Kaggle 입문하기 - 데이터 분석 입문

학습 내용

- 캐글에 대해 이해하기
- 기본 모델과 정규화를 적용해 보기
- 피처를 추가 생성하는 것에 대해 기본 이해
- 여러 모델 비교 실습

목차

- URL : https://www.kaggle.com/
- Competitions 선택하면 다양한 대회 확인 가능.
- 대회 주제 : Bike Sharing Demand
- https://www.kaggle.com/c/bike-sharing-demand

Data Fields

필드명	설명
datetime	hourly date + timestamp
season	1 = spring(봄), 2 = summer(여름), 3 = fall(가을), 4 = winter(겨울)
holiday	whether the day is considered a holiday(휴일인지 아닌지)
workingday	whether the day is neither a weekend nor holiday(주말도 휴일도 아닌 날인지)
weather	1: Clear, Few clouds, Partly cloudy, Partly cloudy 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp	temperature in Celsius (온도)
atemp	"feels like" temperature in Celsius (체감온도)
humidity	relative humidity (습도)
windspeed	wind speed (바람속도)
casual	number of non-registered user rentals initiated (비가입자 사용유저)
registered	number of registered user rentals initiated (가입자 사용유저)
count	number of total rentals (전체 렌탈 대수)

01 기본 모델 만들고 제출

목차로 이동하기

In [51]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsRegressor
```

In [10]:
```
train = pd.read_csv("../bike/train.csv", parse_dates=['datetime'])
test = pd.read_csv("../bike/test.csv", parse_dates=['datetime'])
sub = pd.read_csv("../bike/sampleSubmission.csv")
```

입력 & 출력 특징(피처) 선택

In [3]:
```
sel = ['season', 'weather', 'temp']
X_tr_all = train[sel]         # 학습용 데이터의 변수 선택
y_tr_all = train['count']     # 학습용 데이터의 레이블 변수 선택

last_X_test = test[sel]       # 최종 예측. 테스트 데이터의 변수 선택
```

데이터 나누기

- 학습용 데이터 셋(train)을 학습:테스트(7:3)으로 나누기

In [5]:
```
X_train, X_test, y_train, y_test = train_test_split(X_tr_all,
                                                    y_tr_all,
                                                    test_size=0.3,
                                                    random_state=77)
```

모델 만들기 및 제출

모델 만들기 및 예측 순서

- 모델을 생성한다. model = 모델명()
- 모델을 학습한다. model.fit(입력값, 출력값)
- 모델을 이용하여 예측 model.predict(입력값)

In [6]:
```
from sklearn.linear_model import LinearRegression
```

In [7]:
```
model = LinearRegression()
model.fit(X_train, y_train)

# score()함수를 이용 - 결정계수 확인
print("학습용 세트 결정계수: {:.3f}".format(model.score(X_train, y_train)))
print("테스트 세트 결정계수: {:.3f}".format(model.score(X_test, y_test)))

model.predict(X_test)         # 예측(새로운 데이터로)
```

Out[7]:
```
학습용 세트 결정계수: 0.175
테스트 세트 결정계수: 0.163
array([207.35014841, 159.81017357, 235.16616427, ..., 145.59903806,
       182.62193532,  53.83291236])
```

In [8]:
```
print( model.coef_ )          # 모델(선형회귀의 계수)
print( model.intercept_)      # 모델(선형 회귀의 교차점)
```

```
[ 11.40588087 -30.62127051   8.66532654]
32.6138949247285
```

학습된 모델로 테스트 데이터 count를 예측 후, 제출하기

In [11]:
```
# sub = pd.read_csv("../bike/sampleSubmission.csv")
pred = model.predict(last_X_test)   # 예측
sub['count'] = pred
sub
```

Out[11]:

	datetime	count
0	2011-01-20 00:00:00	105.770886
1	2011-01-20 01:00:00	105.770886
2	2011-01-20 02:00:00	105.770886
3	2011-01-20 03:00:00	105.770886
4	2011-01-20 04:00:00	105.770886
...	...	...
6488	2012-12-31 19:00:00	75.149616
6489	2012-12-31 20:00:00	75.149616
6490	2012-12-31 21:00:00	105.770886
6491	2012-12-31 22:00:00	105.770886
6492	2012-12-31 23:00:00	105.770886

6493 rows × 2 columns

csv 파일(제출용 파일) 생성 후, 제출

In [12]:
```
# 처음 만는 제출용 csv 파일, 행번호를 없애기
sub.to_csv("firstsubmission.csv", index=False)
```

02 모델 성능 개선 - 다항회귀

목차로 이동하기

모델 성능 개선을 위해 다수의 특징(피처 or 변수)를 사용해 보기

In [13]:
```
train.columns
```

Out[13]:
```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count'],
      dtype='object')
```

In [14]:
```
sel = ['season', 'holiday', 'workingday', 'weather', 'temp',
           'atemp', 'humidity', 'windspeed']

X_tr_all = train[sel]         # 학습용 데이터의 변수 선택
```

```
last_X_test = test[sel]       # 테스트 데이터의 변수 선택

y_tr_all = train['count']
```

데이터 나누기

In [15]:
```
from sklearn.model_selection import train_test_split
```

In [16]:
```
X_train, X_test, y_train, y_test = train_test_split(X_tr_all,
                                                    y_tr_all,
                                                    test_size=0.3,
                                                    random_state=77)
```

In [17]:
```
model = LinearRegression()
model.fit(X_train, y_train)
# 결정계수 확인
print("학습용 세트 결정계수 : {:.3f}".format(model.score(X_train, y_train)))
print("테스트 세트 결정계수 : {:.3f}".format(model.score(X_test, y_test)))
```

```
학습용 세트 결정계수 : 0.262
테스트 세트 결정계수 : 0.257
```

MAE 구해보기

In [43]:
```
from sklearn.metrics import mean_absolute_error

preds = model.predict(X_test)
mean_absolute_error(preds, y_test)
```

Out[43]:
```
115.49633229240679
```

다항회귀 - PolynomialFeatures

- 다항식 및 상호작용 피처를 생성한다.
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html

In [21]:
```
from sklearn.preprocessing import PolynomialFeatures
```

In [22]:
```
sel = ['season', 'weather', 'temp']

X_tr = train[sel]         # 학습용 데이터의 변수 선택
y = train['count']

last_X_test = test[sel]       # 테스트 데이터의 변수 선택
```

PolynomialFeatures의 데이터 feature 추가 생성

```
transform from (x1, x2) to (1, x1, x2, x1^2, x1*x2, x2^2)
(x1, x2, x3) to (x1, x2, x3, x1^2, x2^2, x3^2, x1*x2, x1*x3,
x2*x3) 3->9
include_bias=True일 경우, 1 추가
(1, x1, x2, x3, x1^2, x2^2, x3^2, x1*x2, x1*x3, x2*x3) 3->10
```

In [23]:
```
ex_X_tr = PolynomialFeatures(degree=2, include_bias=False).fit_transform(X_tr)

X_tr.shape, ex_X_tr.shape
```

Out[23]:
```
((10886, 3), (10886, 9))
```

- 최종 예측을 위한 테스트 데이터 셋

In [24]:
```
ex_X_test = PolynomialFeatures(degree=2,
                               include_bias=False).fit_transform(last_X_test)

last_X_test.shape, ex_X_test.shape
```

Out[24]:
```
((6493, 3), (6493, 9))
```

In [25]:
```
X_train, X_test, y_train, y_test = train_test_split(ex_X_tr,
                                                    y,
                                                    test_size=0.3,
                                                    random_state=77)
```

In [26]:
```
model = LinearRegression()
model.fit(X_train, y_train)

# score()함수를 이용 - 결정계수 확인
print("학습용 세트 결정계수: {:.3f}".format(model.score(X_train, y_train)))
print("테스트 세트 결정계수: {:.3f}".format(model.score(X_test, y_test)))
```

```
학습용 세트 결정계수: 0.199
테스트 세트 결정계수: 0.182
```

다항회귀 적용 전 결정계수

```
학습용 세트 결정계수: 0.175
테스트 세트 결정계수: 0.163
```

실습해보기

- 8개의 변수를 다항회귀를 통해 피처를 생성하고, 모델을 만들어보자.

03 모델 성능 개선 - 정규화

목차로 이동하기

MinMaxScaler (정규화)

- 입력 데이터의 값의 범위를 0-1 사이로 변환

기본 모델

In [57]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [69]:
```
sel = ['season', 'holiday', 'workingday', 'weather', 'temp',
           'atemp', 'humidity', 'windspeed']
```

```
X_tr = train[sel]         # 학습용 데이터의 변수 선택
y_tr = train['count']

last_X_test = test[sel]       # 테스트 데이터의 변수 선택
```

In [70]:
```
X_train, X_test, y_train, y_test = train_test_split(X_tr,
                                                    y_tr,
                                                    test_size=0.3,
                                                    random_state=77)
```

In [71]:
```
m1 = KNeighborsRegressor()
m1.fit(X_train, y_train)
pred = m1.predict(X_test)

# score()함수를 이용 - 결정계수 확인
print("학습용 세트 결정계수: {:.3f}".format(m1.score(X_train, y_train)))
print("테스트 세트 결정계수: {:.3f}".format(m1.score(X_test, y_test)))
```

```
학습용 세트 결정계수: 0.474
테스트 세트 결정계수: 0.212
```

In [72]:
```
from sklearn.metrics import mean_absolute_error

preds = m1.predict(X_test)
mean_absolute_error(pred, y_test)
```

Out[72]:
```
116.17372933251684
```

정규화 적용

In [73]:
```
scaler = MinMaxScaler().fit(X_tr)
nor_X_tr = scaler.transform(X_tr)
```

In [74]:
```
X_train, X_test, y_train, y_test = train_test_split(nor_X_tr,
                                                    y_tr,
                                                    test_size=0.3,
                                                    random_state=77)
```

In [75]:
```
X_train.min(), X_train.max(), X_test.min(), X_test.max()
```

Out[75]:
```
(0.0, 1.0, 0.0, 1.0)
```

In [76]:
```
m2 = KNeighborsRegressor()
m2.fit(X_train, y_train)
pred = m2.predict(X_test)

# score()함수를 이용 - 결정계수 확인
print("학습용 세트 결정계수: {:.3f}".format(m2.score(X_train, y_train)))
print("테스트 세트 결정계수: {:.3f}".format(m2.score(X_test, y_test)))
```

```
학습용 세트 결정계수: 0.527
테스트 세트 결정계수: 0.292
```

In [77]:
```
from sklearn.metrics import mean_absolute_error

preds = m2.predict(X_test)
mean_absolute_error(pred, y_test)
```

Out[77]:
```
107.95119412124923
```

- 정규화를 적용하기 전의 모델보다 정규화 적용 후, 모델의 성능의 개선이 있다.

04 모델 성능 개선 - Ridge, Lasso

목차로 이동하기

In [78]:
```
from sklearn.linear_model import Ridge, Lasso
```

In [79]:
```
train.columns
```

Out[79]:
```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count'],
      dtype='object')
```

In [80]:
```
f_names = ['season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed']

X_tr = train[f_names]         # 학습용 데이터의 변수 선택
y = train['count']

last_X_test = test[f_names]       # 테스트 데이터의 변수 선택
```

In [81]:
```
ex_X_tr = PolynomialFeatures(degree=3, include_bias=False).fit_transform(X_tr)
ex_X_test = PolynomialFeatures(degree=3,
                               include_bias=False).fit_transform(last_X_test)

X_tr.shape, ex_X_tr.shape, last_X_test.shape, ex_X_test.shape
```

Out[81]:
```
((10886, 8), (10886, 164), (6493, 8), (6493, 164))
```

In [107…
```
X_train, X_test, y_train, y_test = train_test_split(ex_X_tr,
                                                    y,
                                                    test_size=0.4,
                                                    random_state=11)
```

In [108…
```
model = LinearRegression()
model.fit(X_train, y_train)

# 결정계수 확인
print("학습용 세트 결정계수: {:.3f}".format(model.score(X_train, y_train)))
print("테스트 세트 결정계수: {:.3f}".format(model.score(X_test, y_test)))
```

```
학습용 세트 결정계수: 0.351
테스트 세트 결정계수: 0.308
```

In [109…
```
m1 = Ridge(alpha=0.10)
m1.fit(X_train, y_train)

# 결정계수 확인
print("학습용 세트 결정계수: {:.3f}".format(m1.score(X_train, y_train)))
print("테스트 세트 결정계수: {:.3f}".format(m1.score(X_test, y_test)))
```

```
학습용 세트 결정계수: 0.351
테스트 세트 결정계수: 0.308
```

In [110…
```
model = Lasso(alpha=0.001)
model.fit(X_train, y_train)

# 결정계수 확인
```

```
print("학습용 세트 결정계수: {:.3f}".format(model.score(X_train, y_train)))
print("테스트 세트 결정계수: {:.3f}".format(model.score(X_test, y_test)))
```

```
학습용 세트 결정계수: 0.343
테스트 세트 결정계수: 0.310
```

```
C:₩Users₩totofriend₩anaconda3₩lib₩site-packages₩sklearn₩linear_model₩_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 6.977e+07, tolerance: 2.123e+04
  model = cd_fast.enet_coordinate_descent(
```

- 기존의 8개의 특징(변수)가 164개의 특징(변수)로 변환.
- Lasso 모델이 선형회귀보다 약간의 성능 향상이 있음.

05 여러 모델 성능 비교하기

목차로 이동하기

In [113…
```
# Linear Regression(선형 회귀), Ridge(리지), Lasso(라소)
model_list = [LinearRegression(), KNeighborsRegressor(),
              Ridge(alpha=10), Lasso(alpha=0.001)]
```

In [114…
```
for model in model_list:
    model.fit(X_train, y_train)

    print("모델 : ", model)
    # 결정계수 확인
    print("학습용 세트 결정계수: {:.3f}".format(model.score(X_train, y_train)))
    print("테스트 세트 결정계수: {:.3f}".format(model.score(X_test, y_test)))
```

```
모델 :  LinearRegression()
학습용 세트 결정계수: 0.351
테스트 세트 결정계수: 0.308
모델 :  KNeighborsRegressor()
학습용 세트 결정계수: 0.482
테스트 세트 결정계수: 0.156
모델 :  Ridge(alpha=10)
학습용 세트 결정계수: 0.350
테스트 세트 결정계수: 0.309
모델 :  Lasso(alpha=0.001)
학습용 세트 결정계수: 0.343
테스트 세트 결정계수: 0.310
```

```
C:₩Users₩totofriend₩anaconda3₩lib₩site-packages₩sklearn₩linear_model₩_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 6.977e+07, tolerance: 2.123e+04
  model = cd_fast.enet_coordinate_descent(
```

- 모델 비교한 결과 현재 학습 데이터로는 Lasso(alpha=0.001)의 성능이 상대적으로 좋은 편이다.

In [115…
```
model = Lasso(alpha=0.001)
model.fit(X_train, y_train)
```

```
C:₩Users₩totofriend₩anaconda3₩lib₩site-packages₩sklearn₩linear_model₩_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 6.977e+07, tolerance: 2.123e+04
  model = cd_fast.enet_coordinate_descent(
```

Out[115]:
```
Lasso(alpha=0.001)
```

In [116…
```
pred = model.predict(ex_X_test)   # 예측
sub['count'] = pred
sub.loc[sub['count'] < 0, 'count'] = 0
sub.head(3)
```

Out[116]:

	datetime	count
0	2011-01-20 00:00:00	105.416193
1	2011-01-20 01:00:00	66.081767
2	2011-01-20 02:00:00	66.081767

In [117…
```
# 두번째 제출
sub.to_csv("second_sub.csv", index=False)
```

제출 Score: 1.34721

History

- 최종 업데이트 : 2022/10