



넷째마당

딥러닝 기본기 다지기

13장 모델 성능 검증하기

1 데이터의 확인과 예측 실행

2 과적합 이해하기

3 학습셋과 테스트셋

4 모델 저장과 재사용

5 k겹 교차 검증



모델 성능 검증하기

- 모델 성능 검증하기





모델 성능 검증하기

● 모델 성능 검증하기

- 1986년 제프리 힌튼 교수가 오차 역전파를 발표한 직후, 존스 홉킨스의 세즈노프스키(Sejnowski) 교수는 오차 역전파가 은닉층의 가중치를 실제로 업데이트시키는 것을 확인하고 싶었음
- 그는 광석과 일반 암석에 수중 음파 탐지기를 쏜 후 결과를 모아 데이터셋을 준비했고, 음파 탐지기의 수신 결과만 보고 광석인지 일반 암석인지를 구분하는 모델을 만들었음
- 그가 측정한 결과의 정확도는 몇이었을까?
- 이 장에서는 세즈노프스키 교수가 했던 초음파 광물 예측 실험을 텐서플로로 재현해 보고, 이렇게 구해진 실험 정확도를 평가하는 방법과 성능을 향상시키는 중요한 머신 러닝 기법들에 대해 알아보자



1 데이터의 확인과 예측 실행



1 데이터의 확인과 예측 실행

```
# pandas 라이브러리를 불러옵니다.  
import pandas as pd  
  
# 광물 데이터를 불러옵니다.  
df = pd.read_csv('./data/sonar3.csv', header=None)  
  
df.head() # 첫 다섯 줄을 봅니다.
```

실행 결과

	0	1	2	3	4	5	…	55	56	57	58	59	60
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	…	0.0167	0.0180	0.0084	0.0090	0.0032	0
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	…	0.0191	0.0140	0.0049	0.0052	0.0044	0
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	…	0.0244	0.0316	0.0164	0.0095	0.0078	0
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	…	0.0073	0.0050	0.0044	0.0040	0.0117	0
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	…	0.0015	0.0072	0.0048	0.0107	0.0094	0

5 rows × 61 columns



1 데이터의 확인과 예측 실행

● 데이터의 확인과 예측 실행

- 광석이 111개, 일반 암석이 97개, 총 208개의 샘플이 준비되어 있는 것을 알 수 있음

```
x = df.iloc[:,0:60]  
y = df.iloc[:,60]
```



1 데이터의 확인과 예측 실행

● 데이터의 확인과 예측 실행

- 이후 앞서 했던 그대로 딥러닝을 실행

실습 | 초음파 광물 예측하기: 데이터 확인과 실행



```
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense  
  
# 모델을 설정합니다.  
model = Sequential()  
model.add(Dense(24, input_dim=60, activation='relu'))
```



1 데이터의 확인과 예측 실행

● 데이터의 확인과 예측 실행

```
model.add(Dense(10, activation='relu'))  
model.add(Dense(1, activation='sigmoid'))  
  
# 모델을 컴파일합니다.  
model.compile(loss='binary_crossentropy', optimizer='adam',  
metrics=['accuracy'])  
  
# 모델을 실행합니다.  
history = model.fit(X, y, epochs=200, batch_size=10)
```



1 데이터의 확인과 예측 실행

● 데이터의 확인과 예측 실행

실행 결과

Epoch 1/200

21/21 [=====] - 4s 4ms/step - loss: 0.6951 -
accuracy: 0.5000

... (중략) ...

Epoch 200/200

21/21 [=====] - 0s 2ms/step - loss: 0.0327 -
accuracy: 1.0000



1 데이터의 확인과 예측 실행

● 데이터의 확인과 예측 실행

- 200번 반복되었을 때의 결과를 보니 정확도가 100%
- 정말로 어떤 광물이든 100%의 확률로 판별해 내는 모델이 만들어진 것일까?
- 다음 장을 보기 바람



2 과적합 이해하기



2 과적합 이해하기

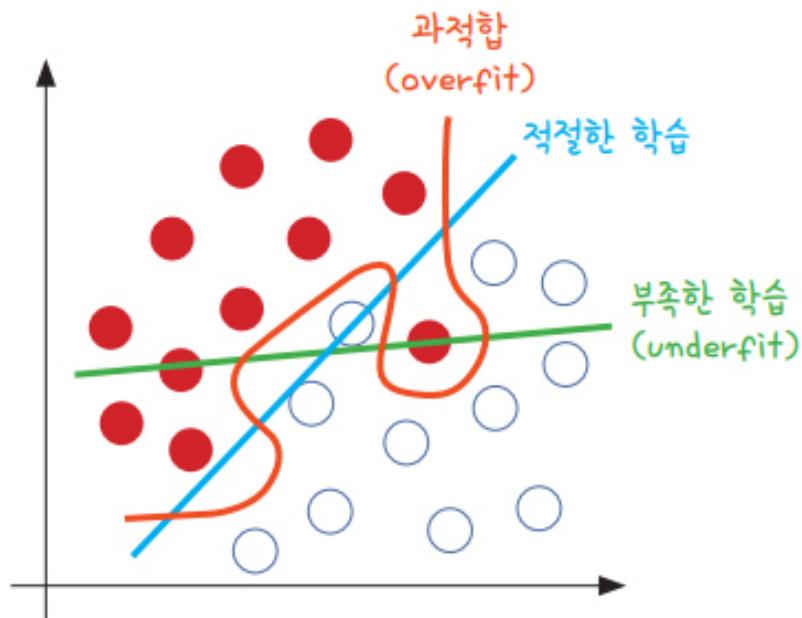
● 과적합 이해하기

- 이제 과적합 문제가 무엇인지 알아보고 이를 어떻게 해결하는지 살펴보자
- **과적합**(overfitting)이란 모델이 학습 데이터셋 안에서는 일정 수준 이상의 예측 정확도를 보이지만, 새로운 데이터에 적용하면 잘 맞지 않는 것을 의미
- 그림 13-1의 그래프에서 빨간색 선을 보면 주어진 샘플에 정확히 맞게끔 그어져 있음
- 이 선은 너무 주어진 샘플에만 최적화되어 있음
- 지금 그어진 선을 새로운 데이터에 적용하면 정확한 분류가 어려워진다는 의미



2 과적합 이해하기

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





3 학습셋과 테스트셋

● 학습셋과 테스트셋

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

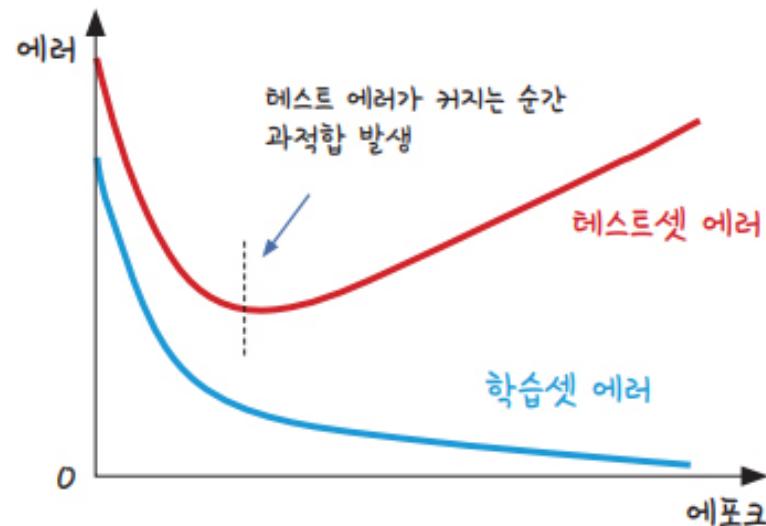
70개의 샘플은 학습셋으로

30개의 샘플은 테스트셋으로



3 학습셋과 테스트셋

▼ 그림 13-4 | 학습이 계속되면 학습셋에서의 에러는 계속해서 작아지지만,
테스트셋에서는
과적합 발생!





3 학습셋과 테스트셋

● 학습셋과 테스트셋

- !pip install sklearn
- 저장된 X 데이터와 y 데이터에서 각각 정해진 비율(%)만큼 학습셋과 테스트셋으로 분리시키는 함수가 사이킷런의 train_test_split() 함수
- 다음과 같이 학습셋과 테스트셋을 만들 수 있음
- 학습셋을 70%, 테스트셋을 30%로 설정했을 때의 예

```
from sklearn.model_selection import train_test_split

# 학습셋과 테스트셋을 구분합니다.

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
shuffle=True)
```



3 학습셋과 테스트셋

● 학습셋과 테스트셋

- 만들어진 모델을 테스트셋에 적용하려면 `model.evaluate()` 함수를 사용하면 됨
- 모델을 테스트셋에 적용해 정확도를 구합니다.

```
score = model.evaluate(X_test, y_test)
print('Test accuracy:', score[1])
```

실행 결과

```
Epoch 1/200
15/15 [=====] - 0s 3ms/step - loss: 0.7158 -
accuracy: 0.4828
... (중략) ...
Epoch 200/200
15/15 [=====] - 0s 2ms/step - loss: 0.0590 -
accuracy: 0.9931
①
2/2 [=====] - 0s 2ms/step - loss: 0.4214 -
accuracy: 0.8413
Test accuracy: 0.841269850730896
②
```



3 학습셋과 테스트셋

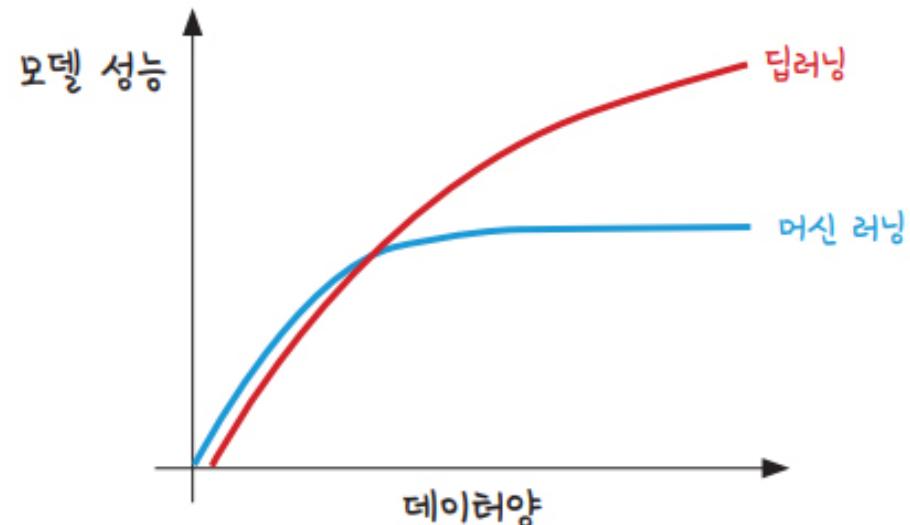
● 학습셋과 테스트셋

- 딥러닝, 머신 러닝의 목표는 학습셋에서만 잘 작동하는 모델을 만드는 것이 아님
- 새로운 데이터에 대해 높은 정확도를 안정되게 보여 주는 모델을 만드는 것이 목표
- 어떻게 하면 이러한 모델을 만들 수 있을까?
- 모델 성능의 향상을 위한 방법에는 크게 데이터를 보강하는 방법과 알고리즘을 최적화하는 방법이 있음
- 데이터를 이용해 성능을 향상시키려면 우선 충분한 데이터를 가져와 추가하면 됨
- 많이 알려진 다음 그래프는 특히 딥러닝의 경우 샘플 수가 많을수록 성능이 좋아짐을 보여 줌



3 학습셋과 테스트셋

▼ 그림 13-6 | 데이터의 증가와 딥러닝, 머신 러닝 성능의 상관관계





3 학습셋과 테스트셋

● 학습셋과 테스트셋

- 데이터를 추가하는 것 자체가 어렵거나 데이터 추가만으로는 성능에 한계가 있을 수 있음
- 가지고 있는 데이터를 적절히 보완해 주는 방법을 사용
- 예를 들어 사진의 경우 사진 크기를 확대/축소한 것을 더해 보거나 위아래로 조금씩 움직여 넣어 보는 것
- 테이블형 데이터의 경우 너무 크거나 낮은 이상치가 모델에 영향을 줄 수 없도록 크기를 적절히 조절할 수 있음
- 시그모이드 함수를 사용해 전체를 0~1 사이의 값으로 변환하는 것이 좋은 예
- 또 교차 검증 방법을 사용해 가지고 있는 데이터를 충분히 이용하는 방법도 있음



3 학습셋과 테스트셋

● 학습셋과 테스트셋

- 다음으로 알고리즘을 이용해 성능을 향상하는 방법은 먼저 다른 구조로 모델을 바꾸어 가며 최적의 구조를 찾는 것
- 예를 들어 은닉층의 개수라든지, 그 안에 들어갈 노드의 수, 최적화 함수의 종류를 바꾸어 보는 것
- 앞서 이야기한 바 있지만, 딥러닝 설정에 정답은 없음
- 자신의 데이터에 꼭 맞는 구조를 계속해서 테스트해 보며 찾는 것이 중요
- 데이터에 따라서는 딥러닝이 아닌 랜덤 포레스트, XGBoost, SVM 등 다른 알고리즘이 더 좋은 성과를 보일 때도 있음
- 일반적인 머신 러닝과 딥러닝을 합해서 더 좋은 결과를 만드는 것도 가능
- 많은 경험을 통해 최적의 성능을 보이는 모델을 만드는 것이 중요



4 모델 저장과 재사용



4 모델 저장과 재사용

● 모델 저장과 재사용

- 학습이 끝난 후 지금 만든 모델을 저장하면 언제든 이를 불러와 다시 사용할 수 있음
- 학습 결과를 저장하려면 `model.save()` 함수를 이용해 모델 이름을 적어 저장

```
# 모델 이름과 저장할 위치를 함께 지정합니다.
```

```
model.save('./data/model/my_model.hdf5')
```



4 모델 저장과 재사용

● 모델 저장과 재사용

- hdf5 파일 포맷은 주로 과학 기술 데이터 작업에서 사용되는데, 크고 복잡한 데이터를 저장하는 데 사용
- 이를 다시 불러오려면 케라스 API의 load_model 함수를 사용
- 앞서 Sequential 함수를 불러온 모델 클래스 안에 함께 들어 있으므로, Sequential 뒤에 load_model을 추가해 다시 불러옴

```
from tensorflow.keras.models import Sequential, load_model
```

①



4 모델 저장과 재사용

● 모델 저장과 재사용

- 테스트를 위해 조금 전 만든 모델을 메모리에서 삭제

```
del model
```



4 모델 저장과 재사용

● 모델 저장과 재사용

- `load_model()` 함수를 사용해서 조금 전 저장한 모델을 불러옴

```
# 모델이 저장된 위치와 이름까지 적어 줍니다.
```

```
model = load_model('./data/model/my_model.hdf5')
```



4 모델 저장과 재사용

● 모델 저장과 재사용

- 불러온 모델을 테스트셋에 적용해 정확도를 구함

```
score = model.evaluate(X_test, y_test)
print('Test accuracy:', score[1])
```



4 모델 저장과 재사용

● 모델 저장과 재사용

실행 결과

```
2/2 [=====] - 0s 2ms/step - loss: 0.4214 -  
accuracy: 0.8413  
Test accuracy: 0.841269850730896
```

- 이전 절에서 실행한 것과 같은 값이 나오는 것을 확인할 수 있음



5 k겹 교차 검증



5 k겹 교차 검증

● k겹 교차 검증

- 앞서 데이터가 충분히 많아야 모델 성능도 향상된다고 했음
- 이는 학습과 테스트를 위한 데이터를 충분히 확보할수록 세상에 나왔을 때 더 잘 작동하기 때문임
- 실제 프로젝트에서는 데이터를 확보하는 것이 쉽지 않거나 많은 비용이 발생하는 경우도 있음
- 가지고 있는 데이터를 십분 활용하는 것이 중요
- 특히 학습셋을 70%, 테스트셋을 30%로 설정할 경우 30%의 테스트셋은 학습에 이용할 수 없다는 단점이 있음
- 이를 해결하기 위해 고안된 방법이 k겹 교차 검증(k-fold cross validation)



5 k겹 교차 검증

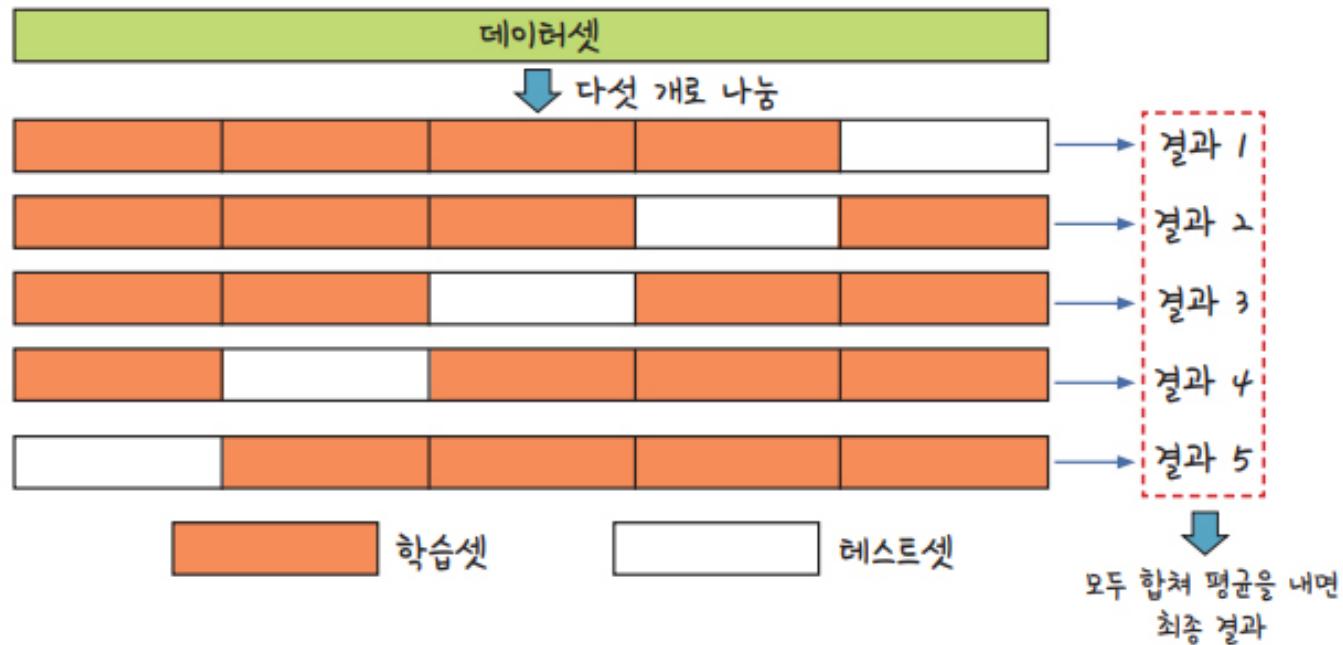
● k겹 교차 검증

- k겹 교차 검증이란 데이터셋을 여러 개로 나누어 하나씩 테스트셋으로 사용하고 나머지를 모두 합해서 학습셋으로 사용하는 방법
- 이렇게 하면 가지고 있는 데이터의 100%를 학습셋으로 사용할 수 있고, 또 동시에 테스트셋으로도 사용할 수 있음
- 예를 들어 5겹 교차 검증(5-fold cross validation)의 예가 그림 13-7에 설명되어 있음



5 k겹 교차 검증

▼ 그림 13-7 | 5겹 교차 검증의 도식





5 k겹 교차 검증

● k겹 교차 검증

- 데이터를 원하는 수만큼 나누어 각각 학습셋과 테스트셋으로 사용되게 하는 함수는 사이킷런 라이브러리의 KFold() 함수
- 실습 코드에서 KFold()를 활용하는 부분만 뽑아 보면 다음과 같음

```
from sklearn.model_selection import KFold
k = 5 .....❶
kfold = KFold(n_splits=k, shuffle=True) .....❷
acc_score = [] .....❸

for train_index, test_index in kfold.split(X): .....❹
    X_train, X_test = X.iloc[train_index,:], X.iloc[test_index,:]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
```



5 k겹 교차 검증

● k겹 교차 검증

- 먼저 몇 개의 파일로 나눌 것인지 ①해 k 변수에 넣음
- 사이킷런의 ② KFold() 함수를 불러옴
- 샘플이 어느 한쪽에 치우치지 않도록 shuffle 옵션을 True로 설정해 줌
- 정확도가 채워 ③ acc_score라는 이름의 빈 리스트를 준비
- ④ split()에 의해 k개의 학습셋, 테스트셋으로 분리되며 for 문에 의해 k번 반복



5 k겹 교차 검증

● k겹 교차 검증

- 반복되는 각 학습마다 정확도를 구해 다음과 같이 acc_score 리스트를 채움

```
accuracy = model.evaluate(X_test, y_test)[1] # 정확도를 구합니다.  
acc_score.append(accuracy) # acc_score 리스트에 저장합니다.
```

- k번의 학습이 끝나면 각 정확도들을 취합해 모델 성능을 평가



5 k겹 교차 검증

● k겹 교차 검증

- 모든 코드를 모아 실행하면 다음과 같음

실습 | 초음파 광물 예측하기: k겹 교차 검증



```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

import pandas as pd

# 깃허브에 준비된 데이터를 가져옵니다.
!git clone https://github.com/taehojo/data.git
```



5 k겹 교차 검증

● k겹 교차 검증

```
# 광물 데이터를 불러옵니다.  
df = pd.read_csv('./data/sonar3.csv', header=None)  
  
# 음파 관련 속성을 X로, 광물의 종류를 y로 저장합니다.  
X = df.iloc[:,0:60]  
y = df.iloc[:,60]  
  
# 몇 겹으로 나눌 것인지 정합니다.  
k = 5  
  
# KFold 함수를 불러옵니다. 분할하기 전에 샘플이 치우치지 않도록 섞어 줍니다.  
kfold = KFold(n_splits=k, shuffle=True)
```



5 k겹 교차 검증

● k겹 교차 검증

```
# 정확도가 채워질 빈 리스트를 준비합니다.  
acc_score = []  
  
def model_fn():  
    model = Sequential() # 딥러닝 모델의 구조를 시작합니다.  
    model.add(Dense(24, input_dim=60, activation='relu'))  
    model.add(Dense(10, activation='relu'))  
    model.add(Dense(1, activation='sigmoid'))  
    return model  
  
# k겹 교차 검증을 이용해 k번의 학습을 실행합니다.  
# for 문에 의해 k번 반복합니다.
```



5 k겹 교차 검증

● k겹 교차 검증

```
# split()에 의해 k개의 학습셋, 테스트셋으로 분리됩니다.  
  
for train_index, test_index in kfold.split(X):  
    X_train, X_test = X.iloc[train_index,:], X.iloc[test_index,:]  
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]  
  
    model = model_fn()  
    model.compile(loss='binary_crossentropy', optimizer='adam',  
    metrics=['accuracy'])  
    history = model.fit(X_train, y_train, epochs=200, batch_size=10,  
    verbose=0)  
  
    accuracy = model.evaluate(X_test, y_test)[1] # 정확도를 구합니다.  
    acc_score.append(accuracy) # 정확도 리스트에 저장합니다.
```



5 k겹 교차 검증

● k겹 교차 검증

```
# k번 실시된 정확도의 평균을 구합니다.  
avg_acc_score = sum(acc_score) / k  
  
# 결과를 출력합니다.  
print('정확도: ', acc_score)  
print('정확도 평균: ', avg_acc_score)
```



5 k겹 교차 검증

● k겹 교차 검증

실행 결과

```
2/2 [=====] - 0s 2ms/step - loss: 1.0080 -  
accuracy: 0.7381  
2/2 [=====] - 0s 2ms/step - loss: 0.7071 -  
accuracy: 0.8095  
2/2 [=====] - 0s 2ms/step - loss: 0.3312 -  
accuracy: 0.8810  
2/2 [=====] - 0s 2ms/step - loss: 0.4377 -  
accuracy: 0.9024  
2/2 [=====] - 0s 3ms/step - loss: 0.6416 - accuracy: 0.7317  
정확도: [0.738095223903656, 0.8095238208770752, 0.8809523582458496,  
0.9024389982223511, 0.7317073345184326]  
정확도 평균: 0.8125435471534729
```



5 k겹 교차 검증

● k겹 교차 검증

- 다섯 번의 정확도를 구했음
- 학습이 진행되는 과정이 길어서 model.fit 부분에 verbose=0 옵션을 주어 학습 과정의 출력력을 생략



5 k겹 교차 검증

● k겹 교차 검증

- 텐서플로 함수가 for 문에 포함되는 경우 다음과 같이 WARNING 메시지가 나오는 경우가 있음
- 텐서플로 구동에는 문제가 없으므로 그냥 진행하면 됨

WARNING:tensorflow:5 out of the last 9 calls to <function Model.make_test_function.<locals>.test_function at 0x000001EDE50D60D0> triggered tf.function retracing...



5 k겹 교차 검증

● k겹 교차 검증

- 이렇게 해서 가지고 있는 데이터를 모두 사용해 학습과 테스트를 진행
- 이제 다음 장을 통해 학습 과정을 시각화해 보는 방법과 학습을 몇 번 반복할지(epochs) 스스로 판단하게 하는 방법 등을 알아보며 모델 성능을 향상시켜 보겠음