Pycaret를 활용한 Tripadvisor Hotel 데이터 분석

201901719 헝가리어과

박화랑

먼저 데이터에 맞는 기계학습 모형을 찾고 성능평가를 위해 구글 코랩(Google Colaboratory) 개발환경에서 Python 오픈소스 로우코드 라이브러리인 Pycaret 3.2.0버전을 사용하였다. 평가지표는 MAE, MSE, RMSE, R2, RMSLE, MAPE가 있으며 우리는 그 중에서도 RMSE를 사용하여 평가하겠다. RMSE는 값이 낮아 질수록 예측 결과가 좋음을 나타내며 성능 지표는 다음과 같다.

폰트, 텍스트, 화이트, 도표이(가) 표시된 사진

자동 생성된 설명

Pycaret 3.2.0에서 지원하는 Scikit-learn을 활용하여 Regression을 진행을 하였다. 코드는 다음과 같다.

!pip install pycaret

!pip install --upgrade pycaret

from pycaret.regression import \*

이후 데이터 전처리는 간단하게 target인 “bubble\_rating”과 연관성이 없어 보이는 호텔이름, 링크와 순서 컬럼을 drop을 사용해서 버려주고 class 컬럼에 no\_stars를 전부 0으로 대체하였다.

from google.colab import drive

drive.mount('/content/drive')

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

result = pd.read\_csv("/content/drive/MyDrive/비즈니스 머신러닝/final\_project\_hotelReview.csv", encoding="CP949")

result = result.drop(columns = "Unnamed: 0")

result = result.drop(columns = "hotel\_url")

result = result.drop(columns = "name")

result.loc[result['class'] == "no stars", 'class'] = 0

result = result.astype({"class":int})

result["class"]

그리고 결측값은 dropna()를 사용하여 제거하였다.

result = result.dropna()

그 후에는 Pycaret의 set\_up함수로 모델을 생성해 주었다. target은 “bubble\_rating”으로 훈련데이터와 테스트데이터 비율은 8:2로 맞추고 random\_state는 123고정하고 빠른 계산을 위해 gpu를 사용하였다.

model = setup(data = result, target = "bubble\_rating",session\_id=123, train\_size=0.8,use\_gpu=True)

compare\_model을 사용해서 여러 회귀모델들의 평가지표를 비교하였고 그 결과 다음과 같다.

|  |  |
| --- | --- |
| Model | RMSE |
| Gradient Boosting Regressor | 0.5743 |
| Random Forest Regressor | 0.5793 |
| Extra Trees Regressor | 0.5952 |
| Light Gradient Boosting Machine | 0.5991 |
| Bayesian Ridge | 0.6124 |
| Orthogonal Matching Pursuit | 0.6188 |
| Extreme Gradient Boosting | 0.6187 |

맨 위 가장 성능이 좋은 Gradient Boosting Regressor(이하 gbr)을 생성하고 최적의 파타미터 튜닝을 하기 위해서 자동 튜닝인 tune\_model을 사용하였다.

gbr = create\_model('gbr')

gbr\_tune = tune\_model(gbr)

튜닝을 한 결과 초기모형의 RMSE값인 0.5743에서 0.5993으로 오히려 결과 값이 안좋아져서 초기모형을 사용하기로 했다. 이후 전체 데이터 학습을 진행하고 test데이터의 예측을 진행하였다.

final\_model = finalize\_model(gbr)

prediction = predict\_model(final\_model)

결과는 다음과 같다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

0.5743이던 RMSE가 0.4558이 나왔다. 더 좋은 성능을 뽑아내기 위해서 blend\_models()을 활용하여

이번에는 모델끼리 섞어보았다.

gbr은 Boosting 모델이니 바로 아래에 다음으로 성능이 좋은 Bagging 모델인 Random Forest Regressor(이하 rf)를 결합하였다. 결합하기전에 똑같이 튜닝을 진행해보고 성능이 상승하였는지 확인하였으나 gbr과 마찬가지로 초기 성능이 더 좋아 초기모형을 사용하였다. (초기: 0.5793, 튜닝 후: 0.5881)

rf = create\_model('rf')

rf\_tune = tune\_model(rf)

blend = blend\_models(estimator\_list= [gbr, rf])

합쳐진 모델의 RMSE값은 0.5712가 나오면서 가장 성능이 좋았던 gbr의 RMSE값보다 성능이 좋게 나왔다. 이제 튜닝을 하고 전체 학습 데이터로 학습 후 예측을 진행해 보겠다. 튜닝한 결과 0.5712에서 0.5709로 아주 근소하게 좋아졌기 때문에 모델은 튜닝한 모델을 활용하였다.

blend\_tune = tune\_model(blend)

final\_model = finalize\_model(blend\_tune)

prediction = predict\_model(final\_model)

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

단일 모델을 활용한 것 보다 섞은 모델을 사용한 것이 성능이 더 좋게 나왔다. (0.4558 -> 0.3649)

그렇다면 이제 마지막으로 상위 6개 모델을 섞어서 더 좋은 성능을 나타내는 지 확인해 보겠다.

상위 5개 모델을 섞을 때 모델 생성중에 오류가 나던 Light Gradient Boosting Machine을 제외하였고 5개 모델은 다음과 같다. gbr, rf, Extra Trees Regressor(이하 et), Bayesian Ridge(이하 br), Orthogonal Matching Pursuit(이하 omp), Extreme Gradient Boosting(이하 xgboost)

et = create\_model('et')

br = create\_model('br')

omp = create\_model('omp')

xgboost = create\_model('xgboost')

blend2 = blend\_models(estimator\_list= [gbr, rf, et, br, omp, xgboost])

합쳐진 모델의 RMSE가 0.5671이 나오면서 다시 한번 더 성능이 좋아졌다. 이제 이를 튜닝을 하고 전체 학습데이터로 학습 후 테스트 데이터 예측을 진행해 보겠다. 튜닝한 결과 0.5671에서 0.5640으로 아주 근소하게 좋아졌기 때문에 모델은 튜닝한 모델을 활용하였다.

blend2\_tune = tune\_model(blend2)

final\_model = finalize\_model(blend2\_tune)

prediction = predict\_model(final\_model)

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

그 결과 지금까지 모델 중에서 가장 성능이 좋게 나왔다. (0.3649 -> 0.2915)

여기서 추가적으로 xgboost 다음으로 성능이 좋은 ridge를 섞은 모델을 만들었으나 튜닝하기 전 RMSE가 0.5693이 나오면서 xgboost까지 섞었던 모델(튜닝 전: 0.5671)이 성능이 더 좋아 사용하지 않았다.

**결론**

단일 모델을 활용했을 때는 Gradient Boosting Regressor이 가장 좋은 성능을 보여줬고 테스트까지 진행했을 때 RMSE가 0.4558이 나왔다.

모델들을 활용하여 Voting을 했을 때 단일 모델보다 좋은 성능을 자랑하였다.

모델 2개 활용 0.3649, 모델 6개 활용 0.2915

하지만 모델이 많아지면 많아 질수록 계산 속도가 더디고 성능이 무조건 상승하는 것은 아니다.

Ex) ridge를 추가를 했을 때 오히려 성능이 감소함 0.5671 -> 0.5693(ridge 추가)

**모든 결과를 종합해봤을 때 가장 좋은 성능을 보여주는 것은 RMSE가 0.2915가 나온 Voting Regressor 모델(Gradient Boosting Regressor, Random Forest Regressor, Extra Trees Regressor, Bayesian Ridge, Orthogonal Matching Pursuit, Extreme Gradient Boosting)이다.**

**전체코드**

!pip install pycaret

!pip install --upgrade pycaret

from pycaret.regression import \*

from google.colab import drive

drive.mount('/content/drive')

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

result = pd.read\_csv("/content/drive/MyDrive/비즈니스 머신러닝/final\_project\_hotelReview.csv", encoding="CP949")

result = result.drop(columns = "Unnamed: 0")

result = result.drop(columns = "hotel\_url")

result = result.drop(columns = "name")

result.loc[result['class'] == "no stars", 'class'] = 0

result = result.astype({"class":int})

result["class"]

result = result.dropna()

model = setup(data = result, target = "bubble\_rating",session\_id=123, train\_size=0.8,use\_gpu=True)

compare\_models()

gbr = create\_model('gbr')

gbr\_tune = tune\_model(gbr)

final\_model = finalize\_model(gbr)

prediction = predict\_model(final\_model)

rf = create\_model('rf')

rf\_tune = tune\_model(rf)

blend = blend\_models(estimator\_list= [gbr, rf])

blend\_tune = tune\_model(blend)

final\_model = finalize\_model(blend\_tune)

prediction = predict\_model(final\_model)

et = create\_model('et')

br = create\_model('br')

omp = create\_model('omp')

blend2 = blend\_models(estimator\_list= [gbr, rf, et, br, omp])

blend2\_tune = tune\_model(blend2)

final\_model = finalize\_model(blend2\_tune)

prediction = predict\_model(final\_model)

xgboost = create\_model('xgboost')

blend2 = blend\_models(estimator\_list= [gbr, rf, et, br, omp, xgboost])

blend2\_tune = tune\_model(blend2)

final\_model = finalize\_model(blend2\_tune)

prediction = predict\_model(final\_model)