AI 라이브러리 활용

14장 베스트 모델 만들기

이 찬 우

학습내용 : 베스트 모델 만들기

1 | 데이터의 확인과 실행

2 | 모델 업데이트하기

3 | 그래프로 확인하기

베스트 모델 만들기

- **실습 데이터** 와인의 종류 예측
  - dataset/wine.csv

- 포르투갈 서북쪽의 대서양을 맞닿고 위치한 비뉴 베르드(Vinho Verde) 지방에서 만들어진 와인을 측정한 데이터임
  → 레드와인 샘플 1,599개를 등급과 맛, 산도를 측정해 분석하고 화이트와인 샘플4,898개를 마찬가지로 분석해 데이터를 만듦

1 | 데이터의 확인과 실행

- 먼저 df_pre라는 공간에 데이터를 불러옴
- 그런 다음 sample( ) 함수를 사용하여 원본 데이터의 몇 %를 사용할지를 지정함

```
df_pre = pd.read_csv('../dataset/wine.csv', header=None)
df = df_pre.sample(frac=1)
```

1 | 데이터의 확인과 실행

- sample( ) 함수는 원본 데이터에서 정해진 비율만큼 랜덤으로 뽑아오는 함수임
- frac = 1이라고 지정하면 원본 데이터의 100%를 불러오라는 의미임
- frac = 0.5로 지정하면 50%만 랜덤으로 불러옴

```
print(df.head(5))
```

1 | 데이터의 확인과 실행

	0	1	2	3	4	5	6	7	8	9	10	11	12
964	8.5	0.47	0.27	1.9	0.058	18	38	0.99518	3.16	0.85	11.1	6	1
664	12.1	0.4	0.52	2	0.092	15	54	1	3.03	0.66	10.2	5	1
1692	6.9	0.21	0.33	1.8	0.034	48	136	0.9899	3.25	0.41	12.6	7	0
5801	6.7	0.24	0.31	2.3	0.044	37	113	0.99013	3.29	0.46	12.9	6	0
2207	6.1	0.28	0.25	17.75	0.044	48	161	0.9993	3.34	0.48	9.5	5	0

1 | 데이터의 확인과 실행

- 한 줄당 모두 13개의 정보가 있음(0~12)
- 이어서 전체 정보를 출력해보자

```
print(df.info( ))
```

1 | 데이터의 확인과 실행

Data	columns (total 13 columns):		
0	6497	non-null	float64
1	6497	non-null	float64
2	6497	non-null	float64
3	6497	non-null	float64
4	6497	non-null	float64
5	6497	non-null	float64
6	6497	non-null	float64
7	6497	non-null	float64
8	6497	non-null	float64
9	6497	non-null	float64
10	6497	non-null	float64
11	6497	non-null	int64
12	6497	non-null	int64
dtypes: float64(11), int64(2)			
memory usage: 710.6 KB			

1 | 데이터의 확인과 실행

- 총 6497개의 샘플이 있음을 알 수 있음
- 13개의 속성이 각각 무엇인지는 데이터를 내려받은 UCI 머신러닝 저장소에서 확인할 수 있음

0	주석산 농도	7	밀도
1	아세트산 농도	8	pH
2	구연산 농도	9	황산칼륨 농도
3	잔류 당분 농도	10	알코올 도수
4	염화나트륨 농도	11	와인의 맛(0~10등급)
5	유리 아황산 농도	12	class (1: 레드와인, 0: 화이트와인)
6	총 아황산 농도		

1 | 데이터의 확인과 실행

- 이제 0~11까지에 해당하는 12개의 정보를 가지고 13번째 클래스를 맞추는 과제임을 확인함
- 이 정보를 토대로 X 값과 Y 값을 다음과 같이 정함

```
dataset = df.values
X = dataset[:,0:12]
Y = dataset[:,12]
```

1 | 데이터의 확인과 실행

코드 14-1 와인의 종류 예측하기: 데이터 확인과 실행

- 예제 소스: run_project/08_Wine.ipynb

```
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint, EarlyStopping

import pandas as pd
import numpy
import tensorflow as tf
import matplotlib.pyplot as plt
```

1 | 데이터의 확인과 실행

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


# 데이터 입력
df_pre = pd.read_csv('../dataset/wine.csv', header=None)
df = df_pre.sample(frac=1)
dataset = df.values
X = dataset[:,0:12]
Y = dataset[:,12]
```

1 | 데이터의 확인과 실행

```
# 모델 설정
model = Sequential( )
model.add(Dense(30, input_dim=12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# 모델 컴파일
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

1 | 데이터의 확인과 실행

```
# 모델 실행
model.fit(X, Y, epochs=200, batch_size=200)


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

1 | 데이터의 확인과 실행

실행 결과

```
Epoch 198/200
6497/6497 [==============================] - 0s 6us/step - loss: 0.0496
- accuracy: 0.9858
Epoch 199/200
6497/6497 [==============================] - 0s 7us/step - loss: 0.0499
- accuracy: 0.9848
Epoch 200/200
6497/6497 [==============================] - 0s 6us/step - loss: 0.0493
- accuracy: 0.9865
6497/6497 [==============================] - 0s 18us/step

 Accuracy: 0.9858
```

2 | 모델 업데이트하기

- 모델을 그냥 저장하는 것이 아니라 에포크(epoch)마다
  모델의 정확도를 함께 기록하면서 저장해 보자

```
import os

MODEL_DIR = './model/'                  # 모델을 저장하는 폴더
if not os.path.exists(MODEL_DIR):       # 만일 위의 폴더가 존재하지 않으면
    os.mkdir(MODEL_DIR)                 # 이 이름의 폴더를 만들어 줌

modelpath="./model/{epoch:02d}-{val_loss:.4f}.hdf5"
```

2 | 모델 업데이트하기

- 이제 모델을 저장하기 위해 케라스의 콜백 함수 중 ModelCheckpoint( ) 함수를 불러옴

```
from keras.callbacks import ModelCheckpoint
```

2 | 모델 업데이트하기

- checkpointer라는 변수를 만들어 이곳에 모니터할 값을 지정함
- 테스트 오차는 케라스 내부에서 val_loss로 기록됨

  → 참고로 학습 정확도는 acc, 테스트셋 정확도는 val_acc,

  학습셋 오차는 loss로 각각 기록됨

```
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_
loss', verbose=1)
```

2 | 모델 업데이트하기

- 이제 모델을 학습할 때마다 위에서 정한 checkpointer의 값을 받아 지정된 곳에 모델을 저장함

```
model.fit(X, Y, validation_split=0.2, epochs=200, batch_size=200,
verbose=0, callbacks=[checkpointer])
```

2 | 모델 업데이트하기

```
Epoch 00194: saving model to ./model/194-0.0629.hdf5
Epoch 00195: saving model to ./model/195-0.0636.hdf5
Epoch 00196: saving model to ./model/196-0.0630.hdf5
Epoch 00197: saving model to ./model/197-0.0695.hdf5
Epoch 00198: saving model to ./model/198-0.0724.hdf5
Epoch 00199: saving model to ./model/199-0.0635.hdf5
```

2 | 모델 업데이트하기

- 저장된 파일의 이름이 곧 에포크 수와 이때의 테스트셋 오차 값임
- 이때 ModelCheckpoint( ) 함수에 모델이 앞서 저장한 모델보다 나아졌을 때만 저장하게끔 하려면 save_best_only 값을 True로 지정함

```
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss', verbose=1, save_best_only=True)
```

2 | 모델 업데이트하기

코드 14-2 와인의 종류 예측하기: 모델 업데이트

- 예제 소스: run_project/09_Wine_Checkpoint.ipynb

```
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint

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

2 | 모델 업데이트하기

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

df_pre = pd.read_csv('../dataset/wine.csv', header=None)
df = df_pre.sample(frac=1)

dataset = df.values
X = dataset[:,0:12]
Y = dataset[:,12]
```

2 | 모델 업데이트하기

```
# 모델 설정
model = Sequential( )
model.add(Dense(30, input_dim=12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# 모델 컴파일
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

2 | 모델 업데이트하기

```
# 모델 저장 폴더 설정
MODEL_DIR = './model/'
if not os.path.exists(MODEL_DIR):
    os.mkdir(MODEL_DIR)

# 모델 저장 조건 설정
modelpath="./model/{epoch:02d}-{val_loss:.4f}.hdf5"
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_
loss', verbose=1, save_best_only=True)

# 모델 실행 및 저장
model.fit(X, Y, validation_split=0.2, epochs=200, batch_size=200,
verbose=0, callbacks=[checkpointer])
```

2 | 모델 업데이트하기

실행 결과

(중략)

Epoch 00183: val_loss improved from 0.04916 to 0.04863, saving model to ./model/183-0.0486.hdf5

Epoch 00184: val_loss did not improve

Epoch 00185: val_loss did not improve

Epoch 00186: val_loss did not improve

Epoch 00187: val_loss did not improve

Epoch 00188: val_loss did not improve

Epoch 00189: val_loss did not improve

2 | 모델 업데이트하기

```
Epoch 00190: val_loss did not improve
Epoch 00191: val_loss did not improve
Epoch 00192: val_loss did not improve
Epoch 00193: val_loss improved from 0.04863 to 0.04855, saving model to ./
model/193-0.0486.hdf5
Epoch 00194: val_loss did not improve
Epoch 00195: val_loss did not improve
Epoch 00196: val_loss did not improve
Epoch 00197: val_loss did not improve
Epoch 00198: val_loss improved from 0.04855 to 0.04823, saving model to ./
model/198-0.0482.hdf5
Epoch 00199: val_loss did not improve
```

3 | 그래프로 확인하기

- 딥러닝 프레임워크가 만들어 낸 모델을 업데이트 하는 과정임
- 이를 위해서는 에포크를 얼마나 지정할지를 결정해야 함
- 학습을 반복하는 횟수가 너무 적어도 안 되고

  또 너무 많아도 과적합을 일으키므로 문제가 있음
- 모델의 학습 시간에 따른 정확도와 테스트 결과를 그래프를 통해 확인해 보자

```
df = df_pre.sample(frac=0.15)

history = model.fit(X, Y, validation_split=0.33, epochs=3500,
batch_size=500)
```

3 | 그래프로 확인하기

- 다음으로 그래프로 표현하기 위한 라이브러리를 불러오고 오차와 정확도의 값을정함
- y_vloss에 테스트셋(33%)으로 실험한 결과의 오차 값을 저장함
- y_acc에 학습셋(67%)으로 측정한 정확도의 값을 저장함

```
import matplotlib.pyplot as plt


y_vloss=history.history['val_loss']
y_acc=history.history['acc']
```

3 | 그래프로 확인하기

- x 값을 지정하고 정확도를 파란색으로, 오차를 빨간색으로 표시해보자

```
x_len = numpy.arange(len(y_acc))
plt.plot(x_len, y_vloss, "o", c="red", markersize=3)
plt.plot(x_len, y_acc, "o", c="blue", markersize=3)
```

3 | 그래프로 확인하기

코드 14-3 와인의 종류 예측하기: 그래프 표현

- 예제 소스: run_project/10_Wine_Overfit_Graph.ipynb

```
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint

import pandas as pd
import numpy
import os
import matplotlib.pyplot as plt
import tensorflow as tf
```

3 | 그래프로 확인하기

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

df_pre = pd.read_csv('../dataset/wine.csv', header=None)
df = df_pre.sample(frac=0.15)

dataset = df.values
X = dataset[:,0:12]
Y = dataset[:,12]
```

3 | 그래프로 확인하기

```
# 모델의 설정
model = Sequential()
model.add(Dense(30, input_dim=12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# 모델 컴파일
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

3 | 그래프로 확인하기

```
# 모델 저장 폴더 설정
MODEL_DIR = './model/'
if not os.path.exists(MODEL_DIR):
    os.mkdir(MODEL_DIR)

# 모델 저장 조건 설정
modelpath="./model/{epoch:02d}-{val_loss:.4f}.hdf5"
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss', verbose=1, save_best_only=True)

# 모델 실행 및 저장
history = model.fit(X, Y, validation_split=0.33, epochs=3500, batch_size=500)
```

3 | 그래프로 확인하기

```
# y_vloss에 테스트셋으로 실험 결과의 오차 값을 저장
y_vloss=history.history['val_loss']

# y_acc에 학습셋으로 측정한 정확도의 값을 저장
y_acc=history.history['acc']

# x 값을 지정하고 정확도를 파란색으로, 오차를 빨간색으로 표시
x_len = numpy.arange(len(y_acc))
plt.plot(x_len, y_vloss, "o", c="red", markersize=3)
plt.plot(x_len, y_acc, "o", c="blue", markersize=3)

plt.show()
```

3 | 그래프로 확인하기

실행 결과

```
Epoch 3497/3500
653/653 [==============================] - 0s - loss: 0.0119 - acc:
0.9954 - val_loss: 0.0976 - val_acc: 0.9783
Epoch 3498/3500
653/653 [==============================] - 0s - loss: 0.0102 - acc:
0.9985 - val_loss: 0.0885 - val_acc: 0.9783
Epoch 3499/3500
653/653 [==============================] - 0s - loss: 0.0123 - acc:
0.9954 - val_loss: 0.0933 - val_acc: 0.9783
Epoch 3500/3500
653/653 [==============================] - 0s - loss: 0.0097 - acc:
0.9969 - val_loss: 0.0958 - val_acc: 0.9783
```

3 | 그래프로 확인하기

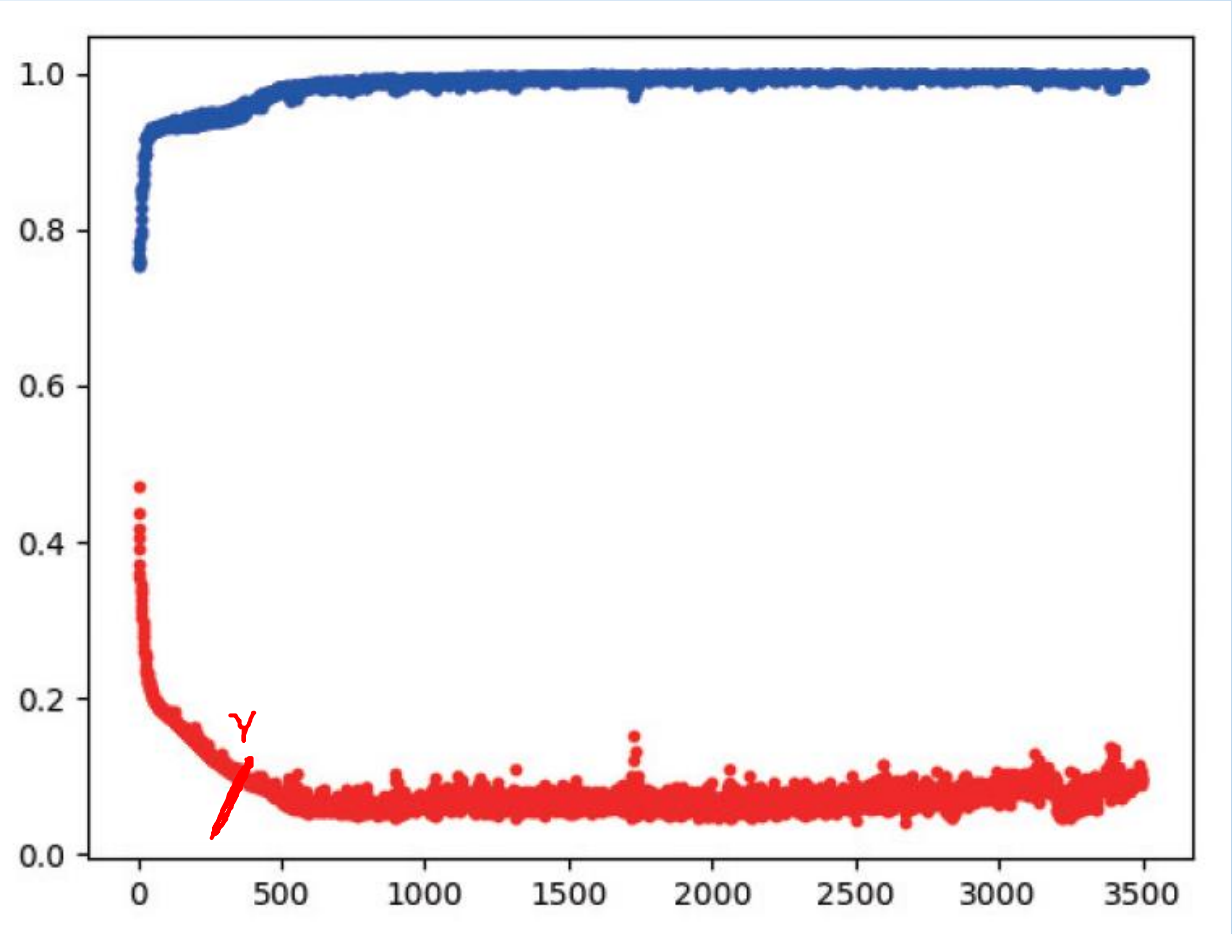

그림 14-1 학습 진행에 따른 학습셋과 테스트셋의 정확도 그래프

3 | 그래프로 확인하기

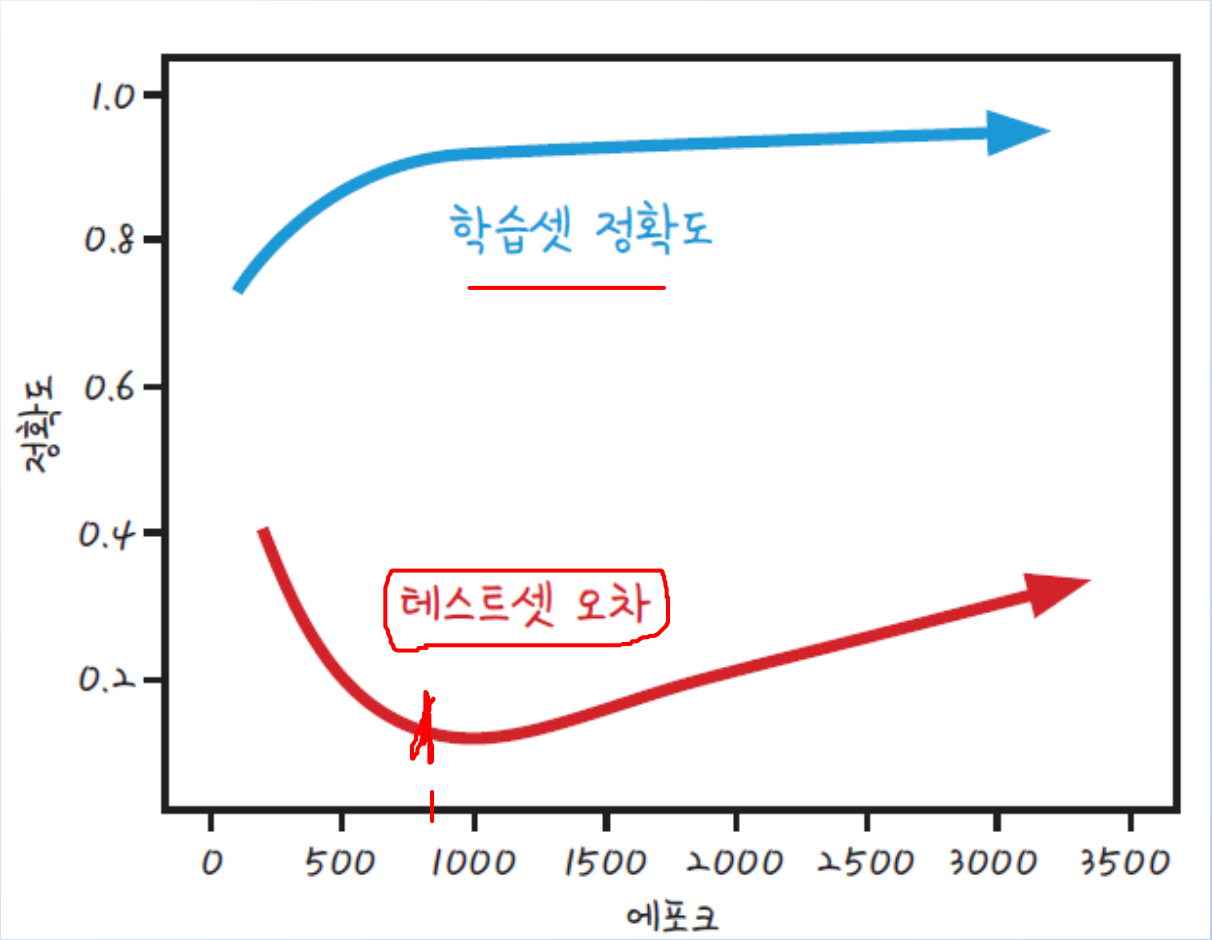

그림 14-2 학습이 진행될수록 학습셋의 정확도는 오르지만 테스트셋에서는 과적합 발생

정리 학습 : 베스트 모델 만들기

1 | 데이터의 확인과 실행

2 | 모델 업데이트하기

3 | 그래프로 확인하기

다음 수업

학습의 자동 중단
선형회귀 적용하기