# 2부 : 답을 알려줘야 학습하는 지도 학습 알고리즘

#### 학습 목표

지도 학습과 관련된 8가지 알고리즘을 알아봅니다. 지도 학습은 학습 데이터에 답(종속변수)이 포함되어 있습니다. 그 답을 잘 예측할 수 있도록 모델을 훈련시키는 방법을 문제해결 관점에서 알아보겠습니다. 가장 기초 알고리즘인 선형 회귀부터 캐글 컴피티션 및 실무에서도 유용한 최신 기법인 XGBoost와 LightGBM까지 폭넓게 다룹니다.

# 4장 선형 회귀 분석 : 보험 데이터셋

#### 학습 목표

선형 회귀 모델을 파이썬으로 구현하여 결과물을 뽑아내고, 작동 원리를 이해합니다.

#### 학습 순서



#### 선형 회귀 소개

선형 회귀Linear Regression는 머신 러닝 모델 중 가장 기초적인 모델입니다. 여러 가지 데이터를 활용하여 연속형 변수인 목표 변수를 예측해 내는 것이 목적입니다. 예를 들어 몸무게, 나이, BMI, 성별 등을 데이터로 활용하여 키와 같은 연속형 변수를 예측하는 겁니다. 연속형 변수는 165.5cm, 172.3cm, 182.9cm와 같이 연속적으로 이어질 수 있는 변수를 의미합니다. 반면 남성/여성으로 구분되는 성별은 연속형 변수가 아닙니다. 선형 회귀 모델에서는 예측할 종속 변수만 연속형 변수면 족합니다. 예측하기 위해 사용되는 그외 변수들은 연속형일 필요는 없습니다(종속 변수가 아니면서, 예측에 사용되는 변수를 독립변수Independent Variable, 피처/속성Feature 등으로 부릅니다).

#### 선형 회귀 분석 예시 그래프



#### 장단점

| **장점** | **단점** |
| --- | --- |
| 모델이 간단하기 때문에 구현과 해석이 쉽습니다. | 최신 알고리즘에 비해 예측력이 떨어집니다. |
| 같은 이유로 모델링하는 데 오랜 시간이 걸리지 않습니다. | 독립변수와 예측변수의 선형 관계를 전제로 하기 때문에, 이러한 전제에서 벗어나는 데이터에서는 좋은 예측을 보여주기 어렵습니다(선형 관계를 전제로 한다는 부분은 이 장의 마지막까지 보면 이해할 수 있습니다). |

#### 유용한 곳

* 연속된 변수를 예측하는 데 사용됩니다. 예를 들어 BMI(체질량지수), 매출액, 전력 사용량과 같은 변수를 떠올리시면 됩니다.

#### TOP 10 선정 이유

* 복잡한 알고리즘에 비해서는 예측력이 떨어지지만 머신러닝 기초 알고리즘입니다. 데이터의 특성이 복잡하지 않을 때는 쉽고 빠른 예측이 가능하기 때문에 많이 사용됩니다. 다른 모델과의 성능을 비교하는 데 사용할 베이스라인으로 사용하기도 합니다.

## 4.1 문제 정의 : 한눈에 보는 분석 목표

<금토끼의 문제 정의> 금토끼는 새해를 맞이하여 올해 예산을 세우던 중 의료비 항목에서 고민이 생겼습니다. 보험 상품마다 본인부담금 비율이 달랐습니다. 올해 병원을 얼마나 갈지도 예측되지 않았습니다. 금토끼는 보험회사에서 개인에게 의료비를 청구한 데이터를 구해서 연령, 성별, BMI 등의 정보를 활용해 보험회사가 개인에게 청구하는 의료비를 예측해보기로 마음 먹었습니다.

| **난이도** | ⭐☆☆ | | |
| --- | --- | --- | --- |
| **알고리즘** | Linear Regression (선형 회귀) | | |
| **데이터셋 파일명** | insurance.csv | **종속 변수** | charges(청구비용) |
| **데이터셋 소개** | 보험과 관련된 데이터입니다. 보험사에서 청구하는 병원 비용이 종속 변수이며, 나이, 성별, BMI, 자녀 수, 흡연 여부를 독립변수로 사용합니다. | | |
| **문제 유형** | 회귀 | **평가지표** | RMSE(평균 제곱근 편차) |
| **사용한 모델** | LinearRegression | | |
| **사용 라이브러리** | * numpy (numpy==1.19.5) * pandas (pandas==1.3.5) * seaborn (seaborn==0.11.2) * matplotlib (matplotlib==3.2.2) * sklearn (scikit-learn==1.0.2) | | |
| **예제 코드 노트북** | 위치 : https://github.com/musthave-ML10/notebooks  파일 : 04\_Linear Regression.ipynb | | |

## 4.2 라이브러리 및 데이터 불러오기

파이썬에서 데이터를 다룰 때 가장 기본적으로 사용되는 라이브러리인 판다스pandas를 불러오겠습니다. 라이브러리를 불러오는 걸 프로그래밍 용어로 ‘임포트(한다)’라고 합니다.

| import pandas as pd # 판다스 라이브러리 임포트 |
| --- |

판다스를 불러왔으니 데이터를 불러오는 코드를 작성하겠습니다. 이번에 사용할 데이터는 insurance.csv 파일입니다. 해당 url을 사용하여 아래와 같이 불러오겠습니다.

| file\_url = 'https://media.githubusercontent.com/media/musthave-ML10/data\_source/main/insurance.csv'  data = pd.read\_csv(file\_url) # 데이터셋 읽기 |
| --- |

pd.read\_csv()를 사용하면 판다스 데이터프레임 형태로 데이터를 불러오게 됩니다.

## 4.3 데이터 확인하기

데이터를 불러왔으니, 불러온 데이터가 어떻게 생겼는지부터 다양한 방법으로 확인하겠습니다.

가장 직관적이고 단순한 방법은 저장한 데이터(data)를 그대로 입력해 출력하는 방식입니다.

| data # 전체 데이터 출력 |
| --- |

그럼 다음과 같은 결과물을 볼 수 있습니다.

|  | 연령 | 성별 | 체질량지수 | 자녀수 | 흡연여부 | 청구 비용 |
| --- | --- | --- | --- | --- | --- | --- |
|  | age | sex | bmi | children | smoker | charges |
| 0 | 19 | 0 | 27.900 | 0 | 1 | 16,884.92400 |
| 1 | 18 | 1 | 33.770 | 1 | 0 | 1,725.55230 |
| 2 | 28 | 1 | 33.000 | 3 | 0 | 4,449.46200 |
| 3 | 33 | 1 | 22.705 | 0 | 0 | 21,984.47061 |
| 4 | 32 | 1 | 28.880 | 0 | 0 | 3,866.85520 |
| ... | ... | ... | ... | ... | ... | ... |
| 1,333 | 50 | 1 | 30.970 | 3 | 0 | 10,600.54830 |
| 1,334 | 18 | 0 | 31.920 | 0 | 0 | 2,205.98080 |
| 1,335 | 18 | 0 | 36.850 | 0 | 0 | 1,629.83350 |
| 1,336 | 21 | 0 | 25.800 | 0 | 0 | 2,007.94500 |
| 1337 | 61 | 0 | 29.070 | 0 | 1 | 29,141.36030 |



엑셀에서 보는 테이블과 비슷합니다. 우선 테이블 하단 ❶ 1338 rows x 6 columns에서 볼 수 있듯이 data에는 총 1338줄, 즉 1338명에 대한 데이터가 있습니다. 각 데이터에는 ❷ 6개의 변수가 있습니다. ❸ 0부터 1337까지있는 왼쪽의 숫자를 인덱스라고 부르며, 기본값은 줄 번호입니다. 파이썬에서는 대부분 0부터 시작하는 것이 특징입니다. 1338줄 모두를 출력하지 않고 중간에 생략되었습니다.

이번에는 판다스에서 제공하는 여러 함수를 사용하여 데이터를 확인하겠습니다. 우선 head() 함수로 상위 5줄을 확인해보겠습니다.

| data.head() # 상위 5줄 출력 |
| --- |

|  | **age** | **sex** | **bmi** | **children** | **smoker** | **charges** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 19 | 0 | 27.9 | 0 | 1 | 16884.924 |
| **1** | 18 | 1 | 33.77 | 1 | 0 | 1725.5523 |
| **2** | 28 | 1 | 33 | 3 | 0 | 4449.462 |
| **3** | 33 | 1 | 22.705 | 0 | 0 | 21984.47061 |
| **4** | 32 | 1 | 28.88 | 0 | 0 | 3866.8552 |

data를 출력했을 때와 같은 형태이지만 5줄만 출력해 더 간결하게 볼 수 있다는 장점이 있습니다. sex와 smoker는 사실 숫자로 표현되는 연속형 변수가 아닌, 범주형 변수입니다. sex는 남성/여성으로 표시되어야 하고, smoker는 흡연자/비흡연자로 표시되어야 하는데, 컴퓨터로 학습하려면 (문자 형태가 아니라) 숫자 형태여야 해서 위와 같이 표기한 겁니다. sex에서 1이 남자, 0이 여자를 뜻합니다. smoker에서는 1이 흡연자, 0이 비흡연자입니다.

| **연속형 변수와 범주형 변수**  연속형 변수는 나이, 키와 같이 연속적으로 이어지는 변수입니다. 반면 범주형 변수는 이어지는 숫자가 아닌 각 카테고리로 구성된 변수입니다. 예를 들어 계절이나 성별 같은 것은 범주형 변수에 속합니다. 연속형 변수에서는 데이터 간의 크고 작음을 비교하거나 사칙연산 등을 할 수 있습니다. 예를 들어 키 180은 170보다 크다할 수 있고, 둘 사이의 평균도 구할 수 있습니다. 범주형 데이터에서는, 겨울이 여름보다 크거나 작다고 할 수 없으며, 평균이라는 개념 또한 존재할 수 없습니다. |
| --- |

이번에는 info() 함수로 데이터가 가지고 있는 변수를 확인해봅시다.

| data.info() # 컬럼 정보 출력 |
| --- |

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1338 entries, 0 to 1337

Data columns (total 6 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 age 1338 non-null int64

1 sex 1338 non-null int64

2 bmi 1338 non-null float64

3 children 1338 non-null int64

4 smoker 1338 non-null int64

5 charges 1338 non-null float64

dtypes: float64(2), int64(4)

memory usage: 62.8 KB



❶ Column에서는 data가 가진 변수 이름을 보여줍니다. 이미 앞에서 확인한 내용이라 특이사항은 없습니다.

❷ Non-Null Count에서는 결측치를 보여줍니다. Non-Null Count에서 Null은 결측치, 즉 비어 있는 값을 말합니다. non-null은 빈 값이 없다는 뜻입니다. 모든 변수들이 1338입니다. 따라서 모든 변수에 빈 값이 없다는 사실을 확인할 수 있습니다.

<용어/>

**Null**

값이 비어 있는 것을 뜻합니다. 널값, Null value, 결측치 등으로도 부르며, N/A, NA, NaN, 등 다양한 방식으로 표현됩니다. Null 값은 비어 있어서 알 수 없는 값이지, 0이 아니니 주의하시기 바랍니다.

</>

마지막 ❸ Dtype은 자료형입니다. 모든 변수가 숫자형 데이터이기 때문에 float과 int로만 구성되어 있습니다. 여기에서는 딱히 자료형에 대해서 고려해야 할 사항은 없으니 확인만 하고 넘어가도록 합시다.

마지막으로 describe()를 사용해 통계적인 정보를 살펴보겠습니다.

| data.describe() #통계 정보 출력 |
| --- |

|  | **age** | **sex** | **bmi** | **children** | **smoker** | **charges** |
| --- | --- | --- | --- | --- | --- | --- |
| **count** | 1338.000000 | 1338.000000 | 1338.000000 | 1338.000000 | 1338.000000 | 1338.000000 |
| **mean** | 39.207025 | 0.505232 | 30.663397 | 1.094918 | 0.204783 | 13270.422265 |
| **std** | 14.049960 | 0.500160 | 6.098187 | 1.205493 | 0.403694 | 12110.011237 |
| **min** | 18.000000 | 0.000000 | 15.960000 | 0.000000 | 0.000000 | 1121.873900 |
| **25%** | 27.000000 | 0.000000 | 26.296250 | 0.000000 | 0.000000 | 4740.287150 |
| **50%** | 39.000000 | 1.000000 | 30.400000 | 1.000000 | 0.000000 | 9382.033000 |
| **75%** | 51.000000 | 1.000000 | 34.693750 | 2.000000 | 0.000000 | 16639.912515 |
| **max** | 64.000000 | 1.000000 | 53.130000 | 5.000000 | 1.000000 | 63770.428010 |

불필요하게 긴 소수점 아래 숫자가 눈에 들어옵니다. 읽기에 편하지 않으니 파이썬에서 기본적으로 제공되는 round() 함수를 이용해 반올림해서 읽기 쉬운 형태로 다시 불러오겠습니다. 우리가 반올림할 데이터 테이블은 data.describe()이니, 다음과 같은 코드로 소수점 2자릿수까지 반올림하겠습니다.

| round(data.describe(), 2) # 소수점 2째자리까지만 표시해 통계 정보 출력 |
| --- |

파이썬에서는 이와 같이 여러 함수를 괄호 안에 계속 넣어서, 혹은 마침표로 계속 연결해가며 사용할 수 있습니다. 여러 개의 함수를 동시에 사용하는 예제는 앞으로도 꾸준히 보여드릴 예정입니다. 다음으로 describe() 함수의 결과물에 대한 얘기를 계속 하겠습니다.

|  | age | sex | bmi | children | smoker | charges |
| --- | --- | --- | --- | --- | --- | --- |
| count | 1,338.00 | 1,338.00 | 1,338.00 | 1,338.00 | 1,338.00 | 1,338.00 |
| mean | 39.21 | 0.51 | 30.66 | 1.09 | 0.20 | 13,270.42 |
| std | 14.05 | 0.50 | 6.10 | 1.21 | 0.40 | 12,110.01 |
| min | 18.00 | 0.00 | 15.96 | 0.00 | 0.00 | 1,121.87 |
| 25% | 27.00 | 0.00 | 26.30 | 0.00 | 0.00 | 4,740.29 |
| 50% | 39.0 | 1 | 30.4 | 1.0 | 0 | 9382.033 |
| 75% | 51.0 | 1 | 34.69375 | 2.0 | 0 | 16639.91252 |
| max | 64.0 | 1 | 53.13 | 5.0 | 1 | 63770.42801 |



각 변수에 대해 count부터 max까지 다양한 정보를 일목요연하게 확인할 수 있습니다. 우선 ❶ 개수(count)는 모든 변수가 1338로 같은데, 여기에서도 데이터에 결측치가 없음을 확인할 수 있습니다. 그 밑으로는 각각 ❷ 평균(mean), ❸ 표준편차(std), ➍ 최솟값(min), ➎ 사분위수 25% 50% 75%, 그리고 ➎ 최댓값(max)을 보여줍니다.

<용어/>

**사분위수(Quantile)**

데이터를 오름차순으로 정리했을 때 25%, 50%, 75% 위치에서 확인한 값입니다. 예를 들어 100개의 값들이 있다고 하면 가장 낮은 숫자부터 하나씩 세어 25번째 데이터, 50번째 데이터, 75번째 데이터가 각각 사분위수 25%, 50%, 75%에 해당합니다. 이는 Q1, Q2, Q3라고도 표현합니다.

</>

## 4.4 전처리 : 학습셋과 시험셋 나누기

일반적으로 데이터를 손보는 데에 상당한 시간을 들여야 하지만 첫 장이니만큼 이미 정리된 데이터를 사용하겠습니다(그래서 데이터를 정제하는 작업인 데이터 클리닝 및 피처 엔지니어링을 생략합니다). 모델링에 들어가기 앞서 데이터를 나누는 작업을 할 것인데, 우선 왜 굳이 데이터를 나누어야 하는지를 알아보겠습니다.

<용어/>

**데이터 클리닝 및 피처 엔지니어링**

데이터 클리닝은 지저분한 데이터를 정리하는 과정입니다. 이는 결측치를 처리하는 과정부터, 오탈자 수정, 불필요한 문자 제거 등을 포괄합니다. 피처 엔지니어링은 가지고 있는 독립변수들을 활용해서 더욱 풍성하고 유용한 독립변수들을 만들어내는 작업입니다.

</>

데이터를 나누는 작업은 크게 2가지 차원으로 진행됩니다. 첫째는 종속변수와 독립변수 분리입니다. 둘째는 학습용 데이터셋인 학습셋Train set과 평가용 시험셋Test set을 나누는 겁니다. 이렇게 2x2 조합으로 총 4개 데이터셋으로 나뉘게 됩니다.

|  | **독립변수** | **종속변수** |
| --- | --- | --- |
| **학습셋** | X\_train | y\_train |
| **시험셋** | X\_test | y\_test |

### 4.4.1 변수와 데이터셋을 나누는 이유

기본적으로 지도 학습에 속하는 머신러닝 모델은 독립변수를 통하여 종속변수를 예측하는 것이므로, 모델링할 때 어떤 변수가 종속변수인지 명확히 알려주어야 합니다. 이런 이유로 많은 머신러닝 알고리즘이 독립변수와 종속변수를 각각 별도의 데이터로 입력받습니다.



그렇다면 학습셋과 시험셋을 나누는 이유는 무얼까요? 예를 들어 학습셋과 시험셋을 구분하지 않고 예측 모델을 만든다고 가정해보겠습니다. 전체 데이터를 가지고 모델링(학습)을 하고, 또 다시 전체 데이터에 대해서 예측값을 만들어서 종속변수와 비교해 예측이 잘 되었는지 평가한다고 합시다. 그 평가 결과가 어느 정도 괜찮았다고 해서 새로운 데이터에 대해서도 좋은 예측력을 보일까요? 보험 청구비를 예측하는 모델을 만들고 새로운 고객 정보(독립변수들)를 받아서 예측을 했을 때, 실제 발생하는 보험 청구비를 제대로 잘 예측할 수 있을지는 장담할 수 없습니다.

왜 장담할 수 없냐하면 학습에 사용한 데이터와 평가에 사용한 데이터가 동일하다는 것은 모델을 만들고 나서 새로운 데이터에도 맞는지 검증하지 않은 거나 다름 없기 때문입니다.



이러한 (검증하지 않은 상태라는) 불확실성을 줄일 목적으로 준비하는 것이 시험셋입니다. 예를 들어 1338개 데이터 중 1000개는 학습셋으로 나누어서 모델을 학습시키는 데 사용하고, 나머지 338개 데이터는 모델 학습이 완료된 이후에 평가에 사용할 수 있습니다. 이렇게 하면 학습된 모델에 있어 시험셋의 338개 데이터는 처음 만나게 되는 데이터인 겁니다. 시험셋으로 예측/평가를 했을 때도 예측력이 좋게 나타난다면, 향후 예측하게 될 새로운 데이터에 대해서도 잘 작동할 거라는 기대를 가질 수 있습니다.



이와 같은 이유로 학습셋과 시험셋을 나눕니다. 일반적으로 학습셋:시험셋을 각각 7:3 혹은 8:2 정도 비율로 나눕니다. 비율은 데이터 크기에 따라 달라지기 마련인데, 전체 데이터 크기가 작을수록 시험셋 비율을 낮게 잡습니다. 두 데이터셋 중에 더 중요한 쪽을 따지자면 학습셋입니다. 실제 모델을 만드는 데 사용되는 데이터이기 때문에 충분한 양이 보장되지 않으면 모델 학습이 제대로 진행되지 않을 수 있습니다.

따라서 전체 데이터 크기가 작다면, 학습셋의 비율을 높여서 최대한 학습셋을 많이 확보해야 합니다. 때에 따라서는 9:1의 비율도 가능하며, 반대로 데이터가 아주 방대할 때는 6:4나 5:5로 나눌 수도 있습니다. 비율은 경험적으로 판단하는 영역이기 때문에 정해진 답을 말씀드릴 순 없습니다. 앞으로 이 책으로 공부하다 보면 적당한 비율에 대한 감을 잡을 수 있게 될 겁니다. 정답이 없기 때문에, 실제 분석을 하면서 비율을 수정해가며 다양한 시도를 해야 합니다.

### 4.4.2 데이터셋 나누기

이제 데이터셋을 나누어보겠습니다. 우선 종속변수와 독립변수를 나눈 후에 학습셋과 시험셋으로 나누겠습니다.

우선 독립변수를 X, 종속변수를 y로 나누겠습니다.

| X = data[['age', 'sex', 'bmi', 'children', 'smoker']] #독립변수 y = data['charges'] #종속변수 |
| --- |

통상적으로 이런 상황에서 X는 대문자, y는 소문자로 쓰는데, X는 변수가 여러 개 있는 데이터프레임DataFrame이기 때문에 대문자로, y는 변수가 하나인 시리즈Series이기 때문에 소문자로 씁니다. 관용적인 표현일 뿐, 다르게 쓴다고 해도 코드에서 문제는 없습니다.

그럼 이제 학습셋과 시험셋을 나누어줄 텐데 사이킷런sklearn에서는 관련 모듈(train\_test\_split)을 제공하므로 모듈을 불러옵시다.

| from sklearn.model\_selection import train\_test\_split #사이킷런 임포트 |
| --- |

데이터셋을 분할합시다.

| X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state=100) # 데이터셋 분할 |
| --- |

독립변수/종속변수, 그리고 학습셋/시험셋 조합으로 총 4개 데이터셋(X\_train, X\_test, y\_train, y\_test)이 나왔습니다.

= 왼쪽(좌항)에 보통은 변수가 하나이기 마련인데 여기에서는 4개나 들어 있습니다. 이런 형태 코드가 조금 낯설 수 있습니다. 이부분은 나중에 함수를 만드는 연습을 할 때 더 구체적으로 설명하겠지만, 여기서 간단히 말씀드리자면, train\_test\_split()이라는 함수가 4개 데이터셋을 결과물로 내보내므로, 이를 받아줄 변수 4개가 필요합니다.



= 오른쪽 코드(우항)를 살펴보겠습니다. 괄호 안에 독립변수(X)와 종속변수(y)를 순서대로 입력해야 합니다. 다음은 test\_size인데, 시험셋의 비율을 의미합니다. 여기에서는 0.2를 넣었으므로 20%, 즉 8:2로 데이터를 나누겠다는 의미입니다. 마지막으로 random\_state은 랜덤 샘플링과 관련이 있는데, 우선 랜덤 샘플링을 알아봅시다.

기본적으로 train\_test\_split() 함수는 랜덤 샘플링을 지원합니다. 랜덤 샘플링은 데이터를 특정 비율로 나눌 때 마구잡이로 뒤섞어서 나누는 겁니다. 만약 랜덤 샘플링을 사용하지 않으면 데이터를 있는 순서 그대로 분할합니다. 예를 들어 1338개 데이터의 앞에서부터 80%가 학습셋이 되고, 뒷부분인 나머지 20%가 시험셋이 됩니다. 이 데이터에서는 크게 문제가 없을 수 있습니다만, 종종 데이터가 특정 순서로 정렬된 경우도 있습니다. 예를 들어 데이터 앞쪽에는 여자, 뒷쪽에는 남자를 두는 식으로 정리를 해둘 수 있죠. 이런 식으로 특정 기준에 따라 정렬된 데이터를 순서대로 분류해버리면 학습셋과 시험셋의 특징이 확연히 다를 수밖에 없습니다. 샘플링된 데이터셋은 그 특성이 최대한 전체 데이터셋과 비슷하게 유지되어야 하기 때문에 위험한 방식입니다. 이와 같은 이유로 랜덤 샘플링을 일반적으로 널리 사용합니다. train\_test\_split() 함수는 기존 데이터의 순서와 상관없이 마구잡이로 섞어서 데이터를 분류시킵니다. 그렇기 때문에 매번 실행할 때마다 train\_set과 test\_set에 들어가는 데이터가 달라집니다.

학습셋과 시험셋 각각에 들어가는 데이터가 매번 달라져도 랜덤 샘플링만 잘됐다면 크게 상관없을 수도 있습니다. 하지만 우리가 분석을 하다 보면 결과의 일관성을 유지해야 할 필요가 있습니다. 내가 작성한 코드를 동료에게 공유했는데, 동료가 같은 코드를 돌리고 나와 다른 결과를 보게 된다면 뭔가 이상할 수밖에 없습니다. train\_test\_split() 함수는 랜덤하게 샘플링하면서도, 지속적으로 같은 데이터 분류를 지원합니다. 바로 random\_state 옵션입니다. 여기에는 그 어떤 임의의 숫자를 넣어도 상관없습니다. 같은 숫자라면 같은 형태로 분류된 데이터셋들을 얻게 됩니다. 여기에서는 100을 넣어서 나누었습니다. 여러분이 다른 숫자를 넣는다면, 결괏값이 책과 조금 다르게 나타날 겁니다. 똑같이 100을 입력하면 같은 결과를 볼 수 있습니다(간혹 패키지 버전 차이로 결과가 미세하게 다른 경우도 있습니다).

## 4.5 모델링

모델링은 머신러닝 알고리즘으로 모델을 학습시키는 과정이며, 그 결과물이 머신러닝 모델이 됩니다. 모델링에 사용할 머신러닝 알고리즘을 선택하고, 독립변수와 종속변수를 fit() 함수에 인수로 주어 학습합니다.



이번 데이터셋에는 선형 회귀 알고리즘을 사용합니다. 따라서 sklearn.linear\_model에서 선형 회귀 라이브러리를 불러옵니다.

<notice>

파이썬은 대소문자에 민감하므로 반드시 대소문자 구분해 사용해주세요.

</>

| from sklearn.linear\_model import LinearRegression |
| --- |

이제 모델을 만들어주어야 합니다. 여기서는 model이라는 이름의 객체에 선형 회귀의 속성을 부여하겠습니다.

| model = LinearRegression() |
| --- |

선형 회귀에 사용할 model 객체를 생성했으니 model 객체를 사용해서 선형 회귀로 학습하고 예측할 수 있게 됩니다(물론 객체 이름을 model 대신 다른 이름으로 지어도 됩니다).

모델을 학습시킬 객체가 준비가 되었으니 학습을 시키는 fit() 함수를 알아보겠습니다.

| model.fit(독립변수, 종속변수) |
| --- |

fit() 함수의 인수로 독립변수와 종속변수를 입력합니다. 이미 데이터를 나누었기 때문에 쉽게 독립변수와 종속변수를 나눠 입력할 수 있습니다. 학습 과정이므로 학습셋을 사용합니다.

| model.fit(X\_train, y\_train) |
| --- |

여기서 ‘학습시킨다’함은, 데이터를 모델 안에 넣어서 독립변수와 종속변수 간의 관계들을 분석해 새로운 데이터를 예측할 수 있는 상태로 만드는 겁니다. 이로써 model은 데이터를 통해 학습을 완료했고, 예측을 할 수 있게 되었습니다.

## 4.6 모델을 활용해 예측하기

이제 예측하는 실습을 하겠습니다. 원래는 예측 및 평가에서 학습셋과 시험셋을 각각 사용해 오버피팅 문제를 확인하는데, 이번 장에서는 간단하게 시험셋만 가지고 예측/평가를 하겠습니다.



<용어/>

**오버피팅(overfitting)**

모델이 학습셋에 지나치게 잘 맞도록 학습되어서 새로운 데이터에 대한 예측력이 떨어지는 현상을 의미합니다. 과적합, 과학습으로도 부릅니다. 5장에서 더 자세하게 다룹니다.

</>

| pred = model.predict(X\_test) |
| --- |

predict() 함수로 예측을 할 수 있으며, 괄호 안에는 예측 대상을 넣어주면 됩니다. 목표 변수가 예측 대상이라고 했죠? 목표 변수를 예측해야 하므로 여기에 들어가는 데이터에는 당연히 목표 변수가 포함되어서는 안 됩니다. 그러면 정답을 알려주는 꼴이 되기 때문입니다. 따라서 학습 때 사용했던 독립변수들을 가진 데이터를 넣어주어야 합니다. train\_test\_split() 함수를 사용하면 X\_train과 X\_test가 같은 변수를 가지기 때문에 이부분은 염려할 필요가 없습니다만, 향후 정말 새로운 데이터로 예측할 때는 주의해야 합니다. 독립변수 중 하나라도 빠진 나머지 데이터로 예측을 시도한다면 모델은 예측 과정에서 오류를 발생하게 됩니다.

## 4.7 예측 모델 평가하기

모델을 평가하는 방법으로 ‘테이블로 평가하기, 그래프로 평가하기, 통계(RMSE)적인 방법으로 평가하기’가 있습니다. 각 방법을 알아보겠습니다.

▽ 예측 모델을 평가하는 3가지 방법



#### 테이블로 평가하기

예측한 값은 pred에, 각각 관측치에 대한 실제 정보는 y\_test에 저장되어 있습니다. 예측값이 얼마나 정확한지는 pred와 y\_test를 비교하는 것으로 단순하게나마 확인할 수 있습니다. pred와 y\_test를 각각 별도로 출력해 확인할 수도 있겠지만, 보기 편하게 두 데이터를 합쳐서 테이블 하나로 만들겠습니다.

| comparison = pd.DataFrame({'actual': y\_test, 'pred': pred}) |
| --- |

판다스의 DataFrame() 함수로 테이블을 만들었습니다. 'actual'이라는 컬럼 이름으로 y값을 넣고, 'pred'라는 컬럼 이름으로 pred 데이터값을 넣는 겁니다. 그리고 이 테이블을 comparison에 저장했습니다. 그럼 comparison을 출력하겠습니다.

| comparison |
| --- |

|  | **actual** | **pred** |
| --- | --- | --- |
| **12** | 1826.84300 | 4765.249466 |
| **306** | 20177.67113 | 4957.730865 |
| **318** | 7421.19455 | 8298.988153 |
| **815** | 1877.92940 | 3078.811868 |
| **157** | 15518.18025 | 24165.956542 |
| **...** | ... | ... |
| **713** | 1984.45330 | 5776.764928 |
| **1282** | 14283.45940 | 23102.847340 |
| **531** | 14043.47670 | 14280.732585 |
| **537** | 8825.08600 | 10527.417291 |
| **1015** | 12124.99240 | 11638.260006 |

268 rows × 2 columns

첫 번째 관측치를 보면 실젯값이 1826이고 예측값은 4833 정도로 차이가 큽니다. 마지막 관측치는 실젯값 7153, 예측값 약 8259로 그나마 좀 비슷합니다. 사실 예측 결과를 이런 식으로 하나하나 확인하는 방식에는 한계가 있습니다. 수많은 데이터를 눈으로 다 볼 수는 없으니까요. 산점도Scatter plot 그래프를 이용해 한눈에 파악해보겠습니다.

#### 그래프로 평가하기

파이썬에서 그래프를 그리는 데 맷플롯립matplotlib과 시본seaborn 라이브러리를 가장 많이 사용합니다. 두 라이브러리를 임포트합시다.

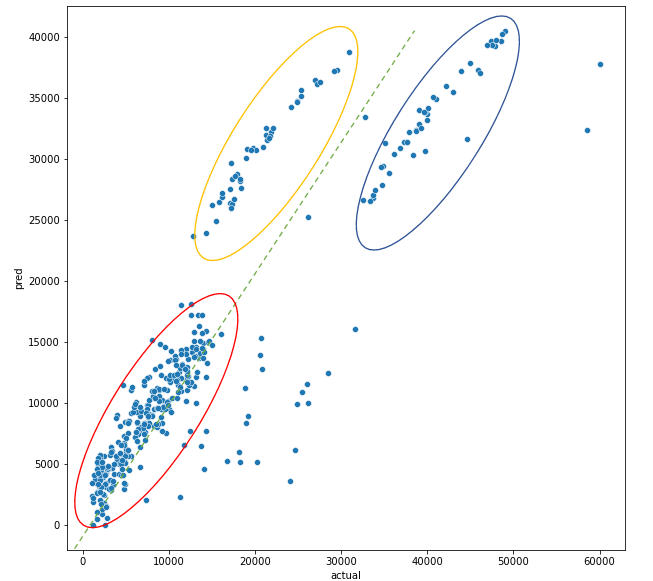
| import matplotlib.pyplot as plt # ❶ import seaborn as sns # ❷ |
| --- |

관행으로 ❶ matplotlib은 plt, ❷ seaborn은 sns로 줄여 사용합니다.

그리고 그래프를 만들어볼 텐데, 여기에서는 그래프의 크기를 정하는 코드와 scatter plot을 만드는 코드 두줄을 써보겠습니다.

| plt.figure(figsize=(10,10)) # ❶ 그래프 크기를 정의 sns.scatterplot(x = 'actual', y = 'pred', data = comparison) # ❷ |
| --- |

먼저 ❶ 그래프 크기를 정하고, ❷ scatterplot() 함수를 이용하여 산점도 그래프를 만들었습니다. 인수로 x축과 y축에 들어갈 데이터 컬럼을 지정합니다. x축에 실젯값인 actual, y축에는 예측값인 pred를 지정했습니다. 그리고 x, y축에 지정해준 컬럼이 속한 데이터를 마지막에 넣어주었습니다.



이해를 돕고자 그래프에 점선과 동그라미를 추가로 그렸습니다. 녹색 점선은 실젯값과 예측값이 정확히 같을 때, 즉 1:1로 매칭되었을 때를 의미합니다. 이 선에 가까울수록 더 잘 예측된 점이라고 해석할 수 있습니다. 이 그래프는 크게 3개의 영역으로 구분해 해석할 수 있습니다. 빨간 타원의 데이터는 녹색 점선에 가까우므로 실젯값과 예측값이 비슷한, 즉 비교적 예측이 잘된 경우입니다. 반면 노란 타원의 데이터는 전반적으로 실젯값보다 예측값이 더 높게 나타난 경우입니다. 반대로 파란 타원은 실젯값보다 예측값이 더 낮은 경우입니다.

그래프로 그렸더니 테이블로 일일이 확인할 때보다 훨씬 평가가 수월합니다. 그런데 그래프로 평가를 하는 방식은 어디까지나 직관적으로 예측력을 확인할 뿐이지, 객관적인 기준이 되지는 않습니다.

#### 통계적인 방법으로 평가하기 : RMSE

이번에는 더 통계적인 방법으로 접근하겠습니다. 연속형 변수를 예측하고 평가할 때 가장 흔하게 쓰이는 RMSERoot Mean Squared Error(루트 평균 제곱근 오차, 평균 제곱근 편차)를 사용해보겠습니다. RMSE를 아주 단순하게 말하면 실젯값과 예측값 사이의 오차를 각각 합산하는 개념입니다. 예를 들어 2개 데이터가 있고 예측값과 실젯값이 다음과 같다고 가정해봅시다.

|  | 예측값 | 실젯값 | 오차 |
| --- | --- | --- | --- |
| 0 | 10 | 12 | -2 |
| 1 | 20 | 18 | +2 |

각 차이가 -2와 +2로, 이 둘을 더해버리면 0이 됩니다. 차이가 0이라고 하면 완전히 정확하게 예측한 것처럼 보이지만 실제로는 그렇지 않죠? +- 부호 때문에 단순히 차이를 합산하면 이런 문제가 발생합니다. 부호 문제를 없애기 위해 절댓값을 쓰거나 제곱한 값을 사용할 수 있는데, 일반적으로 제곱한 값을 사용합니다. 이를 설명하기 위하여 다음과 같은 두 테이블을 예를 들어 살펴보겠습니다.

| Table A | 예측값 | 실젯값 | 오차 | 오차(절댓값) | 오차(제곱) |
| --- | --- | --- | --- | --- | --- |
| 0 | 10 | 5 | +5 | 5 | 25 |
| 1 | 15 | 5 | +10 | 10 | 100 |
| 2 | 10 | 5 | +5 | 5 | 25 |
| 3 | 15 | 5 | +10 | 10 | 100 |
| 합계 |  |  |  | 30 | 250 |

| Table B | 예측값 | 실젯값 | 오차 | 오차(절댓값) | 오차(제곱) |
| --- | --- | --- | --- | --- | --- |
| 0 | 5 | 5 | 0 | 0 | 0 |
| 1 | 20 | 5 | +15 | 15 | 225 |
| 2 | 5 | 5 | 0 | 0 | 0 |
| 3 | 20 | 5 | +15 | 15 | 225 |
| 합계 |  |  |  | 30 | 450 |

Table A는 오차가 각각 +5, +10, +5, +10으로 비교적 고르게 나왔습니다. 반면 Table B에서는 오차가 0인게 2건, 나머지 2건이 +15로 꽤 크게 났습니다. 더 오차의 분포가 크다고 할 수 있습니다. 이제 각 테이블의 오차 (절댓값)을 보면 두 경우 모두 30으로 같습니다. 즉, 오차에 대한 분포에 상관없이 합산된 값이 같습니다. 반면 오차(제곱)는 Table A에서는 250, B에서는 450으로, Table B에서 훨씬 더 큽니다. 이는 오차가 크게 난 경우(1행과 3행)에 제곱을 통해 훨씬 큰값이 나오게 만들기 때문입니다. 통상 오차가 더 큰 때에 더 큰 패널티를 주고자 제곱의 차이를 사용하곤 합니다. 또한 제곱을 사용하는 수식의 장점은 미분이 가능하다는 것입니다. 이 장에서는 왜 오차계산의 수식에 미분이 필요한지에 대해서 다루지 않지만, 10.6에서 해당 내용을 확인하실 수 있습니다.

이제 조금 전문적인 용어를 사용하겠습니다. 절댓값 차이를 이용하는 방법을 MAEMean Absolute Error(평균 절대 오차)라고 하며, 제곱 차이를 활용하는 방법은 MSEMean Squared Error(평균 제곱 오차)라고 부릅니다. 앞에 Mean이 붙은 이유는, 차이의 합을 총 개수로 나누어 평균을 내기 때문입니다. Table A를 예로 들어 설명하겠습니다. Table A에서 MAE는 오차(절댓값)의 총합 30을 4로 나눈 7.5가 됩니다. MSE는 오차(제곱)의 합 250을 4로 나누어 62.5가 됩니다.

앞서 MAE보다는 MSE를 더 일반적으로 사용한다고 말씀드렸는데, 여기서 딱 한걸음만 더 나아가보겠습니다. MSE의 단점은 제곱으로 인해 그 숫자의 규모가 실제 데이터의 스케일에 비해 너무 커진다는 겁니다. 즉, 두 테이블에서의 데이터의 차이는 0, 5, 10, 15 정도 수준인데 MSE는 평균을 낸 값임에도 불구하고 제곱한 값이므로 훨씬 더 큰 62.5라는 숫자를 보여줍니다. 반면 MAE는 7.5로 뭔가 더 합리적으로 보이는 크기의 숫자입니다(실제 데이터에 더 근사한 오차를 보여줍니다). 이 부분을 해소해주기 위해 MSE에 루트를 한 번 씌워줍니다. 그러면 본래 데이터와 스케일도 맞아 떨어집니다. 이렇게 MSE에 루트를 씌워준 값을 RMSERoot Mean Squared Error(루트 평균 제곱 오차)라고 부르며, 이 지표가 연속형 변수를 예측할 때 가장 일반적으로 쓰이는 평가지표입니다.

▽통계적 평가지표

| **평가지표** | **설명** |
| --- | --- |
| MAE | 평균 절대 오차. 실젯값과 예측값의 사이의 오차에 절댓값을 씌운 뒤 이에 대한 평균을 계산. 값이 작을수록 좋은 지표입니다(0에 가까울수록). |
| MSE | 평균 제곱 오차. 실젯값과 예측값의 사이의 오차를 제곱한 뒤 이에 대한 평균을 계산. 값이 작을수록 좋은 지표입니다(0에 가까울수록). |
| RMSE | 루트 평균 제곱 오차. MSE에 루트를 씌운 값으로 가장 일반적으로 사용됨. 값이 작을수록 좋은 지표입니다(0에 가까울수록). |
| R2 | 결정 계수. 독립변수가 종속변수를 얼마만큼 설명해 주는지를 가리키는 지표로, 즉 설명력을 나타냄. 값이 1에 가까울수록 좋은 지표입니다. |

그럼 이제 RMSE를 구해봅시다. 다행히 사이킷런sklearn 라이브러리가 MSE, MAE 등 여러 평가지표 함수를 제공합니다. 우선 MSE를 구하는 함수를 쓰고 거기에 루트를 씌워주는 방식으로 RMSE를 계산해보겠습니다.

| from sklearn.metrics import mean\_squared\_error # ❶ MSE 라이브러리 임포트 mean\_squared\_error(y\_test, pred) \*\* 0.5 # ❷ RMSE 계산 실행 |
| --- |

5684.927776334485

❶ MSE 라이브러리를 불러오는 코드입니다. ❷ MSE를 사용하는 코드인데, 괄호 안에 실젯값 데이터, 예측값 데이터를 순서대로 넣어주면 됩니다. **\*\* 0.5**는 루트입니다. 파이썬에서는 \*\*가 제곱이므로 0.5를 제곱하면 루트를 씌운 값을 계산합니다.

또 다른 방법으로는 MSE 함수 안에서 squared 매개변수를 False로 설정하면 RMSE를 구할 수 있습니다.

| mean\_squared\_error(y\_test, pred, squared = False) |
| --- |

5684.927776334485

약 5684라는 RMSE를 얻을 겁니다. RMSE는 근본적으로 에러에 대한 합을 계산한 것이기 때문에, 작을수록 예측력이 좋다고 할 수 있습니다. 그럼 5684는 작은편에 속할까요, 큰편에 속할까요? 안타깝게도 RMSE를 평가하는 데에 절대적인 기준은 없습니다. 이는 데이터의 특성에 따라 천차만별로 달라질 수 있기 때문에, 어느 수준 이하면 좋은 예측을 보인다는 등의 말을 하기가 어렵습니다. 그래서 RMSE는 절대 평가보다는 상대 평가에 사용합니다. 앞으로 다양한 알고리즘과 활용법을 배울 텐데, 같은 데이터에 여러 가지 모델링을 해보고, 그중 어떤 모델이 가장 뛰어난 예측력을 보이는지를 판단할 때 RMSE가 가장 낮은 모델을 선택하면 됩니다.

마지막으로 R2라는 평가 지표를 알아보겠습니다. R2는 독립변수로 설명되는 종속변수의 분산 비율을 나타내는 통계적 측정값입니다. 무슨 말인지 쉽게 와닿지 않죠? 그림을 보면서 다시 설명해보겠습니다.



이 그래프에서 ❶번 점선은 종속변수의 평균값으로 모델의 성능을 평가하는 비교 대상, 즉 일종의 기준선 역할입니다. ❶번 점선으로부터 관측치까지 차이를 SSTSum of Squares Total이라고 부릅니다. 그리고 ❷번 실선은 우리가 만든 예측 모델입니다. 특정 데이터의 x값을 기준으로 ❶번 점선부터 ❷번 실선까지의 거리가 SSRSum of Squares Regression입니다. 즉, 평균으로 대충 때려 맞추었을 때와, 우리가 만든 모델을 이용했을 때의 차이죠. 마지막으로 ❷번 실선과 실제 데이터까지의 거리는 SSESum of Squares Error입니다. 우리가 만든 모델이 예측해내지 못한 에러를 나타냅니다. R2는 SST에서 SSR가 차지하는 비율을 나타냅니다.

2

즉, 대충 평균값으로 넣었을 때, 예측값(평균값)과 실젯값의 차이 중 우리 모델이 얼마만큼의 비율로 실젯값에 가깝게 예측하는지를 의미합니다. 파이썬에서는 아래와 같은 간단한 코드로 R2를 계산할 수 있습니다.

| model.score(X\_train, y\_train) |
| --- |

0.7368220127747351

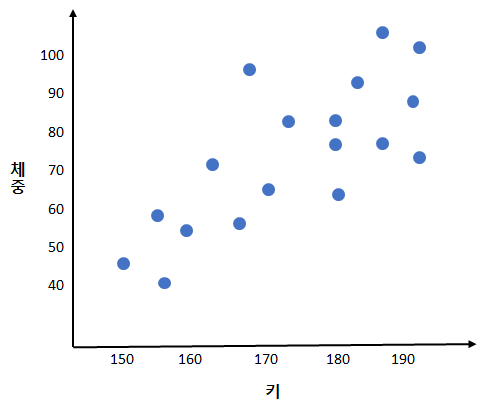
약 0.74의 값이 나왔습니다. R2는 비율이므로 최대 1까지 나올 수 있으며, 좋은 모델일수록 1에 가깝고 0.7~0.8 이상이면 일반적으로 괜찮은 수치라고 볼 수 있습니다. 우리가 만든 모델은 0.74가 나왔으므로 괜찮은 수준이라고 할 수 있겠네요.

지금까지 크게 세 가지 방법으로 모델을 평가해보았습니다. 사실 가장 먼저 살펴본 테이블을 출력한 결괏값 비교는 데이터를 하나하나 확인해야 해서 거의 사용하지 않습니다만 예측 결과가 실제와 얼마나 다른지 직접 보여드리려는 의도로 소개했습니다. 실제로 사용되는 평가 방법은 RMSE와 R2입니다. 산점도 그래프 같은 그래프를 사용하면 RMSE에 대하여 익숙지 않는 사람에게 설명하기가 편합니다.

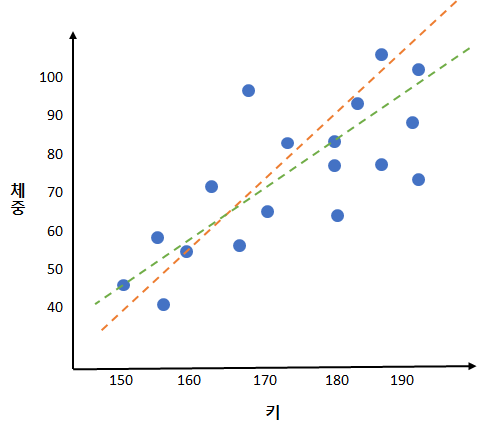
## 4.8 이해하기 : 선형 회귀(Linear Regression)

선형 회귀는 독립변수와 종속변수 간에 선형 관계가 있음을 가정하여 최적의 선을 그리고 예측하는 방법입니다. 흔히 선형 관계가 있을 것이라 예측할 만한 예로 키와 체중을 들 수 있습니다. 같은 키라도 사람마다 체중은 천차만별이겠지만, 평균적으로보면 키가 크면 큰만큼 평균 체중 또한 더 많이 나갈 겁니다.

아래 그래프와 같은 데이터가 있다고 가정해봅시다.



키와 체중이 완벽하게 정비례하지는 않지만, 얼추 선형 관계를 보이는 듯한 분포입니다. 선형 회귀는 여기에 최적의 선을 찾아 그어서 예측하는 겁니다. 그런데 다음 중 어떤 선이 더 최적의 선일까요?

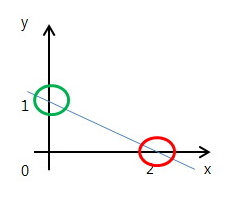


이 부분은 사람의 눈으로 알기가 어렵습니다만, 머신러닝에서는 손실 함수Loss Function를 최소화하는 선을 찾아서 모델을 만들어냅니다. 여기서 손실 함수란 예측값과 실젯값의 차이, 즉 오차를 평가하는 방법을 말합니다. 위 그래프에서는 선과 각 점 간의 거리가 오차가 되고, 우리가 앞서 배웠던 MSE나 RMSE 등이 손실 함수가 됩니다. 예측한 선의 기울기나 y절편에 따라서 실젯값과 예측값의 차이가 달라집니다. 머신 러닝은 이 손실 함수를 최소화하는 방향으로 최적의 선을 단시간에 찾아냅니다.

<용어/>

**절편과 기울기**

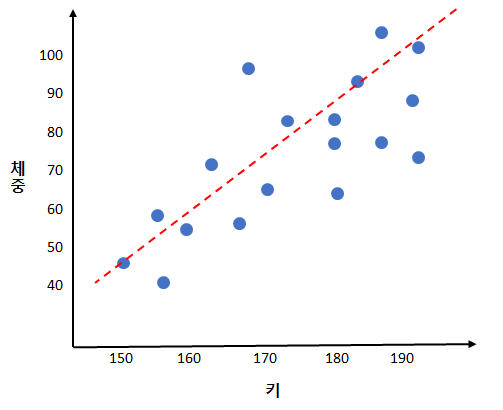
절편은 x축 또는 y축과 만나는 점의 좌표입니다. 다음 그림에서 y절편은 1, x절편은 2입니다.



기울기는 y의 증가량을 x의 증가량으로 나눈 겁니다. 앞의 그래프에서 x가 2 증가할 때, y는 -1 증가했으므로 기울기는 입니다.

</>

예를 들어 키를 독립변수로 두고, 체중을 종속변수로 하는 선형 회귀를 만들고 다음과 같은 그림의 결과를 얻었다고 가정하겠습니다.



여기서 빨간 점선이 우리의 예측 모델이 되는 겁니다. 이 모델을 가지고 새로운 데이터를 예측을 한다면 다음과 같이 설명할 수 있습니다. 예를 들어 키가 170인 새로운 사람에 대한 체중을 예측한다면 노란 점선에서 보이는 것처럼 해당 키에 해당하는 체중값으로 약 72kg 정도의 예측값을 보여줄 겁니다.



선형 회귀는 상대적으로 단순한 알고리즘이기 때문에 수식으로 표현하기도 쉽습니다. 기초교육과정에서 배운 1차 함수로 표현할 수 있습니다.

<수식/>



</>

여기서 x는 키, y는 체중이 되며 a는 예측 모델(점선)의 기울기, b는 점선의 y절편입니다. 따라서 x에 170이라는 숫자를 넣으면 약 72 정도의 y값이 나오는 1차 함수인 셈입니다.

이 예시는 독립변수가 ‘키’ 단 한 개뿐이라서 그리기도 쉽고 이해하도 쉽습니다. 우리는 코딩 파트에서 독립변수가 ‘age’, ‘sex’, ‘bmi’, ‘children’, ‘smoker’로 5개인 모델을 만들었습니다. 독립변수가 많아지게 되면 현실적으로 그림으로 표현해 보여주기가 어렵습니다. 하지만 적어도 수식으로는 표현할 수 있으며, 다음과 같은 형태의 수식을 생각해볼 수 있습니다.

<수식>

</>

여기에서는 y절편을 b 대신 iintercept로 표시해주었고, A, B, C, D, E는 각 변수에 대한 기울기입니다. 이 기울기 값을 계수coefficient라고도 합니다. 사이킷런의 선형 회귀 모델은 예측뿐만 아니라, 학습을 통해 생성된 계수 또한 제공해줍니다.

다음 코드를 실행하면 다음과 같이 5개의 독립변수에 대한 계수를 볼 수 있습니다.

| model.coef\_ |
| --- |

array([2.64799803e+02, 1.73446608e+01, 2.97514806e+02, 4.69339602e+02,

2.34692802e+04])

넘파이 형태로 출력이 되기 때문에 변수의 이름이 따로 나타나지 않는데, 기존 데이터의 독립변수와 같은 순서로 배열되어 있습니다. 더 보기 편하도록 변수 이름을 포함하는 판다스 형태로 변경하겠습니다. 한 줄짜리 데이터이므로 시리즈 형태로 바꿔보겠습니다.

| pd.Series(model.coef\_, index = X.columns) |
| --- |

age 264.799803

sex 17.344661

bmi 297.514806

children 469.339602

smoker 23469.280173

dtype: float64

변수 이름이 함께 나오니 훨씬 해석이 용이해졌습니다. 몇 가지 변수를 해석하자면, age가 1만큼 증가하면 charges는 약 265만큼 증가합니다. sex는 0과 1로만 구성된 데이터이기 때문에 1만큼 증가할 때 charges가 17만큼 증가한다기보다는, 남자(1)의 경우 여자(0)보다 charges가 보통 17정도 높다고 해석하는 게 좋습니다. smoker도 sex와 마찬가지로 해석할 수 있습니다.

학습된 모델은 계수뿐만 아니라 y절편 또한 제공해줍니다.

| model.intercept\_ |
| --- |

-11576.999976112367

그럼 위의 정보를 이용해 앞서 언급했던 수식을 제대로 완성하겠습니다.

charges = 264.799803 \* age + 17.344661 \* sex + 297.514806 \* bmi + 469.339602 \* children + 23469.280173 \* smoker - 11576.999976112367

데이터의 특정 행을 정해서 각 변수의 값을 위 수식에 넣으면 모델이 보여주는 예측값과 같은 결과를 얻으실 수 있습니다. 이처럼 선형 회귀는 수식을 도출하기고 매우 쉽기 때문에 그 해석도 매우 직관적이라는 장점이 있습니다.

## 학습 마무리

#### 되짚어보기

4.1 성별, 나이 등의 정보를 활용하여 보험 청구비용을 예측해봅니다.

4.2 판다스와 프로젝트에 쓸 예제 데이터셋을 불러옵니다.

4.3 데이터를 확인하는 다양한 방법을 확인해보았습니다. 결측치 등 특이사항은 없었습니다.

4.4 지도 학습이므로 평가에 사용할 데이터를 학습셋과 시험셋으로 나누었습니다.

4.5 선형 회귀 모델을 사용하여 예측모델을 만들었습니다.

4.6 시험셋에 대한 예측값을 얻었습니다.

4.7 산점도와 테이블을 사용해 직관적으로 결과를 살펴보고, RMSE와 R2값도 알아보았습니다. R2는 약 0.74로 괜찮은 모델이라고 볼 수 있습니다.

#### 



#### 유의할 점

모델의 계수를 해석할 때 부호의 영향에 유의해야 합니다. 부호와 상관없이 계수의 절댓값이 클수록 영향이 크다고 할 수 있고, 절댓값이 0에 가까울수록 영향력이 거의 없는 겁니다. 다만, 여러 계수를 서로 비교할 때 단순히 절댓값이 더 크면 영향력이 더 크다고 보기에는 무리가 있습니다. 이유는 각 변수의 스케일이 다르기 때문입니다. 예를 들어 성별은 0과 1로만 되어 있는 반면 나이는 20부터 60 등 십의 자리 숫자를 가지고 있습니다. 즉, 성별이 1 커질 때와 나이가 1커질 때가 가지는 영향력이 다르다는 겁니다. 이 부분을 명료하게 비교하려면 스케일링 작업이 필요하며, 이는 6장 ‘K-최근접 이웃(KNN)’에서 다룹니다.

#### 관련 모델 안내

1. **릿지 회귀**Ridge Regression패키지: from sklearn.linear\_model import Ridge  
   선형 회귀모델에 L2 정규화[[1]](#footnote-0)를 적용한 모델로 오버피팅을 억제하는 효과가 있습니다.
2. **라쏘 회귀**Lasso Regression패키지: from sklearn.linear\_model import Lasso  
   선형 회귀모델에 L2 정규화[[2]](#footnote-1)를 적용한 모델로 피처 셀렉션[[3]](#footnote-2) 및 오버피팅[[4]](#footnote-3)을 억제하는 효과가 있습니다.
3. **엘라스틱 넷**Elastic Net  
   패키지: from sklearn.linear\_model import ElasticNet  
   릿지 회귀와 라쏘 회귀의 단점을 절충시킨 모델입니다.

#### 핵심 용어

1. **선형 회귀** : 독립변수와 종속변수 간의 선형 관계를 전제로 한 모델입니다. 구현 및 이해가 용이한 장점이 있습니다.
2. **Null** : 값이 비어 있는 것을 뜻합니다. 널값, Null value, 결측치 등으로도 부르며, N/A, NA, NaN, 등 다양한 방식으로 표현됩니다. Null 값은 비어 있어서 알 수 없는 값이지, 0이 아닌 점을 주의하시기 바랍니다.
3. **사분위수**Quantile : 사분위수는 데이터를 오름차순으로 정리했을 때 25%, 50%, 75% 위치에서 확인한 값입니다. 예를 들어 100개의 값들이 있다고 하면 가장 낮은 숫자부터 하나씩 세어 25번째 데이터, 50번째 데이터, 75번째 데이터가 각각 사분위수 25%, 50%, 75%에 해당합니다. 이는 Q1, Q2, Q3라고도 표현합니다.
4. **오버피팅** : 모델이 학습셋에 지나치게 잘 맞도록 학습되어서 새로운 데이터에 대한 예측력이 떨어지는 현상을 의미합니다. 8장에서 더 자세하게 다룰 예정입니다.

#### 새로운 함수와 라이브러리

1. **round()** : 반올림
2. **pandas.DataFrame()** : 데이터를 DataFrame 형태로 변환
3. **pandas.DataFrame.head()** : 데이터의 초반부 호출(기본값 5줄)
4. **pandas.DataFrame.info()** :변수에 대한 결측치, 데이터 타입 정보 확인
5. **pandas.DataFrame.describe()** : 데이터프레임의 통계적 요약 확인
6. **sklearn.model\_selection.train\_test\_split()** : 훈련셋과 시험셋 분류
7. **sklearn.metrics.mean\_squared\_error()** : 평균 제곱 오차 계산
8. **모델.fit()** : 모델 학습
9. **모델.predict()** : 학습된 모델로 예측
10. **matplotlib.pyplot.figure()** : 새로운 도표를 만들거나 기존 도표를 활성화
11. **seaborn.scatterplot()** : 산점도 그래프 생성

## 연습문제

1. 다음 중 데이터의 통계 정보를 보여주는 함수는 무엇인가요?

① head()

② info()

③ describe()

④ tail()

2. 훈련셋과 시험셋을 나누는 train\_test\_split()에 대한 설명 중 틀린 것은?

① test\_size로 훈련셋과 시험셋의 비율을 정의할 수 있습니다.

② 시험셋은 모델 학습에 사용하는 데이터입니다.

③ 랜덤 샘플링으로 훈련셋과 시험셋을 나눕니다.

④ random\_state을 사용하면 매번 같은 모습의 훈련셋/시험셋을 얻을 수 있습니다.

3.다음 중 연속형 변수의 예측 결과를 평가하는 방법이 아닌것은?

① AUC

② MAE

③ MSE

④ RMSE

4. 선형 회귀 모델의 계수(coef)에 대한 설명으로 옳지 않은 것은?

① 회귀모델의 수식을 그래프로 그렸을 때, 기울기값에 해당합니다.

② 각 변수의 영향도를 대변해주는 값입니다.

③ 계수가 3인 경우는 -9인 경우보다 영향도가 크다고 할 수 있습니다.

④ 계수를 통한 변수 영향도 비교시에는, 변수의 스케일도 고려해야합니다.

#### 정답 및 해설

1. 3

① head() ← 상위 5행 출력

② info() ← 변수 정보 출력

③ describe() ← 통계 정보 출력

④ tail() ← 하위 5행 출력

2. 2

시험셋은 모델 평가에 사용하는 데이터입니다.

3. 1

① AUC ← Area Under the Curve. 이진분류의 예측 결과를 평가할 때 사용되며 12장에서 다룹니다.

② MAE ← Mean Absolute Error. 연속형 변수의 예측 결과를 평가할 때 사용됩니다.

③ MSE ← Mean Squared Error. 연속형 변수의 예측 결과를 평가할 때 사용됩니다.

④ RMSE ← Root Mean Squared Error. 연속형 변수의 예측 결과를 평가할 때 사용됩니다.

4. 3

③ 계수가 3인 경우는 -9인 경우보다 영향도가 크다고 할 수 있습니다. ← 계수를 가지고 영향도를 평가할 때는 절대값을 기준으로 평가하기 때문에, 3은 -9보다 작은 영향도를 의미합니다.

1. 11.6장 참조 [↑](#footnote-ref-0)
2. 11.6장 참조 [↑](#footnote-ref-1)
3. 11.6장 참조 [↑](#footnote-ref-2)
4. 8.7장 참조 [↑](#footnote-ref-3)