따릉이 데이터를 활용한 머신러닝 알고리즘 성능 비교

학술연구 스터디 그룹 **SML**

구성원 구보희, 김은지, 노승혜, 박지원, 송지우

1. 서론

연구배경 및 목적

머신러닝은 광범위한 데이터를 분석하여 정보를 얻는 방법으로, 마케팅, 의료, 교통 등 다양한 분야에서 활발하게 활용되고 있다. 특히 4차 산업혁명을 시작으로 IT에 대한 관심도가 높아지면서 빅데이터와 머신러닝의 중요성 또한 부각되고 있다.

이에, 전공 여부와 관계없이 다양한 곳에서 지식을 얻고, 특히 전공자들은 수업을 통해 지식을 주로 얻을 수 있다. 이렇게 얻은 지식과 더불어 추가로 학습한 내용을 바탕으로 직접 데이터를 분석하여 결과를 도출함으로써 빠르게 성장하고 있는 인공지능, 머신러닝 분야를 경험하고 학습하되, 연구자들의 사전지식과 능력이 모두 다르다는 점을 고려하여 머신러닝의 대표적인 데이터를 사용했으며 기초지식을 단단히 하는 것을 목적으로 하였다.

본 연구는 기상상황의 차이에 따른 따릉이 대여 수의 증감을 예측한다. 이를 위해 머신러닝을 활용하여 모형별 정확도와 예측력을 비교하였고, 예측을 통해 따릉이 이용에 더 나은 편리함을 제공하고자 하였다.

연구범위 및 방법

본 연구는 서울시 마포구의 날짜별, 시간별 기상상황과 따릉이 대여 수를 활용하였다. 이에 대한 분석은 서포트 벡터 머신(support vector machine), 선형회귀분석(linear regression), 의사결정나무(decision tree), 랜덤포레스트(random forest), Xgboost를 이용하였고, 따릉이 대여 수를 추정하여 RMSE(root mean square error)를 통해 모형별 예측에 대한 정확도를 비교하였다.

2. 이론적 고찰

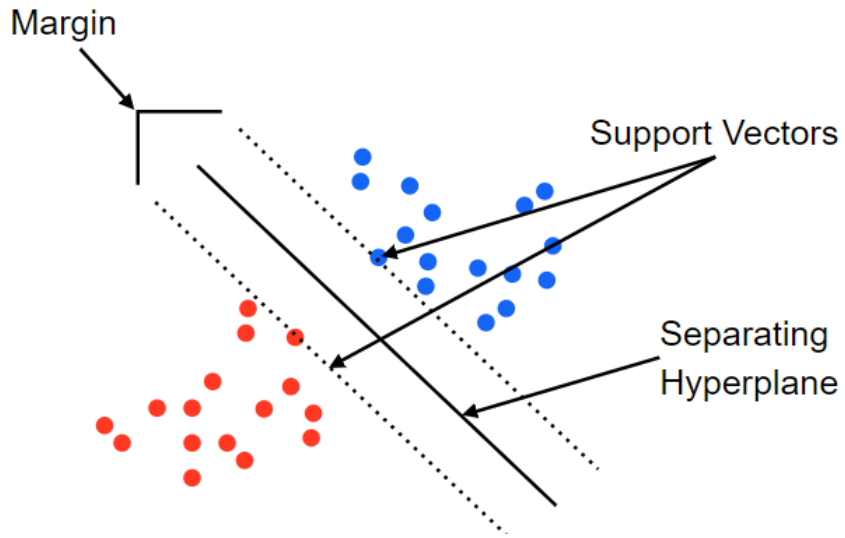
머신러닝이란**?**

머신러닝이란 인공지능의 한 분야이다. 사람이 학습하듯이 컴퓨터에 데이터를 넣어 학습하게 함으로써 새로운 지식을 도출하는 분야이다. 1950년대부터 머신러닝에 대한 연구가 시작되었고, 2000년대 중반에 머신러닝 기술 중 하나인 인공 신경망 분야에서 발전이 가속화되어 딥러닝이 탄생한 후, 머신러닝이 주목을 받게 되었다.

3. 분석모형(이론만)

1) Support Vector Machine

선형이나 비선형 분류, 회귀, 이상치 탐색에도 사용할 수 있는 머신 러닝 방법론으로, 데이터 크기가 중간 이하거나 여러 변수를 기준으로 분류하는 다소 복잡한 분류 문제를 잘 해결하는 머신러닝 모델이다. 레이블의 범주를 선형적 또는 비선형적으로 분류하는 hyperplane을 찾는 것이 모델의 핵심 과제이다.



두 집단의 떨어짐 정도를 margin이라 표현하고, margin은 점들이 포함되지 않은 영역을 최대화해서 class를 분리하는 것이다. support vector는 hyperplane에 가장 가까이 있는 class의 점을 의미 한다.

모든 점들이 완벽하게 구분되지 않으므로 어느 정도의 오류는 허용해야 한다. soft margin이란 일부 데이터가 분류 결과에 어긋나는 경우 일정 수준을 허용하는 방법으로, 하이퍼파라미터 값을 조절하여 조정한다. 따라서 하이퍼파라미터 값이 크면 overfitting 될 가능성이 있고, 너무 작으면 margin오류가 커지는 단점이 있다.

대표적인 하이퍼파라미터는 'C'와 'Kernel' 값이 있다. 'C'의 값이 작을수록 모델이 단순해지고, 커질수록 모델이 복잡해진다. 'Kernel' 값은 rbf, linear, poly 등 다양한 종류가 있다.

2) Linear Regression

선형회귀는 변수들 간의 함수관계를 분석하는 방법으로, 종속 변수 y와 한 개 이상의 독립 변수 (또는 설명 변수) X와의 선형 상관 관계를 모델링하는 회귀분석 기법이다. 독립변수가 종속변수에 미치는 영향력의 크기를 파악하고 이를 통하여 독립변수의 일정한 값에 대응하는 종속변수 값을 예측하는 모형을 산출한다. 일반적으로 최소제곱법을 사용하여 선형회귀 모델을 정립하는데, 두 변수 사이의 관계를 모델링하는 단순 선형회귀 방법과 둘 이상의 독립 변수에 기반하는 다중 선형회귀 방법이 있다.

3) Decision Tree

의사결정나무는 지도학습 기반의 방법론 중 하나로, 설명변수(X) 간의 관계나 척도에 따라 목표변수(Y)를 예측하거나 분류하는 문제에 활용되는 나무 구조의 모델이다. 목표변수(Y)를 예측하거나 분류 문제를 해결함에 있어서 어떤 설명변수가 가장 중요한 영향인자인지 확인할 수 있고, 나아가 각 설명변수별로 어떤 척도에 따라 예측 또는 분류했는지 상세한 기준을 알 수 있다는 장점을 갖는다.

의사결정나무의 분리 기준으로는 지니 지수(Gini Index)와 엔트로피 지수(Entropy Index)가 있다. 지니 지수란 불순도 측정 지수로서 '얼마나 다양한 데이터가 잘 섞여있는지 정도'를 나타내며, 순수도와 반대의 개념이다. 그렇기에 지니지수가 낮을수록 순수도는 높은 것이고 분리가 잘 이루어졌다고 평가할 수 있다. 엔트로피 지수는 '데이터가 섞여있는 정도'를 나타낸다. 따라서 엔트로피가 낮아지는 방향으로 분리하는 것이 좋다.

의사결정나무의 파라미터는 아래 표와 같다.

파라미터 명	설명
criterion	- 의사결정나무 분리 기준
max_depth	- 트리의 최대 깊이 - 완벽하게 클래스 값이 결정될 때 까지 분할 - 데이터 개수가 min_samples_split보다 작아질 때까지 분할 - 깊이가 깊어지면 과적합될 수 있으므로 적절히 제어 필요 - default = None
min_samples_split	- 노드를 분할하기 위한 최소한의 샘플 데이터수 - 과적합을 제어하는데 사용 - 작게 설정할 수록 분할 노드가 많아져 과적합 가능성 증가 - default = 2
min_samples_leaf	- leaf node가 되기 위해 필요한 최소한의 샘플 데이터 수 - min_samples_split과 함께 과적합 제어 용도 - 불균형 데이터의 경우 특정 클래스의 데이터가 극도로 작을 수 있으므로 작게 설정 필요

4) Random Forest

먼저, 랜덤 포레스트를 설명하기에 앞서 결정 트리를 짚어야 한다. 결정 트리란 말그대로 결정을 내리기 위해 사용하는 트리로, 결정 과정을 간단한 문제들로 이루어진 계층 구조로 나눈다. 간단한 문제에 대해서는 매개변수(예: 모든 노드의 테스트 매개변수, 종단 노드에서 매개변수 등)를 사용자가 직접 설정할 수 있지만, 보다 복잡한 문제의 경우 학습 데이터로부터 트리 구조와 매개변수를 모두 자동으로 학습한다는 특징이 있다.

랜덤 포레스트란 분류, 회귀 분석 등에 사용되는 앙상블 학습 방법의 일종으로, 다수의 결정 트리들을 학습한다. 이러한 훈련 과정에서 구성한 다수의 결정 트리로부터 부류(분류) 또는 평균 예측치(회귀 분석)를 출력함으로써 동작하기에, 결정 트리의 집합이라고 보아도 무방하다. 이러한 랜덤 포레스트는 월등히 높은 정확성 및 간편하고 빠른 학습 및 테스트 알고리즘, 변수 소거를 하지 않아도 수천 개의 입력 변수를 다루는 것이 가능한 스케일 등의 장점 덕분에 활용도가 매우 높다.

5) Xgboost

부스팅이란 약한 학습기를 여러 개 연결해 강한 학습기를 만드는 앙상블 방법을 의미한다. 앞의 모델을 보완해 나가면서 일련의 예측기를 학습시키는 것이 아이디어이다. 부스팅 방법 중 가장 선호되는 것은 AdaBoost(Adaptive boosting의 줄임말)과 Gradient Boosting이다. Adaboost는 전의 예측기를 보완하는 새로운 예측기를 만들기 위해 이전 모델이 과소적합했던 훈련 샘플의 가중치를 더 높이는데, 이렇게 하면 새로운 예측기는 학습하기 어려운 샘플에 점점 더 맞춰지게 된다. 이런 방식을 에이다부스트에서 사용한다. 에이다부스트처럼 그레디언트 부스팅은 앙상블에 이전까지의 오차를 보정하도록 예측기를 순차적으로 추가하나, 에이다부스트처럼 반복마다 샘플의 가중치를 수정하는 대신 이전 예측기가 만든 잔여 오차에 새로운 예측기를 학습시킨다. XGBoost는 최적화된 그레디언트 부스팅을 구현한 것으로 Extreme Gradient Boosting의 약자이다. 이 패키지는 톈치 천이 개발했고 매우 빠른 속도, 확장성, 이식성을 목표로 한다.

분석자료

서울시 마포구의 날짜별, 시간별 기상상황과 따릉이 대여 수 데이터를 이용한다. 데이터는 시간, 기온, 비, 풍속, 습도, 시정, 오존, 미세먼지, 시간에 따른 따릉이 대여 수를 포함하고 있다.

분석방법

Random forest, Support vector machine, Linear regression, Decision tree, Xgboost의 따릉이 대여량에 대한 예측을 비교한다.

결측치는 0으로 대체한 후 IQR지수를 이용해 25%부터 75% 내에 들지 않는 이상치들을 제거해 전처리한 데이터를 사용한다. 7.5:2.5의 비율로 train 데이터셋과 test 데이터셋을 분리했으며 각 train 데이터들을 표준화함으로써 컬럼마다 달랐던 데이터들의 분포를 일정하게 맞춰준다. 이후 각 모델들을 적용하여 결과로써 rmse를 구하여 모델을 평가한다.

4. 모형별 적합 결과**(**결과작성**)**

1) Support Vector Machine

```
#train데이터정확도

from sklearn.svm import SVR

model=SVR(kernel='poly')

model.fit(X_train,y_train)

pred_train=model.predict(X_train)

model.score(X_train,y_train)

> 0.4227418049315229


#test데이터정확도

pred_test=model.predict(X_test)

model.score(X_test,y_test)

> 0.4205565595047416


#RMSE

from sklearn.metrics import mean_squared_error

MSE_train=mean_squared_error(y_train,pred_train)

MSE_test=mean_squared_error(y_test,pred_test)

print('train데이터 :',np.sqrt(MSE_train))

print('test데이터 :',np.sqrt(MSE_test))

> train데이터 : 59.137950688476245

test데이터 : 63.729168341972944


#하이퍼파라미터 조정 : Grid Search

from sklearn.model_selection import GridSearchCV
```

```
param_grid={'C':[0.01,0.1,1,10],'kernel':['rbf'],'gamma':[0.01,0.1,1,10
]}

grid_search=GridSearchCV(SVR(),param_grid,cv=5,scoring='neg_mean_square
d_error')

grid_search.fit(X_train,y_train)

> GridSearchCV(cv=5, estimator=SVR(),

             param_grid={'C': [0.01, 0.1, 1, 10], 'gamma': [0.01, 0.1,
1, 10],

                         'kernel': ['rbf']},

             scoring='neg_mean_squared_error')


#최적의 하이퍼파라미터

print('Best Parameter : {}'.format(grid_search.best_params_))

> Best Parameter : {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}


scores_df=pd.DataFrame(grid_search.cv_results_)

scores_df['rmse']=np.sqrt(-scores_df['mean_test_score'])

rmse_df=scores_df.loc[:,['param_C','param_gamma','param_kernel','mean_t
est_score','rmse']].sort_values('mean_test_score',ascending=False)

rmse_df
```

param_C	param_gamma	param_kernel	mean_test_score	rmse
10	0.1	rbf	-2637.387717	51.355503
10	0.01	rbf	-2892.267377	53.779805
1	0.1	rbf	-3796.718388	61.617517
1	0.01	rbf	-4470.847158	66.864394
10	1	rbf	-4643.720871	68.144852
0.1	0.1	rbf	-5648.573846	75.156995

0.1	0.01	rbf	-5914.156914	76.903556
1	1	rbf	-5935.688814	77.043422
0.01	0.1	rbf	-6110.898069	78.172233
0.01	0.01	rbf	-6142.911741	78.376730
0.1	1	rbf	-6145.601928	78.393890
10	10	rbf	-6162.099864	78.499044
0.01	1	rbf	-6167.422583	78.532939
1	10	rbf	-6167.932832	78.536188
0.1	10	rbf	-6169.371139	78.545344
0.01	10	rbf	-6169.816614	78.548180

2) Linear Regression

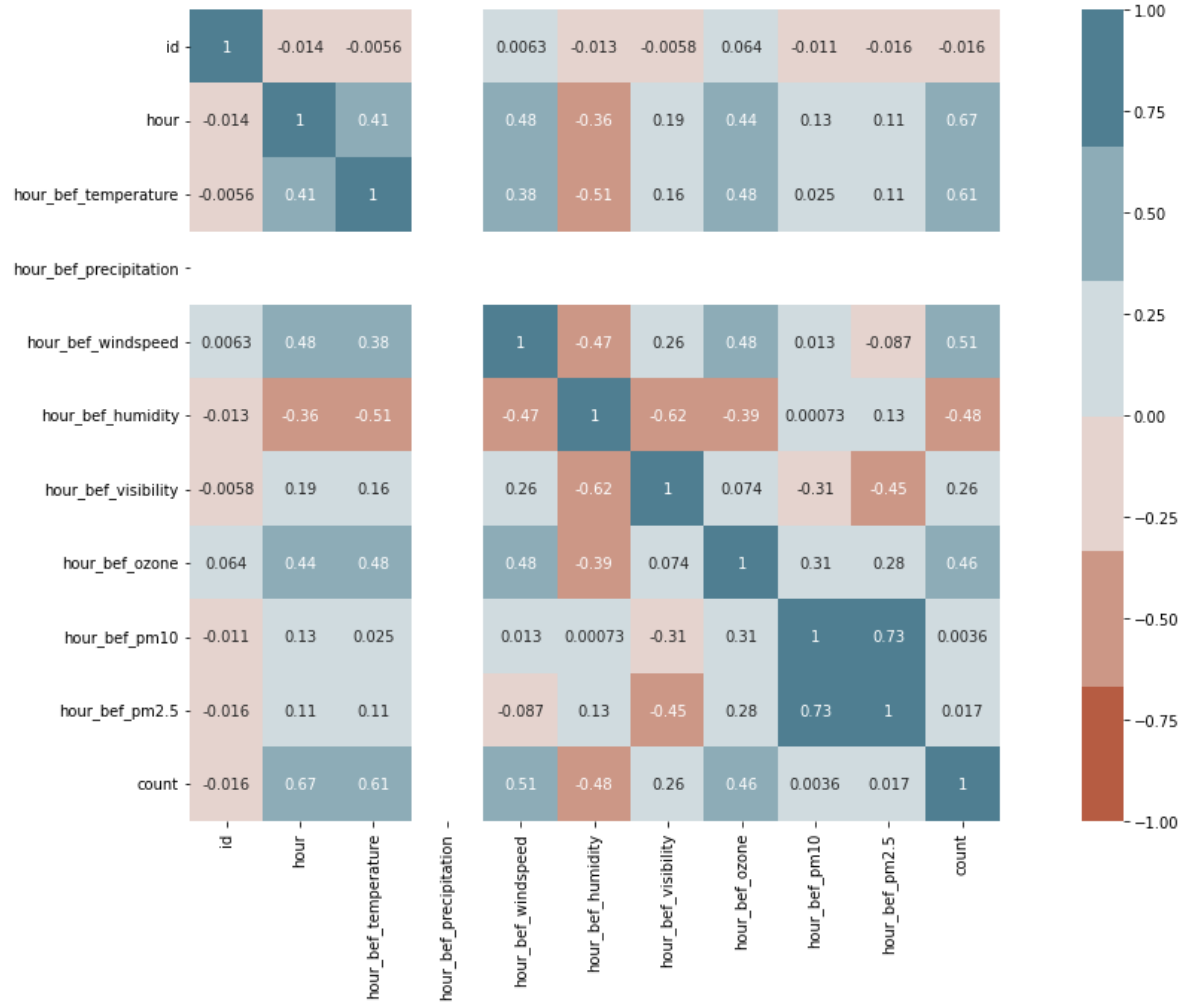
데이터 결측값, 이상치 처리

```
#결측치 0으로 처리
train.fillna(0,inplace = True)
test.fillna(0,inplace = True)

#이상치 처리(IQR 25%~75%)
colL  = ['hour','hour_bef_temperature', 'hour_bef_precipitation',
'hour_bef_windspeed', 'hour_bef_humidity', 'hour_bef_visibility',
'hour_bef_visibility',
      'hour_bef_ozone', 'hour_bef_pm10', 'hour_bef_pm2.5', 'count']
train_iqr = train.copy()
for col in colL:
  Q1 = train_iqr[col].quantile(.25)
  Q3 = train_iqr[col].quantile(.75)
  IQR = Q3 - Q1
  train_del = train_iqr[((Q1-1.5*IQR) > train_iqr[col]) | ((Q3+1.5*IQR)
< train_iqr[col])]
  train_iqr = train_iqr.drop(train_del.index, axis=0)
train_iqr
```

train 데이터 변수간 상관관계

```
import numpy as np
```

```
number = train.select_dtypes(np.number)

corr = number.corr()
plt.figure(figsize=(20, 10))
ax = sns.heatmap(
    corr,
    vmin=-1, vmax=1,
    cmap=sns.diverging_palette(20, 220),
    square=True,
    annot=True
)

plt.show()
```




```
train_iqr.corr()['count']
>
id                          -0.015510
hour                         0.665605
hour_bef_temperature         0.605346
hour_bef_precipitation            NaN
```

```
hour_bef_windspeed        0.510439
hour_bef_humidity        -0.475401
hour_bef_visibility       0.262575
hour_bef_ozone            0.463961
hour_bef_pm10             0.003570
hour_bef_pm2.5            0.017081
count                     1.000000
Name: count, dtype: float64
```

count 변수와 상관관계가 높은 'hour', 'hour_bef_temperature', 'hour_bef_humidity', 'hour_bef_ozone', 'hour_bef_windspeed' 변수만을 사용할 예정

데이터 전처리

```
 # 'hour', 'hour_bef_temperature', 'hour_bef_humidity',
'hour_bef_ozone', 'hour_bef_windspeed' 변수가 포함된 데이터로 처리

X=train_iqr.drop(['id','hour_bef_precipitation', 'hour_bef_visibility',
'hour_bef_pm10','hour_bef_pm2.5','count'], axis=1)
y=train_iqr['count']

 # train 데이터를 모델 학습(*_train)과 예측(*_test)으로 나누어 분석
```

모델을 학습하고 평가하기 위해 전체 데이터를 훈련데이터(75%)와 시험 데이터(25%)로 분리

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42)
```

모델 정의 및 학습, 예측

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train, y_train)

 # 모델로 X_test의 y값 예측
y_hat = model.predict(X_test)
```

RMSE

```
from sklearn.metrics import mean_squared_error
print("RMSE : ",mean_squared_error(y_hat,y_test)**0.5)
```

> RMSE : 50.04604659980739

결과적으로 **linear regression**의 분석 결과 **rmse**는 **50.04604659980739**

3) Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import GridSearchCV


# 모델 생성 및 파라미터 설정

dt_clf = DecisionTreeClassifier(random_state=42)

parameters = {'criterion': ['gini', 'entropy'],

              'max_depth': [x for x in range(3, 15,2)],

              'min_samples_split': [x for x in range(3, 15,2)],

              'min_samples_leaf': [x for x in range(1, 11,2)]

}


# 최적의 파라미터 찾기

grid_dt = GridSearchCV(dt_clf, param_grid = parameters, cv = 5)


grid_dt.fit(X_train, y_train)


#최적의 하이퍼파라미터

print('Best Parameter : {}'.format(grid_dt.best_params_))
```

>> 출력 결과

```
Best Parameter : {'criterion': 'gini', 'max_depth': 3,
'min_samples_leaf': 7, 'min_samples_split': 3}
```

```
# 최적의 parameter 적용하여 모델 생성 (rmse)

dt = DecisionTreeClassifier(random_state=42, criterion='gini',
max_depth= 3, min_samples_leaf= 7, min_samples_split= 3)

dt.fit(X_train, y_train)


# 최적의 파라미터 적용 후 rmse값

from sklearn.metrics import mean_squared_error

pred = dt.predict(X_test)

mse = mean_squared_error(y_test, pred)

rmse = np.sqrt(mse)

print(rmse)
```

>> 출력 결과

```
67.24154516982337
```

```
# 파라미터에 따른 mean_test_socre와 rmse값 비교

params_result = pd.DataFrame(grid_dt.cv_results_['params'])

params_result['mean_test_score'] =
grid_dt.cv_results_['mean_test_score']

params_result['rmse']=np.sqrt(abs(params_result['mean_test_score']))

params_result.sort_values(by='rmse', ascending=True).head(10)
```

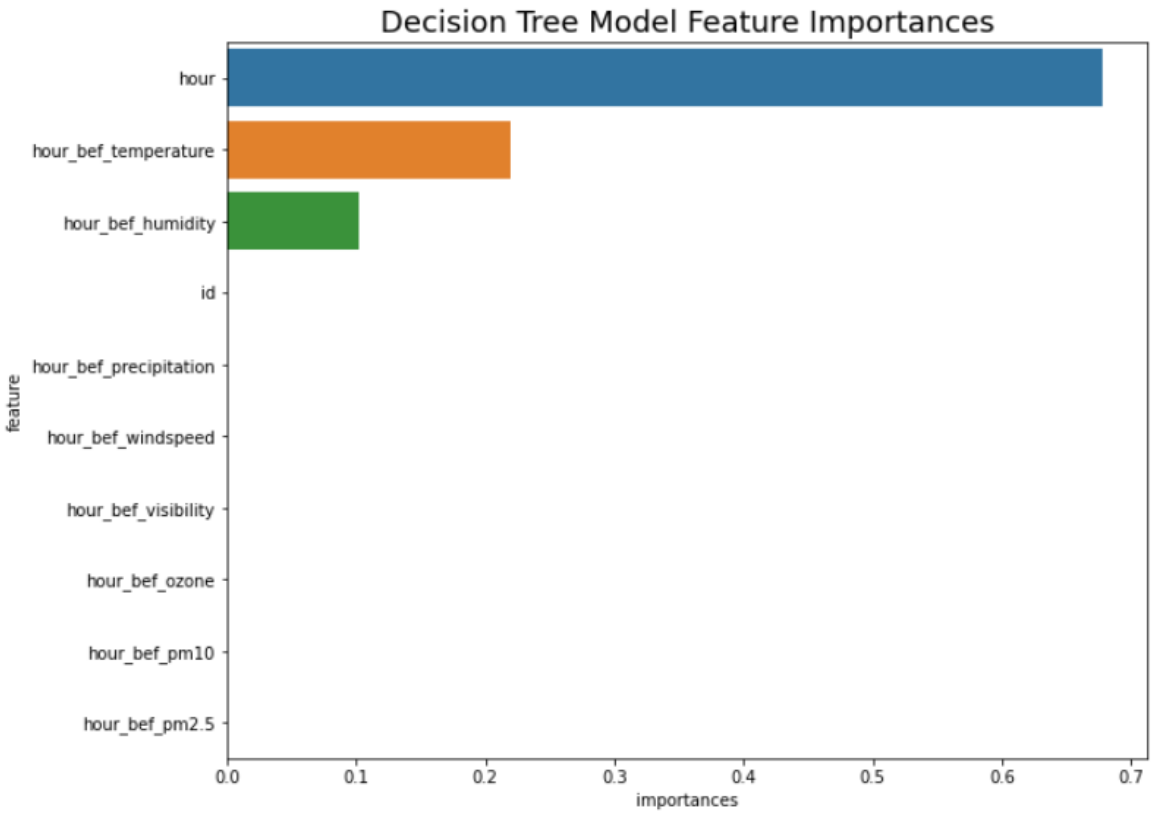
>> 출력 결과

criterion	max_depth	min_samples_split	min_samples_leaf	mean_test_score	rmse
entropy	13	1	3	0.009524	0.097590
entropy	11	1	3	0.009524	0.097590

gini	13	3	13	0.010582	0.102869
gini	11	3	11	0.010582	0.102869
entropy	13	3	9	0.011640	0.107890
entropy	13	1	9	0.011640	0.107890
gini	13	3	11	0.011640	0.107890
entropy	11	1	9	0.011640	0.107890
entropy	11	3	9	0.011640	0.107890
entropy	13	1	11	0.012698	0.112687

```
# Feature Importance 보여주는 함수
import matplotlib.pyplot as plt
import seaborn as sns

def show_importance(model, title):
  importance = model.feature_importances_
  feature = train_x.columns
  importances = pd.DataFrame()
  importances['feature'] = feature
  importances['importances'] = importance
  importances.sort_values('importances', ascending=False, inplace=True)
  importances.reset_index(drop=True, inplace=True)

  plt.figure(figsize=(10, 8))
  sns.barplot(x='importances', y='feature', data=importances)
```

```
  plt.title(title, fontsize=18)

  plt.show()


show_importance(dt, "Decision Tree Model Feature Importances")
```



hour과 hour_bef_temperature, hour_bef_humidity 요인이 따릉이 대여량에 영향을 미치는 정도가 큼을 확인할 수 있다.

4) Random Forest

```
from google.colab import drive
drive.mount('/content/drive')
```

```
train.head()
```

	id	hour	hour_bef_temperature	hour_bef_precipitation	hour_bef_windspeed	hour_bef_humidity	hour_bef_visibility	hour_bef_ozone	hour_bef_pm10	hour_bef_pm2.5	count
0	3	20	16.3	1.0	1.5	89.0	576.0	0.027	76.0	33.0	49.0
1	6	13	20.1	0.0	1.4	48.0	916.0	0.042	73.0	40.0	159.0
2	7	6	13.9	0.0	0.7	79.0	1382.0	0.033	32.0	19.0	26.0
3	8	23	8.1	0.0	2.7	54.0	946.0	0.040	75.0	64.0	57.0
4	9	18	29.5	0.0	4.8	7.0	2000.0	0.057	27.0	11.0	431.0

	A	B
1	id	count
2	0	26
3	1	148
4	2	88
5	4	33
6	5	173
7	10	132
8	11	207
9	12	380
10	15	46
11	17	212
12	18	271
13	23	159
14	25	189
15	26	52
16	31	214
17	39	126
18	40	30
19	41	214
20	42	380
21	43	92
22	51	148

	A	B
1	id	count
2	0	26
3	1	431
4	2	88
5	4	33
6	5	120
7	10	109
8	11	207
9	12	214
10	15	49
11	17	208
12	18	271
13	23	317
14	25	86
15	26	45
16	31	247
17	39	126
18	40	30
19	41	214
20	42	380
21	43	92
22	51	126
23	52	106
24	54	18
25	60	113
26	61	218
27	62	90

<-결과 일부 캡처

최종적으로 심사하였을 때, 성능은 약 60.8%(0.6)으로 좋은 편은 아니었다.

```
60.8040415477
```

```
63.1253498225
```

5) Xgboost

모델 구현

```
import xgboost as xgb
```

```
from xgboost import XGBRegressor

xgb_reg = XGBRegressor(objective='reg:squarederror', n_estimators=200,
colsample_bylevel=0.8, colsample_bytree=0.7, eta=0.1, max_depth=6)

xgb_reg.fit(X_train,y_train)
```

파라미터 설명

colsample_bylevel: 트리의 레벨 별로 훈련 데이터의 변수를 샘플링해주는 비율

colsample_bytree: 트리를 생성할 때 훈련 데이터에서 변수를 샘플링해주는 비율

Eta: 딥러닝에서의 학습률과 같은 개념으로 값이 너무 높으면 학습이 잘 되지 않을 수 있으며 너무 낮으면 학습이 느릴 수 있다.

max_depth: 트리 모델의 최대 깊이

n_estimators: 생성할 트리의 개수

그리드서치 결과

```
from sklearn.model_selection import GridSearchCV

param_grid = {

    'n_estimators':[100,200],

    'max_depth': [6,8,10,12],

    'eta':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8],

    'colsample_bytree':[0.6, 0.7, 0.8, 0.9],

    'colsample_bylevel':[0.6, 0.7, 0.8, 0.9],

}

grid_search = GridSearchCV(XGBRegressor(random_state=42,
objective='reg:squarederror',), param_grid, cv=2, n_jobs=1,
scoring='neg_mean_squared_error')

grid_search.fit(X_train,y_train)
```

최적의 결과: 'colsample_bylevel': 0.8, 'colsample_bytree': 0.7,'eta': 0.1, 'max_depth': 6, 'n_estimators': 200

파라미터 조합 별 결과 비교

colsample_bylevel	colsample_bytree	eta	max_depth	max_depth	test_score	rmse
0.8	0.7	0.1	6	200	-1691.455556	41.127309
0.8	0.7	0.8	6	200	-1691.455556	41.127309
0.8	0.7	0.5	6	200	-1691.455556	41.127309
0.8	0.7	0.7	6	200	-1691.455556	41.127309
0.8	0.7	0.4	6	200	-1691.455556	41.127309
... (위와 동일한 rmse 값이 나오는 일부 행 생략함)						
0.8	0.7	0.8	6	100	-1694.537483	41.164760
0.8	0.7	0.6	6	100	-1694.537483	41.164760
0.8	0.7	0.4	6	100	-1694.537483	41.164760
0.8	0.7	0.3	6	100	-1694.537483	41.164760
0.8	0.7	0.7	6	100	-1694.537483	41.164760
... (위와 동일한 rmse 값이 나오는 일부 행 생략함)						
0.7	0.8	0.3	6	200	-1703.016926	41.267626
0.7	0.8	0.5	6	200	-1703.016926	41.267626
0.7	0.8	0.4	6	200	-1703.016926	41.267626
0.7	0.8	0.1	6	200	-1703.016926	41.267626
0.7	0.8	0.2	6	200	-1703.016926	41.267626
...						

정확도

0.8185366727465971

rmse

테스트 데이터셋에서 이 모델을 적용해 평가한 결과 xgboost의 rmse는 35.203343446692710이 나옴을 확인할 수 있었다.

5. 결론

비교적 쉽고 전처리 등이 용이하며, 참고자료를 찾기 쉬운 데이터셋을 가지고 다양한 알고리즘을 적용시켜 보았다. 결론적으로는, XG boost의 성능이 약 0.8로 가장 좋았다. 일반적으로 random forest의 정확도가 좋다고 생각하는 것과는 다른 결론이 도출되었다.

이러한 원인 및 우리의 연구에 있어 아쉬운 점을 생각해 본다면, 데이터의 전처리도 각자 진행하였기 때문에 해당 성능이 온전히 모델만의 성능은 아닐 수 있다는 점이다. 전처리까지 함께 진행한 후, 각자 모델링하는 방식으로 진행되었다면 성능 수치에 대해 믿을 수 있었을 것이기에 아쉬움이 남는다.

또한, 시간 관계상 '따릉이'로 선정하였지만 이번 프로젝트를 통해 경험치와 지식을 쌓은 지금은, 어느 정도 여유가 된다면 공공데이터 등 공익에게 도움이 되는 데이터 프로젝트도 진행해 보고자 한다.

6. 참고문헌

핸즈온 머신러닝 2판, 오렐리앙 제롱, 한빛미디어

Kaggle 우승작으로 배우는 머신러닝 탐구생활, 정권우, 비제이퍼블릭