10장. 모델 설계하기

• 폐암 수술 환자의 생존율 예측하기

```python
# 딥러닝을 구동하는 데 필요한 케라스 함수를 불러옵니다.
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# 필요한 라이브러리를 불러옵니다.
import numpy as np
import tensorflow as tf

# 실행할 때마다 같은 결과를 출력하기 위해 설정하는 부분입니다.
np.random.seed(3)
tf.random.set_seed(3)

# 준비된 수술 환자 데이터를 불러들입니다.
Data_set = np.loadtxt("ThoraricSurgery.csv", delimiter=",")

# 환자의 기록과 수술 결과를 X와 Y로 구분하여 저장합니다.
X = Data_set[:,0:17]
Y = Data_set[:,17]
1
# 딥러닝 구조를 결정합니다(모델을 설정하고 실행하는 부분입니다).
model = Sequential()
model.add(Dense(30, input_dim=17, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# 딥러닝을 실행합니다.
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, Y, epochs=100, batch_size=10)
```

Q. 클래스는 마지막에 추가?

17 (속성)
샘플 ?

구조, 층 설정

→ 30개의 노드
→ 은닉층 + 입력층
→ 데이터 열개
→ 출력층
두개의 층을 가짐
출력값을 하나로 정해야함
컴파일
→ 실제 수행 부분
→ 열개씩 끊어서 넣음
컴파일 시 수행결과

표 10-1
대표적인 오차 함수

* 실제 값을 yt, 예측 값을 yo라고 가정할 때

평균 제곱 계열	mean_squared_error	평균 제곱 오차 계산: mean(square(yt - yo))
	mean_absolute_error	평균 절대 오차(실제 값과 예측 값 차이의 절댓값 평균) 계산: mean(abs(yt - yo))
	mean_absolute_percentage_error	평균 절대 백분율 오차(절댓값 오차를 절댓값으로 나눈 후 평균) 계산: mean(abs(yt - yo)/abs(yt)) (단, 분모 ≠ 0)
	mean_squared_logarithmic_error	평균 제곱 로그 오차(실제 값과 예측 값에 로그를 적용한 값의 차이를 제곱한 값의 평균) 계산: mean(square((log(yo) + 1) - (log(yt) + 1)))
교차 엔트로피 계열	categorical_crossentropy	범주형 교차 엔트로피(일반적인 분류)
	binary_crossentropy	이항 교차 엔트로피(두 개의 클래스 중에서 예측할 때)

11장. 데이터 다루기

★좋은 데이터

- 한쪽으로 치우치지 않음
- 불필요한 정보X
- 왜곡되지 않음

★피마 인디언 데이터 분석

```
# pandas 라이브러리를 불러옵니다.
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# 피마 인디언 당뇨병 데이터셋을 불러옵니다. 불러올 때 각 컬럼에 해당하는 이름을 지정합니다.
df = pd.read_csv('pima-indians-diabetes.csv',
                 names = ["pregnant", "plasma", "pressure", "thickness", "insulin", "BMI", "pedigree", "age", "class"])
```

```
# 처음 5줄을 봅니다.
print(df.head(5))
```

```
   pregnant  plasma  pressure  thickness  insulin   BMI  pedigree  age  class
0         6     148        72         35        0  33.6     0.627   50      1
1         1      85        66         29        0  26.6     0.351   31      0
2         8     183        64          0        0  23.3     0.672   32      1
3         1      89        66         23       94  28.1     0.167   21      0
4         0     137        40         35      168  43.1     2.288   33      1
```

```
# 데이터의 전반적인 정보를 확인해 봅니다.
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   pregnant   768 non-null    int64
 1   plasma     768 non-null    int64
 2   pressure   768 non-null    int64
 3   thickness  768 non-null    int64
 4   insulin    768 non-null    int64
 5   BMI        768 non-null    float64
 6   pedigree   768 non-null    float64
 7   age        768 non-null    int64
 8   class      768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None
```

```
# 각 정보별 특징을 좀더 자세히 출력합니다.
print(df.describe())
```

```
          pregnant      plasma    pressure  ...    pedigree         age       class
count  768.000000  768.000000  768.000000  ...  768.000000  768.000000  768.000000
mean     3.845052  120.894531   69.105469  ...    0.471876   33.240885    0.348958
std      3.369578   31.972618   19.355807  ...    0.331329   11.760232    0.476951
min      0.000000    0.000000    0.000000  ...    0.078000   21.000000    0.000000
25%      1.000000   99.000000   62.000000  ...    0.243750   24.000000    0.000000
50%      3.000000  117.000000   72.000000  ...    0.372500   29.000000    0.000000
75%      6.000000  140.250000   80.000000  ...    0.626250   41.000000    1.000000
max     17.000000  199.000000  122.000000  ...    2.420000   81.000000    1.000000

[8 rows x 9 columns]
```

```
# 데이터 중 임신 정보와 클래스 만을 출력해 봅니다.
print(df[['plasma', 'class']])
```

```
     plasma  class
0       148      1
1        85      0
2       183      1
3        89      0
4       137      1
..      ...    ...
763     101      0
764     122      0
765     121      0
766     126      1
767      93      0

[768 rows x 2 columns]
```

• 데이터 가공하기 : 당뇨병 발병확률계산

```
print(df[['pregnant','class']].groupby(['pregnant'],
                                       as_index=False).mean().sort_values(by='pregnant',ascending=Tr
```

```
    pregnant     class
0          0  0.342342
1          1  0.214815
2          2  0.184466
3          3  0.360000
4          4  0.338235
5          5  0.368421
6          6  0.320000
7          7  0.555556
8          8  0.578947
9          9  0.642857
10        10  0.416667
11        11  0.636364
12        12  0.444444
13        13  0.500000
14        14  1.000000
15        15  1.000000
16        17  1.000000
```

• 상관관계 그래프 표현

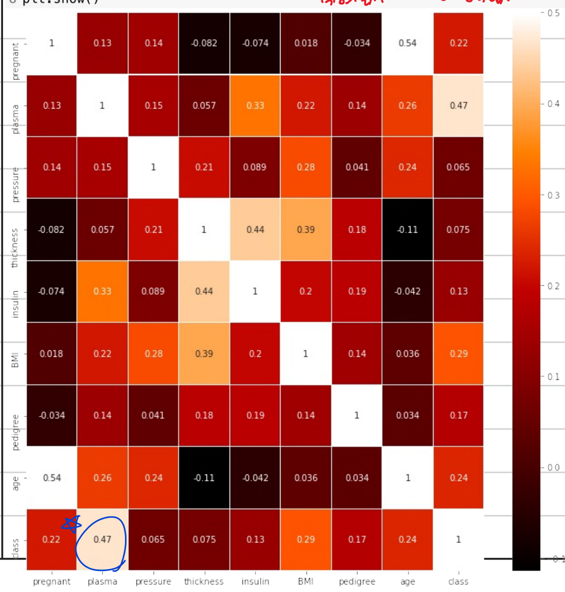
```
# 데이터 간의 상관관계를 그래프로 표현해 봅니다.

colormap = plt.cm.gist_heat   #그래프의 색상 구성을 정합니다.
plt.figure(figsize=(12,12))   #그래프의 크기를 정합니다.

# 그래프의 속성을 결정합니다. vmax의 값을 0.5로 지정해 0.5에 가까울 수록 밝은 색으로 표시되게 합니다.
sns.heatmap(df.corr(),linewidths=0.1,vmax=0.5, cmap=colormap, linecolor='white', annot=True)
plt.show()
```

상관관계 (서로 비슷패턴일수록 1에 가까움)

vmax=0.5 : 색상의 밝기

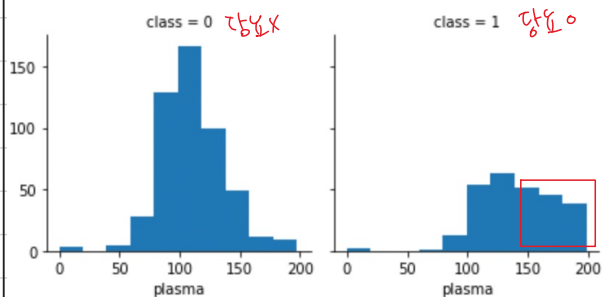
cmap=colormap : 미리정해진색상값

annot=True : 각 셀에 숫자 입력



• 두 항목간의 관계 그래프 확인

```
grid = sns.FacetGrid(df, col='class')
grid.map(plt.hist, 'plasma',  bins=10)
plt.show()
```



∴ 당뇨병 환자의 경우 150 이상의 수치가 높음

• 당뇨병 예측 실행

```
# 딥러닝을 구동하는 데 필요한 케라스 함수를 불러옵니다.
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# 필요한 라이브러리를 불러옵니다.
import numpy
import tensorflow as tf

# 실행할 때마다 같은 결과를 출력하기 위해 설정하는 부분입니다.
numpy.random.seed(3)
tf.random.set_seed(3)

# 데이터를 불러 옵니다.
dataset = numpy.loadtxt("pima-indians-diabetes.csv", delimiter=",")
X = dataset[:,0:8]
Y = dataset[:,8]

# 모델을 설정합니다.
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# 모델을 컴파일합니다.
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# 모델을 실행합니다.
model.fit(X, Y, epochs=200, batch_size=10)

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

```
Epoch 1/200
77/77 [==============================] - 1s 1ms/step - loss: 12.0693 - accuracy: 0.
Epoch 2/200
77/77 [==============================] - 0s 1ms/step - loss: 5.7426 - accuracy: 0.6
Epoch 3/200
77/77 [==============================] - 0s 1ms/step - loss: 3.2875 - accuracy: 0.5
Epoch 4/200
77/77 [==============================] - 0s 685us/step - loss: 1.6549 - accuracy: 0
Epoch 5/200
77/77 [==============================] - 0s 654us/step - loss: 0.8960 - accuracy: 0
Epoch 6/200
```

12장. 다중 분류 문제 해결하기

※다중분류 : 여러개의 답 중 하나를 고르는 분류

<실습>

각 레이어에 하나의 입력 텐서와 하나의 출력 텐서가 있는 경우 → Sequential

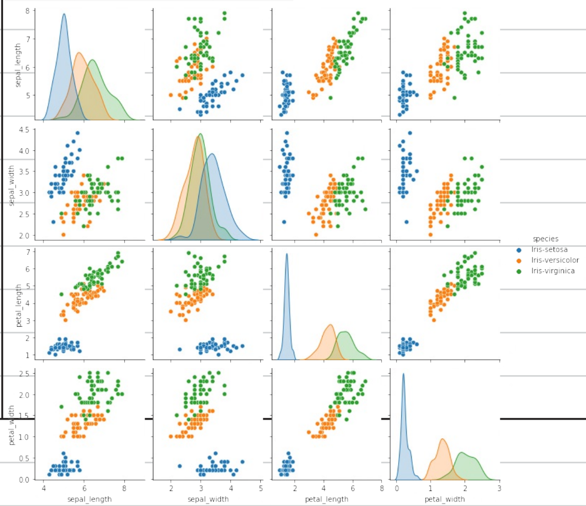
```python
from keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.preprocessing import LabelEncoder

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

↳ 클래스 이름을 숫자 형태로 변경 (라벨링) — LabelEncoder

```python
# 실행할 때마다 같은 결과를 출력하기 위해 설정하는 부분입니다.
np.random.seed(3)
tf.random.set_seed(3)

# 데이터 입력
df = pd.read_csv('iris.csv', names = ["sepal_length", "sepal_width", "petal_length", "petal_width", "species"])

# 데이터 확인
print(df.head())
```

```
   sepal_length  sepal_width  petal_length  petal_width      species
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
```

```python
# 그래프로 확인
sns.pairplot(df, hue='species');
plt.show()
```



#원-핫 인코딩

```
# 데이터 분류
dataset = df.values
X = dataset[:,0:4].astype(float)
Y_obj = dataset[:,4]

# 문자열을 숫자로 변환
e = LabelEncoder()
e.fit(Y_obj)
Y = e.transform(Y_obj)
Y_encoded = tf.keras.utils.to_categorical(Y)

# 모델의 설정
model = Sequential()
model.add(Dense(16,  input_dim=4, activation='relu'))
model.add(Dense(3, activation='softmax'))

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

# 모델 실행
model.fit(X, Y_encoded, epochs=50, batch_size=1)

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

(라벨링)

[1,2,3] ⇒ [1,0,0],[0,1,0],[0,0,1]

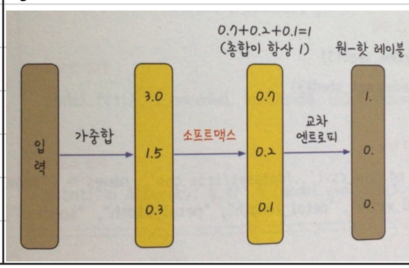
↳활성화함수적용위해 0과 1로 변형

input_dim → ?

categorical_crossentropy ↳다중분류에 적합

☆소프트맥스

총합이 1인 형태로 바꿔서 계산해줌



• 결과

```
Epoch 36/50
150/150 [==============================] - 0s 980us/step - loss: 0.1454 - accuracy: 0.9583
Epoch 37/50
150/150 [==============================] - 0s 977us/step - loss: 0.1459 - accuracy: 0.9488
Epoch 38/50
150/150 [==============================] - 0s 902us/step - loss: 0.1260 - accuracy: 0.9785
Epoch 39/50
150/150 [==============================] - 0s 944us/step - loss: 0.1291 - accuracy: 0.9706
Epoch 40/50
150/150 [==============================] - 0s 1ms/step - loss: 0.1391 - accuracy: 0.9885
Epoch 41/50
150/150 [==============================] - 0s 973us/step - loss: 0.1157 - accuracy: 0.9794
Epoch 42/50
150/150 [==============================] - 0s 1ms/step - loss: 0.1109 - accuracy: 0.9728
Epoch 43/50
150/150 [==============================] - 0s 922us/step - loss: 0.1214 - accuracy: 0.9752
Epoch 44/50
150/150 [==============================] - 0s 983us/step - loss: 0.1119 - accuracy: 0.9651
Epoch 45/50
150/150 [==============================] - 0s 916us/step - loss: 0.1209 - accuracy: 0.9689
Epoch 46/50
150/150 [==============================] - 0s 974us/step - loss: 0.0930 - accuracy: 0.9820
Epoch 47/50
150/150 [==============================] - 0s 889us/step - loss: 0.1204 - accuracy: 0.9622
Epoch 48/50
150/150 [==============================] - 0s 922us/step - loss: 0.0898 - accuracy: 0.9737
Epoch 49/50
150/150 [==============================] - 0s 1ms/step - loss: 0.1079 - accuracy: 0.9547
Epoch 50/50
150/150 [==============================] - 0s 925us/step - loss: 0.0912 - accuracy: 0.9600
5/5 [==============================] - 0s 3ms/step - loss: 0.1024 - accuracy: 0.9733

Accuracy: 0.9733
```