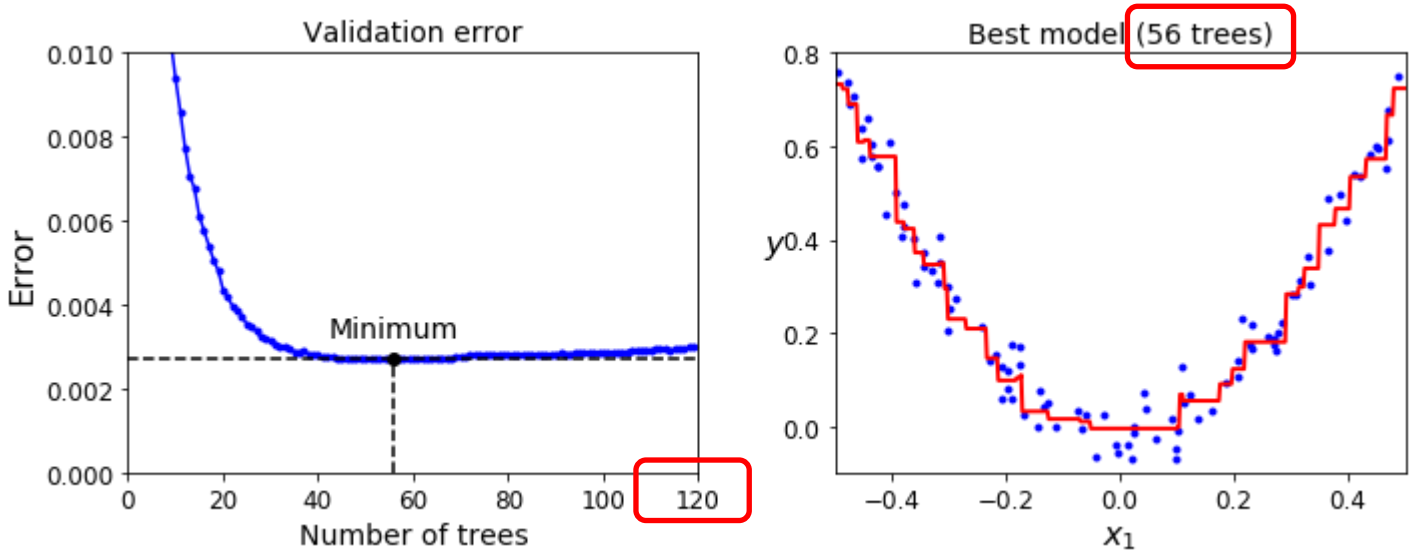
- 각 트리의 기여 정도 조절
- a trade-off between learning_rate and n_estimators
- 트리가 충분하지 않은 반면(왼쪽), 트리가 너무 많아 과대적합(오른쪽)



Ref : Hands-on Machine Learning with Scikit-Learn,keras&tensorflow - O'REILLY

최적의 트리수

- 검증 오차 측정
  - (예제) 120개의 트리로 앙상블 훈련시키고 최적의 트리수를 찾기 위해
- staged_predict() 메서드 : 훈련의 각 단계(트리 하나, 트리 두 개등)에서 앙상블에 의해 만들어진 예측기를 순회하는 반복자를 반환



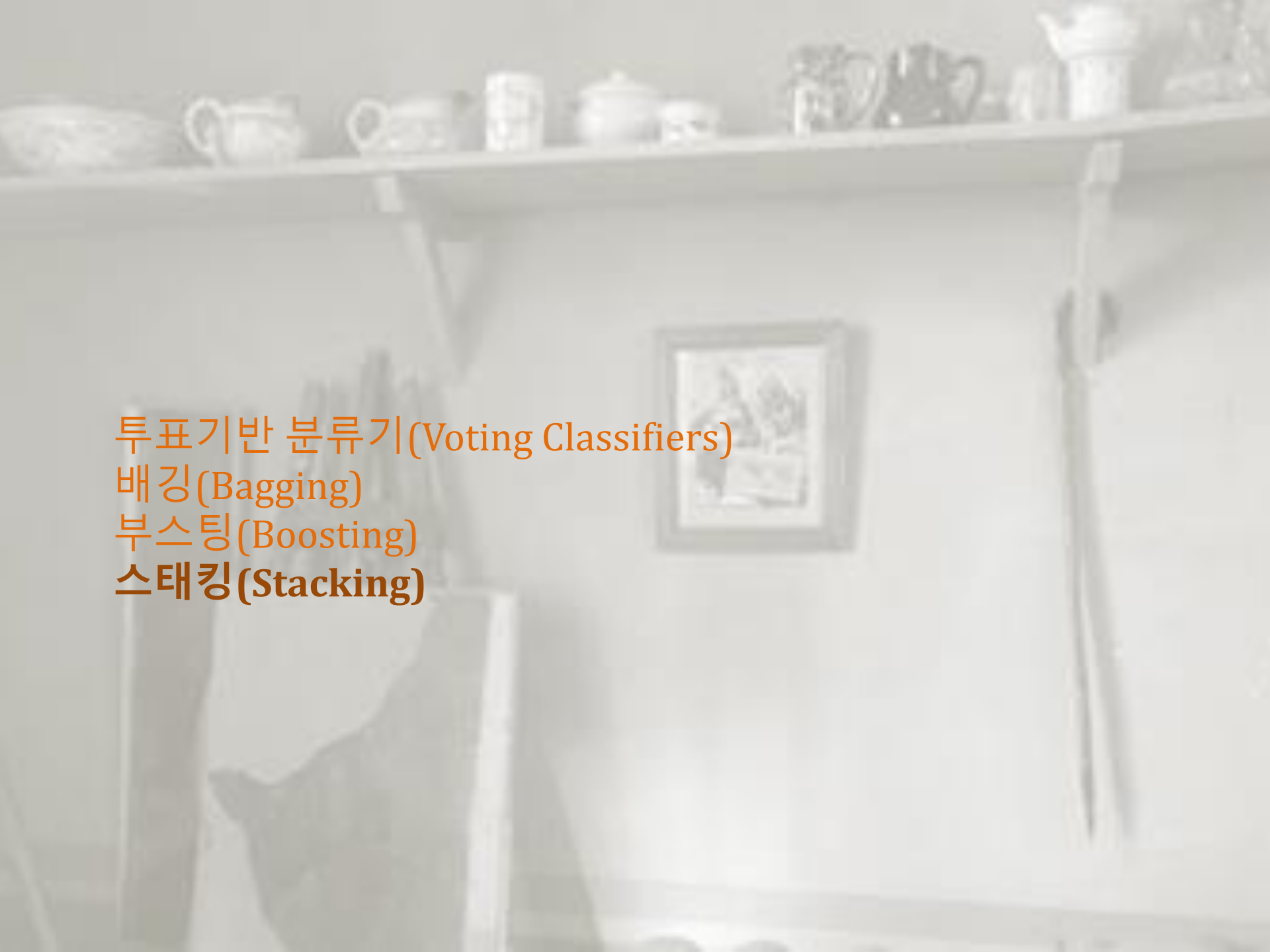
Ref : Hands-on Machine Learning with Scikit-Learn,keras&tensorflow - O'REILLY

최적의 트리수


```
# 최적의 트리수
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_s
tate=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_es
timators, random_state=42)
gbrt_best.fit(X_train, y_train)

min_error = np.min(errors)
```

최적의 트리수

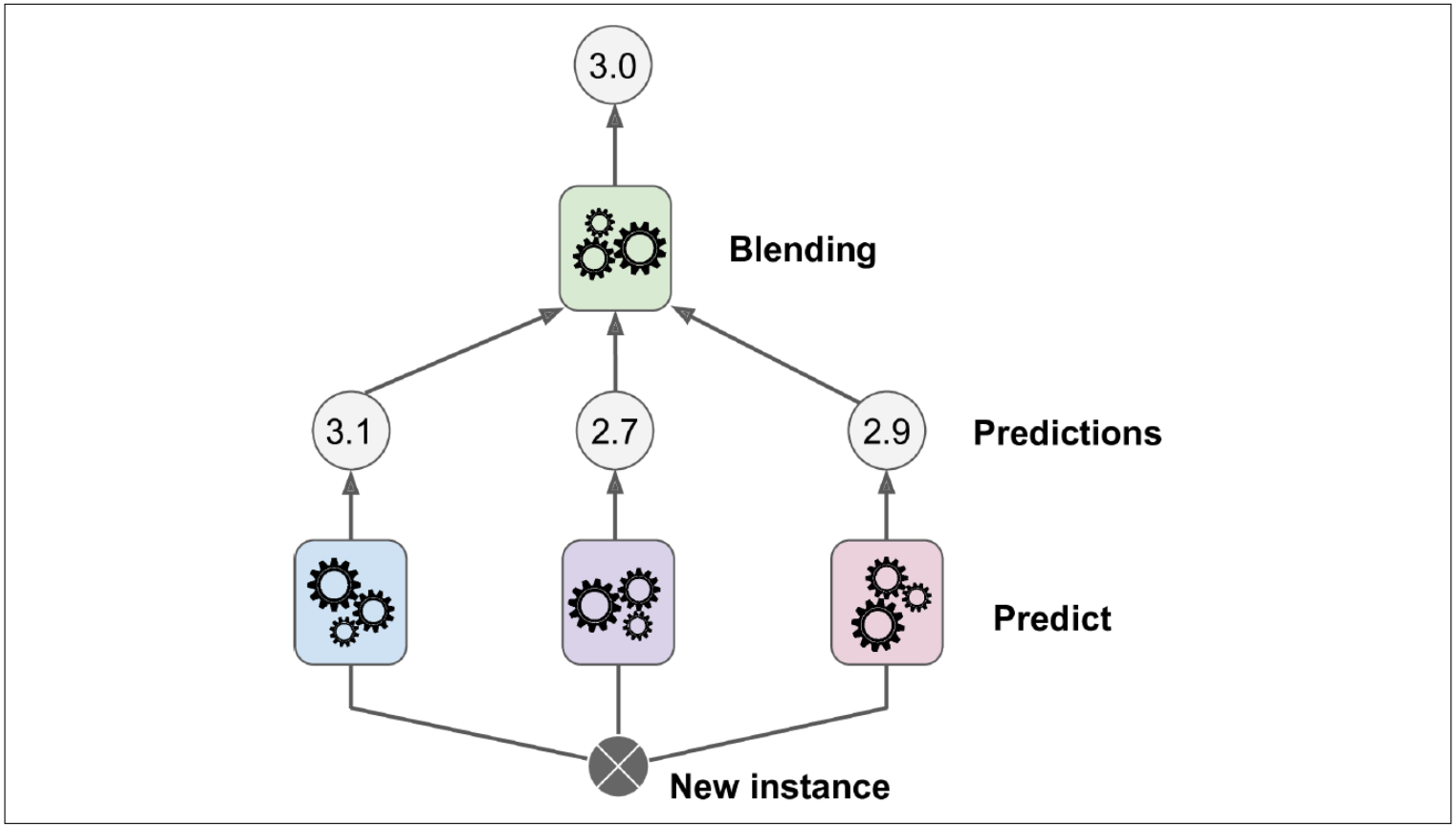

```
plt.figure(figsize=(10, 4))

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", font
size=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([gbrt_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()
```



투표기반 분류기(Voting Classifiers)
배깅(Bagging)
부스팅(Boosting)
스태킹(Stacking)

스태킹(staking, stacked generalization)

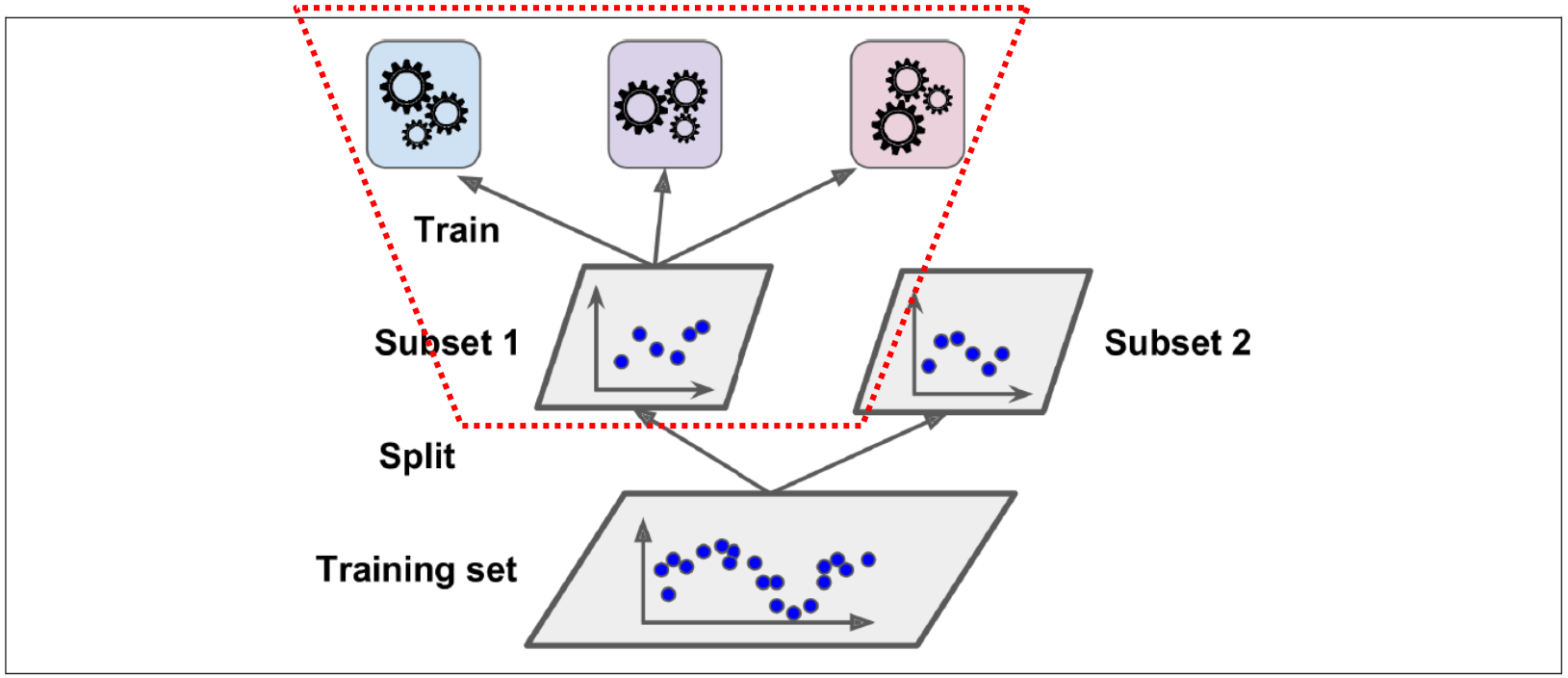
- 기본아이디어
  - 앙상블에 속한 모든 예측기의 예측을 취합하는 간단한 함수(직접투표같은)를 사용하는 대신 취합하는 모델을 훈련시킬 수 없을까?
- 세 예측기는 각각 3.1, 2.7, 2.9를 예측하고 **마지막 예측기**(블렌더 또는 메타학습기)가 이 예측들을 입력으로 받아 최종 예측(3.0)



Ref : Hands-on Machine Learning with Scikit-Learn,keras&tensorflow - O'REILLY

작동원리(1)

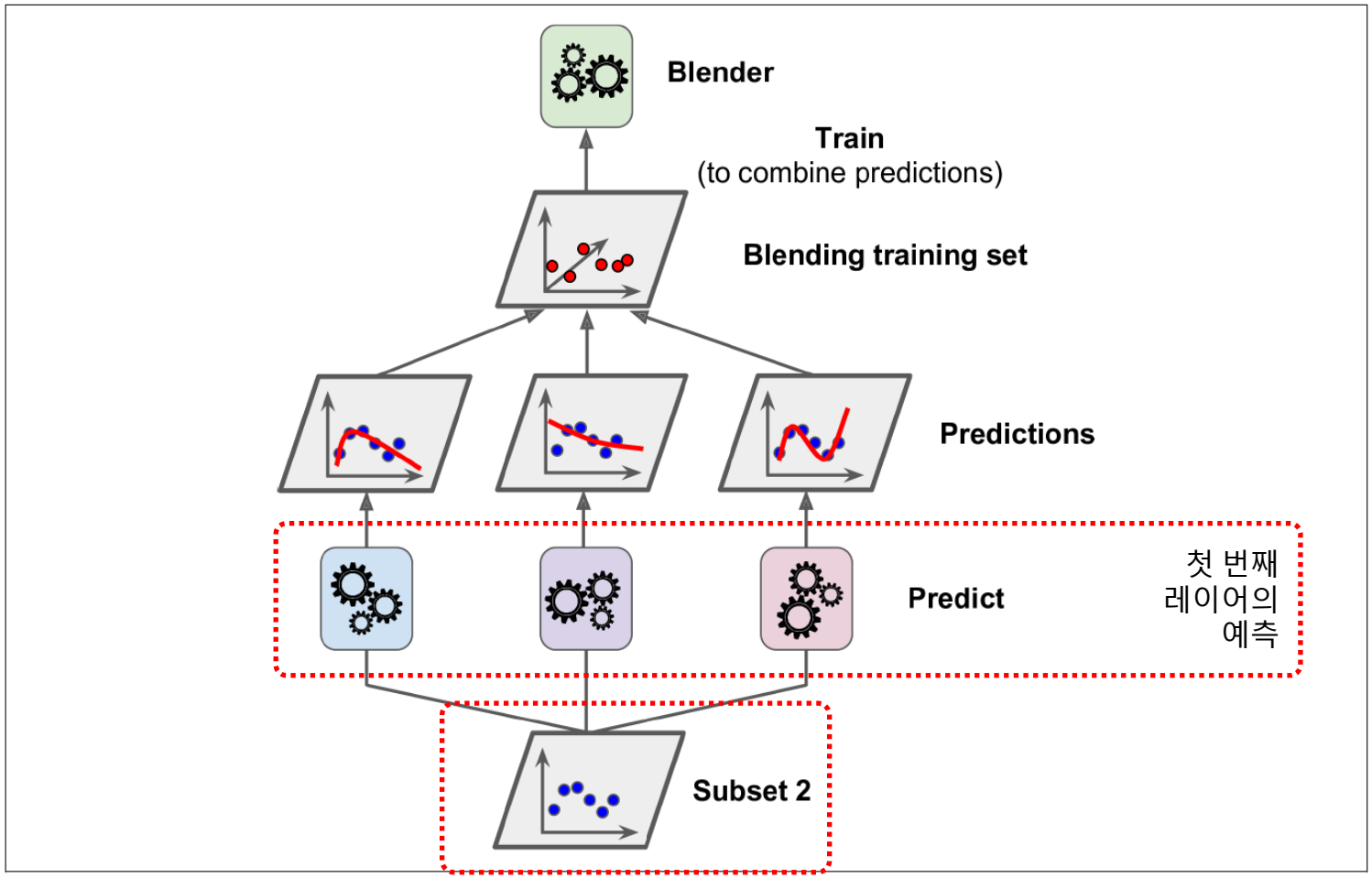
- 훈련세트를 두 개의 서브셋으로 나눈다
- 첫 번째 서브셋 - 첫 번째 레이어의 예측을 훈련



Ref : Hands-on Machine Learning with Scikit-Learn,keras&tensorflow - O'REILLY

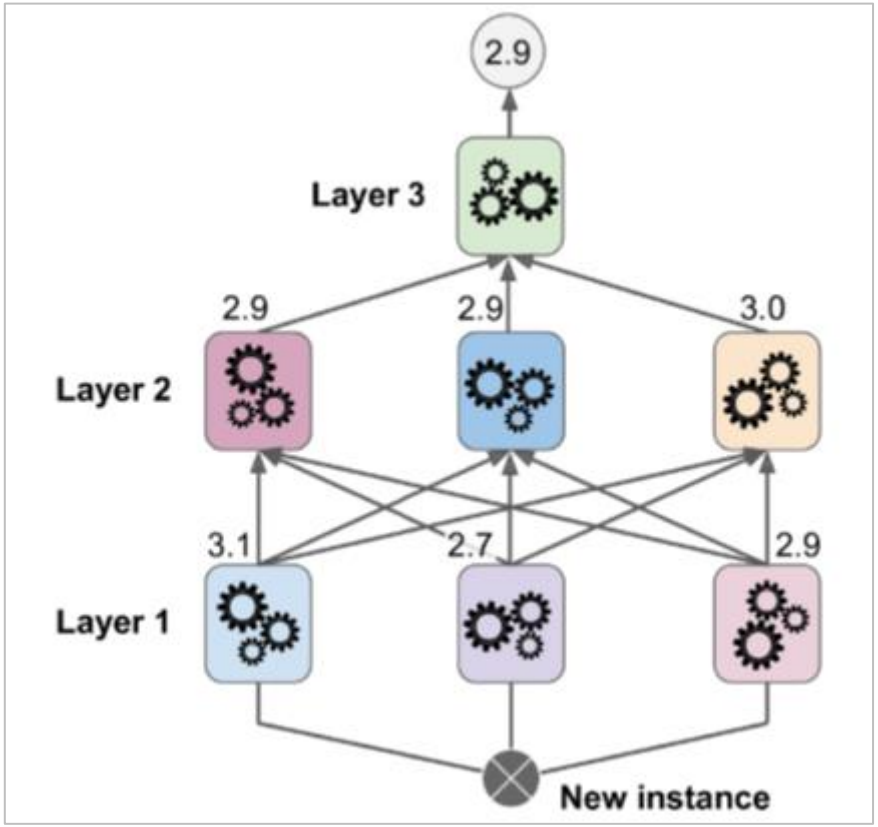
작동원리(2)

- 첫 번째 레이어의 예측기를 사용해 두 번째세트(홀드아웃)에 대한 예측
- 마지막 예측(블렌더) 훈련
  - 타깃값은 그대로 쓰고 앞에서 예측한 값을 입력 특성으로 사용하는 새로운 훈련세트 생성 : 새로운 훈련 세트는 3차원
  - 블렌더가 새 훈련세트로 훈련



여러 개 훈련시키는 것도 가능 : 세 개의 서브셋으로 구분

- 첫 번째 세트 : 첫 번째 레이어를 훈련시키는 데 사용
- 두 번째 세트 : (첫 번째 레이어의 예측기로) 두 번째 레이어를 훈련시키기 위한 훈련 세트를 만드는 데 사용
- 세 번째 세트 : (두번째 레이어의 예측기로) 세 번째 레이어를 훈련시키기 위한 훈련 세트를 만드는 데 사용
- 작업이 끝나면 각 레이어를 차례대로 실행해서 새로운 샘플에 대한 예측을 만든다



- 사이킷런은 스태킹을 지원하지 않는다
- XGboost