



# 13장 과적합 피하기



# 목 차



- 
- 1 데이터의 확인과 실행
  - 2 과적합 이해하기
  - 3 학습셋과 테스트셋
  - 4 모델 저장과 재사용
  - 5 k겹 교차 검증





# 과적합 피하기



- 실습 데이터 초음파 광물 예측

- dataset/sonar.csv



# 과적합 피하기

- 1988년 존스홉킨스대학교의 세즈노프스키(Sejnowski) 교수는 2년 전 힌튼 교수가 발표한 역전파 알고리즘에 관심을 가지고 있었음
- 그는 은닉층과 역전파가 얼마나 큰 효과가 있는지를 직접 실험해 보고 싶었음
- 광석과 일반 돌을 가져다 놓고 음파 탐지기를 쏜 후 그 결과를 데이터로 정리함
- 오차 역전파 알고리즘을 사용한 신경망이 과연 얼마나 광석과 돌을 구분하는 데 효과적인지 알아보기 위해서임

# 1 데이터의 확인과 실행

- dataset/sonar.csv 파일을 가져옴

```
import pandas as pd

df = pd.read_csv('../dataset/sonar.csv', header=None)
print(df.info())
```



# 1 데이터의 확인과 실행



Range Index: 208 entries,0 to 207			
Data columns (total 61 columns):			
0	208	non-null	float64
1	208	non-null	float64
2	208	non-null	float64
3	208	non-null	float64
4	208	non-null	float64



# 1 데이터의 확인과 실행



Range Index: 208 entries, 0 to 207			
Data columns (total 61 columns):			
5	208	non-null	float64
...	...	...	...
58	208	non-null	float64
59	208	non-null	float64
60	208	non-null	object
Dtypes: float64(60), object(1)			
memory usage: 99.2+ KB			

# 1 데이터의 확인과 실행

- 모든 컬럼이 실수형(float64)인데, 맨 마지막 컬럼만 객체형인 것으로 보아

마지막에 나오는 컬럼은 클래스이며 데이터형 변환이 필요한 것을 알 수 있음

- 실제로 맞는지 일부를 출력해 확인해 보자

```
print(df.head())
```



# 1 데이터의 확인과 실행

	0	1	2	3	...	59	60
0	0.02	0.0371	0.0428	0.0207	...	0.0032	R
1	0.0453	0.0523	0.0843	0.0689	...	0.0044	R
2	0.0262	0.0582	0.1099	0.1083	...	0.0078	R
3	0.01	0.0171	0.0623	0.0205	...	0.0117	R
4	0.0762	0.0666	0.0481	0.0394	...	0.0094	R

# 1 데이터의 확인과 실행

## 코드 13-1 초음파 광물 예측하기: 데이터 확인과 실행

- 예제 소스: run\_project/04\_Sonar.ipynb

```
from keras.models import Sequential
from keras.layers.core import Dense
from sklearn.preprocessing import LabelEncoder

import pandas as pd
import numpy
import tensorflow as tf

# seed 값 설정
numpy.random.seed(3)
tf.random.set_seed(3)
```

# 1 데이터의 확인과 실행

# 데이터 입력

```
df = pd.read_csv('../dataset/sonar.csv', header=None)
```

```
dataset = df.values
```

```
X = dataset[:,0:60]
```

```
Y_obj = dataset[:,60]
```

# 문자열 변환

```
e = LabelEncoder( )
```

```
e.fit(Y_obj)
```

```
Y = e.transform(Y_obj)
```

# 1 데이터의 확인과 실행

# 모델 설정

```
model = Sequential()  
model.add(Dense(24, input_dim=60, activation='relu'))  
model.add(Dense(10, activation='relu'))  
model.add(Dense(1, activation='sigmoid'))
```

# 모델 컴파일

```
model.compile(loss='mean_squared_error',  
              optimizer='adam',  
              metrics=['accuracy'])
```

# 1 데이터의 확인과 실행

# 모델 실행

```
model.fit(X, Y, epochs=200, batch_size=5)
```

# 결과 출력

```
print("\n Accuracy: %.4f" % (model.evaluate(X, Y)[1]))
```



# 1 데이터의 확인과 실행



실행  
결과



Epoch 1/200

208/208 [=====] - 0s 601us/step - loss: 0.2449

- accuracy: 0.5673

Epoch 2/200

208/208 [=====] - 0s 197us/step - loss: 0.2404

- accuracy: 0.5481

Epoch 3/200

208/208 [=====] - 0s 197us/step - loss: 0.2268

- accuracy: 0.6202

Epoch 4/200

208/208 [=====] - 0s 211us/step - loss: 0.2188

# 1 데이터의 확인과 실행

– accuracy: 0.6394

Epoch 5/200

208/208 [=====] – 0s 216us/step – loss: 0.2102

– accuracy: 0.7019

(중략)

Epoch 196/200

208/208 [=====] – 0s 202us/step – loss:

3.8308e-04 – accuracy: 1.0000

Epoch 197/200

208/208 [=====] – 0s 197us/step – loss:

3.7724e-04 – accuracy: 1.0000

# 1 데이터의 확인과 실행

Epoch 198/200

208/208 [=====] - 0s 197us/step - loss:

3.6022e-04 - accuracy: 1.0000

Epoch 199/200

208/208 [=====] - 0s 207us/step - loss:

3.5173e-04 - accuracy: 1.0000

Epoch 200/200

208/208 [=====] - 0s 207us/step - loss:

3.5247e-04 - accuracy: 1.0000

208/208 [=====] - 0s 86us/step

Accuracy: 1.0000





## 2 과적합 이해하기



- 완전히 새로운 데이터에 적용하면 이 선을 통해 정확히 두 그룹으로 나누지 못하게 됨

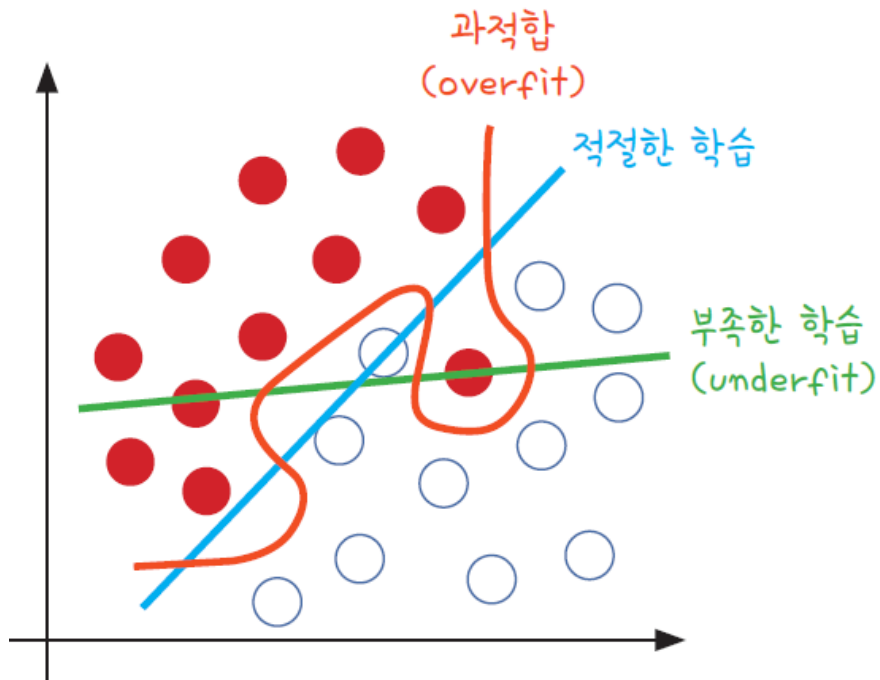


그림 13-1 과적합이 일어난 경우(빨간색)와 학습이 제대로 이루어지지 않은 경우(초록색)



## 2 과적합 이해하기



- 과적합은 층이 너무 많거나 변수가 복잡해서 발생하기도 하고 테스트셋과 학습셋이 중복될 때 생기기도 함
- 딥러닝은 학습 단계에서 입력층, 은닉층, 출력층의 노드들에 상당히 많은 변수들이 투입됨
- 딥러닝을 진행하는 동안 과적합에 빠지지 않게 늘 주의해야 함





### 3 학습셋과 테스트셋



- 과적합을 방지하려면 어떻게 해야 할까?
  - 먼저 학습을 하는 데이터셋과 이를 테스트할 데이터셋을 완전히 구분한 다음 학습과 동시에 테스트를 병행하며 진행하는 것이 한 방법
- 데이터셋이 총 100개의 샘플로 이루어져 있다면 다음과 같이 두 개의

70개 샘플은 학습셋으로

30개 샘플은 테스트셋으로



### 3 학습셋과 테스트셋



- 신경망을 만들어 70개의 샘플로 학습을 진행한 후 이 학습의 결과를 저장함
  - 이렇게 저장된 파일을 '모델'이라고 부름
- 모델은 다른 셋에 적용할 경우 학습 단계에서 각인되었던 그대로 다시 수행함
- 나머지 30개의 샘플로 실험해서 정확도를 살펴보면 학습이 얼마나 잘 되었는지를 알 수 있는 것



### 3 학습셋과 테스트셋

- 딥러닝 같은 알고리즘을 충분히 조절하여 가장 나은 모델이 만들어지면, 이를 실생활에 대입하여 활용하는 것이 바로 머신러닝의 개발 순서

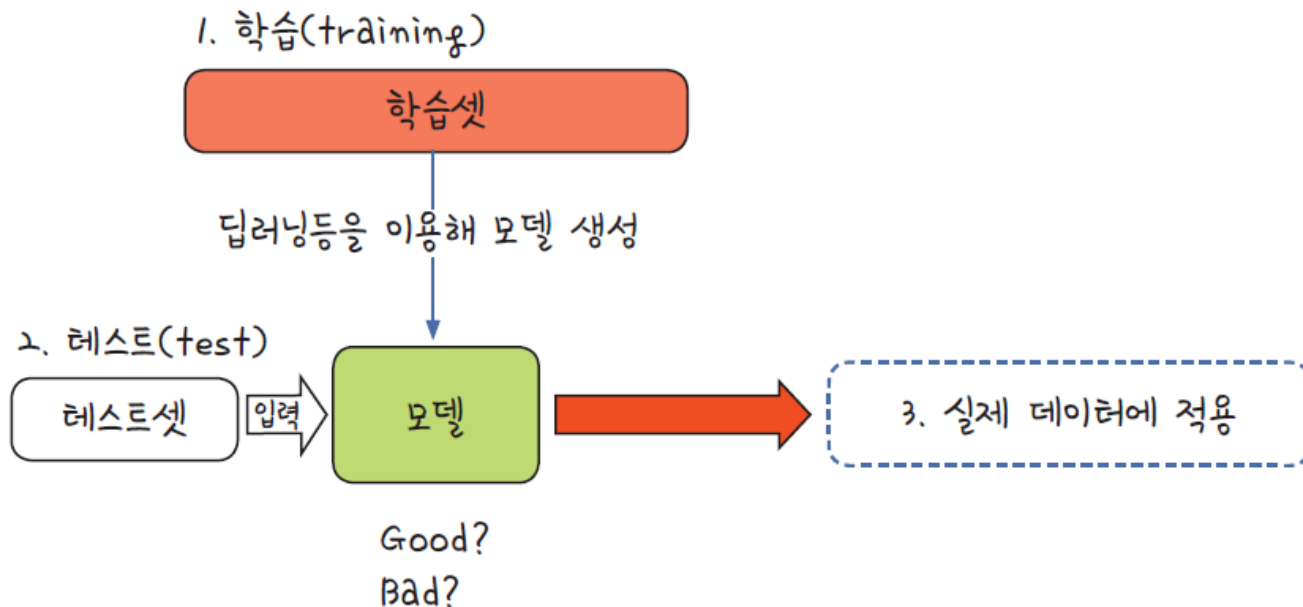


그림 13-2 학습셋과 테스트셋

### 3 학습셋과 테스트셋

- 학습 데이터를 이용해 정확도를 측정한 것은 데이터에 들어있는 모든 샘플을 그대로 테스트에 활용한 결과임
- 학습에 사용된 샘플은 테스트에 쓸 수 없으므로 학습 단계에서 테스트할 샘플은 자동으로 빼고, 이를 테스트한 결과를 모아 정확도를 계산하는 것
- 이러한 방법은 빠른 시간에 모델 성능을 파악하고 수정할 수 있도록 도와 줌
- 머신러닝의 최종 목적은 과거의 데이터를 토대로 새로운 데이터를 예측하는 것
- 테스트셋을 만들어 정확한 평가를 병행하는 것이 매우 중요함



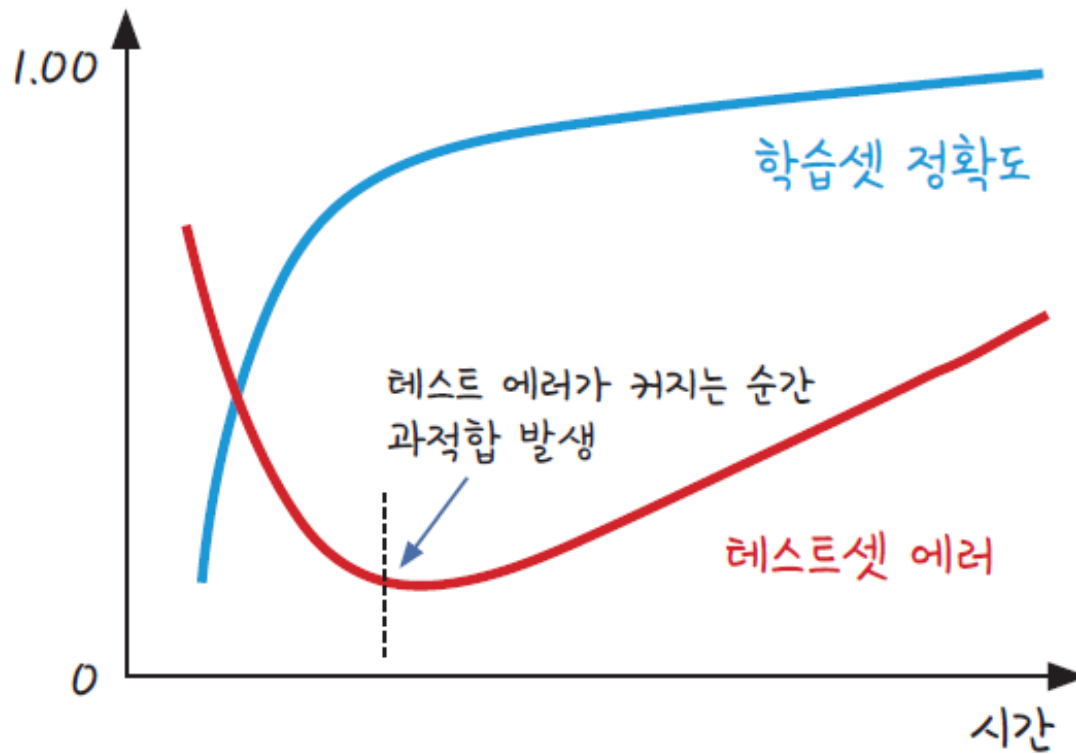
### 3 학습셋과 테스트셋



- 학습셋만 가지고 평가할때, 층을 더하거나 에포크(epoch) 값을 높여 실행 횟수를 늘리면 정확도가 계속해서 올라갈 수 있음
- 학습이 깊어져서 학습셋 내부에서의 성공률은 높아져도 테스트셋에서는 효과가 없다면 과적합이 일어나고 있는 것



### 3 학습셋과 테스트셋



**그림 13-3** 학습이 계속되면 학습셋에서의 정확도는 계속 올라가지만, 테스트셋에서는 과적합이 발생



### 3 학습셋과 테스트셋

- 학습을 진행해도 테스트 결과가 더 이상 좋아지지 않는 지점에서 학습을 멈춰야 함
- 이때의 학습 정도가 가장 적절한 것으로 볼 수 있음
- 우리가 다루는 초음파 광물 예측 데이터를 만든 세즈노프스키 교수가 실험 결과를 발표한 논문의 일부를 가져와 보겠음

TABLE 2 Aspect-Angle Dependent Series				
Number of Hidden Units	Average Performance on Training Sets (%)	Standard Deviation on Training Sets (%)	Average Performance on Testing Sets (%)	Standard Deviation on Testing Sets (%)
0	79.3	3.4	73.1	4.8
2	96.2	2.2	85.7	6.3
3	98.1	1.5	87.6	3.0
6	99.4	0.9	89.3	2.4
12	99.8	0.6	90.4	1.8
24	100.0	0.0	89.2	1.4

Summary of the results of the aspect-angle dependent series of experiments with training and testing sets selected to include all target aspect angles. The standard deviation shown is across networks with different initial conditions.

그림 13-4 학습셋과 테스트셋 정확도 측정의 예(RP Gorman et.al., 1998)

### 3 학습셋과 테스트셋

은닉층 수의 변화	학습셋의 예측률	테스트셋의 예측률
0	79.3	73.1
2	96.2	85.7
3	98.1	87.6
6	99.4	89.3
12	99.8	90.4
24	100	89.2

표 13-1 은닉층 수의 변화에 따른 학습셋 및 테스트셋의 예측률

- 식이 복잡해지고 학습량이 늘어날수록 학습 데이터를 통한 예측률은 계속해서 올라가지만, 테스트셋을 이용한 예측률은 오히려 떨어지는 것을 확인할 수 있음

### 3 학습셋과 테스트셋

- 불러온 X 데이터와 Y 데이터에서 각각 정해진 비율(%)만큼 구분하여 한 그룹은 학습에 사용함
- 다른 한 그룹은 테스트에 사용하게 하는 함수가 sklearn 라이브러리의 `train_test_split( )` 함수임
- 학습셋을 70%, 테스트셋을 30%로 설정했을때의 예

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

```
# 학습셋과 테스트셋의 구분
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=seed)
```

### 3 학습셋과 테스트셋

- 모델을 실행하는 부분에서 위에서 만들어진 학습셋으로 학습을, 테스트셋으로 테스트를 하게 하려면 다음과 같이 실행함

```
model.fit(X_train, Y_train, epochs=130, batch_size=5)

# 테스트셋에 모델 적용
print("\n Test Accuracy: %.4f" % (model.evaluate(X_test, Y_test)
[1]))
```

### 3 학습셋과 테스트셋

#### 코드 13-2 초음파 광물 예측하기: 학습셋과 테스트셋 구분

- 예제 소스: run\_project/05\_Sonar\_Train\_Test.ipynb

```
from keras.models import Sequential
from keras.layers.core import Dense
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

import pandas as pd
import numpy
import tensorflow as tf
```

### 3 학습셋과 테스트셋

```
# seed 값 설정
```

```
seed = 0
```

```
numpy.random.seed(seed)
```

```
tf.random.set_seed(3)
```

```
df = pd.read_csv('../dataset/sonar.csv', header=None)
```

```
dataset = df.values
```

```
X = dataset[:,0:60]
```

```
Y_obj = dataset[:,60]
```

```
e = LabelEncoder( )
```

```
e.fit(Y_obj)
```

```
Y = e.transform(Y_obj)
```

### 3 학습셋과 테스트셋

# 학습셋과 테스트셋의 구분

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=seed)
```

```
model = Sequential()
```

```
model.add(Dense(24, input_dim=60, activation='relu'))
```

```
model.add(Dense(10, activation='relu'))
```

```
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(loss='mean_squared_error',
```

```
              optimizer='adam',
```

```
              metrics=['accuracy'])
```

```
model.fit(X_train, Y_train, epochs=130, batch_size=5)
```



### 3 학습셋과 테스트셋



# 테스트셋에 모델 적용

```
print("\n Test Accuracy: %.4f" % (model.evaluate(X_test, Y_test)
[1]))
```





### 3 학습셋과 테스트셋

실행  
결과



Epoch 1/130

145/145 [=====] – 0s 765us/step – loss: 0.2486

– accuracy: 0.5379

Epoch 2/130

145/145 [=====] – 0s 207us/step – loss: 0.2344

– accuracy: 0.5931

Epoch 3/130

145/145 [=====] – 0s 214us/step – loss: 0.2233

– accuracy: 0.7241

Epoch 4/130

145/145 [=====] – 0s 227us/step – loss: 0.2125

– accuracy: 0.6966

### 3 학습셋과 테스트셋

Epoch 5/130

145/145 [=====] - 0s 227us/step - loss: 0.2040

- accuracy: 0.7310

(중략)

Epoch 126/130

145/145 [=====] - 0s 207us/step - loss: 0.0050

- accuracy: 1.0000

Epoch 127/130

145/145 [=====] - 0s 200us/step - loss: 0.0073

- accuracy: 1.0000

Epoch 128/130

145/145 [=====] - 0s 214us/step - loss: 0.0071

- accuracy: 0.9931

### 3 학습셋과 테스트셋

Epoch 129/130

145/145 [=====] - 0s 193us/step - loss: 0.0054

- accuracy: 1.0000

Epoch 130/130

145/145 [=====] - 0s 200us/step - loss: 0.0046

- accuracy: 1.0000

63/63 [=====] - 0s 238us/step

Test Accuracy: 0.8095

**TIP**

실전에서는 더 정확한 테스트를 위해 테스트셋을 두 개로 나누어, 하나는 앞서 설명한 방식대로 테스트셋으로 사용하고, 나머지 하나는 최종으로 만들어 낸 모델을 다시 한번 테스트하는 용도로 사용하기도 합니다. 추가로 만들어낸 테스트셋을 검증셋(Validation sets)이라고도 부릅니다.

## 4 모델 저장과 재사용

- 학습이 끝난 후 테스트해 본 결과가 만족스러울 때 이를 모델로 저장하여 새로운 데이터에 사용할 수 있음
- 앞서 학습한 결과를 모델로 저장하려면 다음과 같이 실행함

```
from keras.models import load_model
```

```
model.save('my_model.h5')
```

## 4 모델 저장과 재사용

- 이를 불러오려면 다음과 같이 실행함

```
model = load_model('my_model.h5')
```

## 4 모델 저장과 재사용

### 코드 13-3 초음파 광물 예측하기: 모델 저장과 재사용

- 예제 소스: run\_project/06-Sonar-Save-Model.ipynb

```
from keras.models import Sequential, load_model
from keras.layers.core import Dense
from sklearn.preprocessing import LabelEncoder

import pandas as pd
import numpy
import tensorflow as tf
```

## 4 모델 저장과 재사용

```
# seed 값 설정
seed = 0
numpy.random.seed(seed)
tf.random.set_seed(3)

df = pd.read_csv('../dataset/sonar.csv', header=None)

dataset = df.values
X = dataset[:,0:60]
Y_obj = dataset[:,60]

e = LabelEncoder()
e.fit(Y_obj)
Y = e.transform(Y_obj)
```

## 4 모델 저장과 재사용

# 학습셋과 테스트셋을 나눔

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

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=seed)
```

```
model = Sequential()
```

```
model.add(Dense(24, input_dim=60, activation='relu'))
```

```
model.add(Dense(10, activation='relu'))
```

```
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(loss='mean_squared_error',
```

```
              optimizer='adam',
```

```
              metrics=['accuracy'])
```



## 4 모델 저장과 재사용

```
model.fit(X_train, Y_train, epochs=130, batch_size=5)
model.save('my_model.h5') # 모델을 컴퓨터에 저장

del model # 테스트를 위해 메모리 내의 모델을 삭제

model = load_model('my_model.h5') # 모델을 새로 불러옴

print("\n Test Accuracy: %.4f" % (model.evaluate(X_test, Y_test)
[1])) # 불러온 모델로 테스트 실행
```



## 5 k겹 교차 검증



- 딥러닝 혹은 머신러닝 작업을 할 때 늘 어려운 문제 중 하나는 알고리즘을 충분히  
테스트하였어도 데이터가 충분치 않으면 좋은 결과를 내기가 어렵다는 것
- 이러한 단점을 보완하고자 만든 방법이 바로 k겹 교차 검증(k-fold cross validation)
- k겹 교차 검증 :  
데이터셋을 여러 개로 나누어 하나씩 테스트셋으로 사용하고 나머지를 모두 합해서 학습셋으로 사용하는 방법
- 이렇게 하면 가지고 있는 데이터의 100%를 테스트셋으로 사용할 수 있음





## 5 k겹 교차 검증

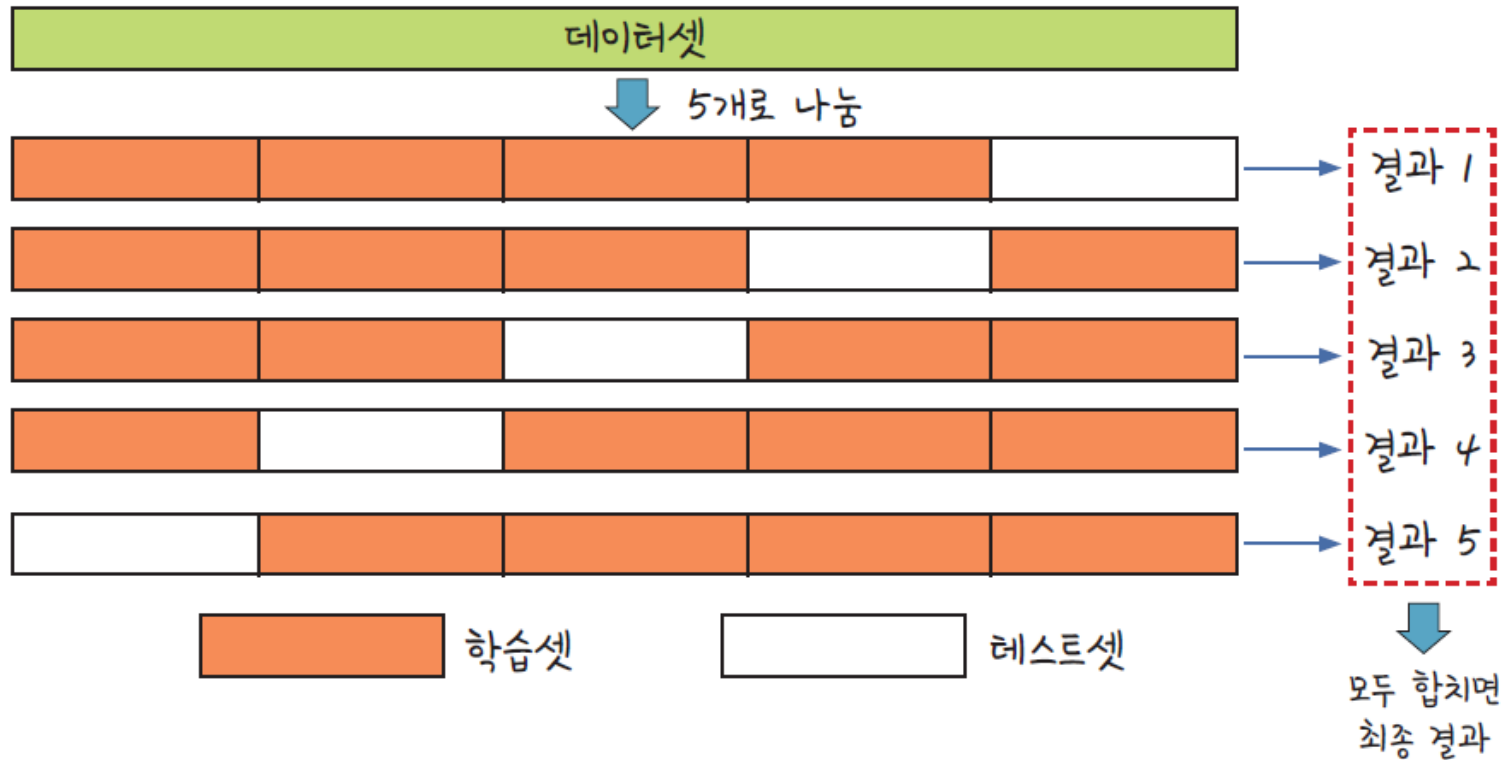


그림 13-5 5겹 교차 검증의 도식



## 5 k겹 교차 검증



- 데이터를 원하는 숫자만큼 쪼개 각각 학습셋과 테스트셋으로 사용되게 만드는 함수는 sklearn의 StratifiedKFold( ) 함수

```
from sklearn.model_selection import StratifiedKFold  
  
n_fold = 10  
skf = StratifiedKFold(n_splits=n_fold, shuffle=True, random_  
state=seed)
```



## 5 k겹 교차 검증



- 모델을 만들고 실행하는 부분을 for 구문으로 묶어 n\_fold만큼 반복되게 함

```
for train, test in skf.split(X, Y):  
    model = Sequential()  
    model.add(Dense(24, input_dim=60, activation='relu'))  
    model.add(Dense(10, activation='relu'))  
    model.add(Dense(1, activation='sigmoid'))  
    model.compile(loss='mean_squared_error',  
                  optimizer='adam',  
                  metrics=['accuracy'])  
    model.fit(X[train], Y[train], epochs=100, batch_size=5)
```



## 5 k겹 교차 검증



- 정확도(Accuracy)를 매번 저장하여 한 번에 보여줄 수 있게  
accuracy 배열을 만듦

```
accuracy = []  
  
for train, test in skf.split(X, Y):  
    (중략)  
    k_accuracy = "%.4f" % (model.evaluate(X[test], Y[test])[1])  
    accuracy.append(k_accuracy)  
  
print("\n %.f fold accuracy:" % n_fold, accuracy)
```



## 5 k겹 교차 검증



### 코드 13-4 초음파 광물 예측하기:k겹 교차 검증

- 예제 소스: run\_projec/07\_Sonar-K-fold.ipynb

```
from keras.models import Sequential
from keras.layers.core import Dense
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold

import numpy
import pandas as pd
import tensorflow as tf
```



## 5 k겹 교차 검증



```
# seed 값 설정
```

```
seed = 0
```

```
numpy.random.seed(seed)
```

```
tf.set_random_seed(seed)
```

```
df = pd.read_csv('../dataset/sonar.csv', header=None)
```

```
dataset = df.values
```

```
X = dataset[:,0:60]
```

```
Y_obj = dataset[:,60]
```

```
e = LabelEncoder()
```

```
e.fit(Y_obj)
```

```
Y = e.transform(Y_obj)
```





## 5 k겹 교차 검증



```
# 10개의 파일로 쪼갬
```

```
n_fold = 10
```

```
skf = StratifiedKFold(n_splits=n_fold, shuffle=True, random_  
state=seed)
```

```
# 빈 accuracy 배열
```

```
accuracy = []
```





## 5 k겹 교차 검증



# 모델의 설정, 컴파일, 실행

```
for train, test in skf.split(X, Y):  
    model = Sequential()  
    model.add(Dense(24, input_dim=60, activation='relu'))  
    model.add(Dense(10, activation='relu'))  
    model.add(Dense(1, activation='sigmoid'))  
    model.compile(loss='mean_squared_error',  
                  optimizer='adam',  
                  metrics=['accuracy'])  
    model.fit(X[train], Y[train], epochs=100, batch_size=5)  
    k_accuracy = "%.4f" % (model.evaluate(X[test], Y[test])[1])  
    accuracy.append(k_accuracy)
```





## 5 k겹 교차 검증



# 결과 출력

```
print("\n %.f fold accuracy:" % n_fold, accuracy)
```

실행  
결과



(중략)

Epoch 98/100

188/188 [=====] - 0s 191us/step - loss: 0.0166

- accuracy: 0.9894

Epoch 99/100

188/188 [=====] - 0s 181us/step - loss: 0.0130

- accuracy: 0.9947



## 5 k겹 교차 검증



Epoch 100/100

188/188 [=====] – 0s 186us/step – loss: 0.0136

– accuracy: 1.0000

20/20 [=====] – 0s 650us/step

10 fold accuracy: ['0.8182', '0.7143', '0.8095', '0.8095', '0.7619', '0.8095', '0.8571',  
'0.9500', '0.7500', '0.8000']

