# 10장 XGBoost : 스피드 데이팅 데이터셋

#### 학습 목표

부스팅 모델 중 가장 유명한 XGBoos를 활용하여 예측 모델을 만들고, 그리드 서치Grid Search로 하이퍼파라미터를 튜닝하여 더 나은 모델을 만드는 방법을 학습합니다.

#### 학습 순서



#### XGBoost 소개

랜덤 포레스트는 각 트리를 독립적으로 만드는 알고리즘입니다. 반면 부스팅은 순차적으로 트리를 만들어 이전 트리로부터 더 나은 트리를 만들어내는 알고리즘입니다. 부스팅 알고리즘은 트리 모델을 기반으로 한 최신 알고리즘 중 하나로, 랜덤 포레스트보다 훨씬 빠른 속도와 더 좋은 예측 능력을 보여줍니다. 이에 속하는 대표적인 알고리즘으로 XG부스트XGBoost, 라이트GBMLightGBM, 캣부스트CatBoost 등이 있습니다. 그중 XGBoost(eXtra Gradient Boost)가 가장 먼저 개발되기도 했고, 가장 널리 활용됩니다. XGBoost는 손실함수뿐만 아니라 모형 복잡도까지 고려합니다.



#### 장단점

| **장점** | **단점** |
| --- | --- |
| 예측 속도가 상당히 빠르며, 예측력 또한 좋습니다. | 복잡한 모델인 만큼, 해석에 어려움이 있습니다. |
| 변수 종류가 많고 데이터가 클수록 상대적으로 뛰어난 성능을 보여줍니다. | 더 나은 성능을 위한 하이퍼파라미터 튜닝이 까다롭습니다. |

#### 유용한 곳

* 종속변수가 연속형 데이터인 경우든 범주형 데이터인 경우든 모두 사용할 수 있습니다.
* 이미지나 자연어가 아닌 표로 정리된 데이터의 경우, 거의 모든 상황에 활용할 수 있습니다.

#### TOP 10 선정 이유

* 캐글 컴피티션의 우승자가 많이 사용하는 성능이 검증된 부스팅 모델입니다. XGBoost 이후로도 다양한 부스팅 모델들이 소개되었지만, 가장 인기있는 모델이기 때문에 구글 검색에서 수많은 참고 자료(활용 예시, 다양한 하이퍼파라미터 튜닝)들을 쉽게 접할 수 있습니다.

<용어/>

**부스팅 알고리즘**

부스팅은 랜덤 포레스트에서 그 다음 세대로 진화하게 되는 중요한 개념입니다. 랜덤 포레스트에서는 각각의 트리를 독립적으로, 즉 서로 관련 없이 만드는 반면, 부스팅 알고리즘에서는 트리를 순차적으로 만들면서 이전 트리에서 학습한 내용이 다음 트리를 만들 때 반영됩니다.

</>

## 10.1 문제 정의 : 한눈에 보는 분석 목표

<금토끼의 문제 정의> “올 크리스마스는 꼭 연인과 함께!” 금토끼의 의지가 아주 단단합니다. 그동안 부정적이던 스피드 데이팅 이벤트까지 신청했습니다. ‘수십 명이 참여하는 데 내 인연 한 명 없겠어’하는 생각으로 희망에 들떠 참석했지만 안타깝게도 커플 성사에 실패하고 말았습니다. 여기서 좌절하면 집념의 토끼가 아닙니다. 다음 스피드 데이팅 이벤트에 참여 신청을 하고 멋진 옷도 샀습니다. 이전 이벤트를 분석해 어떤 사람들끼리 커플 성사가 잘 되는지를 예측하는 모델에 돌입했습니다.

| **난이도** | ⭐⭐⭐ | | |
| --- | --- | --- | --- |
| **알고리즘** | XGBoost(XGBoost) | | |
| **데이터셋 파일명** | dating.csv | **종속 변수** | match(커플 성사 여부) |
| **데이터셋 소개** | 스피드데이팅에 대한 데이터입니다. 스피드 데이팅은 남녀 수십 쌍이 짧은 시간을 보낸 뒤 서로에 대한 호감도를 표현하여 짝을 매칭하는 이벤트입니다. 독립변수로는 상대방과 내 정보, 개인의 취향, 상대방에 대한 평가 등이 있으며, 매칭 성사 여부가 종속변수입니다. | | |
| **문제 유형** | 분류 | **평가지표** | 정확도, 혼동 행렬, 분류 리포트 |
| **사용한 모델** | XGBClassifier | | |
| **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) * xgboost (xgboost==0.90) | | |
| **예제 코드 노트북** | 위치 : <https://github.com/musthave-ML10/notebooks/>  파일명 : 10\_XGBoost.ipynb | | |

## 10.2 라이브러리 및 데이터 불러오기, 데이터 확인하기

4개의 필수 모듈과 dating.csv 파일을 불러오겠습니다.

| import pandas as pd  import numpy as np import matplotlib.pyplot as plt import seaborn as sns  file\_url = 'https://media.githubusercontent.com/media/musthave-ML10/data\_source/main/dating.csv' data = pd.read\_csv(file\_url) # 데이터셋 읽기 |
| --- |

head()를 사용하여 데이터가 어떻게 생겼는지 간단히 살펴보겠습니다.

| data.head() # 상위 5행 출력 |
| --- |

|  | has\_null | gender | age | age\_o | race | race\_o | importance\_same\_race | importance\_same\_religion | pref\_o\_attractive | pref\_o\_sincere | ... | funny\_partner | ambition\_partner | shared\_interests\_partner | interests\_correlate | expected\_happy\_with\_sd\_people | expected\_num\_interested\_in\_me | like | guess\_prob\_liked | met | match |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | female | 21.0 | 27.0 | Asian/PacificIslander/Asian-American | European/Caucasian-American | 2 | 4 | 35 | 20 | ... | 7 | 6.0 | 5 | 0.14 | 3 | 2 | 7 | 6 | 0 | 0 |
| 1 | 0 | female | 21.0 | 22.0 | Asian/PacificIslander/Asian-American | European/Caucasian-American | 2 | 4 | 60 | 0 | ... | 8 | 5.0 | 6 | 0.54 | 3 | 2 | 7 | 5 | 1 | 0 |
| 2 | 1 | female | 21.0 | 22.0 | Asian/PacificIslander/Asian-American | Asian/PacificIslander/Asian-American | 2 | 4 | 19 | 18 | ... | 8 | 5.0 | 7 | 0.16 | 3 | 2 | 7 | NaN | 1 | 1 |
| 3 | 0 | female | 21.0 | 23.0 | Asian/PacificIslander/Asian-American | European/Caucasian-American | 2 | 4 | 30 | 5 | ... | 7 | 6.0 | 8 | 0.61 | 3 | 2 | 7 | 6 | 0 | 1 |
| 4 | 0 | female | 21.0 | 24.0 | Asian/PacificIslander/Asian-American | Latino/HispanicAmerican | 2 | 4 | 30 | 10 | ... | 7 | 6.0 | 6 | 0.21 | 3 | 2 | 6 | 6 | 0 | 1 |

변수가 39개나 되어서 기본 출력 화면에 다 담지 못해 중간에 ...으로 표시된 부분이 있습니다. 이는 판다스는 기본 20개 컬럼까지만 보여주며, 20개를 넘으면 ... 표시 후 생략합니다. 다음과 같이 보고 싶은 컬럼 수를 지정할 수 있습니다. 40개까지 볼 수 있도록 변경하겠습니다.

| pd.options.display.max\_columns = 40 # 총 40개 컬럼까지 출력되도록 설정 |
| --- |

<tip>

컬럼 대신 로우 제한을 변경하려면 max\_columns을 max\_rows로 바꿔주면 됩니다.

</>

info() 함수를 호출해 어떠한 변수가 있는지 구체적으로 살펴봅시다.

| data.info() # 변수 특징 출력 |
| --- |

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

RangeIndex: 8378 entries, 0 to 8377

Data columns (total 39 columns):

# Column Non-Null Count Dtype

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

0 has\_null 8378 non-null int64

1 gender 8378 non-null object

2 age 8283 non-null float64

3 age\_o 8274 non-null float64

4 race 8315 non-null object

5 race\_o 8305 non-null object

6 importance\_same\_race 8299 non-null float64

7 importance\_same\_religion 8299 non-null float64

8 pref\_o\_attractive 8289 non-null float64

9 pref\_o\_sincere 8289 non-null float64

10 pref\_o\_intelligence 8289 non-null float64

11 pref\_o\_funny 8280 non-null float64

12 pref\_o\_ambitious 8271 non-null float64

13 pref\_o\_shared\_interests 8249 non-null float64

14 attractive\_o 8166 non-null float64

15 sincere\_o 8091 non-null float64

16 intelligence\_o 8072 non-null float64

17 funny\_o 8018 non-null float64

18 ambitous\_o 7656 non-null float64

19 shared\_interests\_o 7302 non-null float64

20 attractive\_important 8299 non-null float64

21 sincere\_important 8299 non-null float64

22 intellicence\_important 8299 non-null float64

23 funny\_important 8289 non-null float64

24 ambtition\_important 8279 non-null float64

25 shared\_interests\_important 8257 non-null float64

26 attractive\_partner 8176 non-null float64

27 sincere\_partner 8101 non-null float64

28 intelligence\_partner 8082 non-null float64

29 funny\_partner 8028 non-null float64

30 ambition\_partner 7666 non-null float64

31 shared\_interests\_partner 7311 non-null float64

32 interests\_correlate 8220 non-null float64

33 expected\_happy\_with\_sd\_people 8277 non-null float64

34 expected\_num\_interested\_in\_me 1800 non-null float64

35 like 8138 non-null float64

36 guess\_prob\_liked 8069 non-null float64

37 met 8003 non-null float64

38 match 8378 non-null int64

dtypes: float64(34), int64(2), object(3)

memory usage: 2.5+ MB

총 8378개 관측치가 있고 39개 변수가 있습니다. 마지막 변수 match가 종속 변수이며 나머지는 독립변수입니다. 피처 엔지니어링에 사용할 주요 변수를 알아보고 넘어가겠습니다.

* has\_null : 변수 중 Null값이 있는지 여부. 단, 이 데이터는 기존 데이터에서 일부 변수들이 생략된 축소판이기 때문에, 여기서 보이는 Null값 여부와 다소 차이가 있을 수 있습니다. 전반적으로 무응답 항목이 있는지에 대한 정보이므로 그대로 사용하겠습니다.
* age / age\_o : age는 본인 나이이며 age\_o는 상대방 나이입니다.
* race / race\_o : 마찬가지로 본인과 상대의 인종 정보입니다.
* importance\_same\_race / importance\_same\_religion : 인종과 종교를 중요시 여기는지에 대한 응답입니다.
* attractive(매력적인), sincere(성실한), intelligence(지적), funny(재미난), ambitious(야심찬), shared\_interests(공통관심사) : 이 항목들은 4가지 차원에서 평가되어 총 변수가 24(6 × 4)개입니다. 4가지 차원에 대한 설명을 드리겠습니다.
  + pref\_o\_xxx(예 : pref\_o\_attractive) : 상대방이 xxx 항목을 얼마나 중요하게 생각하는가에 대한 응답입니다.
  + xxx\_o(예: attractive\_o) : 상대방이 본인에 대한 xxx 항목을 평가한 항목입니다.
  + xxx\_important(예 : attractive\_important) : xxx 항목에 대해 본인이 얼마나 중요하게 생각하는가에 대한 응답입니다.
  + xxx\_partner(예 : attractive\_partner) : 본인이 상대방에 대한 xxx 항목을 평가한 항목입니다.
* interests\_correlate : 관심사(취미 등) 연관도
* expected\_happy\_with\_sd\_people : 스피드 데이팅을 통해 만난 사람과 함께할 때 얼마나 좋을지에 대한 기대치
* expected\_num\_interested\_in\_me : 얼마나 많은 사람이 나에게 관심을 보일지에 대한 기대치
* like : 파트너가 마음에 들었는지 여부
* guess\_prob\_liked : 파트너가 나를 마음에 들어했을지에 대한 예상
* met : 파트너를 스피드 데이팅 이벤트 이전에 만난 적이 있는지 여부

다음은 describe() 함수로 통계적 정보를 확인하겠습니다.

| round(data.describe(), 2) # describe를 반올림하여 출력 |
| --- |

|  | has\_null | age | age\_o | importance\_same\_race | importance\_same\_religion | pref\_o\_attractive | pref\_o\_sincere | pref\_o\_intelligence | pref\_o\_funny | pref\_o\_ambitious | ... | funny\_partner | ambition\_partner | shared\_interests\_partner | interests\_correlate | expected\_happy\_with\_sd\_people | expected\_num\_interested\_in\_me | like | guess\_prob\_liked | met | match |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| count | 8378.00 | 8283.00 | 8274.00 | 8299.00 | 8299 | 8289 | 8289 | 8289 | 8280 | 8271 | ... | 8028 | 7666.0 | 7311 | 8220 | 8277 | 1800 | 8138 | 8069 | 8003 | 8378 |
| mean | 0.87 | 26.36 | 26.36 | 3.78 | 3.65 | 22.5 | 17.4 | 20.27 | 17.46 | 10.69 | ... | 6.4 | 6.8 | 5.47 | 0.2 | 5.53 | 5.57 | 6.13 | 5.21 | 0.05 | 0.16 |
| std | 0.33 | 3.57 | 3.56 | 2.85 | 2.81 | 12.57 | 7.04 | 6.78 | 6.09 | 6.13 | ... | 1.95 | 1.8 | 2.16 | 0.3 | 1.73 | 4.76 | 1.84 | 2.13 | 0.28 | 0.37 |
| min | 0.00 | 18.00 | 18.00 | 0.00 | 1 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0.0 | 0 | -0.83 | 1 | 0 | 0 | 0 | 0 | 0 |
| 25% | 1.00 | 24.00 | 24.00 | 1.00 | 1 | 15 | 15 | 17.39 | 15 | 5 | ... | 5 | 6.0 | 4 | -0.02 | 5 | 2 | 5 | 4 | 0 | 0 |
| 50% | 1.00 | 26.00 | 26.00 | 3.00 | 3 | 20 | 18.37 | 20 | 18 | 10 | ... | 7 | 7 | 6 | 0.21 | 6 | 4 | 6 | 5 | 0 | 0 |
| 75% | 1.00 | 28.00 | 28.00 | 6.00 | 6 | 25 | 20 | 23.81 | 20 | 15 | ... | 8 | 8 | 7 | 0.43 | 7 | 8 | 7 | 7 | 0 | 0 |
| max | 1.00 | 55.00 | 55.00 | 10.00 | 10 | 100 | 60 | 50 | 50 | 53 | ... | 10 | 10 | 10 | 0.91 | 10 | 20 | 10 | 10 | 8 | 1 |

최댓값을 살펴보면 조금 특이한 부분이 있습니다. 본인 및 상대방을 평가하는 변수(xxx\_o, xxx\_partner)에서는 대체로 최댓값이 10으로 일정하나, 중요도와 관련된 변수(pref\_o\_xxx, xxx\_important)에서는 최댓값이 100부터 60, 50 등 변수별로 다양합니다. 이럴 때는 데이터가 어떠한 형식으로 수집되었는지 알아야 향후 데이터 클리닝 및 피처 엔지니어링을 하는 데 도움이 됩니다.

이 데이터에서는 평가 관련 변수는 0~10점까지 개별 항목에 적용합니다. 중요도 관련 변수(pref\_o\_funny, pref\_o\_ambitious, pref\_o\_shared\_interests, attractive\_o, sincere\_o, intelligence\_o)는 총 100점을 각 항목에 분배해 적용합니다. 예를 들어 상대방이 각 항목에 대한 중요도 변수들의 합은 개인별로 총 100이 되어야 합니다.

## 10.3 전처리 : 결측치 처리

결측치를 처리해야 하니 항목별 결측치 비율을 확인하겠습니다.

| data.isna().mean() |
| --- |

has\_null 0.000000

gender 0.000000

age 0.011339

age\_o 0.012413

race 0.007520

race\_o 0.008713

importance\_same\_race 0.009429

importance\_same\_religion 0.009429

pref\_o\_attractive 0.010623

pref\_o\_sincere 0.010623

pref\_o\_intelligence 0.010623

pref\_o\_funny 0.011697

pref\_o\_ambitious 0.012772

pref\_o\_shared\_interests 0.015397

attractive\_o 0.025304

sincere\_o 0.034256

intelligence\_o 0.036524

funny\_o 0.042970

ambitous\_o 0.086178

shared\_interests\_o 0.128432

attractive\_important 0.009429

sincere\_important 0.009429

intellicence\_important 0.009429

funny\_important 0.010623

ambtition\_important 0.011817

shared\_interests\_important 0.014443

attractive\_partner 0.024111

sincere\_partner 0.033063

intelligence\_partner 0.035331

funny\_partner 0.041776

ambition\_partner 0.084984

shared\_interests\_partner 0.127357

interests\_correlate 0.018859

expected\_happy\_with\_sd\_people 0.012055

expected\_num\_interested\_in\_me 0.785152

like 0.028646

guess\_prob\_liked 0.036882

met 0.044760

match 0.000000

dtype: float64

대부분의 변수에서 결측치가 보이나 대체로 5% 미만입니다. 우리가 이 장에서 사용할 XGBoost 알고리즘도 기본적으로 트리 베이스 모델이라서 결측치를 채우기는 까다롭지 않습니다.[[1]](#footnote-0) 데이터에 등장하지 않을 법한 임의의 숫자, 예를 들어 -99와 같은 숫자로 채워넣는 것으로 해당 사람은 해당 항목에 응답하지 않았음을 나타내보겠습니다. 단, 중요도와 관련된 변수들은 결측치를 제거하는 방향으로 처리하겠습니다. 이유는 곧 진행할 피처 엔지니어링에서 ‘중요도 X 점수’로 계산을 하기 때문입니다. 평가 점수에 관한 변수는 무응답(결측치)을 하나의 응답 종류로 간주하여 사용하기로 합니다.

그럼 우선 해당 변수들에 대해 dropna()로 결측치를 제거하겠습니다.

| data = data.dropna(subset=['pref\_o\_attractive', 'pref\_o\_sincere', 'pref\_o\_intelligence', 'pref\_o\_funny', 'pref\_o\_ambitious', 'pref\_o\_shared\_interests','attractive\_important', 'sincere\_important', 'intellicence\_important', 'funny\_important', 'ambtition\_important', 'shared\_interests\_important']) # 일부 변수에서 결측치 제거 |
| --- |

그리고 나머지 변수들의 결측치는 -99로 채워넣겠습니다.

| data = data.fillna(-99) # 남은 결측치는 -99로 대체 |
| --- |

## 10.4 전처리 : 피처 엔지니어링

이번 장에서는 피처 엔지니어링으로 다룰 내용이 꽤 있습니다. 가장 먼저 살펴볼 부분은 나이와 관련된 변수입니다. 데이터에 상대방 나이와 본인 나이가 있기 때문에, 이를 토대로 나이차가 얼마나 나는지를 계산할 수 있습니다. 여기에서 추가적로 고려해야 할 사항은 결측치입니다. 결측치를 -99로 채워넣었으므로 단순히 나이차를 계산해서는 안 됩니다. ‘알 수 없음’의 의미로 -99를 사용하겠습니다. 또 하나는 성별과 관련된 요인입니다. 단순한 나이 차이보다는 남자가 여자보다 많은지, 반대 경우인지도 고려하는 게 좋습니다.

여러 조건을 반영하여 계산해야 하니 함수로 만들고 각 조건에 알맞게 처리하겠습니다.

| def age\_gap(x): # 함수 정의  if x['age'] == -99: # age가 -99면  return -99 # -99 리턴  elif x['age\_o'] == -99: # age\_o가 -99면  return -99 # -99 리턴  elif x['gender'] == 'female': # gender가 female이면  return x['age\_o'] - x['age'] # age\_o에서 age를 뺀 값 리턴  else: # 나머지 경우는  return x['age'] - x['age\_o'] # age에서 age\_o를 뺀 값 리턴 |
| --- |

❶ 남녀 중 한 명이라도 나이가 -99이면 -99를 반환합니다. ❷ 그렇지 않으면 남자가 연상이면 플러스값이, 여자가 연상이면 마이너스값이 반환됩니다.

정의한 age\_gap() 함수를 데이터프레임에 적용시키겠습니다.

| data['age\_gap'] = data.apply(age\_gap, axis=1) # age\_gap 변수에 age\_gap 함수 적용 |
| --- |

남녀 중 어느쪽이 더 나이가 많은지와 상관없이, 나이 차이 자체가 중요한 변수가 될 수도 있으므로, age\_gap 변수에 절댓값을 취한 변수도 추가하겠습니다. 절댓값은 다음과 같이 abs()를 사용하면 구할 수 있습니다.

| data['age\_gap\_abs'] = abs(data['age\_gap']) # 절댓값적용 |
| --- |

다음은 인종 데이터 관련 피처 엔지니어링입니다. 본인과 상대방의 인종이 같으면 1, 다르면 -1으로 처리합니다. 결측치는 -99를 반환하는 함수를 만들겠습니다.

| def same\_race(x): # 함수 정의  if x['race'] == -99: # race가 -99면  return -99 # -99 리턴  elif x['race\_o'] == -99: # race\_o가 -99면  return -99 # -99 리턴  elif x['race'] == x['race\_o']: # race와 race\_o가 같으면  return 1 # 1 리턴  else: # 나머지 경우는  return -1 # -1 리턴 |
| --- |

그리고 위 함수를 적용하여 same\_race라는 이름으로 새 변수를 만들겠습니다.

| data['same\_race'] = data.apply(same\_race, axis=1) # data를 same\_race 함수에 적용하여 결과를 same\_race 변수로 저장 |
| --- |

인종과 관련된 변수로 importance\_same\_race도 있습니다. 동일 인종 여부가 얼마나 중요한지를 의미하기 때문에, 새로 구한 same\_race 변수와 이 변수를 곱하여 새 변수를 만들겠습니다. 이 계산 결과는 동일 인종이면 양수, 아니면 음수이며 중요할수록 절대값이 큽니다. 중요도가 0인 경우에는 결과값도 0이 나옵니다. 단, 여기서도 결측치가 있는 값은 -99를 반환하도록 함수를 만들겠습니다.

| def same\_race\_point(x): # 함수 정의  if x['same\_race'] == -99: # same\_race가 -99면  return -99 # -99 리턴  else: # 나머지 경우는  return x['same\_race'] \* x['importance\_same\_race'] # same\_race와 importance\_same\_race의 곱을 리턴 |
| --- |

same\_race\_point() 함수를 적용하여 same\_race\_point라는 새 변수를 생성합니다.

| data['same\_race\_point'] = data.apply(same\_race\_point, axis=1) # data에 same\_race\_point 함수를 적용한 결과를 same\_race\_point 변수로 저장 |
| --- |

그런데 왜 same\_race에 1과 0을 대신 1(인종이 같음)과 -1(인종이 다름)을 사용했을까요? 1과 0을 사용하면 importance\_same\_race와 곱하는 과정에서, 인종 여부가 전혀 중요하지 않은 사람은 무조건 0이 나오게 되는데, 이것이 importance\_same\_race가 10인 사람이 동일 인종이 아닌 경우 취하게 되는 값(0)과 같아, 적절한 변별력을 갖기 어렵기 때문입니다. 아래 예시 표를 통해 살펴보겠습니다.

**▽ same\_race가 1과 0인 경우**

|  | **importance\_same\_race**  **(동일 인종 여부의 중요성)** | **same\_race**  **(인종이 같을 때1, 다를 때 0)** | **same\_race\_point** |
| --- | --- | --- | --- |
| a | 10 | 1 | 10 |
| **b** | **10** | **0** | **0** |
| c | 5 | 1 | 5 |
| d | 5 | 0 | 0 |
| **e** | **0** | **1** | **0** |
| f | 0 | 0 | 0 |

❶ b는 인종 여부가 굉장히 중요한데, 동일 인종이 아니어서 0점을 받게 됐습니다. 반면 ❷ e는 동일 인종이지만 전혀 중요하지 않아서 역시 0점을 가지게 됩니다. ❷ e보다 ❶ b가 더 부정적인 상황인데, 이 둘이 같은 0점을 가져 변별력이 떨어집니다.

**▽ same\_race가 1과 -1인 경우**

|  | importance\_same\_race  (동일 인종 여부의 중요성) | same\_race  (인종이 같을 때1, 다를 때 0) | same\_race\_point |
| --- | --- | --- | --- |
| a | 10 | 1 | 10 |
| b | 10 | -1 | -10 |
| c | 5 | 1 | 5 |
| d | 5 | -1 | -5 |
| e | 0 | 1 | 0 |
| f | 0 | -1 | 0 |

이번에는 ❶ b에서 -10이 나옵니다. 동일 인종임이 매우 중요한데 동일 인종이 아니므로 음수가 더욱 적절해보입니다. 반면 동일 인종 여부가 중요하지 않은 ❷ f는 0입니다. 딱히 좋을 것도 나쁠 것도 없다는 의미로 적절합니다. 1과 0을 사용했을 때보다 훨씬 변별력이 있고 합리적인 수치입니다.

마지막으로 attractive, sincere 등에 대한 평가/중요도 변수들을 다루겠습니다. 간단하게 **평가 점수ｘ중요도**로 계산하여 새로운 변수를 만들 수 있습니다. 같은 계산을 여러 변수에 반복하므로 이번에도 함수로 만들겠습니다. 결측치는 역시나 -99를 반환합니다.

| def rating(data, importance, score): # 함수 정의  if data[importance] == -99: # importance가 -99면  return -99 # -99 리턴  elif data[score] == -99: # score가 -99면  return -99 # -99 리턴  else: # 나머지 경우는  return data[importance] \* data[score] # importance와 score의 곱을 리턴 |
| --- |

매개변수가 3개입니다. 첫 번째 매개변수인 data는 데이터프레임을, importance와 score는 각각 중요도와 평가 변수를 받습니다. 이 함수로 여러 변수들을 계산할 겁니다. 그래서 importance와 score 매개변수를 입력했습니다(그래서 변수 별로 새 함수를 만들 필요 없이 호출할 때 중요도와 평가 항목만 바꿔주면 됩니다). 각각에 들어갈 변수명들을 리스트 형태로 취합하여 for문으로 일괄 계산하겠습니다.

for 문에 적용할 변수 이름을 직접 입력하여 리스트를 만들 수도 있지만, 중요도/평가 항목과 본인/상대방 항목 조합이 데이터프레임 안에 순서대로 나열되어 있어 인덱싱해 리스트를 더 쉽게 만들 수 있습니다. 예를 들어 상대방의 중요도에 대한 변수 이름은 다음과 같이 인덱싱할 수 있습니다.

| data.columns[8:14] # 컬럼의 8자리부터 14이전 자리까지 출력 |
| --- |

Index(['pref\_o\_attractive', 'pref\_o\_sincere', 'pref\_o\_intelligence',

'pref\_o\_funny', 'pref\_o\_ambitious', 'pref\_o\_shared\_interests'],

dtype='object')

보시다시피 pref\_o\_xxx 형태로 된 변수 이름들만 모였습니다. 이 방법을 사용하여 총 4개 카테고리의 변수 이름 리스트를 만들겠습니다.

| partner\_imp = data.columns[8:14] # 상대방의 중요도 partner\_rate\_me = data.columns[14:20] # 본인에 대한 상대방의 평가 my\_imp = data.columns[20:26] # 본인의 중요도 my\_rate\_partner = data.columns[26:32] # 상대방에 대한 본인의 평가 |
| --- |

순서대로 상대방의 중요도, 본인에 대한 상대방의 평가, 본인의 중요도, 상대방에 대한 본인의 평가 변수들이 리스트로 만들어졌습니다.

여기서 하나 더 필요한 것은 계산된 값(평가 점수ｘ중요도)을 받아줄 새 변수의 이름입니다. 이것도 리스트 형태로 직접 새 변수명들을 입력하여 만들겠습니다.

| # 상대방 관련 새 변수 이름을 저장하는 리스트  new\_label\_partner = ['attractive\_p', 'sincere\_partner\_p', 'intelligence\_p', 'funny\_p', 'ambition\_p', 'shared\_interests\_p']  # 본인 관련 새 변수 이름을 저장하는 리스트 new\_label\_me = ['attractive\_m', 'sincere\_partner\_m', 'intelligence\_m', 'funny\_m', 'ambition\_m', 'shared\_interests\_m'] |
| --- |

이제 ‘평가점수 X 중요도’를 계산하면 됩니다. 위 6개 리스트를 한 번에 3가지(새 변수 이름, 중요도, 평가) 리스트를 사용해 계산합니다. 즉, **새 변수 이름 = 중요도 변수 이름 X 평가 변수 이름**이 되어야 하고, 이를 ‘본인 관련 새 변수’에서 한 번, ‘상대방 관련 새 변수’에서 또 한 번 시행해주면 됩니다.

<글상자/>

**for문 in 뒤에 리스트를 2개 이상 사용하기**

zip() 함수를 사용하면 for문에 리스트 3개를 동시에 사용할 수 있습니다. zip() 함수는 가변 인수를 받으므로 2개 이상의 객체(여기서는 리스트)를 받을 수 있습니다. zip()을 사용해 리스트 3개를 순회하면서 원소를 반환하는 for문 예시 코드를 살펴봅시다.

| for i,j,k in zip(new\_label\_partner, partner\_imp, partner\_rate\_me): # 순회  print(i,' & ',j,' & ',k) # 문자 조합 출력 |
| --- |

attractive\_p & pref\_o\_attractive & attractive\_o

sincere\_partner\_p & pref\_o\_sincere & sincere\_o

intelligence\_p & pref\_o\_intelligence & intelligence\_o

funny\_p & pref\_o\_funny & funny\_o

ambition\_p & pref\_o\_ambitious & ambitous\_o

shared\_interests\_p & pref\_o\_shared\_interests & shared\_interests\_o

zip() 안에 들어있는 리스트 3개에서 순서대로 값을 하나씩 뽑아 각각 i, j, k로 반환합니다. 결과물을 보면 첫 번째 줄에 attractive\_p & pref\_o\_attractive & attractive\_o가 하나의 세트로 들어 있습니다.

</>

상대방에 관련 된 변수를 앞서 만든 rating() 함수를 사용하여 ‘중요도 X 평가’를 계산해봅시다.

| for i,j,k in zip(new\_label\_partner, partner\_imp, partner\_rate\_me): # 순회  data[i] = data.apply(lambda x: rating(x, j, k), axis=1) # ❶ rating 함수 적용결과를 i 변수에 저장 |
| --- |

❶ data 전체에 대해 apply()를 활용하여 rating() 함수를 사용했습니다. rating() 함수의 매개변수는 각각 **x = 데이터프레임, j = 중요도 변수 이름, k = 평가 변수**입니다. 계산 결과는 ‘중요도 X 평가’ 변수 이름인 data[i]에 저장합니다. 여기에서 apply 안에 lambda x를 사용하면 해당 변수 i의 한 줄 한 줄의 데이터가 x로 받아져서 rating() 함수에 사용됩니다.

본인 관련된 변수들도 같은 방법으로 계산해줍니다.

| for i,j,k in zip(new\_label\_me, my\_imp, my\_rate\_partner): # 순회  data[i] = data.apply(lambda x: rating(x, j, k), axis=1) # rating 함수 적용결과를 i 변수에 저장 |
| --- |

이제 계산이 필요한 피처 엔지니어링은 모두 끝났습니다. 모델링에 앞서 마지막으로 object 타입 변수들은 문자 형태이기 때문에, 숫자 형태가 되게끔 더미 변수로 바꿔줍니다. 이번에는 gender, race, race\_o 단 3개 변수만이 object 타입입니다.

| data = pd.get\_dummies(data, columns=['gender','race','race\_o'], drop\_first=True) # 더미 변수로 변환 |
| --- |

## 10.5 모델링 및 평가

모델링에 사용할 훈련셋과 시험셋을 분리해주겠습니다.

| from sklearn.model\_selection import train\_test\_split # 임포트 X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.drop('match',axis=1), data['match'], test\_size=0.2, random\_state=100) # 훈련셋/시험셋 분리 |
| --- |

그리고 XGBoost를 xgb라는 이름으로 임포트합니다.

| import xgboost as xgb # 임포트 |
| --- |

XGBoost는 크게 3가지 학습 방법을 제공합니다. 랜덤 포레스트와 마찬가지로 분류와 회귀의 fit() 함수를 제공하고, 추가로 train() 함수도 제공합니다(train() 함수는 11장 ‘LightGBM : 카드 거래 내역 데이터셋’에서 다룹니다). 이 장에서는 분류 함수를 사용하여 모델링합니다.

| model = xgb.XGBClassifier(n\_estimators = 500, max\_depth = 5, random\_state=100) # 모델 객체 생성 |
| --- |

총 3가지 하이퍼파라미터(n\_estimators, max\_depth, random\_state)에 임의의 값을 설정했습니다. XGBoost 모델 객체인 model을 사용하여 기존과 같은 방법으로 훈련과 예측을 차례대로 수행하면 됩니다.

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

XGBClassifier(base\_score=0.5, booster='gbtree', colsample\_bylevel=1,

colsample\_bynode=1, colsample\_bytree=1, enable\_categorical=False,

gamma=0, gpu\_id=-1, importance\_type=None,

interaction\_constraints='', learning\_rate=0.300000012,

max\_delta\_step=0, max\_depth=5, min\_child\_weight=1, missing=nan,

monotone\_constraints='()', n\_estimators=500, n\_jobs=4,

num\_parallel\_tree=1, predictor='auto', random\_state=100,

reg\_alpha=0, reg\_lambda=1, scale\_pos\_weight=1, subsample=1,

tree\_method='exact', validate\_parameters=1, verbosity=None)

학습에 사용된 하이퍼파라미터 정보를 보여줍니다. 임의로 설정한 3개 이외에는 모두 기본값으로 반영됩니다. 학습된 모델을 통해 예측을 하고 정확도를 확인해봅시다.

| pred= model.predict(X\_test) # 예측 from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report # 임포트  accuracy\_score(y\_test, pred) # 정확도 |
| --- |

0.8616236162361623

약 86% 정도의 정확도가 나왔습니다. 수치상으로는 좋아보일 수 있으나 데이터 특성을 고려하면 그렇지 못합니다. 7.2절에서 describe()를 통해 확인한 정보에서 종속 변수 match의 평균값은 0.164입니다. 매칭된 경우가 약 16% 정도라는 겁니다. 나머지 84%는 매칭되지 않았습니다. 즉, 모델링 없이 모든 경우를 0(매칭되지 않았다고)으로만 예측해도 84%는 맞출 수 있는 편향된 데이터입니다. 그래서 86%의 결과는 예측 모델이 없는 것보다 아주 조금 나은 수준입니다.

4.7절 ‘모델링 및 예측하기’에서 배운 confusion\_matrix()를 사용해 정보를 더 구체적으로 살펴봅시다.

| print(confusion\_matrix(y\_test, pred)) # 혼동 행렬 출력 |
| --- |

[[1291 68]

[ 147 114]]

예상대로 실젯값이 0인데 0으로 예측한 경우가 1291건으로 가장 많습니다. 매칭된 경우를 제대로 예측한 경우도 114건으로 무난한 수준입니다. 혼동 행렬에서 1종 오류, 즉 실젯값은 0인데 1로 예측한 경우가 68건, 반대로 2종 오류는 약 147건으로 상당히 많습니다. 모델 간의 비교/평가를 진행할 때는 오류 유형에 따른 평가해야 합니다. 사이킷런에서 제공하는 classification\_report() 함수를 이용하면 이와 관련된 평가 수치를 일목요연하게 확인할 수 있습니다.

| print(classification\_report(y\_test, pred)) |
| --- |

precision recall f1-score support

0 0.90 0.95 0.92 1365

1 0.63 0.44 0.51 261

accuracy 0.87 1626

macro avg 0.76 0.69 0.72 1626

weighted avg 0.85 0.87 0.86 1626

기본적으로 classification\_report() 결과물은 종속 변수의 값인 0과 1 각각에 대하여 한 줄씩 나타납니다. 대부분은 예측하하려는 경우를 1로 두기 때문에 1에 대한 값들을 주로 해석하면 되고, 0에 대한 값들은 필요에 따라 확인하면 됩니다. 이 데이터는 목푯값 중 0의 비율이 87%이기 때문에 정밀도(precision), 재현율(recall), F1-점수(f1-score) 모두 0.9 이상의 높은 값을 보여주므로 중요도가 떨어집니다. 반면 1에 대한 값들은 0.60, 0.42, 0.49 등으로 상대적으로 확연히 낮습니다.

precision, recall, f1-score를 설명하기 위해 혼동 행렬을 다시 한번 살펴보겠습니다.

|  |  | Predicted | |
| --- | --- | --- | --- |
|  |  | 0 | 1 |
| Actual | 0 | 1291 (TN) | 68 (FP) |
| 1 | 147 (FN) | 114 (TP) |

4장에서 혼동 행렬을 설명할 때 TP(양성을 양성으로 판단), FN(양성을 음성으로 판단), FP(음성을 양성으로 판단), TN(음성을 음성으로 판단) 개념도 함께 설명드렸는데, 이 값들로 정밀도와 재현율을 계산합니다.

**정밀도**(precision)는 1로 예측한 경우 중, 얼마만큼이 실제로 1인지를 나타냅니다. 수식은 다음과 같습니다.

= =

FP가 커질수록 분모가 커지기 때문에 정밀도는 낮아집니다. 즉, 1종 오류와 관련됩니다.

**재현율**(recall)은 실제로 1 중에, 얼마만큼을 1로 예측했는지를 나타냅니다. 수식은 다음과 같습니다.

= =

여기서는 FN가 커질수록 recall 값이 작아지므로 Type 2 error와 관련됩니다.

**F1-점수**(f1-score)는 정밀도와 재현율의 조화평균을 의미합니다. 수식은 다음과 같습니다.

=

조화평균값이므로 정밀도와 재현율이 높을 때 당연히 함께 높아지며, 둘의 값이 비슷할수록 더 높은 값을 보여줍니다. 예를 들어 정밀도와 재현율 각각 0.8, 0.6일 때보다 0.7, 0.7일 때 더 높은 값을 나타냅니다.

<용어>

**조화평균**

주어진 수들의 역수의 산술평균의 역수. 주어진 수가 둘일 때 수식은 다음과 같다.



</>

<용어>

**산술평균**

주어진 수의 합을 수의 개수로 나눈 값. 주어진 수가 둘일 때 수식은 다음과 같다.



</>

어떤 오륫값을 살펴봐야 하는지는 분석의 목적에 따라 다릅니다. 1종 오류가 중요하면 정밀도에, 2종 오류가 중요하면 재현율에 더욱 신경 써야 합니다. 특별히 더 중요한 오류 유형이 없다면 F1-점수를 보는 게 무난한 방법입니다. 목푯값이 0과 1로 이루어진 이진분류에 유용하게 사용되는 또 하나의 평가 방법으로 AUC가 있는데, 이는 8장에서 다룹니다.

## 10.6 이해하기 : 경사하강법

하이퍼파라미터 튜닝에 앞서, 경사하강법gradient descent을 먼저 알아보겠습니다. 경사하강법은 머신 러닝이 학습시킬 때 최소의 오차를 찾는 방법입니다. 오차 함수에 대한 경사도(미분계수)를 기준으로 매개변수를 반복적으로 이동해가며 최소 오차를 찾습니다. 여기서 말하는 매개변수는 선형 회귀로 치면 계수(변수에 대한 기울기 값)에 해당합니다. 예를 들어 매개변수(x)에 대한 오차(y) 수식이 다음과 같다고 가정해보겠습니다.

<용어/>

**경사하강법과 보폭**

경사 부스팅의 핵심 개념 중 하나로, 모델이 어떻게 최소 오차가 되는 매개변수들을 학습하는 방법. 오차식에 대한 미분계수를 통해 매개변수의 이동 방향과 보폭을 결정합니다. 보폭은 매개변수를 얼만큼씩 이동할지를 의미합니다. </>

<용어/>

**미분계수**

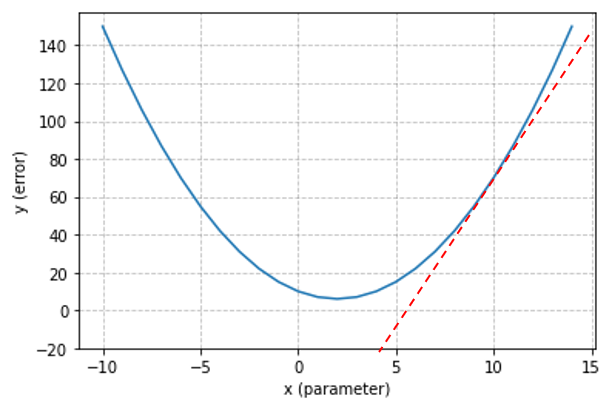
평균변화율에서 x의 증가량을 0으로 가깝게 할 때의 평균변화율을 의미합니다. 기울기, 계수와 같은 의미다.

</>

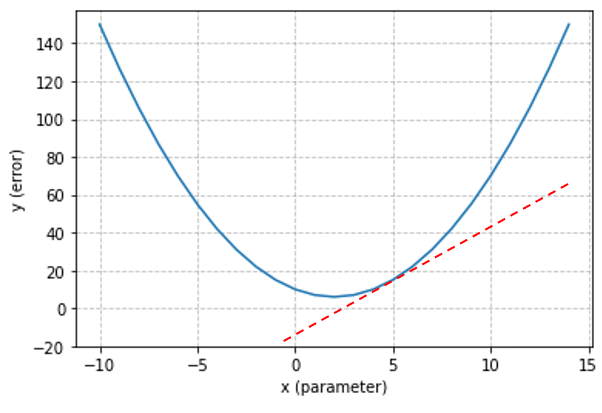
이를 그래프로 그리면 다음과 같습니다.



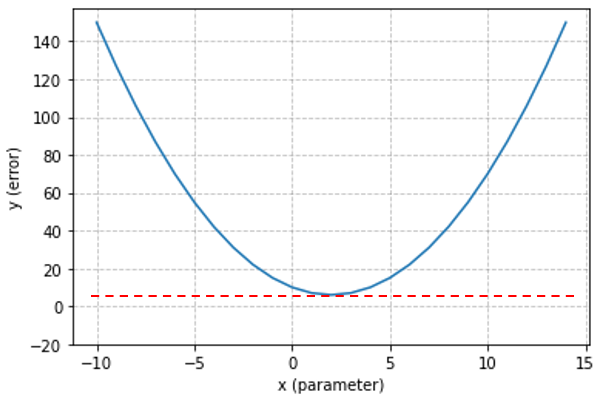
여기서 오차 y의 최솟값은 위 수식을 x에 대해 미분하면 됩니다. 이므로 x가 2일 때 최솟값 6이 나온다는 사실을 쉽게 알 수 있습니다. 그래프에서 미분값은 접선의 기울기를 의미하고, y가 최솟값이 되는 지점에서 기울기는 0입니다. 경사하강법은 임의의 매개변수에서 시작하여 미분값, 즉 오찻값 그래프에서 접선의 기울기를 확인하여 그 기울기가 0인 지점으로 매개변수를 이동시킵니다. 예를 들어 x가 10인 지점에서 미분값을 확인하면 16이고, 이를 그래프에서 접선으로 그려보면 다음과 같습니다.



이 접선의 기울기가 0에 가깝게 되려면 매개변수는 10에서 왼쪽으로 움직여야 합니다. 경사하강법은 이를 미분계수(접선의 기울기)의 부호와 크기로 판단합니다. 이 값이 양수이면 매개변수를 왼쪽으로, 음수이면 오른쪽으로 움직이며, 기울기의 절댓값이 작을수록 더 조금씩 움직입니다. 매개변수 10에서의 미분계수가 16이므로 왼쪽으로 움직이고 절댓값16일 때 움직이는 정도를 5라고 가정하여, 매개변수를 왼쪽으로 5만큼 움직여봅시다. 10에서 왼쪽으로 5만큼 움직였으니 새로 미분계수를 측정할 자리는 매개변수 5가 됩니다. 매개변수가 5인 지점의 미분계수를 확인하면 6이 나옵니다.

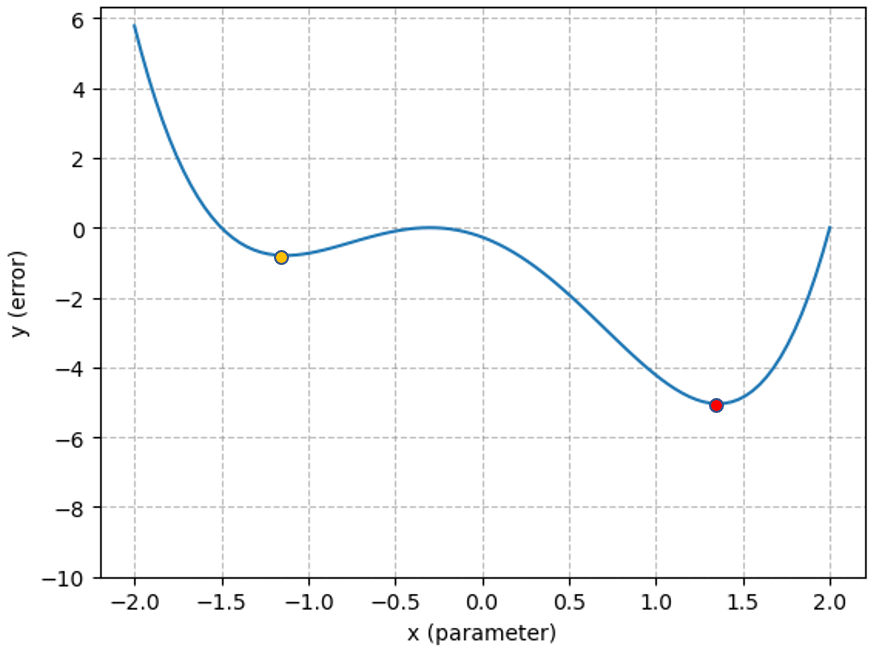


미분계수가 6이므로 기울기가 이전보다 완만해졌으나 아직 0에 가깝지는 않습니다. 미분계수가 16일 때 매개변수를 5만큼 움직였으므로, 미분계수가 6인 지점에서는 더 조금 움직입니다. 이번에는 매개변수를 2만큼 왼쪽으로 움직여서 x값 2에서의 기울기를 확인하면 0, 즉 최솟값의 위치에 다다릅니다.



이는 어디까지나 설명을 위한 단순한 예시 경우이고, 실제로 경사하강법은 정확히 최솟값을 찾아낸다기보다 미분계수가 최대한 0에 가깝도록 계속 움직여서 최솟값의 근삿값에 다다르는 방법입니다. 수식이 복잡할수록 실제 최솟값의 근처에 다다르기 어렵기도 합니다.

가령 다음과 같이 에러에 대한 그래프가 4차 함수의 형식을 띄고 있다면, 경사하강법이 최솟값이라고 여길만한 부분, 즉 미분계수가 0인 지점이 두 곳입니다.



여기서 노란점을 지역 최솟값local minimum이라하고 빨간점을 전역 최솟값global minimum이라고 합니다. 이런 상황에서 학습률이 충분히 크지 않다면 지역 최솟값을 최솟값으로 판단할 수 있으니, 학습률을 충분한 크기로 주어 지역 최솟값을 지나 전역 최솟값으로 향할 수 있게 해야 합니다.

여기에 보여진 예시에는 매개변수에 대한 오차 수식이 아주 간단하기 때문에 미분을 하면 쉽게 기울기가 0이되는 지점을 찾을 수 있지만, 수많은 변수를 가지고 있는 실제 데이터에서는 이러한 방법으로 최소 오차를 찾기 매우 어렵습니다. 따라서 경사하강법을 사용하는 것이 훨씬 효율적입니다.

## 10.7 하이퍼파라미터 튜닝 : 그리드 서치

이번 장에서는 그리드 서치를 활용한 하이퍼파라미터 튜닝을 배워보겠습니다. 지금까지 소개한 하이퍼파라미터 튜닝은 임의의 값들을 넣어 더 나은 결과를 찾는 방식이었습니다. 이런 식으로 하나하나 확인하면 모델링 과정을 기다리고 재시도하는 단순 작업을 반복해야 합니다. 그리드 서치를 이용하면 한 번 시도로 수백 가지 하이퍼파라미터값을 시도해볼 수 있습니다.

그리드 서치의 원리는 간단합니다. 그리드 서치에 입력할 하이퍼파라미터 후보들을 입력하면, 각 조합에 대해 모두 모델링해보고 최적의 결과가 나오는 하이퍼파라미터 조합을 알려줍니다. 예를 들어 하이퍼파라미터로 max\_depth와 learning\_rate를 사용한다고 가정합시다. 다음과 같이 하이퍼파라미터별로 다양한 값들을 지정해줍니다.

<코드/>

max\_depth = [3, 5, 10]

learaning\_rate = [0.01, 0.05, 0.1]

</>

이를 그리드 서치로 적용하면 다음과 같이 9가지 조합이 만들어집니다.

|  | max\_depth | learning\_rate |
| --- | --- | --- |
| 조합 1 | 3 | 0.01 |
| 조합 2 | 5 | 0.01 |
| 조합 3 | 10 | 0.01 |
| 조합 4 | 3 | 0.05 |
| 조합 5 | 5 | 0.05 |
| 조합 6 | 10 | 0.05 |
| 조합 7 | 3 | 0.1 |
| 조합 8 | 5 | 0.1 |
| 조합 9 | 10 | 0.1 |

이렇게 9가지 조합을 각각 모델링하게 되는데, 보통 그리드 서치에서는 6장에서 배운 교차검증도 함께 사용하기 때문에 교차검증의 횟수만큼 곱해진 횟수가 모델링됩니다. 예를 들어 위의 9가지 조합에 K-폴드값을 5로 교차검증을 한다면 9 \* 5 = 45회의 모델링을 진행합니다. 모델링 결과가 좋은 매개변수 조합을 알려주고, 해당 하이퍼파라미터셋으로 예측(predict)까지 지원합니다.

더 많은 하이퍼파라미터 종류와 더 많은 후보를 넣으면 더 좋은 결과를 얻을 수 있는 가능성도 높아지지만, 그만큼 학습 시간이 길어질 수 있습니다. 따라서 소요 시간을 고려하여 적정 수준으로 설정해주는 게 좋습니다.

그리드 서치 모듈은 사이킷런 라이브러리의 model\_selection에서 불러올 수 있습니다.

| from sklearn.model\_selection import GridSearchCV # 임포트 |
| --- |

그리드 서치에 넣어줄 매개변수 4개를 딕셔너리 형태로 입력하면 됩니다.

| parameters = {  'learning\_rate': [0.01, 0.1, 0.3],  'max\_depth': [5,7,10],  'subsample': [0.5,0.7,1],   'n\_estimators': [300, 500, 1000]  } # 하이퍼파라미터 셋 정의 |
| --- |

각 매개변수의 의미는 다음과 같습니다.

* learning\_rate : 경사하강법에서 ‘매개변수’를 얼만큼씩 이동해가면서 최소 오차를 찾을지, 그 보폭의 크기를 결정하는 하이퍼파라미터입니다. 기본적으로 보폭은 미분계수에 의해 결정되지만, learning\_rate를 크게 하면 더 큰 보폭을, 작게 하면 그만큼 작은 보폭으로 움직입니다. learning rate를 우리말로 학습률이라고 합니다. 학습률크기에 따라 최적의 에러를 찾는 과정은 다음과 그림과 같습니다.  
  <note/>  
  **학습률과 보폭  
  학습률은 입력, 보폭은 그 결과입니다. 큰 학습률을 사용하면 결과적으로 보폭도 커집니다.**   
  </>



왼쪽 그래프는 너무 작은 학습률을 쓴 경우입니다. 보폭이 너무 조금씩 움직이다보니 최소 에러값을 찾는 데 상당한 시간이 들고 지역 최소해에 빠질 가능성도 상대적으로 커집니다. 반대로 오른쪽 그래프는 너무 큰 학습률을 사용해서 너무 성큼 움직이기 때문에 최소 에러를 정확히 찾지 못하고 좌우로 계속 넘어다닙니다. 가운데의 그래프가 가장 이상적입니다. 이처럼 적절한 크기의 학습률을 사용해야 큰 시간을 들이지 않고 최소 오차 지점을 찾아낼 수 있습니다.

* max\_depth : 각 트리의 깊이를 제한합니다.
* subsample : 모델을 학습시킬 때 일부 데이터만 사용하여 각 트리를 만듭니다. 0.5를 쓰면 데이터의 절반씩만 랜덤 추출하여 트리를 만듭니다. 이 또한 오버피팅을 방지하는 데 도움이 됩니다.
* n\_estimators : 전체 나무의 개수를 정합니다.

위와 같이 매개변수를 딕셔너리 형태로 설정해준 뒤에는, 알고리즘 속성이 부여된 빈 모델을 만듭니다.

| model = xgb.XGBClassifier() # 모델 객체 생성 |
| --- |

이제 그리드 서치를 수행할 준비를 모두 갖춰졌으니, 모델을 생성할 GridSearchCV() 함수를 다음과 같이 설정해줍니다. 모델(model)과 딕셔너리형의 하이퍼파라미터셋(parameters)은 필수, 그외는 선택 사항입니다.

| gs\_model = GridSearchCV(model, parameters, n\_jobs=-1, scoring='f1', cv = 5) # 그리드서치 객체 생성 |
| --- |

n\_jobs는 사용할 코어 수이고, scoring은 모델링할 때 어떤 기준으로 최적의 모델을 평가할지를 정의합니다. 여기서는 F1-점수를 기준으로 판단하도록 설정했습니다. 그리고 cv는 교차검증에 사용할 K-폴드값입니다. 여기서는 5로 설정했습니다. 이제 gs\_model을 가지고 학습을 시키겠습니다. 학습시키는 코드는 기존의 모델링과 같습니다.

| gs\_model.fit(X\_train, y\_train) # 학습 |
| --- |

이번에는 학습 완료까지 상당한 시간이 소요될 겁니다. 하이퍼파라미터셋이 총 4종류에 3개씩 값이 있으니 으로 총 81번의 모델링을 진행하고 나서 교차검증 5회를 실행하므로 총 405번의 모델링 작업이 이뤄집니다.

그리드 서치는 학습이 완료된 후, 가장 좋은 성능을 보인 하이퍼파라미터 조합 정보를 보관합니다. 아래 코드로 최적의 조합을 확인할 수 있습니다.

| gs\_model.best\_params\_ # 최적의 하이퍼파라미터 출력 |
| --- |

{'learning\_rate': 0.3, 'max\_depth': 5, 'n\_estimators': 1000, 'subsample': 0.5}

또한 이전의 모델링 과정과 동일하게, 그리드 서치를 이용해서 새로운 데이터를 예측할 수도 있습니다. 이때 적용되는 모델은 최적의 하이퍼파라미터 조합이 자동으로 반영됩니다.

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

그럼 예측된 값에 대한 정확도(accuracy\_score)와 분류 리포트(classification\_report)를 확인하겠습니다.

| accuracy\_score(y\_test, pred) # 정확도 계산 |
| --- |

0.8634686346863468

| print(classification\_report(y\_test, pred)) # classification report 출력 |
| --- |

precision recall f1-score support

0 0.90 0.95 0.92 1365

1 0.60 0.44 0.51 261

accuracy 0.86 1626

macro avg 0.75 0.69 0.71 1626

weighted avg 0.85 0.86 0.85 1626

정확도는 아주 미세하게 올라갔고, F1-점수는 0.02 상승했습니다. 일반적으로 하이퍼파라미터 튜닝으로 엄청난 개선을 얻기는 쉽지 않습니다. 예측에는 피처 엔지니어링과 모델 알고리즘 선정이 큰 영향을 미칩니다. 하이퍼파라미터 튜닝은 조금이라도 더 나은 모델을 만드는 역할입니다.

## 10.8 중요 변수 확인

선형 회귀과 로지스틱 회귀에서는 계수로, 결정 트리에서는 노드의 순서로 변수의 영향력을 확인했습니다. 부스팅 모델은 이전 모델들보다 훨씬 복잡한 알고리즘이기 때문에 단순하게 변수의 영향력을 파악하기는 어렵지만, XGBoost에 내장된 함수는 변수의 중요도까지 계산해줍니다. 단, 그리드 서치로 학습된 모델에서는 이 기능을 사용할 수 없으니 그리드 서치에서 찾은 최적의 하이퍼파라미터 조합으로 다시 한번 학습을 시키겠습니다.

| model = xgb.XGBClassifier(learning\_rate = 0.3, max\_depth = 5, n\_estimators = 1000, subsample = 0.5, random\_state=100) # 모델 객체 생성  model.fit(X\_train, y\_train) # 학습 |
| --- |

학습이 완료되었으면 feature\_importances\_에서 변수 중요도(피처 중요도feature importance라고도 합니다)를 확인할 수 있습니다.

| model.feature\_importances\_ # 변수 중요도 확인 |
| --- |

array([0.02178125, 0.01137641, 0.00998134, 0.00984809, 0.01060789,

0.01359383, 0.01065769, 0.01713987, 0.01186322, 0.01206678,

0.01331671, 0.04854793, 0.01233603, 0.01430371, 0.02732428,

0.01440853, 0.02333124, 0.01437325, 0.01104851, 0.01472255,

0.00973702, 0.01481565, 0.01001215, 0.02409385, 0.01538233,

0.01472 , 0.02749152, 0.01478216, 0.01670052, 0.01100918,

0.0106856 , 0.02170104, 0.04928579, 0.01951623, 0.03824322,

0.01167233, 0.01354653, 0.01223037, 0.01456301, 0.0113123 ,

0.01188447, 0.01298039, 0.01511254, 0.01037562, 0.01001647,

0.01314105, 0.01188815, 0.01242248, 0.01119815, 0.01119024,

0.0116582 , 0.00895496, 0.01707342, 0.01282681, 0.03765631,

0.02832991, 0.03117039, 0.01136704, 0.02328539, 0.0133381 ],

dtype=float32)

결괏값은 넘파이 형태이기 때문에 변수 이름 없이 순서대로 숫자만 나열됩니다. X\_train의 변수 이름을 사용하여 feature\_imp라는 이름의 데이터프레임으로 만들어 변수 이름과 매칭시키겠습니다.

| feature\_imp = pd.DataFrame({'features': X\_train.columns, 'values': model.feature\_importances\_}) # 데이터프레임으로 변환 |
| --- |

feature\_imp에 head()를 호출해 제대로 만들어졌는지 확인해봅시다.

| feature\_imp.head() # 상위 5줄 출력 |
| --- |

|  | features | values |
| --- | --- | --- |
| 0 | has\_null | 0.021781 |
| 1 | age | 0.011376 |
| 2 | age\_o | 0.009981 |
| 3 | importance\_same\_race | 0.009848 |
| 4 | importance\_same\_religion | 0.010608 |

변수 중요도를 내림차순으로 정리하여확인해봅시다. barplot로 바 그래프를 그려 상위 10개의 중요 변수를 확인하겠습니다.

| plt.figure(figsize=(20, 10)) # 그래프 크기 설정 sns.barplot(x='values', y='features', # ❶  data=feature\_imp.sort\_values(by='values', ascending=False).head(10)) # ❷ |
| --- |



❶ x축에는 values(중요도), y축에는 features 컬럼을 설정했습니다. ❷ sort\_values()를 사용해 컬럼 ‘values’를 기준으로 내림차순 정렬합니다. 그 뒤에 head()를 이어 붙여서 상위 10개 항목만 불러 그래프를 출력합니다.

선형 회귀와 로지스틱 회귀에서는 계수 부호와 크기로 직관적으로 이해할 수 있습니다. 결정 트리에서 각 노드는 어떤 변수에서 어떤 값 기준으로 다음 노드를 나누는지 보여주기 때문에 역시 직관적으로 이해할 수 있습니다. 반면 여기서 나타나는 values는 상대적인 중요도를 계산한 값이기 때문에 직관적으로 이해하기가 어렵습니다. 또한 중요하다고 나타난 변수들이 긍정적인 영향인지 부정적인 영향인지까지는 보여주지 않으므로 해석에 유의할 필요가 있습니다.

## 10.9 이해하기 : XGBoost

XGBoost를 얘기하기에 앞서 트리 모델의 진화 과정을 간략하게 알아보겠습니다.

<그림/>



</>

결정 트리와 랜덤 포레스트는 지난 장에서 다뤘습니다. 배깅bagging, 부스팅boosting, 경사 부스팅gradient boosting은 이 책에서 선정한 톱 10 알고리즘에 포함되지 않았으니 간략하게 소개하겠습니다.

### 10.9.1 배깅

배깅은 부트스트랩bootstrap 훈련셋을 사용하는 트리 모델입니다. 부트스트랩은 데이터의 일부분을 무작위로 반복 추출하는 방법입니다. 이러한 식으로 추출한 데이터의 여러 부분집합을 사용해 여러 트리를 만들어 오버피팅을 방지합니다. 이 방법은 랜덤 포레스트에도 포함된 내용입니다. 랜덤 포레스트는 배깅에서 한 단계 더 발전된 모델입니다.

▽ 배깅 학습 방법



### 10.9.2 부스팅과 에이다부스트

부스팅Boosting은 랜덤 포레스트에서 한 단계 더 발전한 방법으로 역시 여러 트리를 만드는 모델입니다. 가장 큰 차이점은 랜덤 포레스트에서 각 트리는 독립적이나, 부스팅에서는 그렇지 않다는 겁니다. 다시 말해 랜덤 포레스트에서는 각 트리를 만들 때 이전에 만든 트리와 상관없이 새로운 데이터 부분집합과 변수 부분집합을 이용합니다. 반면 부스팅은 각 트리를 순차적으로 만들면서 이전 트리의 정보를 이용합니다. 부분집합을 이용해 첫 트리를 만들고 난 후, 해당 트리의 예측 결과를 반영하여 두 번째 트리를 만들어서 첫 번째 트리와의 시너지 효과를 키웁니다. 트리를 계속하여 만들 때마다 이런 식으로 이전 트리의 정보를 이용한다는 점이 랜덤 포레스트와 다릅니다.

부스팅의 대표 알고리즘인 에이다부스트AdaBoost(Adaptive Boosting)는 단계적으로 트리를 만들 때 이전 단계에서의 분류 결과에 따라 각 데이터에 가중치를 부여/수정합니다. 예를 들어 이전 트리에서 가중치가 덜 부여되고 잘못 분류된 데이터들에 더 높은 가중치를 부여하고, 후속 트리에서는 가중치가 높은 데이터를 분류하는 데 우선순위를 둡니다. 이러한 방식으로 트리 여러 개를 만들면 분류가 복잡한 데이터셋도 세부적으로 나눌 수 있는 모델이 만들어집니다.

▽ 에이다부스트 학습 방법



### 10.9.3 경사 부스팅과 XGBoost

에이다부스트는 각 데이터에 가중치를 부여/수정하는 방식으로 트리를 만듭니다. 반면 경사 부스팅gradient boosting은 경사하강법을 이용하여, 이전 모델의 에러를 기반으로 다음 트리를 만들어갑니다. 경사 부스팅으로 구현한 알고리즘으로 XGBoost, LightGBM, 캣부스트Catboost 등이 있습니다.

▽ 경사 부스팅 학습 방법



XGBoost는 Extreme Gradient Boosting을 줄인 것으로, XGBoost가 기존의 경사 부스팅보다 특별한 이유는 계산 성능 최적화와 알고리즘 개선을 함께 이루었기 때문입니다. 우선 계산 성능 최적화에 있어서, XGBoost 이전의 부스팅 모델은 트리를 순차적으로 만들어내기 때문에 모델링 속도가 느립니다. XGBoost도 마찬가지로 순차적으로 트리를 만들지만, 병렬화Parallelization, 분산 컴퓨팅Distributed Computing, 캐시 최적화Cache Optimization / Cache-Aware Access 등을 활용해 계산 속도가 훨씬 빠릅니다.

또한 알고리즘도 개선해 경사하강법보다 더 발전된 형태로 최솟값을 찾아냅니다. 기존 경사하강법이 접점의 기울기를 계산하고 매개변수를 이동한 반면, XGBoost에서는 2차 도함수(2번 미분한 함수)를 활용하여 더 많은 정보를 수집하고, 이를 활용해 더 적절한 이동 방향과 이동 크기를 찾아내어 더 빠른 시간에 전역 최솟값에 도달합니다.

또 한 가지 중요한 개선 사항은 정규화 하이퍼파라미터regularization hyperparameter 지원입니다. 트리 모델이 진화할수록 더 좋은 예측 성능을 보이는 동시에 반대급부로 오버피팅 문제가 더 심각해질 수 있습니다. XGBoost는 이러한 부작용을 줄일 목적으로 LASSO(L1)와 Ridge(L2) 정규화 하이퍼파라미터를 지원합니다(더 구체적인 설명은 11.6절 ‘하이퍼파라미터 튜닝’ 참조). 그 밖에도 애매하게 예측된 관측치에 높은 가중치를 부여하는 가중치 분위수 스케치Weighted Quantile Sketch, 결측치를 유연하게 처리해내는 희소성 인식Sparsity Aware 등을 포함하여 성능을 개선시켰습니다.

<용어/>

가중치 분위수 스케치 : 최적의 분할을 찾기 위해 각 변수에 대한 히스토그램을 만듭니다. 히스토그램의 기둥의 경계는 최상의 분할지점을 찾기 위한 후보로 사용되는데, 가중 분위수 스케치에서는 각 기둥이 동일한 가중치를 갖도록 만들어집니다.

</>

그리드 서치에 다른 하이퍼파라미터값들을 넣어 더 나은 예측을 보이는 조합을 찾을 수 있는지 시도해보세요.

## 학습 마무리

#### 되짚어보기

10.1 스피드데이팅 데이터셋으로 남여의 커플 성사를 예측하는 문제입니다.

10.2 판다스, 넘파이, 맷플롯립, 시본 라이브러리를 임포트했습니다. 프로젝트에 쓸 예제 데이터셋을 불러옵니다.

10.3 변수 특성을 고려하여, 결측치가 있는 변수 일부를 데이터에서 제거했습니다.

10.4 나이, 인종, ‘중요하게 생각하는 특성’ 등을 반영하는 추가 변수를 만들었습니다.

10.5 XGBoost로 모델링한 결과 86% 정확도를 얻었으나, F1-점수는 0.49로 낮았습니다.

10.7 그리드 서치를 사용해 최적의 하이퍼파라미터 조합을 찾아 적용하니 F1-점수가 아주 조금 상승했습니다.

10.8 어떤 변수가 더 중요하게 작용했는지를 확인해보았습니다(영향도의 방향성은 알 수 없었습니다).



#### 과제

피처 엔지니어링에는 정답이 없습니다. 여기에 제시된 방법 이외에, 여러분만의 아이디어로 변수들을 다양하게 조합해보고 모델링하여 결과를 확인해봅시다.

#### 핵심 용어 정리

1. **XGBoost** : 부스팅 기법의 하나로, 각 트리가 독립적인 랜덤 포레스트와 달리 이전 트리를 기반으로 새로운 트리를 생성합니다.
2. **부스팅 알고리즘** : 부스팅은 랜덤 포레스트에서 그 다음 세대로 진화하게 되는 중요한 개념입니다. 랜덤 포레스트에서는 각각의 트리를 독립적으로, 즉 서로 관련 없이 만드는 반면, 부스팅 알고리즘에서는 트리를 순차적으로 만들면서 이전 트리에서 학습한 내용이 다음 트리를 만들 때 반영됩니다.
3. **경사하강법** : 경사 부스팅의 핵심 개념 중 하나로, 모델이 어떻게 최소 오차가 되는 매개변수들을 학습하는지에 대한 방법론입니다. 오차식에 대한 미분계수를 통해 매개변수의 이동 방향과 보폭을 결정합니다. 보폭은 러닝 레이트라는 하이퍼파라미터로 조절할 수 있습니다.

#### 새로운 함수와 라이브러리

* **zip()** : 둘 이상의 데이터(예를 들어 리스트 두개)의 각 요소들을 순서대로 짝을 지어 한쌍의 튜플로 묶어냅니다.
* **abs()** : 절댓값을 보여줍니다.
* **XGB모델.feature\_importances\_** : 학습된 xg부스트 모델에서 변수의 중요도를 보여줍니다.

#### 

## 연습문제

1. 아래의 코드를 실행했을 때 예상되는 출력물은?

| a = ['a','b','c'] b = [0,1,2]  for i, j in zip(a,b):  print(i, j) |
| --- |

① a 0

b 1

c 2

② 0 a

1 b

2 c

③ 0 1 2

a b c

④ a b c

0 1 2

2. 다음 정밀도, 재현율, F1-점수에 대한 설명 중 옳지 않은 것은?

① 정밀도는 1종 오류와 관련이 있다.

② 2종 오류가 중요할 때는 재현율을 더욱 고려해야 한다.

③ 1종 오류와 2종 오류를 동등한 수준으로 고려해야 한다면 F1-점수를 확인하는 것이 좋다.

④ F1-점수는 정밀도와 재현율의 산술평균이다.

3. 다음 XGBoost의 하이퍼파라미터 중, 경사하강법의 보폭을 결정하는 것은?

① learning\_rate

② max\_depth

③ subsample

④ n\_estimators

4. 다음 XGBoost의 설명 중 옳지 않은 것은?

① 부스팅 모델의 일종으로, 트리모델을 기반으로 한다.

② 랜덤 포레스트와 동일하게 여러 개의 트리를 사용하는 알고리즘이다.

③ 랜덤 포레스트와 동일하게 각각의 트리는 서로 완전히 독립적이다.

④ 모델링 후 변수의 중요도를 확인할 수 있다.

#### 정답 및 해설

1. 1

① a 0

b 1

c 2

2. 4

④ F1-점수는 정밀도와 재현율의 산술평균이다. ← 산술평균이 아닌 조화평균으로 계산됩니다.

3. 1

① learning\_rate

② max\_depth ← 트리의 깊이를 결정합니다.

③ subsample ← 서브 샘플의 비율을 결정합니다.

④ n\_estimators ← 사용할 트리 개수를 결정합니다.

4. 3

③ 랜덤 포레스트와 동일하게, 각각의 트리는 서로 완전히 독립적이다. ← XGBoost는 랜덤 포레스트와는 달리 이전의 트리를 기반으로 다음 트리를 만들어냅니다.

1. 가령 -99로 결측치를 대체한다면, 선형모델에서는 해당 숫자가 아웃라이어로써 작용하겠지만 트리 모델에서는 결측치라는 사실 자체가 유의미한 차이를 보인다면 -99를 분류하는 노드가 생겨날 겁니다. [↑](#footnote-ref-0)