실습 문제2 & 랜덤 포레스트

2022.11.14

Chung-Ang University
AI/ML Innovation Research Center
Hyun-soon Lee

<실습 문제2>

01 실습문제2

■ 타이타닉(Titanic) 데이터셋을 캐글(Kaggle)에서 필요한 파일들을 다운로드 (https://www.kaggle.com/c/titanic) 한 후에 승객의 나이, 성별, 승객 등급, 승선 위치 같은 속성을 기반으로 하여 승객의 생존 여부를 예측하시오.

1. PassengerId 열을 인덱스 열로 지정한다.

2. 누락된 데이터를 확인해본다.

3. Random Forest를 적용하여 n_estimators=100, random_state=42로 예측하고 10-fold cross validation을 사용하여 forest 점수의 평균값을 구하시오.

4. Support vector machine(SVM)을 적용하여 gamma="auto"로 예측하고 10-fold cross validation을 사용하여 SVM 점수의 평균값을 구하시오.

5. 3, 4번의 결과에 대하여 accuracy, precision, F1을 각각 구하시오.

앙상블
<랜덤 포레스트>

목차

01

앙상블의 이해

01 앙상블의 이해

1. 대중적인 데이터 분석 알고리즘

- 최근 머신러닝/딥러닝 분야에서 딥러닝 다음으로 부스팅(boosting) 알고리즘이 핵심적으로 사용됨
- 선형회귀나 로지스틱 회귀는 가장 대중적인 알고리즘이고, 그 다음이 의사결정트리와 앙상블 계열 알고리즘, 딥러닝

01 앙상블의 이해

1. 대중적인 데이터 분석 알고리즘

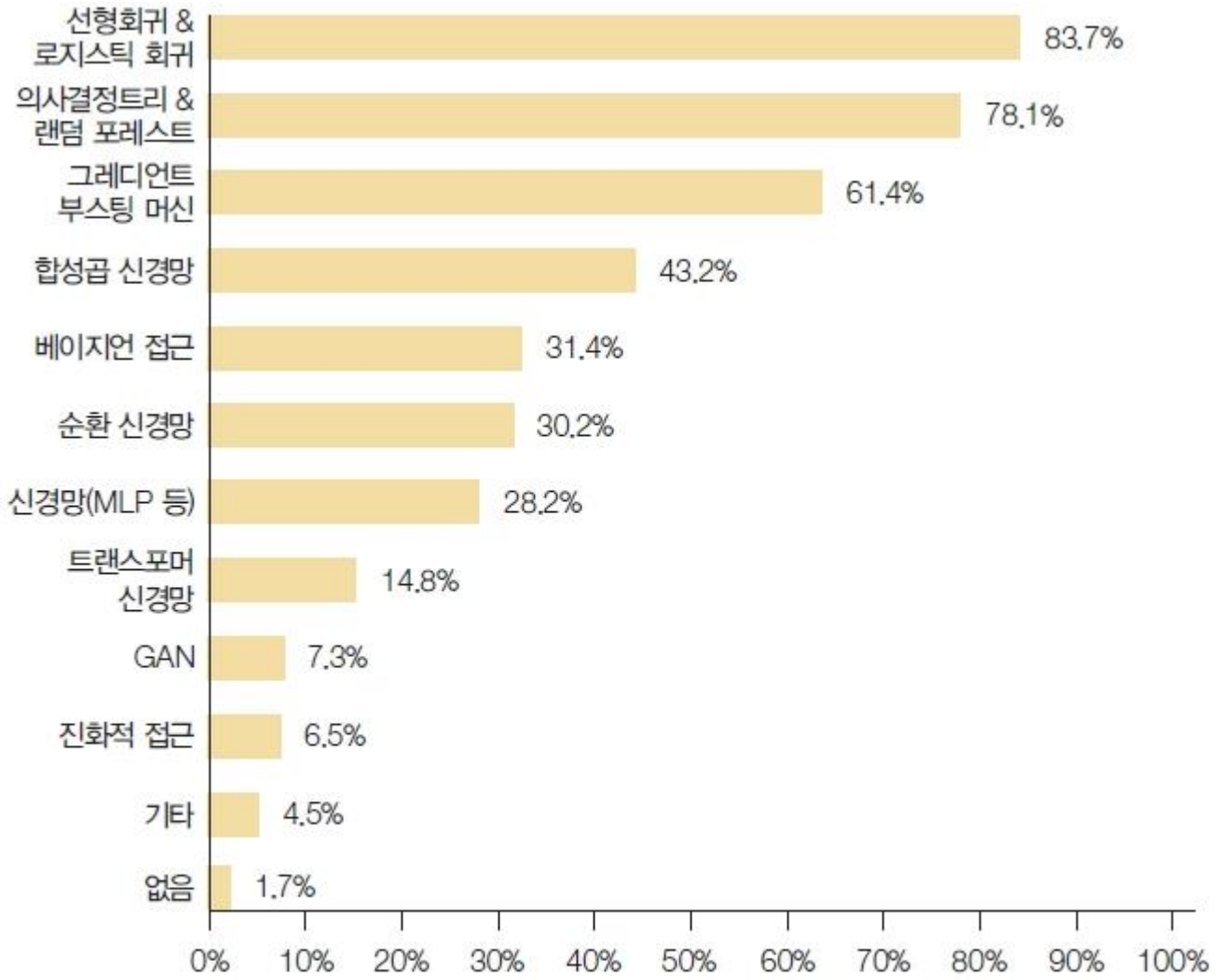


그림 13-1 데이터 분석 시 사용하는 알고리즘 설문조사

01 앙상블의 이해

2. 앙상블의 개념

- 앙상블(ensemble) : 여러 개의 알고리즘들이 하나의 값을 예측하는 기법을 통칭하여 말함
  - 회귀 문제에서는 가중 평균이나 단순 평균을 구하는 방식으로 Y 값을 예측
- 메타 분류기(meta-classifier)라고도 부름
  - 메타(meta)는 일종의 상위 또는 추상화라는 개념. 여러 분류기들을 모아 하나의 분류기를 만들어 이를 메타 분류기라고 부른다
- 시간이 굉장히 오래 걸리지만 비교적 좋은 성능을 냄

01 앙상블의 이해

2. 앙상블의 개념

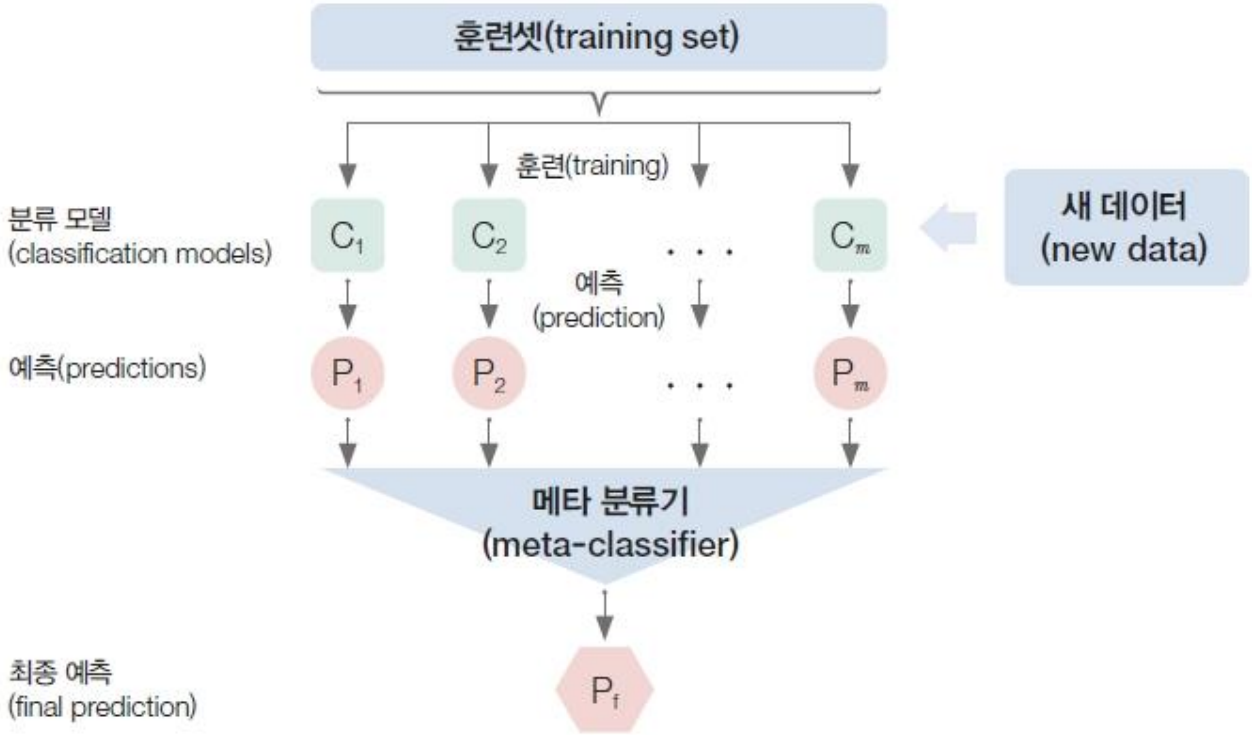


그림 13-2 앙상블 모델

- 하나의 데이터를 넣음 → 이를 여러 모델에 학습시키고 → 테스트 데이터를 각 모델에 입력 → 투표 또는 여러 가중치 기법을 적용하여 최종 선택

01 앙상블의 이해

2. 앙상블의 개념

- 앙상블 기법들
  - 바닐라 앙상블 : 가장 기본적인 앙상블 기법. 바닐라라고 하면 아이스크림에서 아무것도 첨가되지 않은 맛인데, 바닐라 앙상블도 아무것도 처리하지 않은 앙상블 모델을 의미. 일반적으로 가중치 평균이나 투표 방식으로 만들어지는 앙상블 모델
  - 부스팅 : 하나의 모델에서 여러 데이터를 샘플링한 다음 그 샘플링된 데이터로 각각의 모델을 만드는 기법
  - 배깅 : 'boosting aggregation(부스팅 집합)'의 줄임말로 부스팅을 좀 더 발전시킨 기법

02

투표 분류기

02 투표 분류기

1. 투표 분류기의 개념

- 투표 분류기(voting classifier) : 여러 개의 모델을 만들어 모두 같은 데이터를 넣고 결과를 취합하여 가장 많이 선택된 결과를 취함

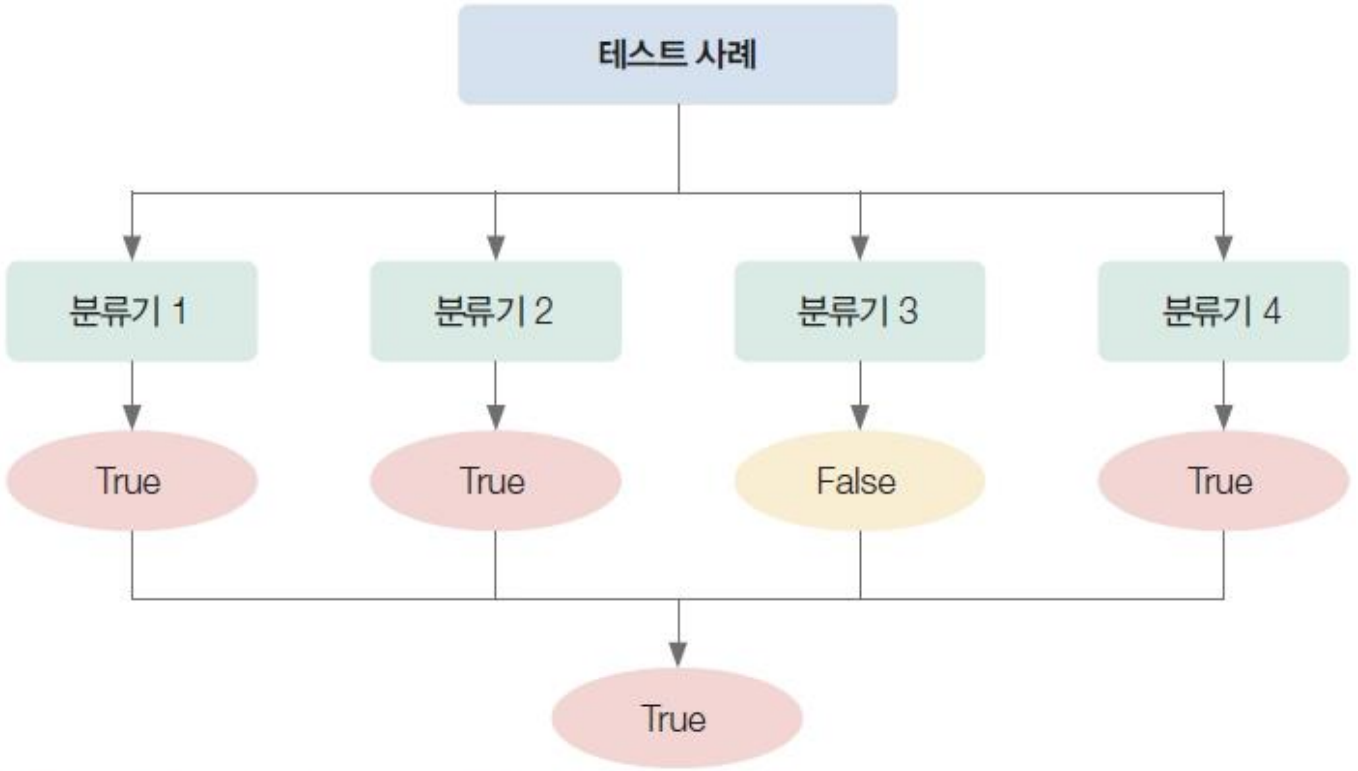


그림 13-3 투표 분류기의 기본 형태

02 투표 분류기

1. 투표 분류기의 개념

- 앙상블 모델의 가장 기본적인 형태
- 다수결 분류기(majority voting classifier)라고도 부름
- 또는 각 분류기마다 가중치를 주고 해당 가중치를 각 모델에 곱하여 가중치의 합을 구하는 방식
- 장점 : 다양한 모델을 만든 후, 다음 단계로 매우 쉽게 만들 수 있음

02 투표 분류기

2. 투표 분류기의 클래스

- 사이킷런에서 제공하는 VotingClassifier 클래스 사용

In [1]:
```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import VotingClassifier
```

- 전처리되어 .npy 파일 형태인 데이터를 호출

In [2]:
```
X = np.load("c:/source/titanic_X_train.npy")
y = np.load("c:/source/titanic_y_train.npy")
```

2. 투표 분류기의 클래스

In [3]:	X[0]
Out [3]:	array([0.27345609, 0.01415106, 0. , 1. , 0. , 0.125 , 0. , 0. , 0. , 1. , 0. , 0. , 0. , 0. , 0. , 1. , 0. , 0. , 1. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.])
In [4]:	y[:10]
Out [4]:	array([0., 1., 1., 1., 0., 0., 0., 0., 1., 1.])

02 투표 분류기

2. 투표 분류기의 클래스

- 기초 모델들을 생성

In [5]:
```
clf1 = LogisticRegression(random_state=1)
clf2 = DecisionTreeClassifier(random_state=1,
max_depth=4)
clf3 = GaussianNB()

eclf = VotingClassifier(
 estimators=[('lr', clf1), ('rf', clf2), ('gnb',clf3)],
voting='hard')
```

02 투표 분류기

2. 투표 분류기의 클래스

- 투표 분류기의 성능과 모델별 성능을 측정

In [6]:	from sklearn.model_selection import cross_val_score cross_val_score(eclf, X, y, cv=5).mean()
Out [6]:	0.8222941661905668
In [7]:	cross_val_score(clf1, X, y, cv=5).mean()
Out [7]:	0.8290420872214816
In [8]:	cross_val_score(clf2, X, y, cv=5).mean()
Out [8]:	0.8223068621849807
In [9]:	cross_val_score(clf3, X, y, cv=5).mean()
Out [9]:	0.4600139655938551

- clf3 제외하면 전체 모델 성능보다 개별 모델 성능이 높다

02 투표 분류기

2. 투표 분류기의 클래스

- GaussianNB은 연속적인 데이터를 다루기 위한 모델로 데이터셋과 맞지 않아 해당 모델을 빼고 성능을 측정

In [10]:	eclf = VotingClassifier(estimators=[('lr', clf1), ('rf', clf2)], voting='hard') cross_val_score(eclf, X, y, cv=5).mean()
Out [10]:	0.8301783787215135

- 앙상블 모델에서는 반드시 많은 수의 모델 조합이 가장 최선의 결과를 내는 것이 아니다

02 투표 분류기

3. 하이퍼 매개변수를 튜닝한 투표 분류기

- 성능이 좋았던 모델 두 개를 각각 VotingClassifer에 할당

In [11]:
```
clf1 = LogisticRegression(random_state=1)
clf2 = DecisionTreeClassifier(random_state=1)
eclf = VotingClassifier(estimators=[('lr', clf1), ('dt', clf2)],
voting='hard')
```

In [12]:
```
c_params = [0.1, 5.0, 7.0, 10.0, 15.0, 20.0, 100.0]

params ={
    "lr__solver" : ['liblinear'], "lr__penalty" : ["l2"], "lr__C" : c_params,
    "dt__criterion" : ["gini", "entropy"],
    "dt__max_depth" : [10,8,7,6,5,4,3,2],
    "dt__min_samples_leaf": [1,2,3,4,5,6,7,8,9]
    }
```

02 투표 분류기

3. 하이퍼 매개변수를 튜닝한 투표 분류기

- 가장 좋은 모델의 성능을 확인

In [13]:	from sklearn.model_selection import GridSearchCV grid = GridSearchCV(estimator=eclf, param_grid=params, cv=5) grid = grid.fit(X, y) grid.best_score_
Out [13]:	0.8425569732749316

02 투표 분류기

3. 하이퍼 매개변수를 튜닝한 투표 분류기

- 가장 좋은 성능을 내는 매개변수 확인

In [14]:	grid.best_params_
Out [14]:	{'dt__criterion': 'gini', 'dt__max_depth': 10, 'dt__min_samples_leaf': 5, 'lr__C': 5.0, 'lr__penalty': 'l2', 'lr__solver': 'liblinear'}

03

배깅과 랜덤 포레스트(Random forest)

03 배깅과 랜덤 포레스트

1. 배깅의 개념

- 배깅(bagging) : 하나의 데이터셋에서 샘플링을 통해 여러 개의 데이터셋을 만든 다음 각 데이터셋마다 모델을 개발하여 투표 분류기로 만드는 기법
  - 단순하면서 성능이 높아 특히 트리 계열 알고리즘과 함께 많이 사용되며 통계적인 샘플링 기법이나 딥러닝 기법과도 함께 사용
- 샘플링(sampling) : 다루고자 하는 데이터가 전체 모수라면 그 모수에서 일부분을 뽑아서 데이터를 분석

03 배깅과 랜덤 포레스트

1. 배깅의 개념

- 배깅의 장점 : 다양한 데이터셋에서 강건한 모델(robust model)을 개발할 수 있다

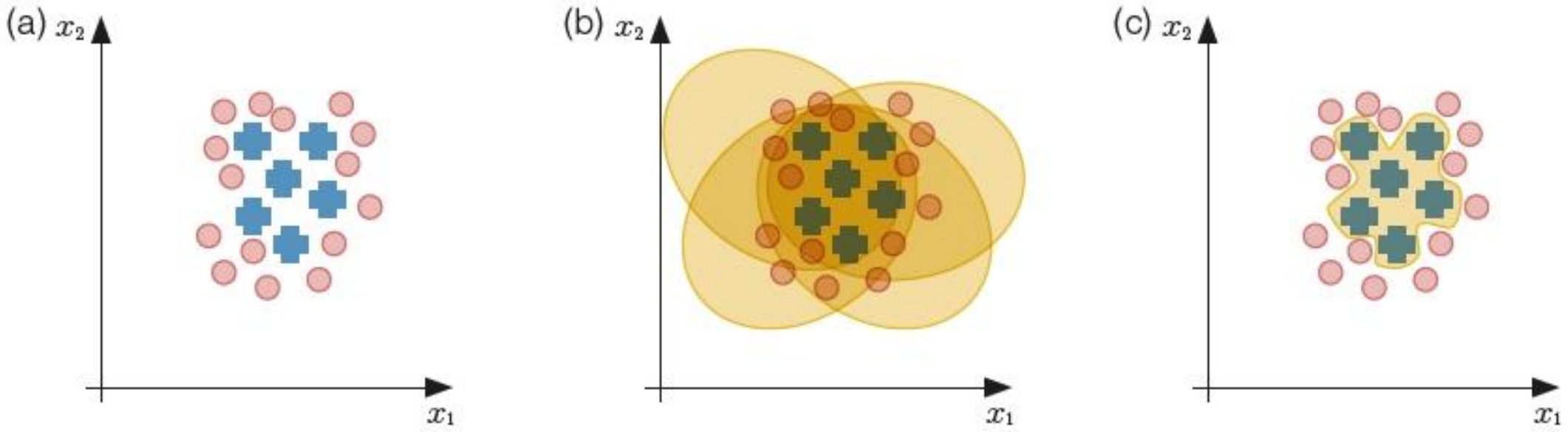


그림 13-4 배깅의 모델링

03 배깅과 랜덤 포레스트

1. 배깅의 개념

[하나 더 알기] 약분류기와 강분류기로 알아보는 배깅 기법

- 배깅 기법은 여러 개의 약분류기(weak learner)로 강분류기(strong learner)를 만드는 것이다.
- 약분류기는 기본적으로 과소적합이 다소 있지만 과적합되어 있지 않은 모델. 다소 느슨하게 경계를 생성하는 여러 개의 약한 분류기를 앙상블한다면 좀 더 정확한 경계를 생성한다.
- 각각의 작은 데이터로는 모든 구체적인 분류 영역을 정할 수 없지만 많은 데이터로 투표한다면 더 높은 성능을 기대할 수 있다.

03 배깅과 랜덤 포레스트

2. 부트스트래핑

- 부트스트래핑(bootstrapping) : 모수 데이터로부터 학습 데이터를 추출할 때 임의의 데이터를 추출한 후 복원추출하는 여러 번의 과정
- 복원추출 : 전체 데이터에서 먼저 일부를 추출하여 이를 '학습 데이터셋 1' 이라고 부른 다음 다시 그 데이터를 모수에 집어넣고 '학습 데이터 셋 2' 를 뽑는 방식

03 배깅과 랜덤 포레스트

2. 부트스트래핑

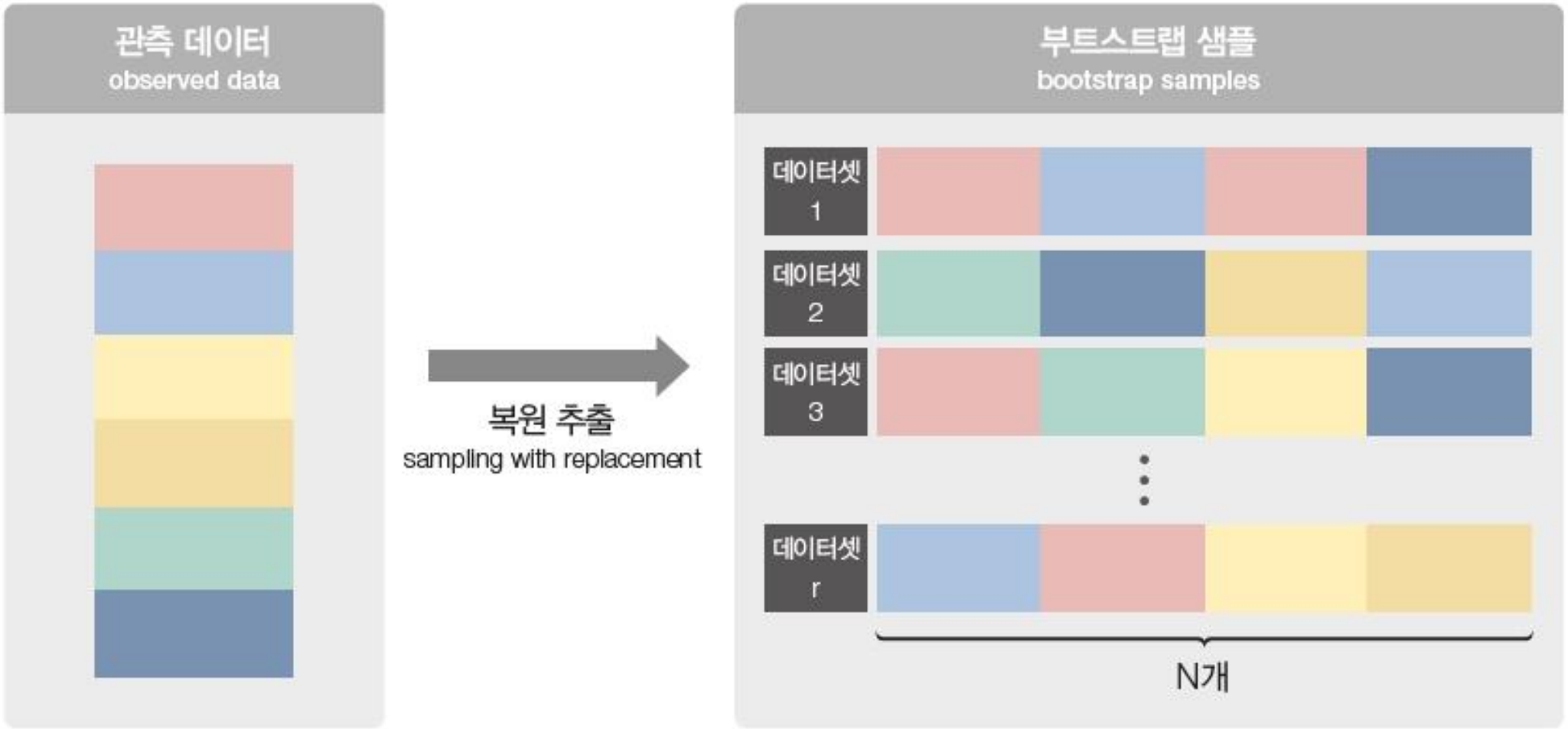


그림 13-5 복원추출

2. 부트스트래핑

[하나 더 알기] 부트스트래핑의 유래

- 부트스트래핑은 원래 카우보이 워크화의 뒤에 달려 있는 조그마한 끈을 말한다. [그림 13-6]과 같이 해당 끈에 손을 넣어서 워크화를 착용했다.
- 남에게 도움 받지 않고 스스로 무엇인가를 시작할 때 이를 지칭하는 대표적인 용어로 사용한다. 데이터 과학에서는 처음 모수 데이터에서만 일부 데이터를 추출하여 사용하는 행위를 의미하며, 컴퓨터 과학에서는 외부데이터의 주입 없이 컴퓨터가 메모리에 저장된 정보만으로 부팅되는 것을 의미한다.

그림 13-6 워크화의 부트스트래핑

03 배깅과 랜덤 포레스트

3. 부트스트랩 집합인 배깅

- 배깅(bagging)은 부트스트랩 집합이라는 의미의 'bootstrap aggregation'의 약자로, 말 그대로 부트스트랩 연산의 집합이라는 개념
- 데이터셋으로부터 부분집합 n개를 추출 → 앙상블 방법과 달리 하나의 모델에 다양한 데이터셋을 넣어서 n개의 모델을 생성
- 높은 분산으로, 일반적인 모델로 만들 경우 과적합이 심한 데이터셋에 좀 더 강건
  - 각 모델들은 해당 데이터셋에 맞춰진 과적합 모델

03 배깅과 랜덤 포레스트

3. 부트스트랩 집합인 배깅

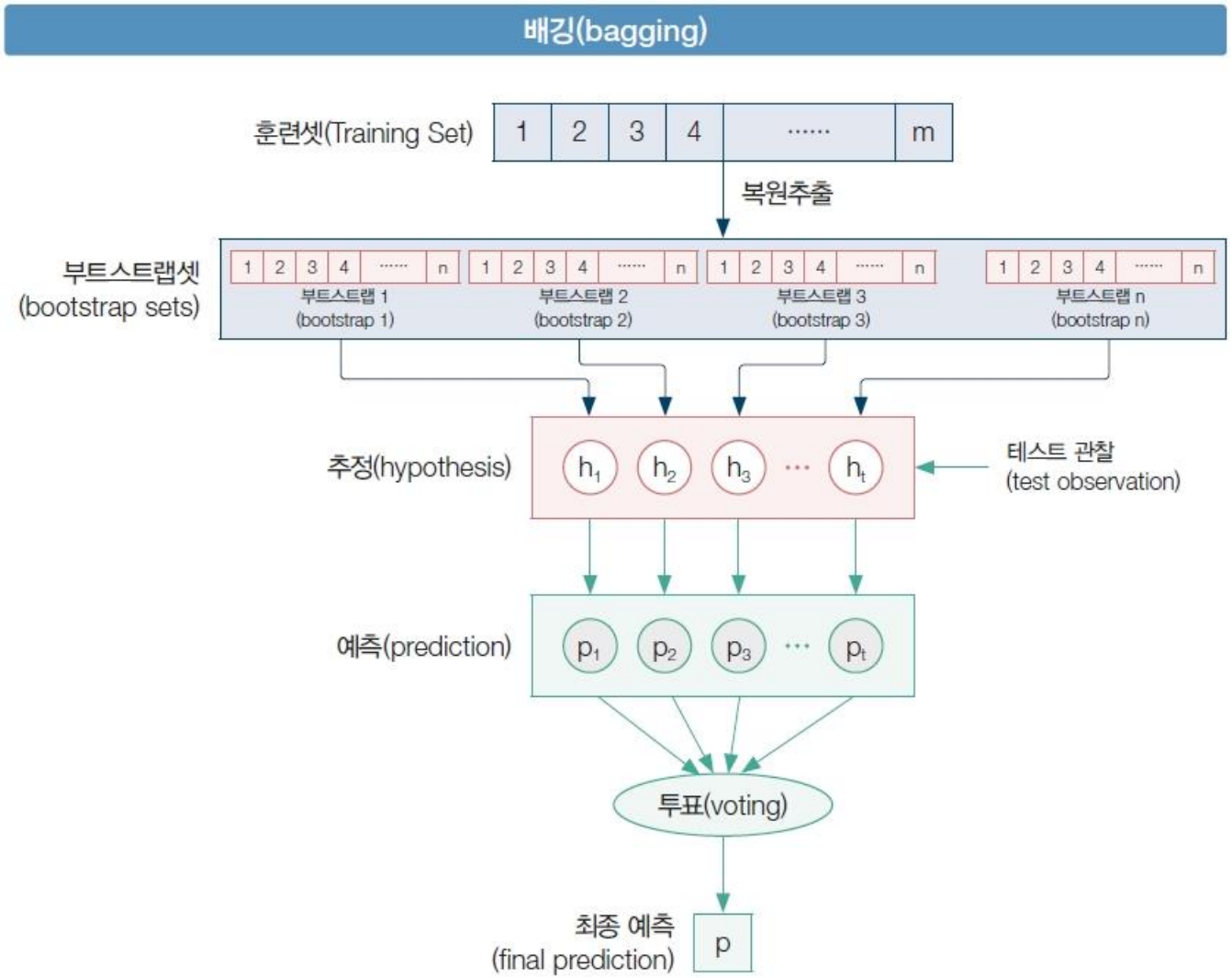


그림 13-7 배깅 업무 순서도

3. 부트스트랩 집합인 배깅

[하나 더 알기] **Out-of-bag Error**

- 배깅 모델의 성능을 측정하기 위해서 정확도나 정밀도 외에 'Out-of-bag Error'라는 지표를 사용한다. 일반적으로 'OOB error estimation'이라고 부른다.
- 배깅에서 부분집합을 생성할 때 일부 데이터만 학습에 사용되는데, 각 부분집합에서 학습에 사용되지 않은 데이터셋에 대해서만 성능을 측정하여 배깅 모델의 효과를 측정하는 것이다.
- 기본적으로 검증셋(validation set)과 유사한 방식으로 학습에 사용하지 않은 데이터를 가지고 학습의 성능을 측정한다고 이해할 수 있다.

$$Acc(M) = \frac{1}{k}\sum_{i=1}^{k}(0.632 \times Acc(M_i)_{test\_set} + 0.368 \times Acc(M_i)_{train\_set})$$

03 배깅과 랜덤 포레스트

4. 랜덤 포레스트

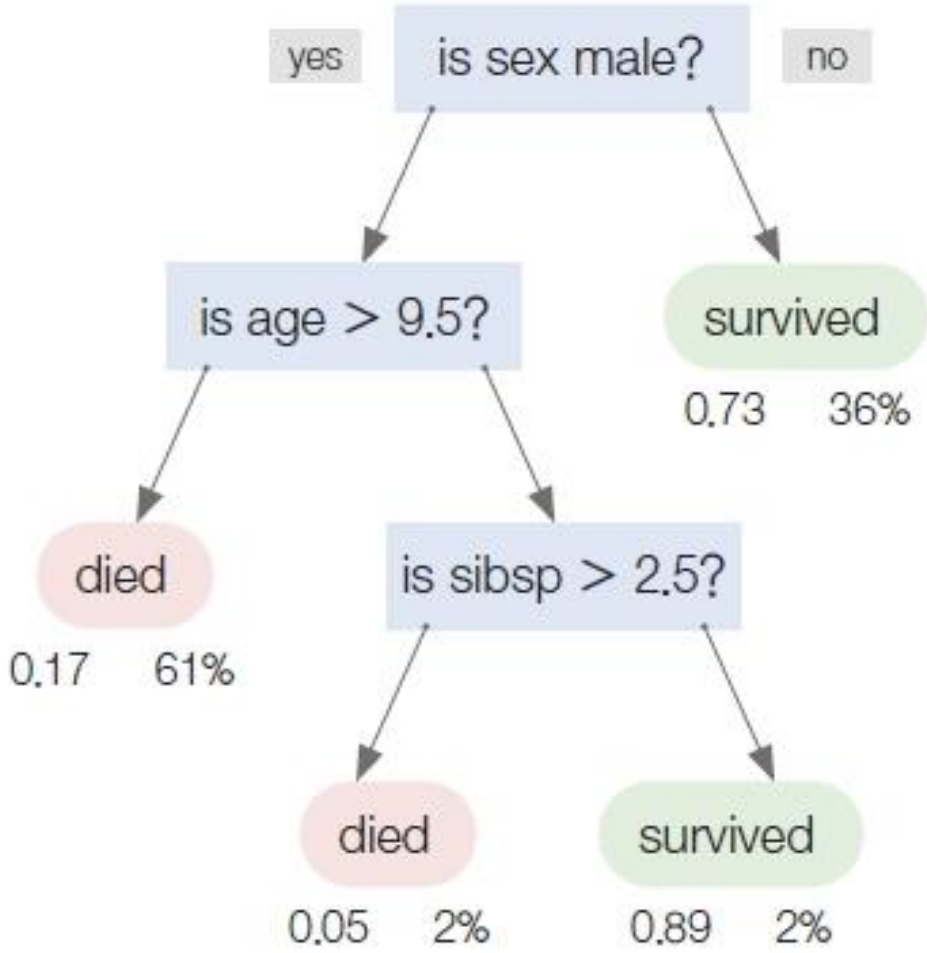
- 랜덤 포레스트(random forest) : 하나의 모델을 나무라고 한다면 이러한 나무들을 이용해 랜덤하게 데이터를 뽑아서 숲을 생성하는 알고리즘
- 배깅 알고리즘을 **의사결정트리**에 적용한 모델
- **의사결정트리(decision tree)** : 어떤 규칙을 하나의 트리(tree) 형태로 표현한 후 이를 바탕으로 분류나 회귀 문제를 해결
  - 규칙은 'if-else' 문으로 표현이 가능
  - 트리는 일종의 경로를 표현하는 것
  - 트리 구조의 마지막 노드에는 분류 문제에서 클래스, 회귀 문제에서는 예측치가 들어감

4. 랜덤 포레스트

- 의사결정트리

```
if age > 30:
    return True
else:
    return False
```

(a) if-else문의 예



(b) if-else문의 경로 표현

그림 12-1 의사결정트리의 이해

03 배깅과 랜덤 포레스트

4. 랜덤 포레스트

- 의사결정트리

그림 12-2 아키네이터(akinator) 게임 © https://kr.akinator.com/

4. 랜덤 포레스트

- **의사결정트리는 딥러닝 기반을 제외한 전통적인 통계 기반의 머신러닝 모델 중 효과와 실용성이 가장 좋음**
  - 테이블형 데이터에 있어 설명력과 성능의 측면에서 딥러닝 모델들과 대등하게 경쟁
  - 앙상블(ensemble) 모델이나 부스팅(boosting) 같은 새로운 기법들이 모델들의 성능을 대폭 향상시키고 있음

4. 랜덤 포레스트

- 사이킷런 배깅 분류기 BaggingClassifier
  - base_estimator : 사용될 수 있는 모델(default=None)
  - n_estimators : int, optional(default=10), subset으로 생성되는 모델의 개수
  - max_samples : int or float, optional(default=1.0), 최대 데이터 개수 또는 비율
  - max_features : int or float, optional(default=1.0), 최대 사용 피쳐 또는 비율
  - bootstrap : boolean, optional(default=True), bootstrap 사용 여부
  - oob_score : boolean, oob score 산출 여부
  - warm_start : booeanl, optional(default=False), 이전에 학습된 모델을 사용할 것인가에 대한 정보

03 배깅과 랜덤 포레스트

4. 랜덤 포레스트

In [1]:
```
import numpy as np

from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import BaggingClassifier

X = np.load("c:/source/titanic_X_train.npy")
y = np.load("c:/source/titanic_y_train.npy")
clf1 = LogisticRegression(random_state=1)
eclf = BaggingClassifier(clf1, oob_score=True)

from sklearn.model_selection import cross_val_score
cross_val_score(eclf, X, y, cv=5).mean()
```

Out [1]:
```
0.8267822002158318
```

4. 랜덤 포레스트

In [2]:
```
params ={
    "n_estimators" : [10,20,30,40,50,55],
    "max_samples" : [0.5,0.6,0.7,0.8,0.9,1]
}

from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(estimator=eclf, param_grid=params, cv=5)
grid = grid.fit(X, y)

grid.best_score_
```

Out [2]:
```
0.8324255697327493
```

0.8301783787215135

03 배깅과 랜덤 포레스트

4. 랜덤 포레스트

In [3]:	grid.best_params_
Out [3]:	{'max_samples': 0.9, 'n_estimators': 20} {'max_samples': 0.9, 'n_estimators': 10}
In [4]:	grid.best_estimator_.oob_score_
Out [4]:	0.8245219347581553

03 배깅과 랜덤 포레스트

4. 랜덤 포레스트

In [5]:	import numpy as np from sklearn.ensemble import RandomForestClassifier X = np.load("c:/source/titanic_X_train.npy") y = np.load("c:/source/titanic_y_train.npy") eclf = RandomForestClassifier(n_estimators=100, max_features=2, n_jobs=7, oob_score=True) from sklearn.model_selection import cross_val_score cross_val_score(eclf, X, y, cv=5).mean()
Out [5]:	0.798685964578156

0.8065574811147084

03 배깅과 랜덤 포레스트

4. 랜덤 포레스트

In [6]:
```
from sklearn.model_selection import GridSearchCV

params ={
    "n_estimators" : [10, 20, 30, 50, 100],
    "max_features" : [1,2,3,4,5,6,7, 10, 15, 20, 25, len(X[0])]
    }

grid = GridSearchCV(estimator=eclf, param_grid=params, cv=5)
grid = grid.fit(X, y)

grid.best_score_
```

Out [6]:
```
0.8234558496794261
```

0.8245667491906303

03 배깅과 랜덤 포레스트

4. 랜덤 포레스트

In [7]:	grid.best_params_
Out [7]:	{'max_features': 25, 'n_estimators': 30} {'max_features': 15, 'n_estimators': 20}
In [8]:	grid.best_estimator_.oob_score_
Out [8]:	0.8053993250843644

0.8076490438695163

[TIP] 기존 로지스틱 분류기 기반의 배깅 모델보다 훨씬 더 시간이 오래 걸린다.
성능 향상은 다른 알고리즘보다 실험에 의해 많이 좌우된다.

04

부스팅

04 부스팅

1. 부스팅의 개념

- 부스팅(boosting) : 학습 라운드를 차례로 진행하면서 각 예측이 틀린 데이터에 점점 가중치를 주는 방식
- 라운드별로 잘못 분류된 데이터를 좀 더 잘 분류하는 모델로 만들어 최종적으로 모델들의 앙상블을 만드는 방식
  - 배깅 알고리즘이 처음 성능을 측정하기 위한 기준(baseline) 알고리즘으로 많이 사용된다면, 부스팅 알고리즘은 높은 성능을 내야 하는 상황에서 가장 좋은 선택지

04 부스팅

1. 부스팅의 개념

- 첫 번째 라운드 결과 모델 (1)에서 A점은 오차가 큰 부분
- 두 번째 라운드에서 오답으로 분류된 A에 가중치를 줘 학습
- 다시 오류가 큰 B 영역에 가중치를 둬 모델 (3) 개발

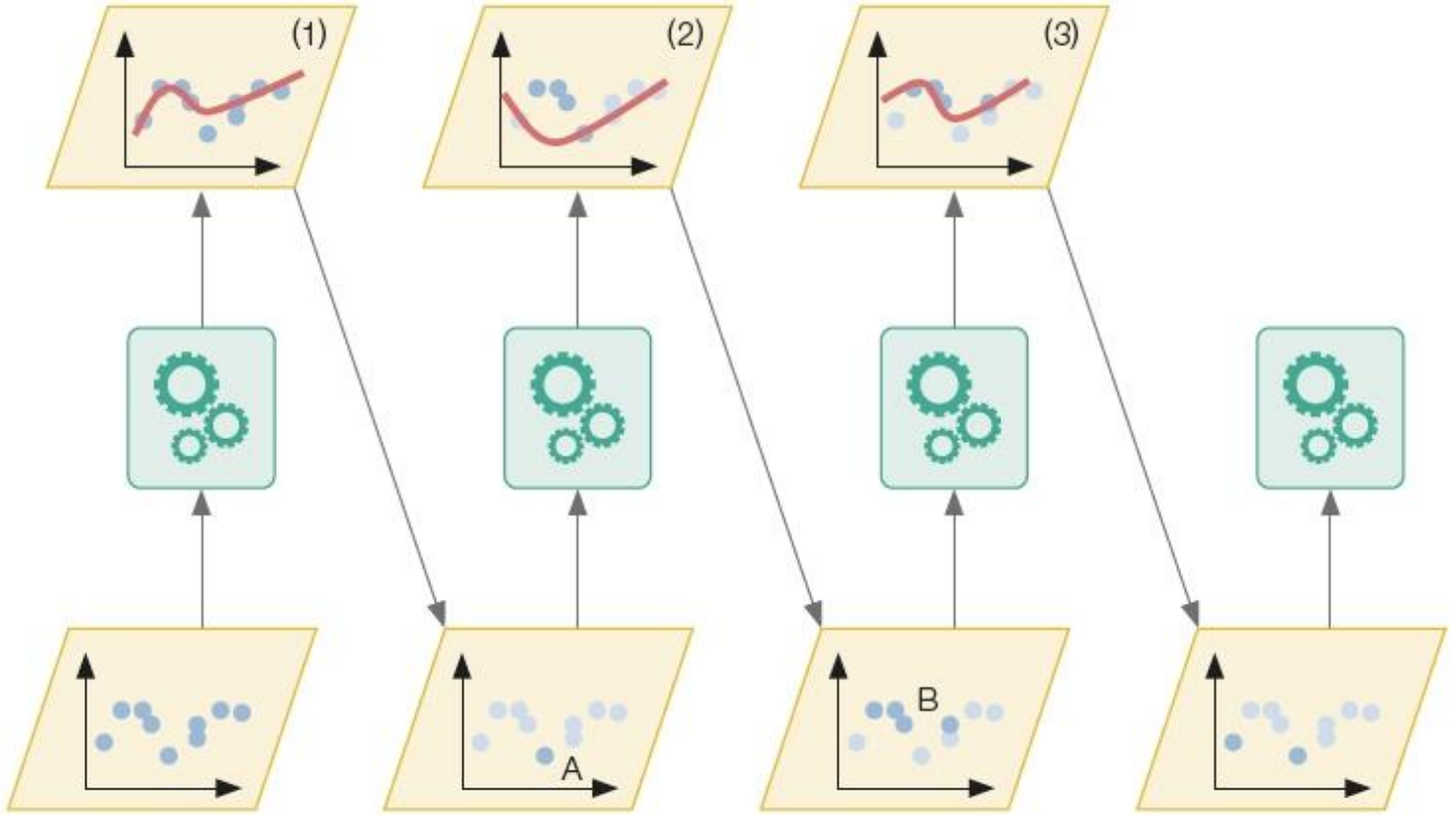


그림 13-8 부스팅

04 부스팅

1. 부스팅의 개념

- 틀린 부분만 집중해서 모델들을 순차적으로 만들고, 해당 모델들은 최종적으로 앙상블

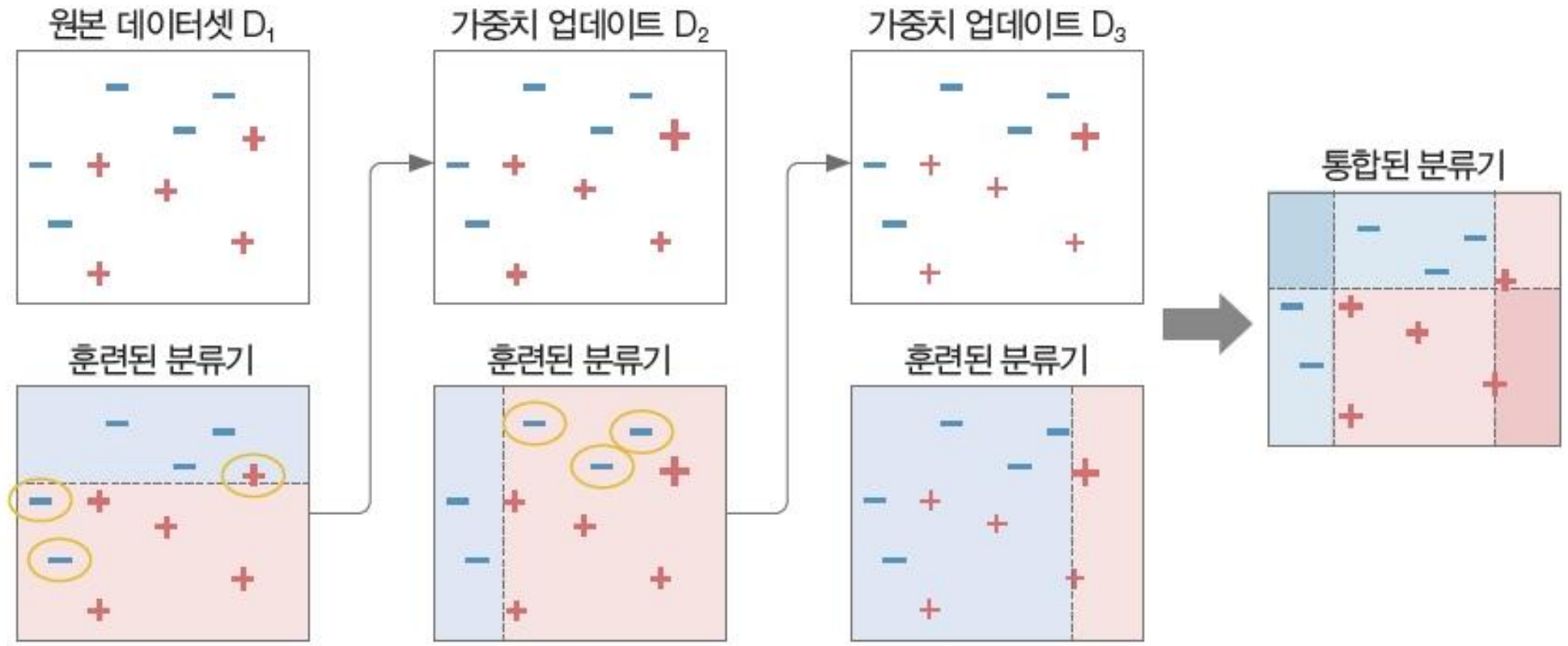


그림 13-9 분류 관점에서 부스팅 정리

04 부스팅

2. 배깅과 부스팅의 차이점

2.1 병렬화 가능 여부

- 배깅은 데이터가 n개라면 n개의 CPU로 한번에 처리하도록 구조를 설계할 수 있음
  - 배깅은 데이터를 나눠 데이터마다 조금씩 다른 모델을 생성
- 부스팅은 단계적으로 모델들을 생성하고 해당 모델들의 성능을 측정한 후 다음 단계로 넘어가 병렬화를 지원하지 않음
- 부스팅은 배깅에 비해 속도가 매우 떨어짐

2. 배깅과 부스팅의 차이점

2.2 기준 추정치

- 배깅 개별 모델들은 높은 과대적합으로 모델의 분산이 높음
- 부스팅은 각각의 모델에 편향이 높은 기준 추정치를 사용하여 개별 모델들은 과소적합이 발생하지만 전체적으로 높은 성능을 낼 수 있는 방향으로 학습
  - 부스팅 모델의 이러한 특징을 약한 학습자(weak learner)라고 부름

2. 배깅과 부스팅의 차이점

2.3 성능 차이

- 부스팅은 기본적으로 비용이 높은 알고리즘
  - 비용은 속도나 시간을 말함
- 배깅은 데이터의 부분집합에 대해 학습을 수행하기 때문에 부스팅보다 좋은 성능을 내기는 어려움
- 초기 성능을 측정할 때는 배깅, 이후의 성능 측정은 부스팅으로 하는 것이 가장 일반적인 접근

04 부스팅

3. 에이다부스트

- 부스팅 알고리즘 중 대표적인 알고리즘
- 에이다부스트(AdaBoost) : 매 라운드마다 인스턴스, 즉 개별 데이터의 가중치를 계산하는 방식
- 매 라운드마다 틀린 값이 존재하고 해당 인스턴스에 가중치를 추가로 주어 가중치를 기준으로 재샘플링(resampling)

3. 에이다부스트

3.1 알고리즘

- 모든 샘플의 가중치 값을 데이터 개수를 기준으로 $\frac{1}{N}$로 초기화한 다음

$$w_i = \frac{1}{N}, \quad i = 1, 2, 3, 4, \cdots, N$$

① 데이터의 가중치를 사용해 모델 $G_m(x_i)$을 학습시킨다

② 해당 분류기의 오류를 계산한다

$$err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$$

$$I(y_i, G_m(x_i)) = \begin{cases} 0 & \text{if } y_i = G_m(x_i) \\ 1 & \text{if } y_i \neq G_m(x_i) \end{cases}$$

3. 에이다부스트

3.1 알고리즘

③ 해당 분류기의 가중치를 생성한다

$$a_m = \log((1 - err_m)/err_m)$$

④ 모델의 가중치를 사용하여 각 데이터의 가중치를 업데이트

$$w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \cdots, N$$

최종 결과물은 각 모델 가중치와 모델 결과값의 가중합을 연산하여 계산

$$G(x) = sign\left[\sum_{m=1}^{M} a_m G_m(x)\right]$$

3. 에이다부스트

3.2 에이다부스트와 스텀프

- 스텀프(stump)는 '그루터기'라는 뜻으로 나무의 윗부분을 자르고 아랫부분만 남은 상태

그림 13-10 스텀프(stump)의 개념

04 부스팅

3. 에이다부스트

3.2 에이다부스트와 스텀프

- 에이다부스트에서 스텀프는 학습할 때 큰 나무를 사용하여 학습하는 것이 아니라 나무의 그루터기만을 사용하여 학습한다는 개념
- 1뎁스(depth) 또는 2뎁스 정도의 매우 간단한 모델을 여러 개 만들어 학습한 후, 해당 모델들의 성능을 에이다부스트 알고리즘을 적용하여 학습하는 형태

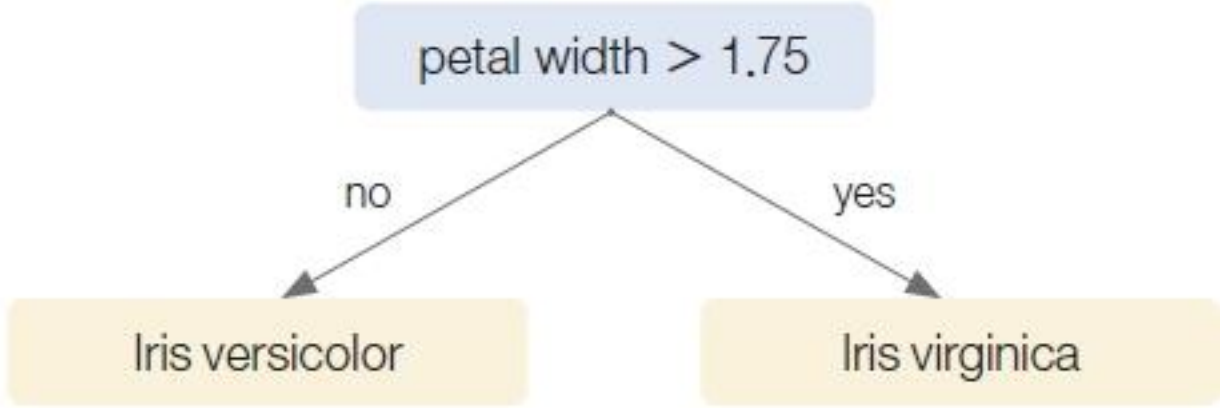


그림 13-11 간단한 모델

3. 에이다부스트

3.3 사이킷런으로 에이다부스트 실습하기

- 사이킷런에서 제공하는 AdaBoostClassifier를 사용

In [1]:	import numpy as np X = np.load("c:/source/titanic_X_train.npy") y = np.load("c:/source/titanic_y_train.npy")
In [2]:	from sklearn.ensemble import AdaBoostClassifier from sklearn.tree import DecisionTreeClassifier eclf = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=2), n_estimators=500)
In [3]:	from sklearn.model_selection import cross_val_score cross_val_score(eclf, X, y, cv=5).mean()
Out [3]:	0.7896908525360249 0.7908207960388498

04 부스팅

3.3 사이킷런으로 에이다부스트 실습하기

- 비교군으로 RandomForestClassifier를 생성

In [4]:	**from sklearn.ensemble import RandomForestClassifier** **eclf = RandomForestClassifier(n_estimators=500)** **cross_val_score(eclf, X, y, cv=5).mean()**
Out [4]:	**0.803186694597854 4**

0.8054402336062972

04 부스팅

3.3 사이킷런으로 에이다부스트 실습하기

- GridSearchCV를 사용하여 가장 좋은 모델을 찾아 모델의 성능을 향상

In [5]:

```
eclf = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=2),
n_estimators=500)

params = {"base_estimator__criterion" : ["gini", "entropy"],
"base_estimator__max_features" : [7,8,],
          "base_estimator__max_depth" : [1,2],
          "n_estimators": [23,24, 25, 26, 27],
          "learning_rate": [0.4, 0.45, 0.5, 0.55, 0.6]
          }

from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(estimator=eclf, param_grid=params, cv=5, n_jobs=7)
grid = grid.fit(X, y)

grid.best_score_
```

Out [5]:

```
0.8324382657271631      0.8346791087411922
```

04 부스팅

3.3 사이킷런으로 에이다부스트 실습하기

- GridSearchCV를 사용하여 가장 좋은 모델을 찾아 모델의 성능을 향상

In [6]:	grid.best_params_
Out [6]:	{'base_estimator__criterion': 'gini', 'base_estimator__max_depth': 2, 'base_estimator__max_features': 7, 'learning_rate': 0.6, 'n_estimators': 27}

```
{'base_estimator__criterion': 'entropy',
 'base_estimator__max_depth': 2,
 'base_estimator__max_features': 7,
 'learning_rate': 0.45,
 'n_estimators': 26}
```

04 부스팅

3.3 사이킷런으로 에이다부스트 실습하기

- feature_importances_로 각 피쳐(feature)들이 모델에 영향을 미치는 정도를 나타낼 수 있다

In [7]:	grid.best_estimator_.feature_importances_
Out [7]:	array([0.19297248, 0.17038992, 0.03419741, 0.0978764 , 0.03066311, 0.08343991, 0.0385496 , 0.00700623, 0.01984721, 0. , 0. , 0. , 0.01669575, 0.05762587, 0.04374955, 0.06470532, 0.03188512, 0.01193647, 0.01636659, 0. , 0.00727312, 0.00486091, 0.03454949, 0.00868942, 0.02230592, 0. , 0.0044142])

```
array([0.19234007, 0.18832578, 0.01666046, 0.05983134, 0.07185324,
       0.10614953, 0.05328338, 0.0229881 , 0.        , 0.00574226,
       0.        , 0.        , 0.00914371, 0.03884135, 0.0293809 ,
       0.03832089, 0.03319968, 0.01830455, 0.02064887, 0.        ,
       0.00697715, 0.02563929, 0.01558522, 0.03471563, 0.        ,
       0.01206861, 0.        ])
```

Thank You !