# 5장 로지스틱 회귀 분석 : 타이타닉 데이터셋

#### 학습 목표

파이썬을 통해 직접 로지스틱 회귀 분석 모델을 구현하여 결과물을 뽑아내고, 작동 원리를 이해합니다. 또한 분류Classification 문제를 예측하고 평가합니다.

#### 학습 순서



#### 로지스틱 회귀 분석 소개

로지스틱 회귀 분석Logistic Regression 또한 선형 회귀 분석처럼 기본 분석 모델입니다. 로지스틱 회귀 분석은 알고리즘의 근간을 선형 회귀 분석에 두고 있어서 선형 회귀 분석과 상당히 유사하지만 다루는 문제가 다릅니다. 선형 회귀 분석은 연속된 변수를 예측하는 반면, 로지스틱 회귀 분석은 Yes/No처럼 두 가지로 나뉘는 분류 문제를 다룹니다.



#### 장단점

| **장점** | **단점** |
| --- | --- |
| * 선형 회귀 분석만큼 구현하기 용이합니다. * 계수Coefficient(기울기)를 사용해 각 변수의 중요성을 쉽게 파악할 수 있습니다. | * 선형 회귀 분석을 근간으로 하고 있기 때문에, 선형 관계가 아닌 데이터에서는 예측력이 떨어집니다. |

#### 유용한 곳

* Yes/No, True/False와 같은 두 가지 범주로 나뉜 값을 예측하는 데 사용합니다.
* 분류 문제에 있어서 기준선Baseline으로 자주 활용됩니다(타 모델과 비교할 목적).

#### TOP 10 선정 이유

* 선형 회귀 분석과 마찬가지로, 너무나도 기본 알고리즘이라서 꼭 알고 있어야 합니다. 실제 이진분류가 필요한 상황이 많기 때문에 두 가지 범주를 구분하는 간단한 예측에 유용하며 딥러닝에서도 기본 지식입니다.

## 5.1 문제 정의 : 한눈에 보는 분석 목표

<금토끼의 문제 정의> 금토끼는 넷플릭스에서 〈타이타닉〉 영화를 보았습니다. 레오나르도 디카프리오가 바닷속으로 점점 깊게 빠져드는 장면에서 그만 눈물이 찔끔 나버렸지 뭡니까? 다행히 옆에 있던 꽃토끼에게 눈물을 들키지 않은 것 같습니다. 영화는 끝났지만 여운이 가시지 않았습니다. 디카프리오가 살 방법이 없었을까 하는 생각이 머리에서 떠나지 않았습니다. 여자와 아이들을 보트에 먼저 태우던데 성별과 나이 같은 정보가 실제로 생존에 어떤 영향을 미치는지 확인해보고 싶은 생각이 들었습니다. 타이타닉 데이터셋을 입수한 금토끼는 곧바로 분석에 들어갔습니다.

| **난이도** | ⭐☆☆ | | |
| --- | --- | --- | --- |
| **알고리즘** | Logistic Regression(로지스틱 회귀) | | |
| **데이터셋 파일명** | titanic.csv | **종속 변수** | Survived(생존 여부) |
| **데이터셋 소개** | 타이타닉에 승객의 정보를 담은 데이터셋입니다. 각 승객 정보(이름, 성별, 나이, 티켓 번호 등)를 활용하여 생존 여부를 예측합니다. | | |
| **문제 유형** | 분류 | **평가지표** | 정확도 |
| **사용한 모델** | LogisticRegression | | |
| **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 : 05\_Logistic Regression.ipynb | | |

## 5.2 라이브러리 및 데이터 불러오기

판다스를 활용하여 ‘titanic.csv’를 data라는 이름으로 불러옵니다.

| import pandas as pd # 판다스 라이브러리 임포트 file\_url = 'https://media.githubusercontent.com/media/musthave-ML10/data\_source/main/titanic.csv' data = pd.read\_csv(file\_url) # 데이터셋 읽기 |
| --- |

## 5.3 데이터 확인하기

head() 함수로 상위 5행 출력해 데이터를 살펴봅시다.

| data.head() # 상위 5행 출력 |
| --- |

| 티켓 글래스 | 승객이름 | 성별 | 나이 | 함께 탑승한 형제 및 배우자의 수 | 함께 탑승한 부모 및 자녀의 수 | 티켓 번호 | 승선한 항구 | 생존 유무 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |

|  | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Embarked | Survived |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 3.00 | Braund, Mr. Owen Harris | male | 22.00 | 1.00 | 0.00 | A/5 21171 | S | 0.00 |
| 1 | 1.00 | Cumings, Mrs. John Bradley (Florence Briggs Th... | female | 38.00 | 1.00 | 0.00 | PC 17599 | C | 1.00 |
| 2 | 3.00 | Heikkinen, Miss. Laina | female | 26.00 | 0.00 | 0.00 | STON/O2. 3101282 | S | 1.00 |
| 3 | 1.00 | Futrelle, Mrs. Jacques Heath (Lily May Peel) | female | 35.00 | 1.00 | 0.00 | 113,803.00 | S | 1.00 |
| 4 | 3.00 | Allen, Mr. William Henry | male | 35.00 | 0.00 | 0.00 | 373,450.00 | S | 0.00 |



❶ 독립변수 8개와 ❷ 종속변수(Survived) 1개를 볼 수 있습니다. 변수 이름에 대한 설명은 다음과 같습니다.

* Pclass : 비행기처럼, 일종의 티켓 클래스입니다.
* Name : 승객 이름
* Sex : 성별
* Age : 나이
* SibSp : 함께 탑승한 형제 및 배우자의 수
* Parch : 함께 탑승한 부모 및 자녀의 수
* Ticket : 티켓 번호
* Embarked : 승선한 항구(C = Cherbourg, Q = Queenstown, S = Southampton)
* Survived: 생존 유무 (1 = 생존, 0 = 사망)

이번에는 info() 함수를 호출해 각 변수의 특징을 살펴봅시다.

| data.info() # 변수 특징 출력 |
| --- |

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

RangeIndex: 889 entries, 0 to 888

Data columns (total 9 columns):

# Column Non-Null Count Dtype

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

0 Pclass 889 non-null int64

1 Name 889 non-null object

2 Sex 889 non-null object

3 Age 889 non-null float64

4 SibSp 889 non-null int64

5 Parch 889 non-null int64

6 Ticket 889 non-null object

7 Embarked 889 non-null object

8 Survived 889 non-null int64

dtypes: float64(1), int64(4), object(4)

memory usage: 62.6+ KB



이번에도 모든 변수의 ❶ Non-Null Count 값이 889로 모두 같습니다. 즉 빈 값(결측치)이 없습니다.

❷ Dtype에서 자료형을 살펴보니 문자형(object) 변수가 4개입니다. Name, Sex, Ticket, Embarked입니다.

이번에는 통계 정보를 확인하겠습니다.

| data.describe() # 통계 정보 출력 |
| --- |

|  | Pclass | Age | SibSp | Parch | Survived |
| --- | --- | --- | --- | --- | --- |
| count | 889.00 | 889.00 | 889.00 | 889.00 | 889.00 |
| mean | 2.31 | 29.32 | 0.52 | 0.38 | 0.38 |
| std | 0.83 | 12.98 | 1.10 | 0.81 | 0.49 |
| min | 1.00 | 0.42 | 0.00 | 0.00 | 0.00 |
| 25% | 2.00 | 22.00 | 0.00 | 0.00 | 0.00 |
| 50% | 3.000000 | 28 | 0.000000 | 0.000000 | 0 |
| 75% | 3.000000 | 35 | 1.000000 | 0.000000 | 1 |
| max | 3.000000 | 80 | 8.000000 | 6.000000 | 1 |



❶ Pclass에서 min부터 max까지의 값을 보면 1/2/3 총 3가지의 값이 있습니다. ❷ Age는 50%값(중앙값)이 28, 75%(상위 25%)값이 35였다가 max가 80으로 갑자기 높아집니다. 대부분의 승객이 비교적 젊은 층이지만, 일부 나이가 많은 승객들이 있다고 해석할 수 있습니다. ❸ SibSp와 ➍ Parch는 25%(하위 25%)와 50%값이 모두 0입니다. ➍ Parch의 경우 75%값까지 0입니다. 즉, 대부분 승객이 가족을 동반하지 않고 혼자 탑승했습니다.

이번에는 각 변수의 상관관계를 확인하겠습니다. 상관관계는 두 변수 간의 변화가 서로 연관되었을 때, 즉 A가 증가할수록 B가 증가하거나, 반대로 A가 증가할수록 B가 감소하거나 하는 관계를 숫자로 보여줍니다. 변수 이름만 놓고 봤을 때 서로 관련이 있을 것 같은 변수가 보이나요?

<용어/>

**상관관계**

두 변수 간의 관련성. 예를 들어 변수 A가 증가할 때, B도 증가하면 상관관계가 있다고 할 수 있습니다. 반대로 A가 증가할 때 B가 감소한다면 이 또한 음의 방향으로 상관관계가 있다 할 수 있습니다.

</>

| data.corr() # 상관관계 출력 |
| --- |

|  | Pclass | Age | SibSp | Parch | Survived |
| --- | --- | --- | --- | --- | --- |
| Pclass | 1.00 | -0.34 | 0.08 | 0.02 | -0.34 |
| Age | -0.34 | 1.00 | -0.23 | -0.17 | -0.07 |
| SibSp | 0.08 | -0.23 | 1.00 | 0.41 | -0.03 |
| Parch | 0.02 | -0.17 | 0.41 | 1.00 | 0.08 |
| Survived | -0.34 | -0.07 | -0.03 | 0.08 | 1.00 |



결과 테이블을 보면 자료형이 object인 변수 4개가 빠져 있습니다. 상관관계는 숫자가 아니면 계산이 안 되기 때문에, 파이썬에서 자동으로 문자형 변수들을 제거하고 상관관계를 보여줍니다. 0에 가까울수록 상관관계가 없는 것이고, 1 혹은 -1에 가까울수록 상관관계가 큰 겁니다. 양수는 정적 상관, 즉 A가 증가할수록 B도 함께 증가하는 경우이며, 반대로 음수는 부적상관으로 A 증가할수록 B가 감소하는 경우입니다. 따라서 단순히 숫자가 크고 작음으로 상관관계의 크기를 판단하면 안 됩니다. -0.5는 0.1보다 작은 숫자지만, 음의 방향일뿐이지 상관관계는 더 크기 때문입니다.

가장 큰 상관관계를 보이는 부분은 Parch와 SibSp입니다(약 0.41). 아마도 혼자 온 승객들이 상당히 많고, 가족을 동반할 경우 부모와 자녀, 형제와 배우자를 함께 동반하는 경우가 많기 때문이 아닐까 짐작할 수 있습니다. 이정도 수치면 높은 상관관계는 아니고 중간정도의 상관관계입니다. 상관관계의 강도에 대한 (절댓값 기준) 일반적인 해석은 다음과 같습니다.

| * 0.2 이하 : 상관관계가 거의 없음 * 0.2 ~ 0.4 : 낮은 상관관계 * 0.4 ~ 0.6 : 중간 수준의 상관관계 * 0.6 ~ 0.8 : 높은 상관관계 * 0.8 이상 : 매우 높은 상관관계 |
| --- |

그런데 위의 상관관계 테이블로는 상관관계 파악이 쉽지 않습니다. 더 파악하기 쉬운 히트맵heatmap이라는 그래프를 그려보겠습니다. 우선 그래프를 그릴 맷플롯립matplotlib과 시본seaborn 라이브러리를 불러옵니다.

| import matplotlib.pyplot as plt import seaborn as sns |
| --- |

히트맵 생성에는 sns.heatmap() 함수를 사용합니다. 앞서 사용한 상관관계 테이블 생성 코드를 인수로 넣어주면 됩니다.

| sns.heatmap(data.corr()) # 상관관계에 대한 히트맵 생성 plt.show() # 그래프 출력(맷플롯립과 시본이 최근 버전이면 제외해도 됨) |
| --- |



히트맵을 그려 상관관계를 색으로 확인할 수 있게 되었습니다. 히트맵 우측에는 색상이 의미하는 수치를 보여줍니다. 그런데 우리는 상관관계의 강도를 절댓값 기준으로 해석하기 때문에, 0을 기준으로 대칭이 되는 색상 배열을 사용하는 편이 더 좋습니다. 색상을 변경하는 매개변수를 추가해 해결하겠습니다.

| sns.heatmap(data.corr(), cmap='coolwarm') # 히트맵 생성  plt.show() # 그래프 출력 |
| --- |



| <note>  극명하게 대비를 주는 coolwarm 색상 배열을 활용했습니다. 시본에서 제공하는 색상 배열은 170여 가지입니다. 이 책에서 모두 다루기에는 너무 많은 색상 배열들이 있으니 구글에서 ‘seaborn palette’로 검색해 여러 아티클에서 다양한 색상들을 활용해보시기 바랍니다.   * 참고 사이트 : https://seaborn.pydata.org/tutorial/color\_palettes.html |
| --- |

이제 더 보기 편해졌습니다. 양수는 빨강, 음수는 파랑 계열로 표현되고 있으며, 관계가 강할수록 더 진하게 표시됩니다. 우측 범례를 보면 빨강과 파랑 사이에서 가장 밝은 부분이 0보다 조금 위쪽인 ❶ 0.4 부근에 있습니다. 입력된 수치의 범위가 1부터 -0.5까지라서 그래프에서는 -1 ~ +1까지가 아닌, ❷ -0.5 ~ +1까지로 잡아 색상을 표시했기 때문입니다. 그래프에서 표시 범위를 -1에서 1까지로 조정하여 색상의 밸런스를 맞춰보겠습니다.

| sns.heatmap(data.corr(), cmap='coolwarm', vmin=-1, vmax=1) |
| --- |



vmin과 vmax 매개변수를 이용하여 데이터 범위의 최솟값과 최댓값을 지정하여 더 밸런스가 맞는 색상을 얻었습니다. 각 칸 안에 상관관계 수치를 표현하면 더 좋을 것 같습니다. 이 역시 추가하겠습니다.

| sns.heatmap(data.corr(), cmap='coolwarm', vmin=-1, vmax=1, annot=True) |
| --- |



드디어 이상적인 형태의 히트맵이 되었습니다. 파이썬에서는 이와 같이 다양한 매개변수를 추가하여 활용할 수 있습니다. 매개변수는 그래프를 그릴 때 뿐만 아니라, 다양한 모듈에 적용됩니다. 모듈에 따라서 가지고 있는 매개변수가 다르기 때문에, 각 모듈에 맞는 매개변수를 적절히 활용해야 합니다.

## 5.4 전처리 : 카테고리 변수 변환하기(더미 변수와 원-핫 인코딩)

타이타닉 데이터셋에는 자료형이 object인 변수들, 즉 데이터가 숫자가 아닌 문자인 변수가 4개 있습니다. 기본적으로 머신러닝 알고리즘에서는 문자열로 된 데이터를 이해하지 못합니다. object형까지도 처리해주는 일부 알고리즘 대부분은 object 컬럼들을 숫자 데이터로 변환하는 기능을 제공합니다.

object형을 숫자화해봅시다. 단순하게는 각 값(특정 문자)을 숫자로 대체하는 방법이 있습니다. 가령, 계절이라는 변수의 값으로 봄, 여름, 가을, 겨울이 있다면 각각을 1, 2, 3, 4로 대체하는 방법입니다. 때에 따라서는 이 방법이 효과적일 때도 있으나 기본적으로 지양해야 합니다. 특히나 선형 모델에서는 이 방법을 사용하면 숫자가 상대적인 서열로 인식됩니다. 즉 봄(1)보다 겨울(4)이 더 큰 개념으로 학습됩니다.



이러한 문제를 피하는 데 더미 변수를 활용합니다. 가장 쉬워보이는 Sex 변수를 가지고 설명하겠습니다. 이 컬럼이 가장 설명하기 쉬운 이유는 값이 male과 female 두 가지뿐이기 때문입니다. 어떻게 변환하는지 아래 테이블을 살펴봐주세요.

## 

기존에 하나던 컬럼을 각 male과 female 컬럼으로 분리했습니다. 그리고 하나에서 두 개로 늘어난 컬럼에는 변수에 해당하면 1, 해당하지 않으면 0을 숫자로 채웠습니다. 머신러닝에서는 이런 식으로 문자로 된 값을 숫자화하여 이해할 수 있게 됩니다. 이런 식으로 변환하는 것을 ‘더미dummy 변수를 만든다’, 혹은 원-핫 인코딩one-hot encoding이라고 합니다.

<용어/>

**더미 변수와 원-핫 인코딩**

카테고리 형태(혹은 문자 형태)의 변수를 숫자로 표현하는 방법으로, 변수에 속해 있는 고윳값에 대한 새로운 변수들을 만들어 0과 1로 표현합니다.

</>

Sex에는 값이 male과 female 두 가지 뿐이라 더미 변수 2개가 만들어졌습니다. 그럼 봄, 여름, 가을, 겨울 값이 있는 계절은 어떻게 될까요? 당연히 값 종류만큼 4개 컬럼이 생성되어야 합니다. 원-핫 인코딩을 할 때 한 가지 고려할 사항이 있습니다. 예를 들어 값이 수백 수천 가지라면 어떻게 할까요? 새로운 컬럼을 수백 수천 개나 만들어야 할까요? 정말 중요하면 어떤 수단을 써서라도 숫자화해야겠지만, 그렇지 않다면 데이터에서 제외시키거나 다른 방법으로 처리하는 것이 좋습니다.

타이타닉 데이터셋에 있는 object들을 살펴봅시다. Name, Sex, Ticket, Embarked입니다. 각 변수마다 고윳값이 몇 가지인지 살펴보겠습니다. nunique() 함수로 고윳값 개수를 확인할 수 있습니다. Name부터 하나씩 하나씩 살펴보겠습니다.

| data['Name'].nunique() |
| --- |

889

| data['Sex'].nunique() |
| --- |

2

| data['Ticket'].nunique() |
| --- |

680

| data['Embarked'].nunique() |
| --- |

3

Sex에는 이미 알다시피 두 가지 값이 있고, Embarked(승선한 항구)도 3개로 그리 많지 않습니다. 전혀 부담되지 않는 수준입니다. 하지만 Name이나 Ticket은 상황이 좀 다릅니다. 고윳값이 수백 가지라서 더미 변수로 변환시키면 그 수만큼 컬럼이 생깁니다. 여기서 이 변수들이 결과를 도출하는 데 꼭 필요한지를 고민해보아야 합니다. 우선은 이름에 따라 사망 여부가 갈린다고 추론하기는 어렵기 때문에 Name 변수를 큰 고민 없이 제외시킬 수 있습니다. Ticket은 티켓 번호입니다. 중요할 수도 있지만, 이미 Pclass(티켓 클래스)와 컬럼을 가지고 있기 때문에 굳이 Ticket 변수로 무언가 얻어낼 필요는 없을 것 같습니다.

따라서 우리는 Name과 Ticket 변수를 데이터에서 제외하고, 남은 두 object형을 원-핫 인코딩하겠습니다. 우선 drop() 함수를 사용하여 Name과 Ticket을 제거하고 head() 함수로 제대로 제거되었는지 확인해봅시다.

| data = data.drop(['Name','Ticket'], axis=1)  data.head() |
| --- |

|  | **Pclass** | **Sex** | **Age** | **SibSp** | **Parch** | **Embarked** | **Survived** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 3 | male | 22 | 1 | 0 | S | 0 |
| **1** | 1 | female | 38 | 1 | 0 | C | 1 |
| **2** | 3 | female | 26 | 0 | 0 | S | 1 |
| **3** | 1 | female | 35 | 1 | 0 | S | 1 |
| **4** | 3 | male | 35 | 0 | 0 | S | 0 |

이제 판다스의 get\_dummies() 함수를 사용하여 문자 형태의 변수들을 원-핫-인코딩 해보겠습니다.

| pd.get\_dummies(data, columns = ['Sex','Embarked']) |
| --- |

괄호 안에 데이터 프레임(여기서는 data)을 먼저 써주시고, columns라는 매개변수에 변환시킬 컬럼명을 리스트 형태로 넣으면 됩니다. 그럼 다음과 같은 결과물을 얻게 됩니다. 참고로 이 코드의 결과를 data에 저장하지는 않습니다. 변환된 모습만 출력했습니다.

|  | **Pclass** | **Age** | **SibSp** | **Parch** | **Survived** | **Sex\_female** | **Sex\_male** | **Embarked\_C** | **Embarked\_Q** | **Embarked\_S** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 3 | 22 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| **1** | 1 | 38 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| **2** | 3 | 26 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| **3** | 1 | 35 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| **4** | 3 | 35 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **884** | 2 | 27 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| **885** | 1 | 19 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| **886** | 3 | 28 | 1 | 2 | 0 | 1 | 0 | 0 | 0 | 1 |
| **887** | 1 | 26 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| **888** | 3 | 32 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
|  |  |  |  |  |  |  |  |  |  |  |
| 889 rows × 10 columns |  |  |  |  |  |  |  |  |  |  |



예상대로 Sex에서 2개, Embaked에서 3개, 총 5개의 컬럼이 오른쪽에 새로 생겼습니다. 더미 변수가 생기면서 기존에 있던 Sex와 Embarked 컬럼(변수)는 사라졌습니다. 여기서 한 단계만 더 나아가보겠습니다. Sex는 Sex\_female과 Sex\_male로 분리되었죠. 과연 둘 다 필요할까요? 예를 들어 Sex\_male이 0이면 당연히 이 승객은 female에 해당합니다. 둘 중 하나만 남겨도 구분이 가능하겠군요. Embarked도 마찬가지입니다. Embarked\_Q와 Embarked\_S가 모두 0이면 Embarked\_C에 해당하는 승객입니다. 즉, 우리는 더미 변수에서 고윳값 개수보다 하나를 덜 사용해도 구분하는 데 문제가 없습니다. 이렇게 컬럼 개수를 줄여주면 데이터 계산량이 줄어듭니다. 우리가 테이블을 눈으로 확인할 때도 조금이나마 부담을 줄일 수 있습니다. get\_dummies() 함수는 이 기능도 제공합니다. drop\_first 매개변수를 추가하면 됩니다.

| pd.get\_dummies(data, columns = ['Sex','Embarked'], drop\_first = True) |
| --- |

|  | **Pclass** | **Age** | **SibSp** | **Parch** | **Survived** | **Sex\_male** | **Embarked\_Q** | **Embarked\_S** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 3 | 22 | 1 | 0 | 0 | 1 | 0 | 1 |
| **1** | 1 | 38 | 1 | 0 | 1 | 0 | 0 | 0 |
| **2** | 3 | 26 | 0 | 0 | 1 | 0 | 0 | 1 |
| **3** | 1 | 35 | 1 | 0 | 1 | 0 | 0 | 1 |
| **4** | 3 | 35 | 0 | 0 | 0 | 1 | 0 | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **884** | 2 | 27 | 0 | 0 | 0 | 1 | 0 | 1 |
| **885** | 1 | 19 | 0 | 0 | 1 | 0 | 0 | 1 |
| **886** | 3 | 28 | 1 | 2 | 0 | 0 | 0 | 1 |
| **887** | 1 | 26 | 0 | 0 | 1 | 1 | 0 | 0 |
| **888** | 3 | 32 | 0 | 0 | 0 | 1 | 1 | 0 |
|  |  |  |  |  |  |  |  |  |
| 889 rows × 8 columns |  |  |  |  |  |  |  |  |



기존에는 10개 컬럼이 출력되었는데, 8개로 줄어든 모습입니다. 다만, 아까 말씀드렸다시피 이 코드는 변환된 모습을 보여줄 뿐 data에 저장하지 않습니다. data에 최종 데이터를 저장해줍시다.

| data = pd.get\_dummies(data, columns = ['Sex','Embarked'], drop\_first = True) |
| --- |

## 5.5 모델링 및 예측하기

모델링하기에 앞서 독립변수와 종속변수, 그리고 훈련셋과 시험셋으로 나누어주겠습니다.

| from sklearn.model\_selection import train\_test\_split  X = data.drop('Survived', axis = 1) # 데이터셋에서 종속변수 제거 후 저장 y = data['Survived'] # 데이터셋에서 종속변수만 저장 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 100) #학습셋, 시험셋 분리 |
| --- |

로지스틱 회귀 분석 모듈은 선형 회귀 분석과 마찬가지로 sklearn.linear\_model에서 불러올 수 있습니다.

| from sklearn.linear\_model import LogisticRegression # 로지스틱 회귀 임포트 |
| --- |

4장에서와 동일하게 model이라는 이름 안에 로지스틱 회귀 분석 속성을 부여하고, fit() 함수로 훈련시킵니다.

| model = LogisticRegression() # 로지스틱 회귀 모델 생성 model.fit(X\_train, y\_train) # 모델 학습 |
| --- |

예측하는 함수 또한 동일하게 predict()를 사용합니다.

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

## 5.6 예측 모델 평가하기

1장에서는 실젯값과 예측값의 차이를 각각 합산하는 RMSE를 사용해 예측 결과를 평가했습니다. 하지만 이번 데이터 목푯값은 0과 1로 나누어진 이진분류Binary classification이기 때문에 RMSE는 평가에 적합하지 않습니다. 다양한 이진분류 평가 지표로는 정확도accuracy(5장), 오차 행렬(7장), 정밀도precision(10장), 재현율recall(10장), F1 Score(10장), 민감도(11장), 특이도(11장), AUC(11장) 등이 있습니다.

그중 가장 간단한 정확도를 사용하겠습니다. 정확도는 예측값과 실젯값을 비교하여 얼마나 맞추었는지를 확인하는 겁니다. 즉 100개의 시험셋을 예측하고, 그중 90개를 정확히 맞췄다면 정확도는 0.9가 되고, 모두 맞추면 1.0이 됩니다.

| from sklearn.metrics import accuracy\_score # 정확도 라이브러리 임포트 accuracy\_score(y\_test, pred) # 실젯값과 예측값으로 정확도 계산 |
| --- |

0.7808988764044944

sklearn.metrics에서 평가 모듈을 불러왔습니다. accuracy\_score()에 실젯값과 예측값을 매개변수로 넣어주면 됩니다. 결과를 보면 약 78% 정도의 정확도를 보입니다. 그렇게 나쁜 수준은 아니지만, 그렇다고 엄청 잘 예측하는 모델이라 할 수는 없습니다.

정확도의 좋고 나쁨을 결정하는 절대적인 지표는 없습니다. 이는 처한 상황에 따라 다르게 고려되어야 합니다. 예를 들어 예측하려는 종속변수의 고윳값이 2개가 아닌 10개라면 상대적으로 더 낮은 정확도도 용인될 수 있습니다. 또한 고윳값이 2개인 이진분류에서도 각각의 비율이 어떠한가에 따라 평가 기준이 달라집니다. 가령 0이 95%이고 1이 5%로 구성된 이진분류라면, 정확도가 90%가 나온들 좋은 값이라고 볼 수 없습니다. 왜냐하면 이 경우는 머신러닝 모델 없이 모든 값을 0으로 예측하는 편법을 써도 정확도는 95%가 나올 수 있기 때문입니다. 만약 이진분류에서 고윳값이 비슷한 비율로 (약 50:50) 분포되어 있다면, 80% 이상의 정확도 정도면 나쁘지 않다고 보고, 90% 이상의 정확도를 얻어야 괜찮은 결과로 보는 편입니다. 타이타닉 데이터의 경우는 높은 정확도를 기대할 수 있는 조건이기 때문에 78%의 정확도는 아쉬움이 많이 남는 결과입니다. 향후 배울 XGBoost나 LightGBM을 사용하면 더 좋은 정확도를 얻을 수 있을 겁니다.

어떤 변수가 어떤 영향을 미쳤는지 계수를 통하여 확인해봅시다. 방법은 1장 ‘선형 회귀 분석’과 거의 같습니다. 약간 다른 점이 있어 설명드리겠습니다. 다음은 선형 회귀 분석에서 확인했던 model.coef\_의 결과물입니다.

array([2.59757578e+02, 1.82169249e+01, 2.77903898e+02, 4.61169867e+02,

2.39817410e+04])

그럼 로지스틱 회귀 분석 모델의 계수도 확인하겠습니다.

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

array([[-1.18222701, -0.03992812, -0.32136451, 0.00796449, -2.56868467,

-0.07899451, -0.23563186]])

다른 점을 발견했나요? 1장에서는 array( ) 안에 리스트가 [ ]로 된 반면, 이번에는 2중 리스트 형태인 [ [ ] ]입니다. 1장에서와 같이 다음 코드로 실행하면 에러가 발생합니다.

| # X의 컬럼을 사용해 판다스 시리즈로 변환  pd.Series(model.coef\_, index = X.columns) |
| --- |

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

ValueError Traceback (most recent call last)

[<ipython-input-25-4374cac64b97>](https://localhost:8080/#) in <module>()

----> 1 pd.Series(model.coef\_, index = X.columns)

1 frames

[/usr/local/lib/python3.7/dist-packages/pandas/core/common.py](https://localhost:8080/#) in require\_length\_match(data, index)

**530** if len(data) != len(index):

**531** raise ValueError(

--> 532 "Length of values "

**533** f"({len(data)}) "

**534** "does not match length of index "

ValueError: Length of values (1) does not match length of index (7)

에러 메시지의 가장 아랫 줄에 이유가 보입니다. 넣은 데이터 model.coef\_의 길이는 1인데, 인덱스 X.columns의 길이는 7이라서 안 된다는 겁니다. 즉 X.columns으로 7개값을 입력한다고 지정했으니, model.coef\_ 길이가 7이어야 하는 겁니다.

<글상자>

**리스트 길이에 대해서 다시 한번 생각해보기**

예를 들어 [1, 2, 3, 4, 5]의 길이는 5가 됩니다. 하지만 [ [ 1, 2, 3, 4, 5 ] ]는 리스트 1이 됩니다. 겉에 있는 [ ]가 하나의 리스트를 만들고 그 안에는 리스트가 있기 때문입니다. 만약 [ [1,2,3], [4,5] ]라는 리스트가 있다면 길이가 얼마일까요? [1, 2, 3]과 [4, 5] 두 값이 들어 있으므로 2가 됩니다. array나 list의 길이를 len() 함수로 확인할 수 있습니다. 이미 답을 알고 있지만 len() 함수를 사용해 model.coef\_의 길이를 더 확인하겠습니다.

| len(model.coef\_) |
| --- |

1

</>

에러를 피하려면 어떤 방법을 쓰면 될까요? model.coef\_ 길이와 X.columns값을 똑같이 맞추면 됩니다. model.coef의 길이가 1이므로 2중으로 씌워진 리스트 안에 있는 리스트를 콕 지정하면 됩니다. 여기서는 첫 번째 값을 인덱싱하면 되는 거죠.

| len(model.coef\_[0]) # 첫 번째 값의 길이 출력 |
| --- |

7

[0]으로 인덱싱해주어 원하는 형태를 얻었습니다. 원하는 형태를 얻었으니, Series()로 계수를 확인하겠습니다.

| pd.Series(model.coef\_[0], index = X.columns) # model.coef\_를 7개 값이 되도록 풀어서 컬럼 이름을 매핑 |
| --- |

Pclass -1.182227

Age -0.039928

SibSp -0.321365

Parch 0.007964

Sex\_male -2.568685

Embarked\_Q -0.078995

Embarked\_S -0.235632

dtype: float64

Parch를 제외하고는 모두 음수입니다. 목푯값인 survived가 1이면 생존이고, 0이면 사망이라는 점을 유념하고 변수들의 영향을 해석하겠습니다. 우선 Pclass는 음수의 계수를 가지고 있기 때문에 Pclass가 높을수록 생존 가능성이 낮습니다. Pclass는 낮은 숫자일수록 비행기의 퍼스트 클래스처럼 더 비싼 티켓이기 때문에 더 유리하지 작용하지 않았을까 추측해볼 수 있습니다. Age는 낮을수록, 성별은 여성이 생존 가능성이 높습니다. 이미 우리가 알고 있는 타이타닉 이야기와 비슷합니다.

로지스틱 회귀 분석에서는 계수를 단순하게 기울기 값처럼 곱하여 수식을 만들어서는 안 됩니다. 이는 로지스틱 회귀 분석이 선형 회귀 분석에서 한 단계 계산을 더 거치기 때문인데요, 이부분은 5.8절 ‘이해하기 : 로지스틱 회귀 분석’에서 자세히 설명하겠습니다.

## 5.7 이해하기 : 피처 엔지니어링

피처 엔지니어링Feature Engineering이란 기존 데이터를 손보아 더 나은 변수를 만드는 기법입니다(특징 공학 또는 특성 공학이라고도 합니다). 앞에서 경험한 더미 변수를 만드는 일도 일종의 피처 엔지니어링입니다. 더미 변수를 쓰지 않았다면 해당 컬럼(Sex와 Embarked)을 모두 버려야 했으나, 더미 변수로 만들어 예측에 도움이 되는 변수를 얻은 겁니다.

피처 엔지니어링은 머신러닝에 있어서 엄청 중요합니다. 적합한 머신러닝 알고리즘을 선택하고 하이퍼파라미터를 튜닝하는 일도 중요하지만 좋은 피처를 하나 더 마련하는 일만큼 강력한 무기는 없습니다. 여기서 피처라 함은 독립변수의 다른 표현입니다. 독립변수는 흔히 통계 영역에서 쓰이는 용어이고, 피처는 머신러닝에서 더 흔하게 사용됩니다. 이 책에서는 두 용어를 혼용합니다.

피처 엔지니어링에서는 도메인 지식의 활용이 중요합니다. 데이터에 대한 사전 지식이 있으면 어떤 변수를 어떻게 바꾸면 더 나은 피처를 얻을 수 있을지 생각해볼 여지가 있습니다. 물론 도메인 지식 없이 몇몇 변수를 단순히 곱하거나 나누어서 무작위로 다양한 변수들을 만들어내는 것만으로도 도움이 될 때도 있습니다만, 도메인 지식을 바탕으로 정확한 목적을 가지고 수행하는 피처 엔지니어링이 더욱 효율적입니다.

선형 회귀 분석과 로지스틱 회귀 분석을 선형 모델이라고 하는데, 이러한 선형 모델에서는 다중공선성Multicollinearity 문제를 주의해야 합니다. 다중공선성은 독립변수 사이에 상관관계가 높은 때에 발생하는 문제입니다. 예를 들어 두 독립변수 A, B는 모두 목표 변수를 양의 방향으로 이끄는 계수를 가지고 있을 때 A와 B의 상관관계가 매우 높다면, y가 증가한 이유가 A 때문인지 B 때문인지 명확하지 않습니다. 그래서 그때그때 데이터의 특성에 따라 변덕스러운 결과를 보여주는 문제가 발생합니다. 다중공선성 문제는 상관관계가 높은 변수 중 하나를 제거하거나, 둘을 모두 포괄시키는 새로운 변수를 만들거나, PCA와 같은 방법으로 차원을 축소를 수행해 해결할 수 있습니다. 이 장에서 다루는 데이터에서는 Parch와 SibSp가 그나마 조금 강한 상관관계를 보였으므로 이 둘을 새로운 변수로 만들겠습니다.

<용어/>

PCA는 Principal Component Analysis의 약자로, 주성분 분석이라고도 부릅니다. 차원 축소는 데이터의 차원, 즉 독립변수의 개수를 줄이는 방법을 뜻하는데 13장에서 자세히 다룹니다.

</>

<용어/>

다중공선성 문제

변수 간의 강한 상관관계가 있을 때 발생하는 문제. 선형 모델은 독립변수간의 독립성을 전제로 하기 때문에, 다중공선성 문제를 해결해주는 것이 좋습니다.

</>

사실 꼭 상관관계 수치 때문이 아니더라도, 직관적으로 이 두 변수를 합쳐보고 싶다는 생각이 들 수도 있습니다. Parch는 부모와 자식, SibSp는 형제/자매와 배우자로, 결국 모두 가족구성원이라는 공통점이 있습니다. 그렇다면 이 두 변수를 합하여 가족구성원 숫자를 나타내는 변수로 만들어보면 어떨까요? 두 변수를 합하여 family라는 컬럼을 만들고 Parch와 SibSp 컬럼을 제거하겠습니다.

| data['family'] = data['SibSp'] + data['Parch'] # SibSp와 Parch 변수 합치기 data.drop(['SibSp','Parch'], axis=1, inplace=True) # SibSp와 Parch 변수 삭제 data.head() # 5행 출력 |
| --- |

| Pclass | **Age** | **Survived** | **Sex\_male** | **Embarked\_Q** | **Embarked\_S** | **family** |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 3 | 22 | 0 | 1 | 0 | 1 | 1 |
| **1** | 1 | 38 | 1 | 0 | 0 | 0 | 1 |
| **2** | 3 | 26 | 1 | 0 | 0 | 1 | 0 |
| **3** | 1 | 35 | 1 | 0 | 0 | 1 | 1 |
| **4** | 3 | 35 | 0 | 1 | 0 | 1 | 0 |



원하던 형태로 잘 변경되었습니다. 드디어 피처 엔지니어링을 마쳤으니 이 데이터를 가지고 모델링 과정부터 평가까지 다시 진행하겠습니다.

| X = data.drop('Survived', axis = 1) y = data['Survived'] X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 100) # 학습셋, 시험셋 준비 model = LogisticRegression() # 모델 생성 model.fit(X\_train, y\_train) # 학습 pred = model.predict(X\_test) # 예측 accuracy\_score(y\_test, pred) # 정확도 계산 |
| --- |

0.7921348314606742

총 독립변수 수가 하나 줄어지만 기존보다 정확도가 0.012 정도 높아졌습니다. 이렇게 모델링은 한 번만에 끝내는 것이 아니라, 다양한 시도를 해가며 수없이 재반복해 더 나은 결과물을 얻어내는 과정입니다.

여기서는 0.4 정도의 상관관계가 있는 변수를 합쳐주면 더 좋은 결과를 얻을 수 있을 겁니다. 그런데 언제나 유효한 기준은 아닙니다. 피처 엔지니어링에는 정답이 없습니다. 데이터 특성이나 도메인 지식에 따라 무궁무진하게 확장할 수 있습니다. 물론, 그 어떤 피처 엔지니어링도 수행할 수 없는 경우도 있겠죠. 그렇기 때문에 피처 엔지니어링은 매우 중요하면서도 아주 까다롭습니다.

## 5.8 이해하기 : 로지스틱 회귀 분석

우선 아주 근본적인 질문에서부터 시작하겠습니다. 타이타닉 데이터와 같은 이진분류 문제에, 즉 목푯값이 0과 1로 구성된 때는 왜 선형 회귀 분석을 사용하지 않을까요? 0과 1도 어차피 숫자인데 말이죠. 선형 회귀 분석을 적용시킬 수 없는 건 아닙니다만 그 결괏값은 우리가 예상하는 형태가 아니게 됩니다. 아래 그래프를 예시로 들어 설명하겠습니다.



이 그래프에서는 독립변수가 단 1개만 존재하고(x축), 목푯값은 0과 1만 있는 데이터입니다. 각 데이터는 파란 원으로 표시했으며, 이를 선형 회귀 분석으로 모델링하면 빨간선 같은 형태의 모델을 얻을 수 있습니다. 전반적으로 독립변수 x가 클 수록 y 값이 1일 확률이 높아지는 형태입니다. 그런데 문제는, x값이 특정 범위가 넘어서면 크게는 1 이상으로 넘어가고, 작게는 0 아래로 내려가서 마이너스의 값을 예측합니다.

목푯값은 0과 1입니다. 따라서 예측값이 그 사이에 있어야 합니다. 물론 범위를 넘어선 값들을 강제로 1과 0으로 귀속시킬 수도 있습니다. 예를 들어 예측값이 1.7이라면 1로 강제로 변경해주고, -0.3이 나오면 0으로 변경해주도록 알고리즘을 만드는 거죠. 하지만 이러한 예측은 그다지 정확하지 못할 겁니다.

우리는 이진분류 문제에서 목푯값은 0과 1이므로 예측값이 해당 범위 안에서 나오기를 바랍니다. 로지스틱 회귀 분석은 이러한 문제를 해결해줍니다. 로지스틱 회귀 분석은 로지스틱 함수Logistic function를 사용하여 선형 회귀 분석의 직선 형태를 다음과 같은 그림으로 변형시켜줍니다.



곡선을 그리면서 한없이 1과 0에 가까워지는 형태입니다. 때문에 로지스틱 회귀 분석에서의 예측값은 절대 1과 0사이를 벗어나지 않게 됩니다.

그런데 우리가 코딩을 통하여 로지스틱 회귀 분석의 예측값을 확인했을 때 분명 값들이 0과 1로만 구성되어 있습니다. 하지만 위 그래프를 보면 0과 1 사이의 수많은 수들, 예를 들어 0.3이나 0.7 같은 예측값도 나올 수 있을 것으로 보입니다. 이미 알아차리신 분도 계시겠지만, 예측값 0.3은 Yes일 확률이 30%, 0.7은 70%를 의미합니다. 실제로 이 모델은 0과 1로만 구성된 결괏값이 아닌 수많은 다른 예측값을 계산해내지만, 기본적으로 0.5를 기준으로 1과 0으로 변환시킨 값을 보여줍니다. 0과 1사이의 구체적인 수들을 얻고 싶다면 predict() 대신 predict\_proba()를 사용해야 합니다(11장 ‘LightGBM : 카드 거래 내역 데이터셋’에서는 0.5가 아닌 다른 기준값으로 0과 1을 분류하는 실습도 진행합니다).

다음은 로지스틱 회귀 분석의 공식입니다. e는 자연로그의 밑 (2.718)이고, 지수부 는 4장에서 다룬 1차 함수의 형태와 같습니다. 즉, 로지스틱 회귀 분석은 선형 회귀 분석의 공식을 한 번 더 변환한 형태입니다.

이 책이 머신러닝 입문자를 대상으로 하는만큼, 여기에서는 수식에 대한 도출까지는 다루지 않겠습니다. 수식의 도출보다는 알고리즘의 기본적인 원리를 이해하는 데 집중해주세요.

## 학습 마무리

#### 되짚어보기

5.1 타이나틱호 탑승객 정보를 사용하여 생존 여부를 예측하는 문제를 정의합니다.

5.2 판다스와 프로젝트에 쓸 예제 데이터셋을 불러옵니다.

5.3 데이터 특성을 살펴보았습니다. 특이사항으로 SisSp(함께 탑승한 형제 및 배우자의 수)와 Parch(함께 탑승한 부모 및 자녀의 수)에서 0.41정도의 상관관계가 보였습니다.

5.4 더미 변수를 사용하여 카테고리 변수인 Sex와 Embarked(승선한 항구)를 숫자 형태로 변환해주었습니다.

5.5 로지스틱 회귀를 사용하여 예측 모델을 만들었습니다.

5.6 약 78%의 정확도를 얻었습니다.



#### 과제

피처 엔지니어링 과정에서 family 변수를 만드는 대신, Parch나 SibSp 중 하나를 제거하는 방법으로 했을 때의 예측률도 확인해보세요. 피처 엔지니어링에 대한 또 다른 아이디어가 떠오른다면 다양하게 시도해보고 더 나은 예측률이 나오는 모델을 만들어봅시다.

#### 핵심 용어 정리

1. **로지스틱 회귀** : 선형 회귀 분석을 기반으로 한 모델로, 연속형 종속 변수가 아닌 이진분류 문제를 위한 알고리즘입니다.
2. **피처 엔지니어링** : 기존 변수에서 더 나은 변수를 도출해내는 작업입니다. Parch와 SibSp를 사용해 새로운 변수 family를 만들어보았습니다.
3. **상관관계** : 두 변수간의 연관성을 나타내는 것으로, 상관관계가 높으면 절대값이 1에 가깝습니다.
4. **PCA** : Principal Component Analysis의 약자로, 주성분 분석이라고도 부릅니다.
5. **다중공선성 문제** : 변수 간의 강한 상관관계가 있을 때 발생하는 문제. 선형 모델은 독립변수간의 독립성을 전제로 하기 때문에, 다중공선성 문제를 해결해 주는 것이 좋습니다.
6. **더미 변수와 원-핫 인코딩** : 카테고리 형태(혹은 문자 형태)의 변수를 숫자로 표현하는 방법으로, 변수에 속해 있는 고윳값에 대한 새로운 변수들을 만들어 1과으로 표현합니다(해당 변수는 1, 나머지 변수는 0).

#### 새로운 함수와 라이브러리

1. **len()** : 데이터의 길이 확인
2. **pandas.DataFrame.nunique()** : 고윳값의 개수 확인
3. **pandas.DataFrame.drop()** : 데이터프레임의 행/열 제거
4. **pandas.get\_dummies()** : 더미 변수로 변환
5. **sns.heatmap()** : 히트맵 그리기

## 연습문제

1. 다음 상관관계를 설명한 내용 중 틀린 것은?

① 두 변수 간의 상관관계를 나타내는 척도다.

② 상관관계가 작으면 -1, 높으면 1에 가까운 숫자를 보인다.

③ 변수간의 상관관계가 높은 경우 선형 모델에 안 좋은 영향을 미칠 수 있다.

④ 파이썬에서는 corr() 함수로 확인할 수 있다.

2. 더미 변수에 대한 설명 중 옳지 않은 것은?

① 여러 머신러닝 알고리즘이 문자형 데이터를 소화할 수 없기 때문에 필요하다.

② 더미 변수로 변환해도, 총 변수의 개수는 항상 기존과 동일하게 유지된다.

③ 문자형 데이터를 0과 1의 숫자형 데이터로 변환해준다.

④ 새로 생성되는 더미 변수에서 1개를 제외해도 상관없다.

3. 다음 중 선형 회귀분석이 아닌 로지스틱 회귀가 필요한 경우는?

① 100,000~999,000원으로 분포된 매출액 예측

② 150~190cm로 분포된 학생의 키 예측

③ 0과 1로 표현된 암 진단 예측

④ 100점 만점의 시험 점수 예측

#### 정답 및 해설

1. 2

② 상관관계가 작으면 -1, 높으면 1에 가까운 숫자를 보인다. ← 상관관계가 작으면 절대값이 0에 가까운 숫자를 크면 절대값이 1에 가까운 숫자를 보입니다.

2. 더미 변수에 대한 설명 중 옳지 않은 것은? 2

② 더미 변수로 변환해도, 총 변수의 개수는 항상 기존과 동일하게 유지된다. ← 더미 변수로 변환하면, 고윳값 개수만큼 변수가 늘어납니다.

3. 다음 중 선형 회귀분석이 아닌 로지스틱 회귀가 필요한 경우는? 3

① 100,000~999,000원으로 분포된 매출액 예측 ← 연속형 변수이므로 선형 회귀분석을 사용합니다.

② 150~190cm로 분포된 학생의 키 예측 ← 연속형 변수이므로 선형 회귀분석을 사용합니다.

③ 0과 1로 표현된 암 진단 예측 ← 0과 1로 나타낸 이진분류이므로 로지스틱 회귀가 필요합니다.

④ 100점 만점의 시험 점수 예측 ← 연속형 변수이므로 선형 회귀분석을 사용합니다.