데이터문제해결및실습1

7주차-2
Chapter_06_[경진대회1] 자전거 대여 수요 예측_2

세종대학교
인공지능데이터사이언스학과
박동현 교수

- 본 강의는 골든래빗 출판사의 머신러닝/딥러닝 문제해결전략이 제공하는 강의 교안에 기반함.

Chapter

06

[경진대회]
자전거 대여 수요 예측 2

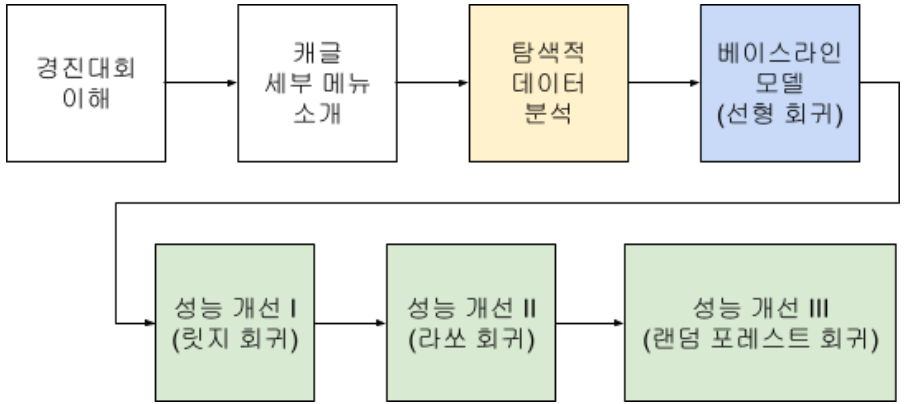
[경진대회] 자전거 대여 수요 예측

난이도	★☆☆		
경진대회명	자전거 대여 수요 예측 경진대회		
미션	날짜, 계절, 근무일 여부, 날씨, 온도, 체감 온도, 풍속 데이터를 활용하여 자전거 대여 수량 예측		
문제 유형	회귀	**평가지표**	RMSLE
데이터 크기	1.1MB	**참가팀 수**	3,242팀
제출 시 사용한 모델	랜덤 포레스트 회귀		
파이썬 버전	3.7.10		
사용 라이브러리 및 버전	• numpy (numpy==1.19.5) • pandas (pandas==1.3.2) • seaborn (seaborn==0.11.2) • matplotlib (matplotlib==3.4.3) • sklearn (scikit-learn==0.23.2) • datetime, calendar		

[경진대회] 자전거 대여 수요 예측

▪ 학습 목표

- 머신러닝 모델링 프로세스를 배워봅시다!
- 기본 회귀 모델을 배우게 됩니다.
- 경진대회 세부 메뉴도 알아봅니다.



▪ 학습 키워드

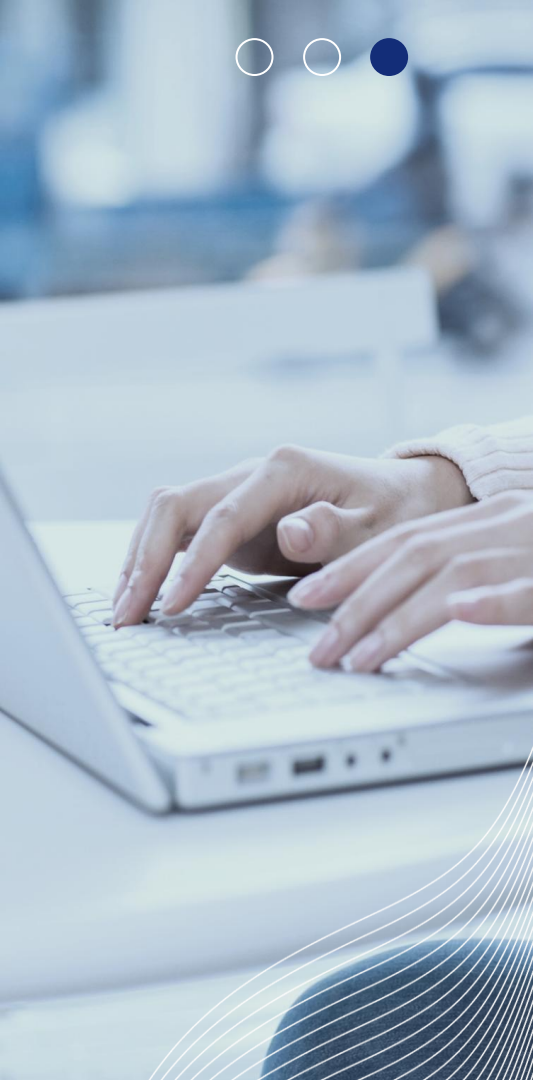
- 유형 및 평가지표 : 회귀, RMSLE
- 탐색적 데이터 분석 : 분포도, 막대 그래프, 박스플롯, 포인트플롯, 산점도, 히트맵
- 머신러닝 모델 : 선형 회귀, 릿지 회귀, 라쏘 회귀, 랜덤 포레스트 회귀
- 피처 엔지니어링 : 파생 피처 추가, 피처 제거
- 하이퍼파라미터 최적화 : 그리드서치

6.3 탐색적 데이터 분석

▪ 분석 정리 및 모델링 전략

- 분석 정리

1. **타깃값 변환** : 타깃값을 count가 아닌 log(count)로 변환해 사용(마지막에 다시 지수변환해 count로 복원해야 함)
2. **파생 피처 추가** : datetime 피처를 분리해 year, month, day, hour, minute, second 피처 생성 가능
3. **파생 피처 추가** : datetime에 숨어 있는 또 다른 정보인 요일(weekday) 피처 추가
4. **피처 제거** : 테스트 데이터에 없는 피처는 훈련에 사용해도 의미 없음. 따라서 훈련 데이터에만 있는 casual, registered 피처 제거
5. **피처 제거** : datetime 피처는 인덱스 역할만 하므로 타깃값 예측에 아무런 도움이 되지 않음
6. **피처 제거** : date 피처가 제공하는 정보는 year, month, day 피처에 담겨 있음
7. **피처 제거** : month는 season 피처의 세부 분류로 볼 수 있어 month 피처 제거
8. **피처 제거** : 막대 그래프 확인 결과, 파생 피처인 day는 분별력이 없음
9. **피처 제거** : 막대 그래프 확인 결과, 파생 피처인 minute와 second에는 아무런 정보가 담겨 있지 않음
10. **이상치 제거** : 포인트 플롯 확인 결과, weather가 4인 데이터는 이상치
11. **피처 제거** : 산점도와 히트맵 확인 결과, windspeed 피처에는 결측값이 많고 대여 수량과의 상관관계가 매우 약함



6.1 경진대회 이해

6.2 경진대회 접속 방법 및 세부 메뉴

6.3 탐색적 데이터 분석

6.4 베이스라인 모델

6.5 성능 개선 I : 릿지 회귀 모델

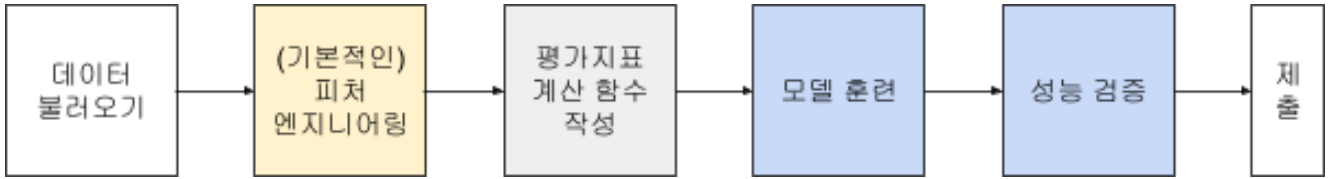
6.6 성능 개선 II : 라쏘 회귀 모델

6.7 성능 개선 III : 랜덤 포레스트 회귀 모델

6.4 베이스라인 모델

▪ 베이스라인 모델

- 베이스라인 모델baseline model이란 뼈대가 되는 가장 기본적인 모델을 의미
- 베이스라인에서 출발해 성능을 점차 향상하는 방향으로 모델링
- 사이킷런이 제공하는 기본 선형 회귀 모델을 베이스라인으로 사용할 계획
- 전체 프로세스는 다음과 같음



6.4 베이스라인 모델

6.4.1 피처 엔지니어링

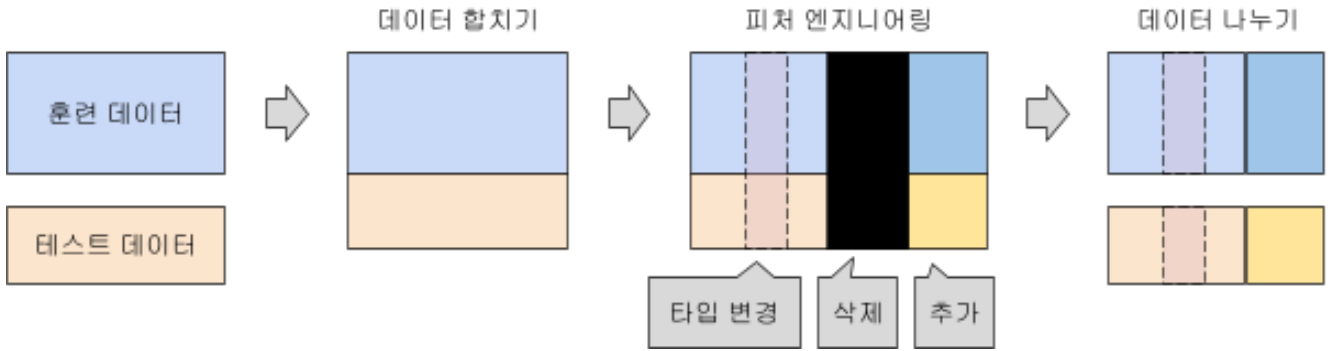
- 피처 엔지니어링은 데이터를 변환하는 작업
- 먼저, 노트북 양식 복사 후 데이터 불러오기

```
import pandas as pd
# 데이터 경로
data_path = '/kaggle/input/bike-sharing-demand/'

train = pd.read_csv(data_path + 'train.csv')
test = pd.read_csv(data_path + 'test.csv')
submission = pd.read_csv(data_path + 'sampleSubmission.csv')
```

6.4 베이스라인 모델

▪ 6.4.1 피처 엔지니어링

- 피처 엔지니어링은 훈련, 테스트 데이터에 모두 적용해야 함
- 따라서 피처 엔지니어링 전에 두 데이터를 합쳤다가 다 끝나면 도로 나눠줘야 함



6.4 베이스라인 모델

6.4.1 피처 엔지니어링

- 이상치 제거
  - 데이터 합치기 전에 이상치 먼저 제거
  - weather가 4인 데이터(폭우, 폭설이 내리는 날 저녁 6시에 대여) 제거(분석 정리 10)

```
# 훈련 데이터에서 weather가 4가 아닌 데이터만 추출
train = train[train['weather'] != 4]
```

6.4 베이스라인 모델

▪ 6.4.1 피처 엔지니어링

• 데이터 합치기

▪ 훈련 데이터와 테스트 데이터 같은 피처 엔지니어링을 적용하기 위해 두 데이터를 하나로 합침

```
all_data = pd.concat([train, test], ignore_index=True)
all_data
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
❶ 0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0000	3.0	13.0	16.0
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0000	8.0	32.0	40.0
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0000	5.0	27.0	32.0
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0000	3.0	10.0	13.0
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0000	0.0	1.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	❷ ...
17373	2012-12-31 19:00:00	1	0	1	2	10.66	12.880	60	11.0014	NaN	NaN	NaN
17374	2012-12-31 20:00:00	1	0	1	2	10.66	12.880	60	11.0014	NaN	NaN	NaN
17375	2012-12-31 21:00:00	1	0	1	1	10.66	12.880	60	11.0014	NaN	NaN	NaN
17376	2012-12-31 22:00:00	1	0	1	1	10.66	13.635	56	8.9981	NaN	NaN	NaN
17377	2012-12-31 23:00:00	1	0	1	1	10.66	13.635	65	8.9981	NaN	NaN	NaN

17378 rows × 12 columns

6.4 베이스라인 모델

6.4.1 피처 엔지니어링

- 파생 피처(변수) 추가
  - 6.3.3에서 다룬 피처 엔지니어링을 비슷한 방식으로 적용(분석 정리 2, 3)

```
from datetime import datetime

# 날짜 피처 생성
all_data['date'] = all_data['datetime'].apply(lambda x: x.split()[0])
# 연도 피처 생성
all_data['year'] = all_data['datetime'].apply(lambda x: x.split()[0].split('-')[0])
# 월 피처 생성
all_data['month'] = all_data['datetime'].apply(lambda x: x.split()[0].split('-')[1])
# 시 피처 생성
all_data['hour'] = all_data['datetime'].apply(lambda x: x.split()[1].split(':')[0])
# 요일 피처 생성
all_data["weekday"] = all_data['date'].apply(lambda dateString :
                                             datetime.strptime(dateString,"%Y-%m-%d").weekday())
```

  - day, minute, second 피처는 생성하지 않음(분석 정리 8, 9)

6.4 베이스라인 모델

6.4.1 피처 엔지니어링

- 필요 없는 피처 제거
  - casual과 registered 피처는 테스트 데이터에 없으므로 제거(분석 정리 4)
  - datetime 피처는 인덱스 역할이고, date 피처가 갖는 정보는 다른 피처들(year, month, day)에도 담겨 있기 때문에 datetime과 date 피처도 필요 없음(분석 정리 5, 6)
  - season 피처가 month 피처의 대분류 성격이라 month 피처도 제거(분석 정리 7)
  - windspeed 피처도 타깃값과 상관관계가 약해서 제거(분석 정리 11)

```
drop_features = ['casual', 'registered', 'datetime', 'date', 'windspeed', 'month']

all_data = all_data.drop(drop_features, axis=1)
```

  - 탐색적 데이터 분석에서 얻은 인사이트를 활용해 의미 있는 피처와 불필요한 피처를 구분(피처 선택)
    - 피처가 많다고 무조건 좋은 게 아님
    - 예측 성능을 높이려면 타깃값과 관련 있는 피처가 필요
    - 탐색적 데이터 분석, 피처 중요도, 상관관계 매트릭스, 배경 지식을 종합적으로 활용해 판단해야 함

6.4 베이스라인 모델

6.4.1 피처 엔지니어링

- 데이터 나누기
  - 모든 피처 엔지니어링을 끝냈으므로 훈련 데이터와 테스트 데이터를 다시 나눔
    - 타깃값이 있으면 훈련 데이터, 없으면 테스트 데이터

```
# 훈련 데이터와 테스트 데이터 나누기
X_train = all_data[~pd.isnull(all_data['count'])]
X_test = all_data[pd.isnull(all_data['count'])]

# 타깃값 count 제거
X_train = X_train.drop(['count'], axis=1)
X_test = X_test.drop(['count'], axis=1)

y = train['count'] # 타깃값
```

	season	holiday	workingday	weather	temp	atemp	humidity	year	hour	weekday
0	1	0	0	1	9.84	14.395	81	2011	00	5
1	1	0	0	1	9.02	13.635	80	2011	01	5
2	1	0	0	1	9.02	13.635	80	2011	02	5

6.4 베이스라인 모델

▪ 6.4.2 평가지표 계산 함수 작성

- 평가지표
  - 훈련이 제대로 됐는지 확인하려면 능력을 평가할 수단, 즉 평가지표가 필요
  - 본격적인 훈련에 앞서 본 경진대회 평가지표인 RMSLE 계산 함수 생성

```
import numpy as np

def rmsle(y_true, y_pred, convertExp=True):
    # 지수변환
    if convertExp:
        y_true = np.exp(y_true)
        y_pred = np.exp(y_pred)

    # 로그변환 후 결측값을 0으로 변환
    log_true = np.nan_to_num(np.log(y_true+1))
    log_pred = np.nan_to_num(np.log(y_pred+1))

    # RMSLE RM계산
    output = np.sqrt(np.mean((log_true - log_pred)**2))
    return output
```

타깃값으로 log(count)를 사용했으니
여기에 지수변환을 하면 count로 복원됨
(분석 정리 1)

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(log(y_i+1)-log(\hat{y}_i+1))^2}$$

6.4 베이스라인 모델

▪ 6.4.3 모델 훈련

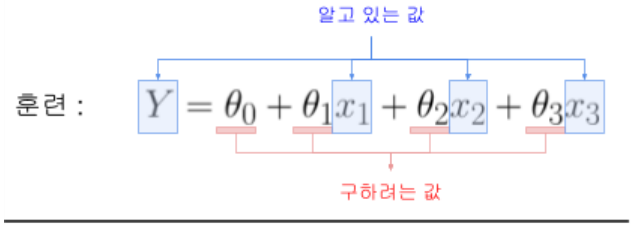
- 모델 생성 후 훈련
  - 가장 간단한 선형 회귀 모델인 LinearRegression 활용
  - 훈련 전 타깃값 로그변환(분석 정리1)
  - fit( )은 사이킷런의 모델 훈련 메서드

```
from sklearn.linear_model import LinearRegression

linear_reg_model = LinearRegression() # 선형 회귀 모델 생성

log_y = np.log(y) # 타깃값 로그변환
linear_reg_model.fit(X_train, log_y) # 모델 훈련
```

6.4 베이스라인 모델

▪ 6.4.3 모델 훈련

- 모델 생성 후 훈련



Y	종속변수(타깃값)
θ	회귀계수(가중치)
x	독립변수(피처)

- 피처(독립변수)와 타깃값(종속변수)이 주어질 때, 최적 가중치(회귀계수)를 찾는 과정
- 최적 가중치를 아는 상태(훈련된 모델)에서 새로운 독립변수(데이터)가 주어질 때, 타깃값을 추정하는 과정

- **탐색적 데이터 분석** : 예측에 도움될 피처를 추리고, 적절한 모델링 방법을 탐색하는 과정
- **피처 엔지니어링** : 추려진 피처를 훈련에 적합하도록, 성능 향상에 도움되도록 가공하는 과정

6.4 베이스라인 모델

6.4.4 모델 성능 검증

- 훈련된 모델로 예측 후 모델 성능 검증
  - 훈련된 모델로 예측 수행

```
preds = linear_reg_model.predict(X_train)
```

- 모델 훈련 – 결과 예측 – RMSLE 도출을 위해 시험 삼아 짠 코드
- 원래는 훈련 시 훈련 데이터를, 검증 시 검증 데이터를, 테스트 시 테스트 데이터를 사용해야 함

  - 예측이 잘 되었는지 RMSLE 계산

```
print (f'선형회귀의 RMSLE 값 : {rmsle(log_y, preds, True):.4f}')
선형회귀의 RMSLE 값 : 1.0205
```

- rmsle( ) 함수의 세 번째 인수로 True를 전달했으므로 지수변환을 적용함
- 단순 선형 회귀 모델의 RMSLE 값은 1.02

6.4 베이스라인 모델

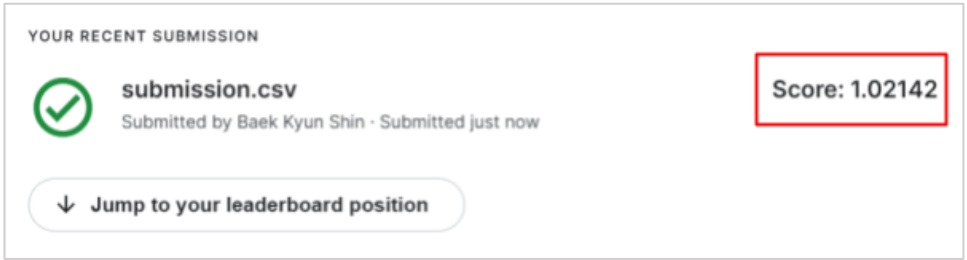
▪ 6.4.5 예측 및 결과 제출

- 베이스라인 모델로 예측한 결과 제출

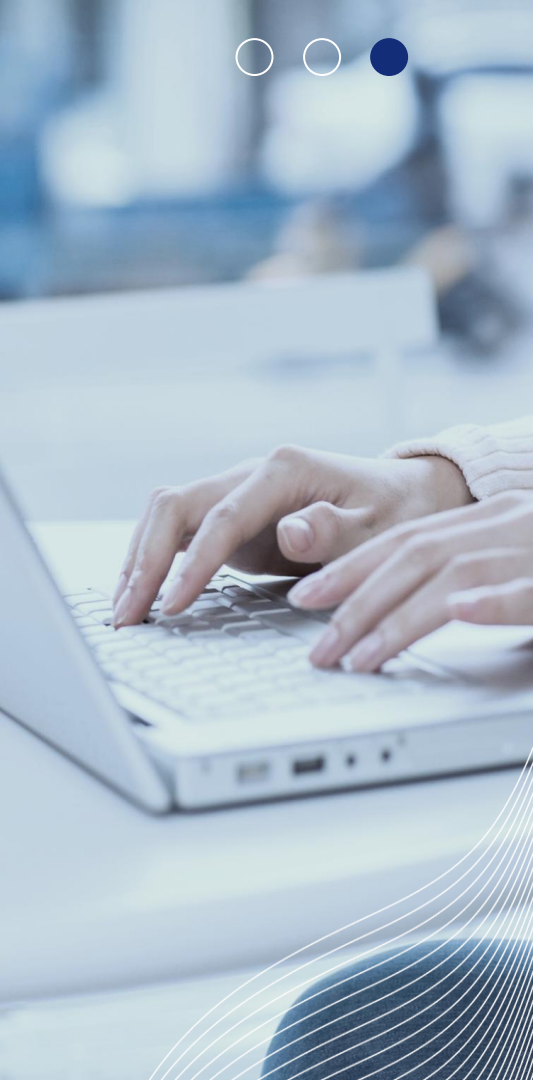
```
linearreg_preds = linear_reg_model.predict(X_test) # 테스트 데이터로 예측

submission['count'] = np.exp(linearreg_preds)     # 지수변환
submission.to_csv('submission.csv', index=False) # 파일로 저장
```

  - to_csv( )는 DataFrame을 csv 파일로 저장하는 함수(index=False로 설정해야 인덱스를 제외하고 저장)

- 커밋 후 제출 결과 확인



  - 평가점수 1.02142면 2,773등으로 전체 3,242명 가운데 상위 85.5%



6.1 경진대회 이해

6.2 경진대회 접속 방법 및 세부 메뉴

6.3 탐색적 데이터 분석

6.4 베이스라인 모델

6.5 성능 개선 I : 릿지 회귀 모델

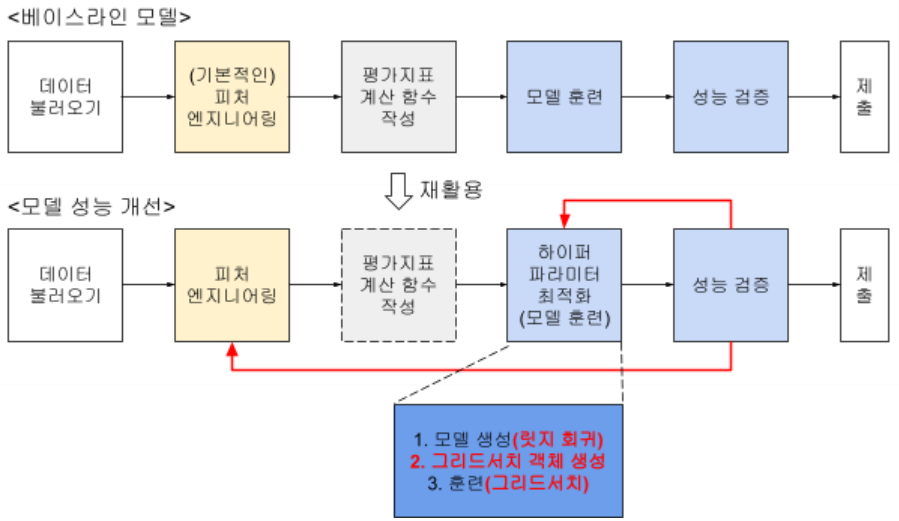
6.6 성능 개선 II : 라쏘 회귀 모델

6.7 성능 개선 III : 랜덤 포레스트 회귀 모델

6.5 성능 개선 I : 릿지 회귀 모델

▪ 릿지 회귀 모델

- L2 규제를 적용한 선형 회귀 모델
  - 규제regularization란 모델이 훈련 데이터에 과대적합overfitting되지 않도록 해주는 방법
  - 훈련 데이터에 과대적합되면 모델이 훈련 데이터에만 너무 잘 들어맞고, 테스트 데이터로는 제대로 예측하지 못함
- 베이스라인 모델과 성능 개선 프로세스 비교
  - 피처 엔지니어링을 본격적으로 수행
  - 모델 훈련 단계에서 하이퍼파라미터 최적화 수행
  - 성능이 만족스럽지 못하면 피처 엔지니어링이나 하이퍼파라미터 최적화를 더 고민해봄
  - 이번 장은 튜토리얼이므로 피처 엔지니어링을 추가로 진행하지 않음
  - 그러니 베이스라인용 노트북을 복사한 뒤, '데이터 불러오기' → '피처 엔지니어링' → '평가지표 계산 함수 작성'까지 실행해주세요!

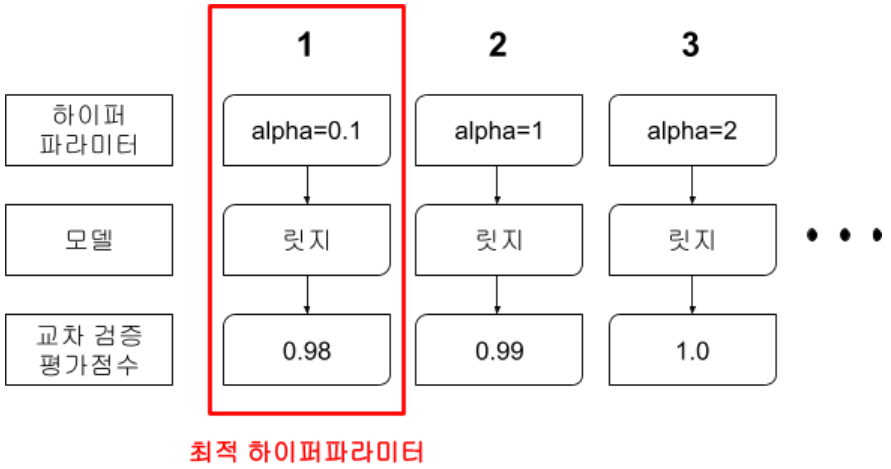


6.5 성능 개선 I : 릿지 회귀 모델

▪ 6.5.1 하이퍼파라미터 최적화(모델 훈련)

- 그리드서치

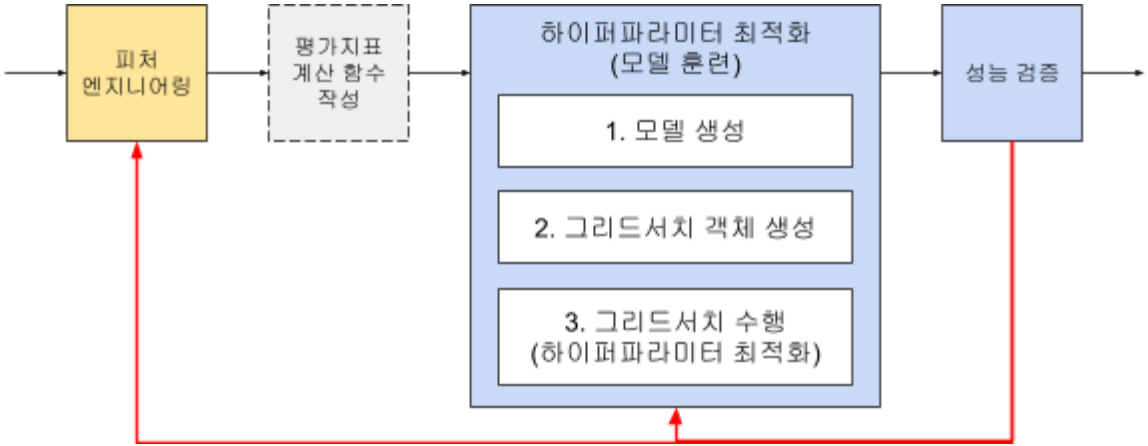
  - 그리드서치grid search는 하이퍼파라미터 최적화 기법 중 하나로, 격자grid처럼 촘촘하게 순회하며 최적 하이퍼파라미터 값을 찾는 기법
  - 각 하이퍼파라미터를 적용한 모델마다 교차 검증cross-validation하며 성능을 측정해 최종적으로 성능이 가장 좋았을 때의 하이퍼파라미터 값을 찾음
  - 그리드서치를 이용하지 않으면 수작업으로 하이퍼파라미터 값을 전달해서 성능을 측정해야 함



6.5 성능 개선 I : 릿지 회귀 모델

▪ 6.5.1 하이퍼파라미터 최적화(모델 훈련)

- 그리드서치
  - 그리드서치가 추가되면서 하이퍼파라미터 최적화 절차는 다음과 같이 세분화됨



6.5 성능 개선 I : 릿지 회귀 모델

6.5.1 하이퍼파라미터 최적화(모델 훈련)

- 모델 생성
  - 릿지 모델 생성

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from sklearn import metrics

ridge_model = Ridge()
```

6.5 성능 개선 I : 릿지 회귀 모델

▪ 6.5.1 하이퍼파라미터 최적화(모델 훈련)

- 그리드서치 객체 생성
  - 그리드서치의 원리
    - '하이퍼파라미터의 값'을 바꿔가며 '모델'의 성능을 교차 검증으로 '평가'해 최적 하이퍼파라미터를 찾음
  - 그리드서치 객체는 다음 세 가지를 알고 있어야 함
    - 1) 비교 검증해볼 하이퍼파라미터 값 목록, 2) 대상 모델, 3) 교차 검증용 평가 수단(평가 함수)

```
# 하이퍼파라미터 값 목록
ridge_params =
{'max_iter':[3000], 'alpha':[0.1, 1, 2, 3, 4, 10, 30, 100, 200, 300, 400, 800, 900, 1000]}

# 교차 검증용 평가 함수(RMSLE 점수 계산)
rmsle_scorer = metrics.make_scorer(rmsle, greater_is_better=False)

# 그리드서치(with 릿지) 객체 생성
gridsearch_ridge_model = GridSearchCV(estimator=ridge_model,   # 릿지 모델
                                      param_grid=ridge_params, # 하이퍼파라미터 값 목록
                                      scoring=rmsle_scorer,    # 평가지표
                                      cv=5)                    # 교차검증 분할 수
```

6.5 성능 개선 I : 릿지 회귀 모델

▪ 6.5.1 하이퍼파라미터 최적화(모델 훈련)

- 그리드서치 수행
  - 방금 만든 그리드서치 객체를 이용해 그리드서치 수행

```
log_y = np.log(y) # 타깃값 로그변환
gridsearch_ridge_model.fit(X_train, log_y) # 훈련(그리드서치)
```

- fit( ) 메서드를 실행하면 객체 생성 시 param_grid에 전달한 값을 순회하면서, 교차 검증으로 평가지표 점수를 계산함
- 이때 가장 좋은 성능을 보인 값을 best_params_ 속성에 저장하며,
  최적 값으로 훈련한 모델(최적 예측기)을 best_estimator_ 속성에 저장함

  - 최적 하이퍼파라미터 값 확인

```
print('최적 하이퍼파라미터 :', gridsearch_ridge_model.best_params_)
최적 하이퍼파라미터 : {'alpha': 0.1, 'max_iter': 3000}
```

6.5 성능 개선 I : 릿지 회귀 모델

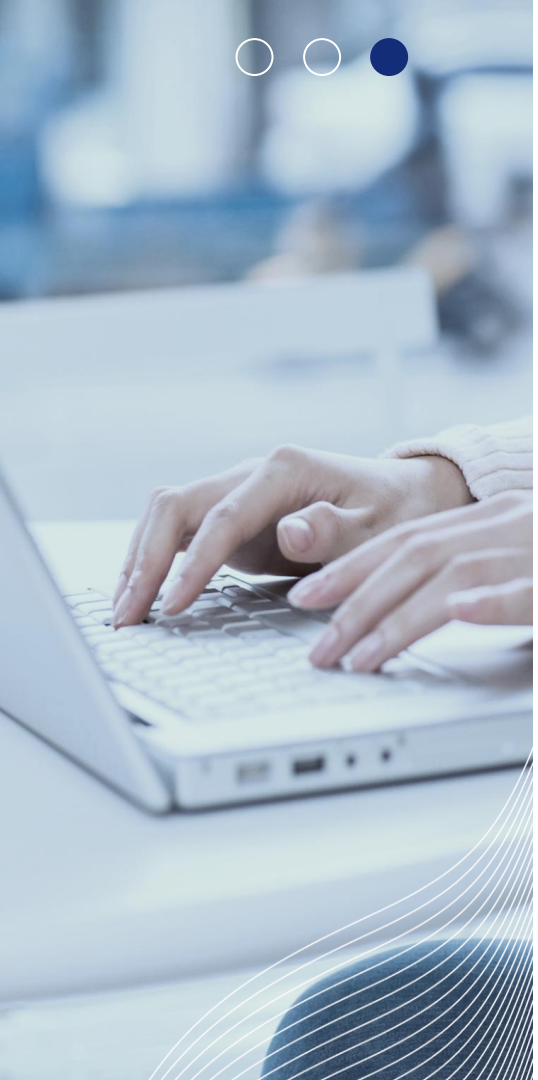
▪ 6.5.2 성능 검증

- 최적 예측기를 활용해 성능 검증

```
# 예측
preds = gridsearch_ridge_model.best_estimator_.predict(X_train)

# 평가
print(f'릿지 회귀 RMSLE 값 : {rmsle(log_y, preds, True):.4f}')
```

```
릿지 회귀 RMSLE 값 : 1.0205
```

- 선형 회귀 모델 결과와 다르지 않아서 결과는 굳이 제출하지 않음



6.1 경진대회 이해

6.2 경진대회 접속 방법 및 세부 메뉴

6.3 탐색적 데이터 분석

6.4 베이스라인 모델

6.5 성능 개선 I : 릿지 회귀 모델

6.6 성능 개선 II : 라쏘 회귀 모델

6.7 성능 개선 III : 랜덤 포레스트 회귀 모델

6.6 성능 개선 II : 라쏘 회귀 모델

▪ 6.6.1 하이퍼파라미터 최적화(모델훈련)

- 전체 흐름은 릿지 회귀 때와 똑같음

```
from sklearn.linear_model import Lasso

# 모델 생성
lasso_model = Lasso()
# 하이퍼파라미터 값 목록
lasso_alpha
= 1/np.array([0.1, 1, 2, 3, 4, 10, 30, 100, 200, 300, 400, 800, 900, 1000])
lasso_params = {'max_iter':[3000], 'alpha':lasso_alpha}
# 그리드서치(with 라쏘) 객체 생성
gridsearch_lasso_model = GridSearchCV(estimator=lasso_model,
                                      param_grid=lasso_params,
                                      scoring=rmsle_scorer,
                                      cv=5)
# 그리드서치 수행
log_y = np.log(y)
gridsearch_lasso_model.fit(X_train, log_y)
```

```
최적 하이퍼파라미터 : {'alpha': 0.00125, 'max_iter': 3000}
```

〈Note〉

'평가지표 계산 함수 작성' 단계까지가 계속 똑같기 때문에 6.5절의 노트북을 그대로 사용

6.6 성능 개선 II : 라쏘 회귀 모델

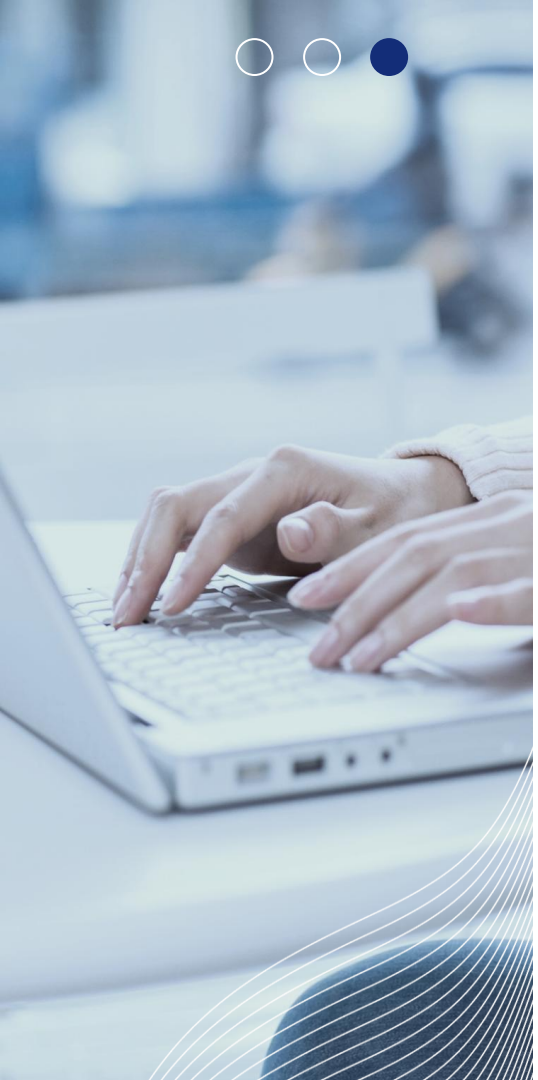
▪ 6.6.2 성능 검증

- 최적 예측기를 활용해 성능 검증

```
# 예측
preds = gridsearch_lasso_model.best_estimator_.predict(X_train)

# 평가
print(f'라쏘 회귀 RMSLE 값 : {rmsle(log_y, preds, True):.4f}')
```

```
라쏘 회귀 RMSLE 값 : 1.0205
```

- 여전히 성능이 개선되지 않아 이번에도 결과를 제출하지 않음



6.1 경진대회 이해

6.2 경진대회 접속 방법 및 세부 메뉴

6.3 탐색적 데이터 분석

6.4 베이스라인 모델

6.5 성능 개선 I : 릿지 회귀 모델

6.6 성능 개선 II : 라쏘 회귀 모델

6.7 성능 개선 III : 랜덤 포레스트 회귀 모델

6.7 성능 개선 III : 랜덤 포레스트 회귀 모델

▪ 6.7.1 하이퍼파라미터 최적화(모델훈련)

- 랜덤 포레스트 회귀 모델로 최적 하이퍼파라미터 값 구하기

```
from sklearn.ensemble import RandomForestRegressor

# 모델 생성
randomforest_model = RandomForestRegressor()
# 그리드서치 객체 생성
rf_params = {'random_state':[42], 'n_estimators':[100, 120, 140]}
gridsearch_random_forest_model = GridSearchCV(estimator=randomforest_model,
                                              param_grid=rf_params,
                                              scoring=rmsle_scorer,
                                              cv=5)
# 그리드서치 수행
log_y = np.log(y)
gridsearch_random_forest_model.fit(X_train, log_y)
print('최적 하이퍼파라미터 :', gridsearch_random_forest_model.best_params_)
```

```
최적 하이퍼파라미터 : {'n_estimators': 140, 'random_state': 42}
```

〈Note〉

'평가지표 계산 함수 작성' 단계까지가 계속 똑같기 때문에 앞 절의 노트북을 그대로 사용

6.7 성능 개선 III : 랜덤 포레스트 회귀 모델

▪ 6.7.2 모델 성능 검증

- 최적 예측기로 성능 확인

```
# 예측
preds = gridsearch_random_forest_model.best_estimator_.predict(X_train)

# 평가
print(f'랜덤 포레스트 회귀 RMSLE 값 : {rmsle(log_y, preds, True):.4f}')
```

```
랜덤 포레스트 회귀 RMSLE 값: 0.1126
```

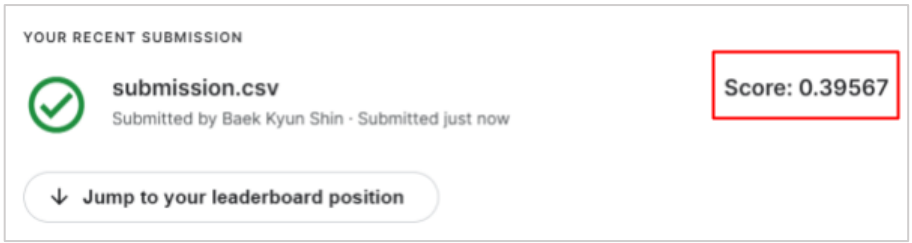
- RMSLE가 1.0205였던 선형 회귀, 릿지, 라쏘에 비해 큰 폭으로 개선됨
- 네 모델 가운데 가장 우수한 모델은 랜덤 포레스트 회귀

6.7 성능 개선 III : 랜덤 포레스트 회귀 모델

▪ 6.7.3 예측 및 결과 제출

```
# 예측
randomforest_preds = gridsearch_random_forest_model.best_estimator_.predict(X_test)

submission['count'] = np.exp(randomforest_preds) # 지수변환
submission.to_csv('submission.csv', index=False) # 예측 결과를 csv 파일로 저장
```

- 커밋 후 제출



- 193등, 상위 6.0% (vs. 베이스라인 모델은 상위 85.5%)
- 동메달권 점수