11장 데이터 다루기

목 차

데이터 다루기

- **실습 데이터** 피마 인디언 당뇨병 예측
  - dataset/pima-indians-diabetes.csv

1 딥러닝과 데이터

- 데이터의 양보다 훨씬 중요한 것은, '필요한' 데이터가 얼마나 많은가임
- 준비된 데이터가 우리가 사용하려는 머신러닝과 딥러닝에 얼마나 효율적으로사용되게끔 가공됐는지가 역시 중요함
- 머신러닝 프로젝트의 성공과 실패는 얼마나 좋은 데이터를 가지고 시작하느냐에 영향을 많이 받음
- 여기서 좋은 데이터란 내가 알아내고자 하는 정보를 잘 담고 있는 데이터를 말함
- 한쪽으로 치우치지 않고, 불필요한 정보를 가지고 있지 않으며, 왜곡되지 않은 데이터여야 함

1 딥러닝과 데이터

- 머신러닝, 딥러닝 개발자들은 데이터를 들여다 보고 분석할 수 있어야 함
- 내가 이루고 싶은 목적에 맞춰 가능한 한 많은 정보를 모았다면 이를 머신러닝과 딥러닝에서 사용할 수 있게 잘 정제된 데이터 형식으로 바꿔야 함
- 이 작업은 모든 머신러닝 프로젝트의 첫 단추이자 가장 중요한 작업

2 피마 인디언 데이터 분석하기

그림 11-1 피마 인디언 옛 모습

2 피마 인디언 데이터 분석하기

- 비만이 유전 및 환경, 모두의 탓이라는 것을 증명하는 좋은 사례가 바로 미국 남서부에 살고 있는 피마 인디언의 사례

속성 / 클래스

	정보 1	정보 2	정보 3	…	정보 8	당뇨병 여부
샘플	1번째 인디언	6	148	72	…	50	1
	2번째 인디언	1	85	66	…	31	0
	3번째 인디언	8	183	64	…	32	1
	…	…	…	…	…	…	…
	768번째 인디언	1	93	70	…	23	0

표 11-1 피마 인디언 데이터의 샘플, 속성, 클래스 구분

2 피마 인디언 데이터 분석하기

- 샘플 수: 768
- 속성: 8
  - 정보 1 (pregnant): 과거 임신 횟수
  - 정보 2 (plasma): 포도당 부하 검사 2시간 후 공복 혈당 농도(mm Hg)
  - 정보 3 (pressure): 확장기 혈압(mm Hg)
  - 정보 4 (thickness): 삼두근 피부 주름 두께(mm)
  - 정보 5 (insulin): 혈청 인슐린(2-hour, mu U/ml)
  - 정보 6 (BMI): 체질량 지수(BMI, weight in kg/(height in $m)^2$)
  - 정보 7 (pedigree): 당뇨병 가족력
  - 정보 8 (age): 나이
- 클래스: 당뇨(1), 당뇨 아님(0)

2 피마 인디언 데이터 분석하기

- 데이터의 각 정보가 의미하는 의학, 생리학 배경 지식을 모두 알 필요는 없지만, 딥러닝을 구동하려면 반드시 속성과 클래스를 먼저 구분해야 함
- 모델의 정확도를 향상시키기 위해서는 데이터의 추가 및 재가공이 필요할 수도 있음
- 딥러닝의 구동에 앞서 데이터의 내용과 구조를 잘 파악하는 것이 중요함

3 pandas를 활용한 데이터 조사

- 데이터를 잘 파악하는 것이 딥러닝을 다루는 기술의 1단계
- 데이터의 크기가 커지고 정보량이 많아지면 데이터를 불러오고 내용을 파악할 수 있는 효과적인 방법이 필요함
- 이때 가장 유용한 방법이 데이터를 시각화해서 눈으로 직접 확인해 보는 것

3 pandas를 활용한 데이터 조사

- 데이터를 다룰 때에는 데이터를 다루기 위해 만들어진 라이브러리를 사용하는 것이 좋음
- 파이썬 데이터 관련 라이브러리 중 pandas를 사용해 데이터를 불러와 보겠음(run_project/02_Pima_Indian.ipynb)

```
import pandas as pd
df = pd.read_csv('../dataset/pima-indians-diabetes.csv',
                 names = ["pregnant", "plasma", "pressure", "thickness",
                 "insulin", "BMI", "pedigree", "age", "class"])
```

3 pandas를 활용한 데이터 조사

- Csv :

  comma separated values file의 약자로, 콤마(,)로 구분된 데이터들의 모음이란 뜻

- 헤더(header) :

  csv 파일에는 데이터를 설명하는 한 줄이 파일 맨 처음에 나옴

- 우리가 가진 csv 파일에는 헤더가 없음
- 이에 names라는 함수를 통해 속성별 키워드를 지정해 줌

3 pandas를 활용한 데이터 조사

- 이제 불러온 데이터의 내용을 간단히 확인하고자 head( ) 함수를 이용하여 데이터의 첫 다섯 줄을 불러옴

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

3 pandas를 활용한 데이터 조사

- 정보마다 이름이 잘 지정된 것을 볼 수 있음
- 파이썬에서는 숫자를 0부터 세기 때문에 맨 첫 번째 행이 1이 아닌 0이

	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

3 pandas를 활용한 데이터 조사

- 이제 이 데이터의 전반적인 정보를 확인해 보자

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

3 pandas를 활용한 데이터 조사

- 항목별로 768개가 빠짐없이 들어있음을 나타내고, 각 정보의 형식을 알려 줌

Range Index: 768 entries, 0 to 767

Data	Columns (total 9)		
pregnant	768	non-null	int64
plasma	768	non-null	int64
pressure	768	non-null	int64
thickness	768	non-null	int64
insulin	768	non-null	int64
BMI	768	non-null	float64
pedigree	768	non-null	float64
age	768	non-null	int64
class	768	non-null	int64
Dtypes: float64(2) int64(7)			
Memory usage: 54.1 KB			

3 pandas를 활용한 데이터 조사

- 정보별 특징을 좀 더 자세히 알고 싶으면 describe( ) 함수를 이용함

```
print(df.describe( ))
```

3 pandas를 활용한 데이터 조사

- 정보별 샘플 수(count) , 평균(mean) , 표준편차(std), 최솟값(min), 백분위 수로 25%, 50%, 75%에 해당하는 값 그리고 최댓값(max)이 정리되어 보임

	pregnant	plasma	pressure	thickness	insulin	BMI	pedigree	age	class
count	768	768	768	768	768	768	768	768	768
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.88416	0.331329	11.760232	0.476951
min	0	0	0	0	0	0	0.078	21	0
25%	1	99	62	0	0	27.3	0.24375	24	0
50%	3	117	72	23	30.5	32	0.3725	29	0
75%	6	140.25	80	32	127.25	36.6	0.62625	41	1
max	17	199	122	99	846	67.1	2.42	81	1

3 pandas를 활용한 데이터 조사

- 데이터의 일부 컬럼만 보고 싶을 때, 예를 들어 임신 횟수(pregnant)와 당뇨병 발병 여부(class)만 확인해 보고 싶다면 다음과 같이 입

```
print(df[['pregnant', 'class']])
```

3 pandas를 활용한 데이터 조사

	pregnant	class
0	6	1
1	1	0
2	8	1
3	1	0
4	0	1
5	5	0
...	...	...
765	5	0
766	1	1
767	1	0

4 데이터 가공하기

- 데이터를 잘 다루려면 데이터를 한 번 더 가공해야 함
- 데이터를 가공할 때 우리가 무엇을 위해 작업을 하는지 그 목적을 잊어서는 안 됨
- 이 프로젝트의 목적은 당뇨병 발병을 예측하는 것
- 모든 정보는 당뇨병 발병과 어떤 관계가 있는지를 중점에 놓아야 함

4 데이터 가공하기

- 앞서 살펴본 임신 횟수와 당뇨병 발병 확률의 경우 다음과 같이 계산할 수 있음

```
print(df[['pregnant','class']].groupby(['pregnant'], as_
index=False).mean().sort_values(by='pregnant', ascending=True))
```

4 데이터 가공하기

- groupby() 함수를 사용해 pregnant 정보를 기준으로 하는 새 그룹을 만듦
- as_index=False는 pregnant 정보 옆에 새로운 인덱스(index)를 만듦
- mean() 함수를 사용해 평균을 구하고 sort_values() 함수를 써서 pregnant 컬럼을 오름차순(ascending)으로 정리하게끔 설정함

4 데이터 가공하기

- 이를 출력하면 다음과 같이 임신 횟수당 당뇨병 발병 확률을 구할 수 있음

	pregnant	class
0	0	0.342342
1	1	0.214815
2	2	0.184466
3	3	0.36
4	4	0.338235
5	5	0.368421
6	6	0.32
7	7	0.555556
8	8	0.578947
9	9	0.642857
10	10	0.416667

	pregnant	class
11	11	0.636364
12	12	0.444444
13	13	0.5
14	14	1
15	15	1
16	17	1

5 matplotlib를 이용해 그래프로 표현하기

- 데이터를 그래프로 표현해서 그 성격을 파악 하는 것이 중요함
- matplotlib는 파이썬에서 그래프를 그릴 때 가장 많이 사용되는 라이브러리임
- matplotlib 라이브러리와 이를 기반으로 좀 더 정교한 그래프를 그리게끔 도와주는 seaborn 라이브러리를 사용해 각 정보끼리 어떤 상관관계가 있는지를 알아보자

```
import matplotlib.pyplot as plt
import seaborn as sns
```

5 matplotlib를 이용해 그래프로 표현하기

- 먼저 그래프의 크기를 결정함

```
plt.figure(figsize=(12,12))
```

5 matplotlib를 이용해 그래프로 표현하기

- heatmap( ) 함수 :

  두 항목씩 짝을 지은 뒤 각각 어떤 패턴으로 변화하는지를 관찰하는 함수

- 두 항목이 전혀 다른 패턴으로 변화하고 있으면 0을, 서로 비슷한 패턴

```
sns.heatmap(df.corr( ), linewidths=0.1, vmax=0.5, cmap=plt.cm.gist_
heat, linecolor='white', annot=True)
```

TIP

vmax는 색상의 밝기를 조절하는 인자입니다. cmap은 미리 정해진 matplotlib 색상의 설정값을 불러옵니다. 색상 설정값은 https://matplotlib.org/users/colormaps.html에서 확인할 수 있습니다.

5 matplotlib를 이용해 그래프로 표현하기

- 이제 그래프를 다음과 같이 표시함

```
plt.show( )
```

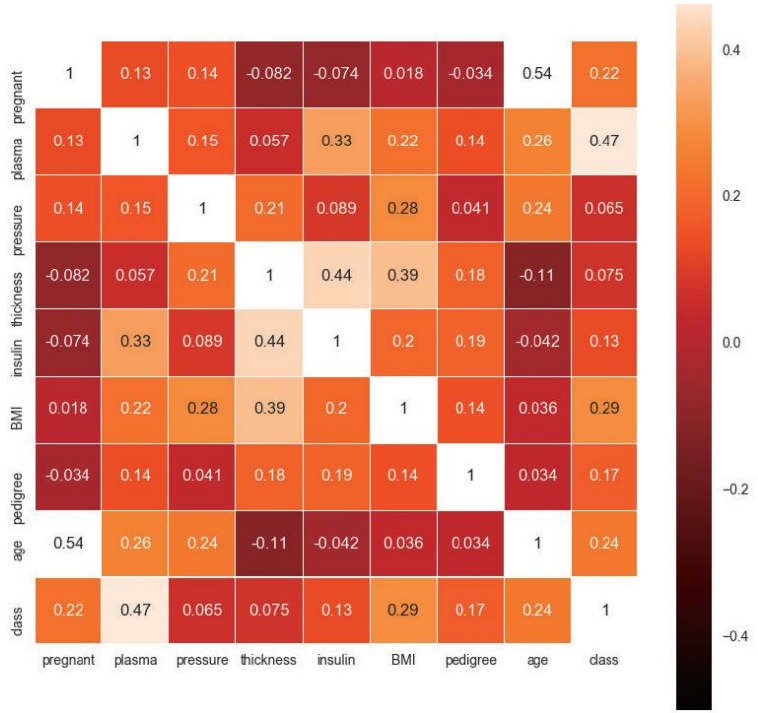

그림 11-2 정보 간 상관관계 그래프

5 matplotlib를 이용해 그래프로 표현하기

- 그래프를 통해 plasma 항목(공복 혈당 농도)이 class 항목과 가장 상관관계가 높다는 것을 알 수 있음
- 즉, 이 항목이 결론을 만드는 데 가장 중요한 역할을 한다는 것을 예측할 수 있음
- 이제 plasma와 class 항목만 따로 떼어 두 항목 간의 관계를 그래프로 다시 한번 확인함

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

5 matplotlib를 이용해 그래프로 표현하기

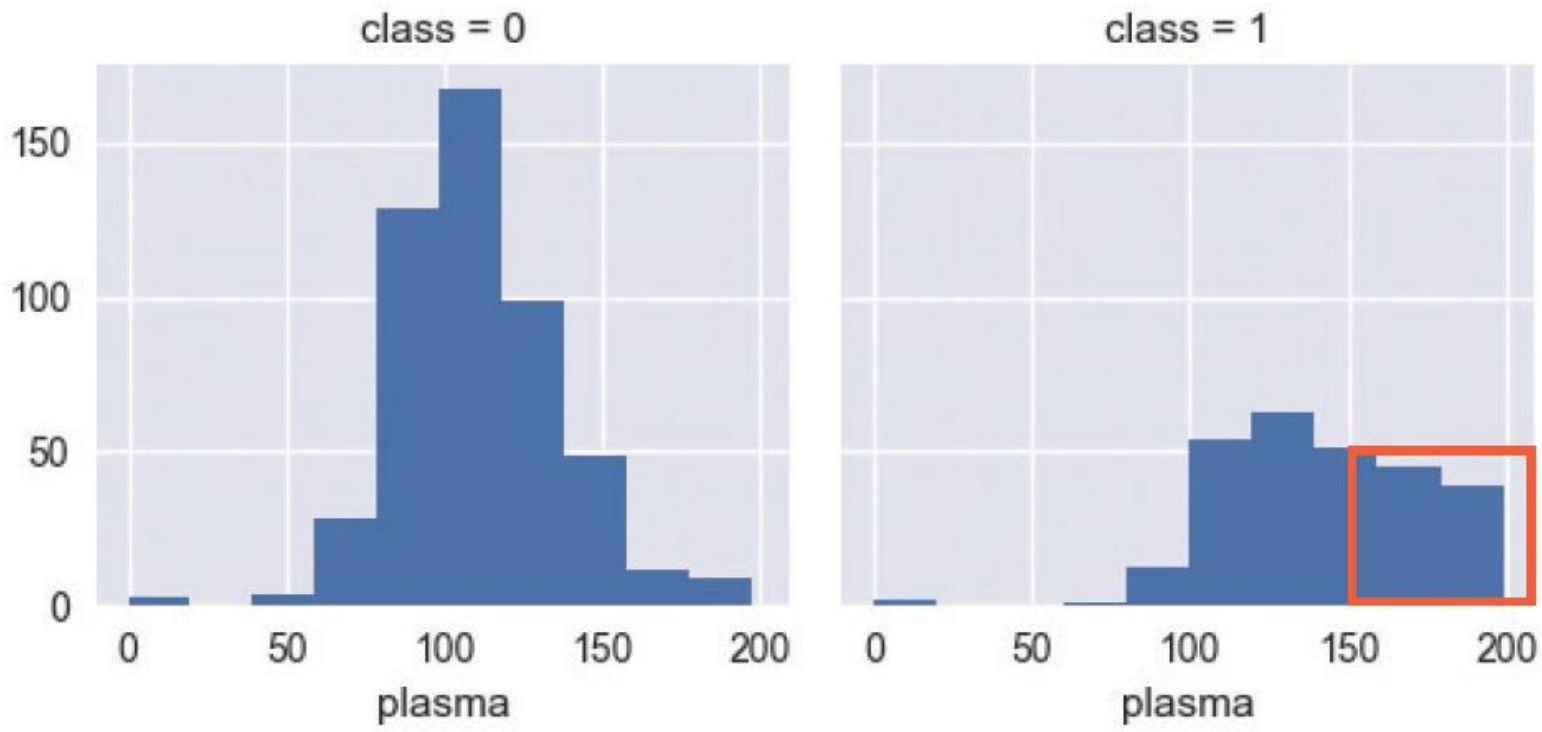

그림 11-3 plasma 정보와 class

5 matplotlib를 이용해 그래프로 표현하기

- 결과에 미치는 영향이 큰 항목을 발견하는 것이 '데이터 전처리 과정'의 한 예
- 데이터 전처리 과정은 딥러닝을 비롯하여 모든 머신러닝의 성능 향상에 중요한 역할을 함

TIP

데이터 전처리 과정은 모아진 데이터에 빠진 값이 있다면 평균이나 중앙값으로 대치하는 등의 과정, 전혀 관계없는 이상 데이터가 끼어 있지는 않은지를 점검하는 과정 등이 포함됩니다. SVM, RF등의 머신러닝 기법은 중요한 속성을 뽑아 내는 Feature extraction 과정도 데이터 전처리에 포함하지만, 딥러닝은 중요한 속성을 내부적으로 뽑아내므로 이 과정은 필요 없습니다.

6 피마 인디언의 당뇨병 예측 실행

- 이제 케라스를 이용해 당뇨병 예측을 실행하면 다음과 같음

```
# seed 값 생성
np.random.seed(seed)
tf.random.set_seed(seed)
```

6 피마 인디언의 당뇨병 예측 실행

- 우리가 random( ) 함수를 써서 임의의 숫자를 만들어 내는 것처럼 보여도 이는 컴퓨터 안에 미리 내장된 수많은 '랜덤 테이블' 중 하나를 불러내 그 표의 순서대로 숫자를 보여 주는 것
- seed 값을 설정한다는 것은 그 랜덤 테이블 중에서 몇 번째 테이블을 불러와 쓸지를 정하는 것
- seed 값이 같으면 똑같은 랜덤 값을 출력함

6 피마 인디언의 당뇨병 예측 실행

- 넘파이 라이브러리를 사용하면서 텐서플로 기반으로 딥러닝을 구현할 때는 일정한 결괏값을 얻기 위해 넘파이 seed 값과 텐서플로 seed 값을 모두 설정해야 함
- 단, seed 값을 이렇게 지정해도 여전히 출력 값이 미세하게 다를 수 있음
- 텐서플로를 구동시키는 cuDNN 등의 내부 소프트웨어가 자체적으로 또 다른 랜덤 테이블을 생성하기 때문인데, 현재 이 부분의 seed 값을 지정할 방법은 없음
- 최종 딥러닝 결과는 여러 번 실행하여 평균을 구하는 것이 가장 적절함

6 피마 인디언의 당뇨병 예측 실행

- 딥러닝 모델에 은닉층 추가할 때는 model.add() 함수를 이용해 새로운 줄을 추가하기만 하면 됨

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

6 피마 인디언의 당뇨병 예측 실행

- 모델의 컴파일 부분을 도식화하면 그림 11-4와 같음

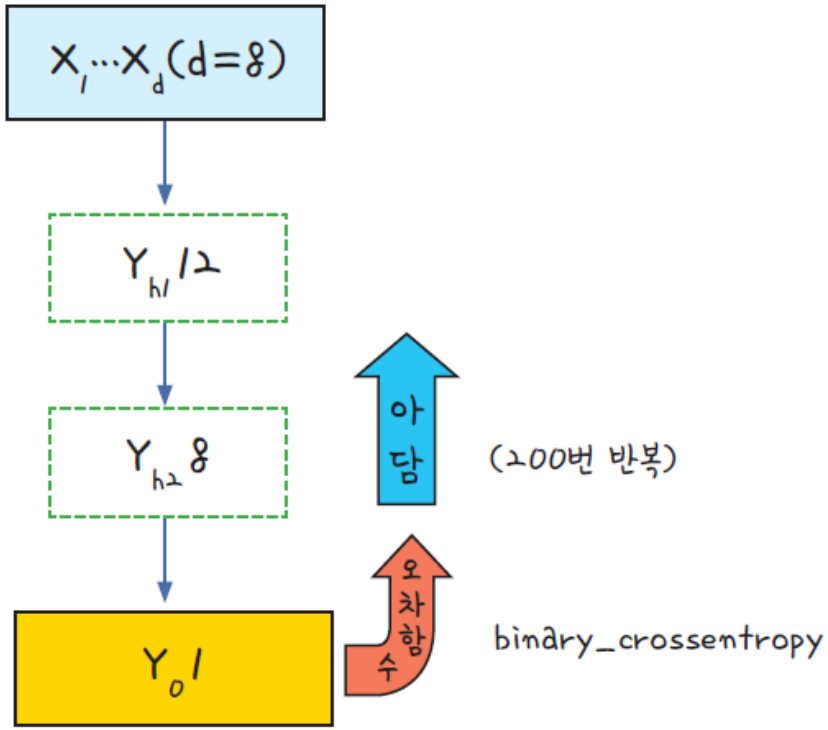

그림 11-4 피마 인디언 신경망 모델의 간단한 도식화

6 피마 인디언의 당뇨병 예측 실행

- 둘 중 하나를 결정하는 이항 분류(binary classification) 문제이므로 오차 함수는 binary_crossentropy를 사용하고, 최적화 함수로 adam을 사용함
- 전체 샘플이 200번 반복해서 입력될 때까지 실험을 반복함

코드 11-1 피마 인디언의 당뇨병 예측하기

- 예제 소스: run_project/02_Pima_Indian.ipynb

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import numpy
import tensorflow as tf
```

6 피마 인디언의 당뇨병 예측 실행

```
# seed 값 생성
np.random.seed(3)
tf.random.set_seed(3)

# 데이터 로드
dataset = numpy.loadtxt("../dataset/pima-indians-diabetes.csv",
delimiter=",")
X = dataset[:,0:8]
Y = dataset[:,8]
```

6 피마 인디언의 당뇨병 예측 실행

```
# 모델의 설정
model = Sequential( )
model.add(Dense(12, input_dim=8, 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'])
# 모델 실행
model.fit(X, Y, epochs=200, batch_size=10)
# 결과 출력
print("\n Accuracy: %.4f" % (model.evaluate(X, Y)[1]))
```

6 피마 인디언의 당뇨병 예측 실행

실행 결과

```
Train on 768 samples
Epoch 1/200
768/768 [==============================] - 1s 721us/sample - loss:
11.4155 - accuracy: 0.6198
Epoch 2/200
768/768 [==============================] - 0s 116us/sample - loss:
6.4242 - accuracy: 0.6159
Epoch 3/200
768/768 [==============================] - 0s 113us/sample - loss:
3.6949 - accuracy: 0.5221
Epoch 4/200
(중략)
```

6 피마 인디언의 당뇨병 예측 실행

```
Epoch 198/200
768/768 [==============================] - 0s 117us/sample - loss:
0.4944 - accuracy: 0.7240
Epoch 199/200
768/768 [==============================] - 0s 118us/sample - loss:
0.4935 - accuracy: 0.7214
Epoch 200/200
768/768 [==============================] - 0s 117us/sample - loss:
0.4979 - accuracy: 0.7357

 Accuracy: 0.7253
```