Data scientist/Dacon

[Dacon] 2020 D CUP Google Analytics 데이터 1위 코드 분석

2021. 6. 16. 17:10 수정 삭제 공개

2020 D CUP Google Analytics 데이터

이 대회는 Dacon에서 진행한 대회로서 **과거의 데이콘 데이터를 활용한 미래의 사용자 행동 패턴을 예측** 하는 대회였다.
데이터 자체는 ga 기반의 데이터이여서 깔끔했다.

데이터

데이터 형태는 이런 식이였고 안에 데이터 형태는

	DateTime	사용자	세션	신규방문자	페이지뷰
0	2018-09-09 00:00:00	19	19	8	206
1	2018-09-09 01:00:00	20	19	9	259
2	2018-09-09 02:00:00	12	9	1	48
3	2018-09-09 03:00:00	10	10	2	102
4	2018-09-09 04:00:00	6	5	3	18

이와 같았다.

평가

심사 기준: Weighted RMSE

사용자 수, 세션 수, 신규 방문자 수, 페이지 뷰 수 4가지 항목을 예측하는 대회입니다.
각 변수의 크기가 다르기 때문에 가중치를 부여한 RMSE로 모델의 성능을 평가합니다.

소스

Private 1위, Private 점수 1.60023점, Linear Regression

(소스는 1위했던 분의 소스 였습니다.)

```
import os
os.chdir('/content/drive/MyDrive/dacon_cup/2차_open_data/open_data')

import pandas as pd
import numpy as np

import warnings
warnings.filterwarnings(action='ignore')

import seaborn as sns
import matplotlib.pyplot as plt
import plotly.graph_objs as go
```

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score

from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
#!pip install lightgbm
from lightgbm import LGBMRegressor

# import zipfile
# with zipfile.ZipFile("open_data.zip") as zf:
#   zf.extractall()
```

일반적인 머신러닝 기법들을 사용 한 것 같다.

```
train1 = pd.read_csv('1차_open_data/train.csv', encoding='cp949')
train2 = pd.read_csv('2차_open_data/2차_train.csv', encoding='cp949')
train = pd.concat([train1,train2],axis=0)
submission = pd.read_csv('1차_open_data/submission.csv', encoding='cp949')

info_comp = pd.read_csv('2차_추가데이터/new_competition_info.csv', encoding='cp949')
info_login = pd.read_csv('2차_추가데이터/new_login_info.csv', encoding='cp949')
info_user = pd.read_csv('2차_추가데이터/new_user_info.csv', encoding='cp949')
info_sub = pd.read_csv('2차_추가데이터/new_submission_info.csv', encoding='cp949')

train.head()
```

```
# train 데이터 일별로 summary

train['DateTime'] = pd.to_datetime(train['DateTime'])
train['date'] = train['DateTime'].dt.date
train  = train.groupby('date').sum().reset_index()
train.tail()
```

```
# 모든 값이 결측치인 행은 제거

comp = info_comp.dropna(how='all')
login = info_login.dropna(how='all')
user = info_user.dropna(how='all')
sub = info_sub.dropna(how='all')
```

여기서는 결측치 자체를 모두 삭제해줬다. 결측치 관해서는 모두 다른것 같다.
결측치로 해당 되는 부분이 적을 경우에는 해당 행자체를 삭제하고
컬럼수준에서 결측치가 많은 경우에는 컬럼자체를 삭제하는 것 같다.
결측치 처리 방법 자체가 딱히 정해진게 없고 너무 각각 다르게 사용하고 있어 이 부분에 대해서는 데이터에 따라 사용자가 어떻게 정하냐에 따라 달라지는 것 같다.
결측치에 대한 좀 더 자세한 내용은 블로그에 게시해놨다.
https://continuous-development.tistory.com/157

```
# comp 데이터 일별로 summary

comp['period_start'] = pd.to_datetime(comp['period_start'])
```

```
comp['period_end'] = pd.to_datetime(comp['period_end'])

comp['start_date'] = comp['period_start'].dt.date
comp['end_date'] = comp['period_end'].dt.date

start_comp_fe = comp.groupby('start_date').count().reset_index()[['start_date','cpt_id']]
start_comp_fe.columns = ['date','start_comp_count']

end_comp_fe = comp.groupby('end_date').count().reset_index()[['end_date','cpt_id']]
end_comp_fe.columns = ['date','end_comp_count']

start_comp_fe.head() # 날짜별 대회 시작 수
```

여기서는 날짜별 데이터를 summary 해줬다. 일자를 기준으로 잡고 하려고 했던 것 같다.

```
end_comp_fe.head() # 날짜별 대회 마감 수
```

```
# login 데이터 일별로 summary

login['c_time'] = pd.to_datetime(login['c_time'])
login['date'] = login['c_time'].dt.date
login_fe = login.groupby('date').nunique().reset_index().drop(['Unnamed: 0','c_time'],axis=1)
login_fe.columns = ['date','login_count','login_user','login_platform','login_browser']
login_fe.head() # 날짜별 총 로그인 수, 접속한 유저 수, 사용한 플랫폼 수, 사용한 브라우저 수

# user 데이터 일별로 summary

user['c_time'] = pd.to_datetime(user['c_time'])
user['date'] = user['c_time'].dt.date
user_fe = user.groupby('date').count()['id'].reset_index()
user_fe.columns = ['date','user_count']
user_fe.head() # 날짜별 가입 유저 수

# submission 데이터 일별로 summary

sub['c_time'] = pd.to_datetime(sub['c_time'])
sub['date'] = sub['c_time'].dt.date
sub_fe = sub.groupby('date').nunique().reset_index().drop(['Unnamed: 0','c_time'],axis=1)
sub_fe.columns = ['date','sub_count','sub_cpt','sub_team','sub_user']
sub_fe.head() # 날짜별 총 제출 횟수, 제출이 발생한 대회 수, 제출한 팀 수, 제출한 유저 수
```

각기 다른 컬럼들도 일별로 summary 해줬다.

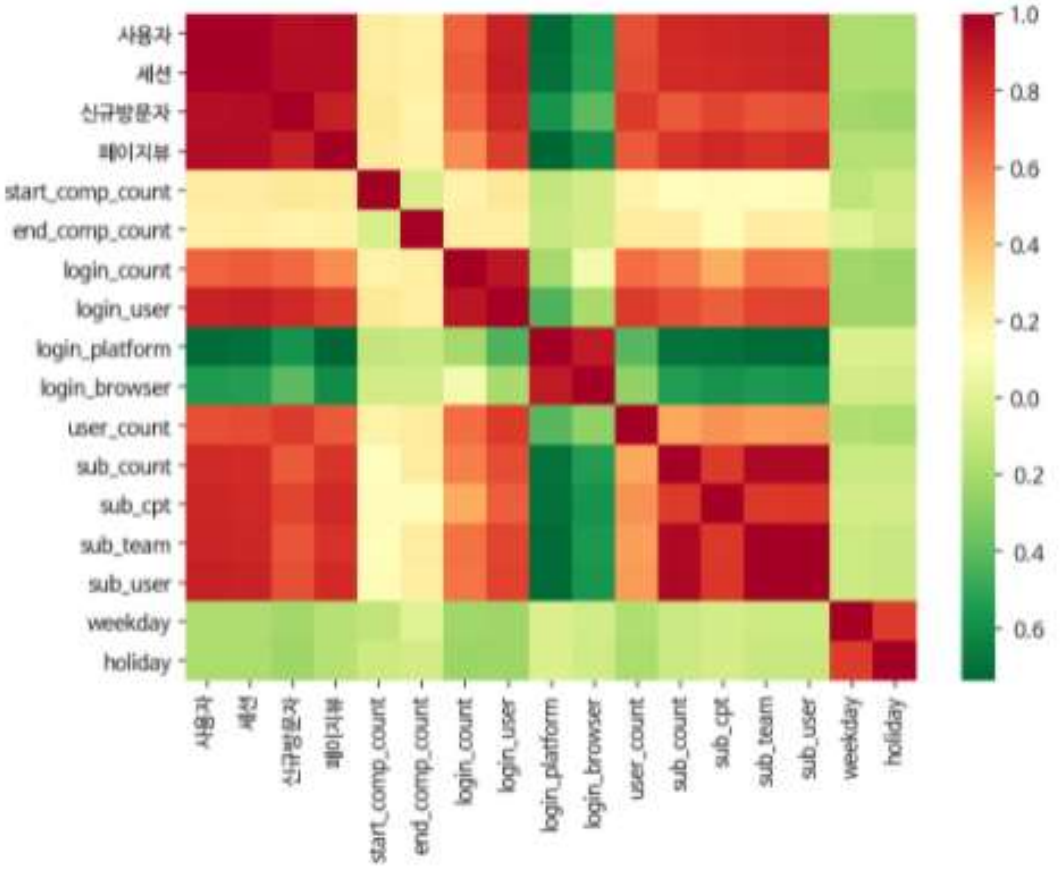
```
# date 기준으로 feauture 전부 결합하여 최종 train 데이터셋 구축

train_X = pd.merge(train,start_comp_fe,on='date',how='left')
train_X = pd.merge(train_X,end_comp_fe,on='date',how='left')
```

```
train_X = pd.merge(train_X,login_fe,on='date',how='left')
train_X = pd.merge(train_X,user_fe,on='date',how='left')
train_X = pd.merge(train_X,sub_fe,on='date',how='left')

# 요일 및 주말여부 feature 생성
train_X['weekday'] = pd.to_datetime(train_X['date']).dt.weekday  # 요일
train_X['holiday'] = [1 if (x == 5) | (x==6) else 0 for x in train_X['weekday']]  # 주말 여부

train_X = train_X.fillna(0)

train_X.head()
```

데이터를 합쳤고 추가적으로 변수를 생성해준 부분은 시계열이여서 그런지 요일과 휴일 정도였다.

```
# test 데이터셋도 동일하게 생성

submission['DateTime'] = submission['DateTime'].astype(str)
start_comp_fe['date'] = start_comp_fe['date'].astype(str)
end_comp_fe['date'] = end_comp_fe['date'].astype(str)
login_fe['date'] = login_fe['date'].astype(str)
user_fe['date'] = user_fe['date'].astype(str)
sub_fe['date'] = sub_fe['date'].astype(str)

temp_sub = submission[['DateTime']]
temp_sub.columns = ['date']

test_X = pd.merge(temp_sub,start_comp_fe,on='date',how='left')
test_X = pd.merge(test_X,end_comp_fe,on='date',how='left')
test_X = pd.merge(test_X,login_fe,on='date',how='left')
test_X = pd.merge(test_X,user_fe,on='date',how='left')
test_X = pd.merge(test_X,sub_fe,on='date',how='left')

# 요일 및 주말여부 feature 생성
test_X['weekday'] = pd.to_datetime(test_X['date']).dt.weekday
test_X['holiday'] = [1 if (x == 5) | (x==6) else 0 for x in test_X['weekday']]

test_X = test_X.fillna(0)

test_X.head()
```

```
plt.figure(figsize=(8,6))
sns.heatmap(train_X.corr(), cmap='RdYlGn_r')

# 사용자,세션,신규방문자,페이지뷰는 서로 비슷한 양상을 보임
# 날짜별 총 로그인 수, 접속한 유저 수, 가입 유저 수, 총 제출 수, 제출이 발생한 대회 수, 제출한 팀 수
```




```
# 독립변수와 종속변수 분할

y_user = train_X.iloc[:,1]
y_session = train_X.iloc[:,2]
y_visit = train_X.iloc[:,3]
y_view = train_X.iloc[:,4]

X_train = train_X.iloc[:,5:]
X_test = test_X.iloc[:,1:]

# 최소-최대 정규화 수행

ms = MinMaxScaler()
X_train_m = pd.DataFrame(ms.fit_transform(X_train))
X_test_m = pd.DataFrame(ms.transform(X_test))
```

여기서는 minmaxScaler를 사용하였다. 일반적으로 많이 사용한 것을 사용한 것 같다.

```
# 개별 모델 성능 (RMSE) 확인

knn_model = KNeighborsRegressor()
svr_model = SVR()
lr_model = LinearRegression()
rf_model = RandomForestRegressor()
lgb_model = LGBMRegressor()

cross_score_knn = cross_val_score(knn_model, X_train_m, y_user, scoring='neg_mean_squared_error', cv=
cross_score_lr = cross_val_score(lr_model, X_train_m, y_user, scoring='neg_mean_squared_error', cv=5)
cross_score_svr = cross_val_score(svr_model, X_train_m, y_user, scoring='neg_mean_squared_error', cv=5
cross_score_rf = cross_val_score(rf_model, X_train_m, y_user, scoring='neg_mean_squared_error', cv=5)
```

```
cross_score_lgb = cross_val_score(lgb_model, X_train_m, y_user, scoring='neg_mean_squared_error', cv=


r_score_knn = np.sqrt(-cross_score_knn)
r_score_lr = np.sqrt(-cross_score_lr)
r_score_svr = np.sqrt(-cross_score_svr)
r_score_rf = np.sqrt(-cross_score_rf)
r_score_lgb = np.sqrt(-cross_score_lgb)

print("KNN score :",r_score_knn, " / KNN mean score", r_score_knn.mean())
print("LR score :",r_score_lr, " / LR mean score", r_score_lr.mean())
print("SVR score :",r_score_svr, " / SVR mean score", r_score_svr.mean())
print("RF score :",r_score_rf, " / SVR mean score", r_score_rf.mean())
print("LGB score :",r_score_lgb, " / SVR mean score", r_score_lgb.mean())

KNN score : [319.5788471  186.80204009 451.77081579 493.64866123 914.11989847]  / KNN mean
LR score : [207.1418869  236.50166626 341.35193628 256.32162566 399.16717389]  / LR mean scor
SVR score : [ 980.54278522  838.44540524  467.66577595 1300.37450078 2315.99100601]  / SVR m
RF score : [276.72644792 176.07377115 552.71587197 323.83386142 686.562731  ]  / SVR mean sco
LGB score : [247.03656244 169.92058529 444.31790294 370.21512513 688.04037992]  / SVR mean s
```

총 5개의 모델을 사용 하였고 부스팅 계열도 들어가 있었다. 그중 LR이 성능이 제일 잘나왔다.

```
# 선형회귀 모델 적합 및 예측 - 사용자

lr_model = LinearRegression()
lr_model.fit(X_train_m,y_user)
submission['사용자'] = lr_model.predict(X_test_m)
submission.head()

# 선형회귀 모델 적합 및 예측 - 세션

lr_model = LinearRegression()
lr_model.fit(X_train_m,y_session)
submission['세션'] = lr_model.predict(X_test_m)
submission.head()

# 선형회귀 모델 적합 및 예측 - 신규방문자

lr_model = LinearRegression()
lr_model.fit(X_train_m,y_visit)
submission['신규방문자'] = lr_model.predict(X_test_m)
submission.head()

# 선형회귀 모델 적합 및 예측 - 페이지뷰

lr_model = LinearRegression()
lr_model.fit(X_train_m,y_view)
submission['페이지뷰'] = lr_model.predict(X_test_m)
submission.head()

submission.to_csv('submission'.csv',index=False,encoding='cp949')
```

성능이 가장 좋았던 모델인 lr로 하나씩 예측을 하였고 나중에 저장을 했다.

정리(사용한 것들)

1. 시간 데이터를 일자로 summary
2. 각 컬럼에 모두 똑같이 summary
3. 요일 및 주말 컬럼 생성

후기

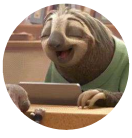
생각보다 기본적으로 했는데 성능이 높게 나온게 의외였다. 또한 모델 자체도 lr 이였는데 높게 나왔다는 것도 신기했다. 보통은 부스팅 계열이 가장 높게 나왔었는데 확실히 데이터에 따라서 모델 자체의 성능이 크게 바뀌는것 같다.
변수 생성 부분에서는 데이터가 워낙 깔끔해서 아마 시간에 관련 한 파생변수 말고는 딱히 없었을 것 같다.
여기서는 하이퍼 파라미터를 찾기위해 무언가를 썼다던지는 안나와 있는데 하이퍼 파라미터를 찾기위해 다른 무언가를 썼다면 성능이 조금 더 오르지 않았을까라는 생각을 한다.
이 작업을 조금 더 해봐야겠다. 그리고 사용되지 않았던 다른 모델도 사용해볼 생각이다.
시계열 데이터에서는 요일 주말로 나누는 작업은 기본적으로 꼭 들어가는 것 같다. 다른것은 일 월 까지도 들어가고 기타 등등 여러가지 기본적으로 하는 작업들이 있는 것 같다.

이 소스와 모든 내용의 출처는 Dacon 입니다.
(https://dacon.io/competitions/official/235683/codeshare/2341?page=1&dtype=recent)

'Data scientist > Dacon' 카테고리의 다른 글

[Dacon] 2020 D CUP Google Analytics 데이터 1위 코드 분석

DACON rmse



나아무늘보
혼자 끄적끄적하는 블로그 입니다.