TalkingData AdTracking Fraud Detection

E조 발표자료
이재후, 임원균, 박종민, 신종환, 이봉호, 전주홍

Contents

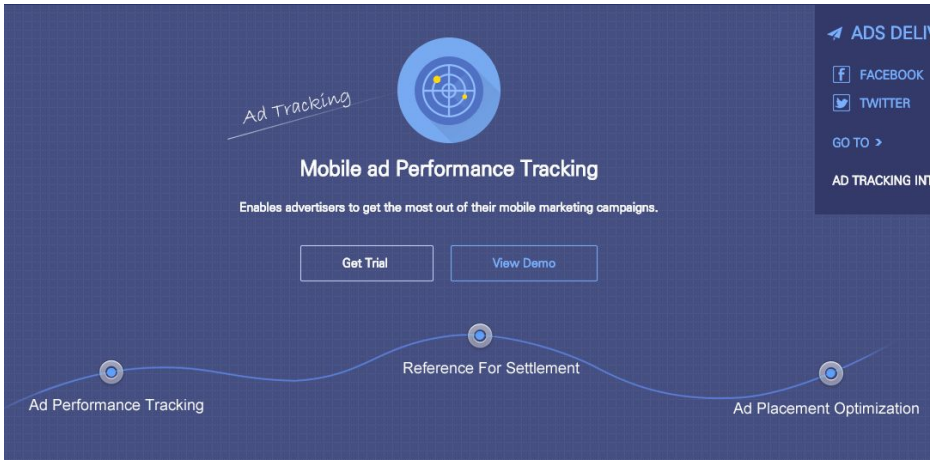
Mobile AD Performance Tracking

What is it?

- 앱을 광고시 광고 채널 별로 광고 실적을 파악

What we need to do?

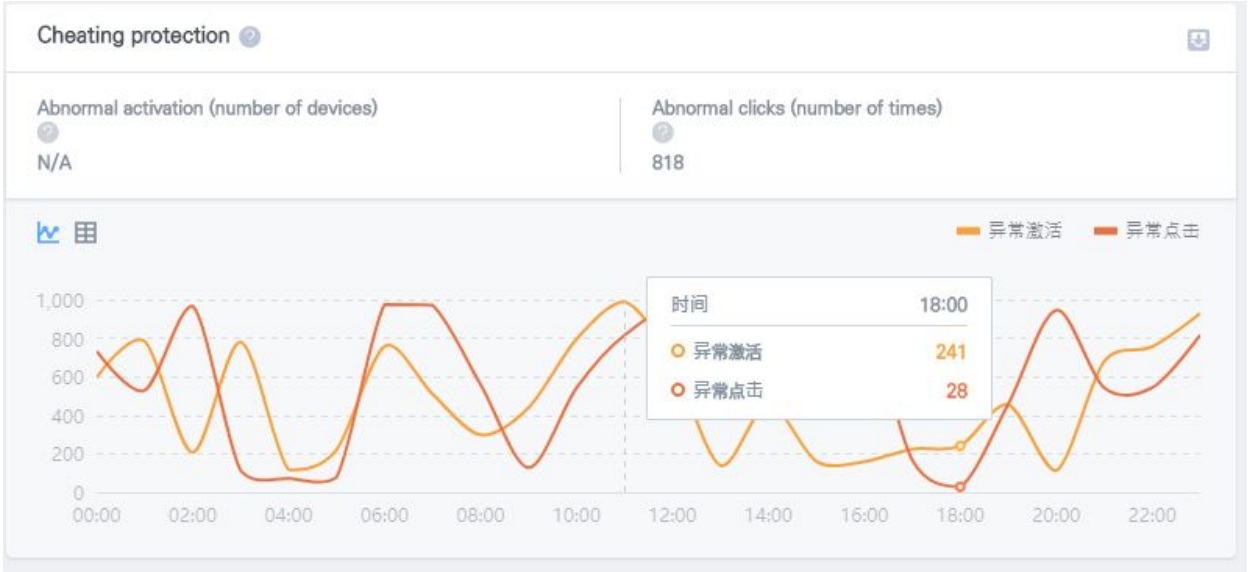
- 앱을 다운로드할 (is_attributed) 사용자 예측



Mobile AD Performance Tracking

TalkingData는 광고의 Performance 측정 솔루션 제공

솔루션의 Cheating Protection 기능



EDA 수행 결과

변수 및 데이터 설명

전체 데이터 설명

- 약 4일 간의 2억개 사용자 클릭 로그

변수 설명
- `ip`: 클릭한 폰의 IP 주소
- `app`: 광고가 켜지는 앱의 ID
- `device`: 모바일 폰 타입
- `os`: 모바일 폰의 OS 버전
- `channel`: 광고 채널(e.g. facebook, google ad 등등)
- `click_time`: 광고를 클릭한 시간(UTC)
- `attributed_time`: 앱 다운로드 시 다운로드 시간
- `is_attributed`: 앱의 실제 다운로드 여부 (실제 추정 값)

변수 및 데이터 설명

전체 데이터 Describe
- **공통사항**
- Ip,app,device,os,channel 는 encoded value
- 모든 필드 Category type 지정 필요

- **Train Dataset**
- 전체 row: 1억 8천개
- Is_attributed NULL X

- **Test Dataset**
- 전체 row: 6400만 개
- Is_attributed, attributed_time X

- 압축된 1.8Gb 의 학습 데이터는 압축을 해제하면 무려 7.5Gb

```
# File sizes
  train.csv                       7537.65MB
  test.csv                        863.27MB
  sample_submission.csv           195.58MB
  train_sample.csv                4.08MB

  # Line count:
  184903891 ../input/train.csv
  18790470 ../input/test.csv
  100001 ../input/train_sample.csv
```

top 는 most common value 를 의미합니다. 그리고 그 반복횟수가 freq 입니다.

```
train.describe()
```

	ip	app	device	os	channel	click_time	attributed_time	is_attributed
count	184903890.0	184903890.0	184903890.0	184903890.0	184903890.0	184903890	456846	184903890.0
unique	277396.0	706.0	3475.0	800.0	202.0	259620	182057	2.0
top	5348.0	3.0	1.0	19.0	280.0	2017-11-07 14:00:11	2017-11-08 14:16:29	0.0
freq	1238734.0	33911780.0	174330052.0	44181914.0	15065927.0	1502	14	184447044.0
first	NaN	NaN	NaN	NaN	NaN	2017-11-06 14:32:21	2017-11-06 16:00:47	NaN
last	NaN	NaN	NaN	NaN	NaN	2017-11-09 16:00:00	2017-11-09 16:00:00	NaN

```
test.describe()
```

	click_id	ip	app	device	os	channel	click_time
count	57537505.0	57537505.0	57537505.0	57537505.0	57537505.0	57537505.0	57537505
unique	57537505.0	126414.0	536.0	3032.0	607.0	186.0	86945
top	57537504.0	5348.0	9.0	1.0	19.0	107.0	2017-11-10 09:00:10
freq	1.0	504147.0	9066687.0	53545440.0	13424328.0	3673594.0	1466
first	NaN	NaN	NaN	NaN	NaN	NaN	2017-11-09 14:23:39
last	NaN	NaN	NaN	NaN	NaN	NaN	2017-11-10 16:00:00

EDA 전략

Focus #1 - **Time**

Focus #2 - **IP and User-agent**

Focus #3 - **Channel**

Time에 따른 데이터 분석

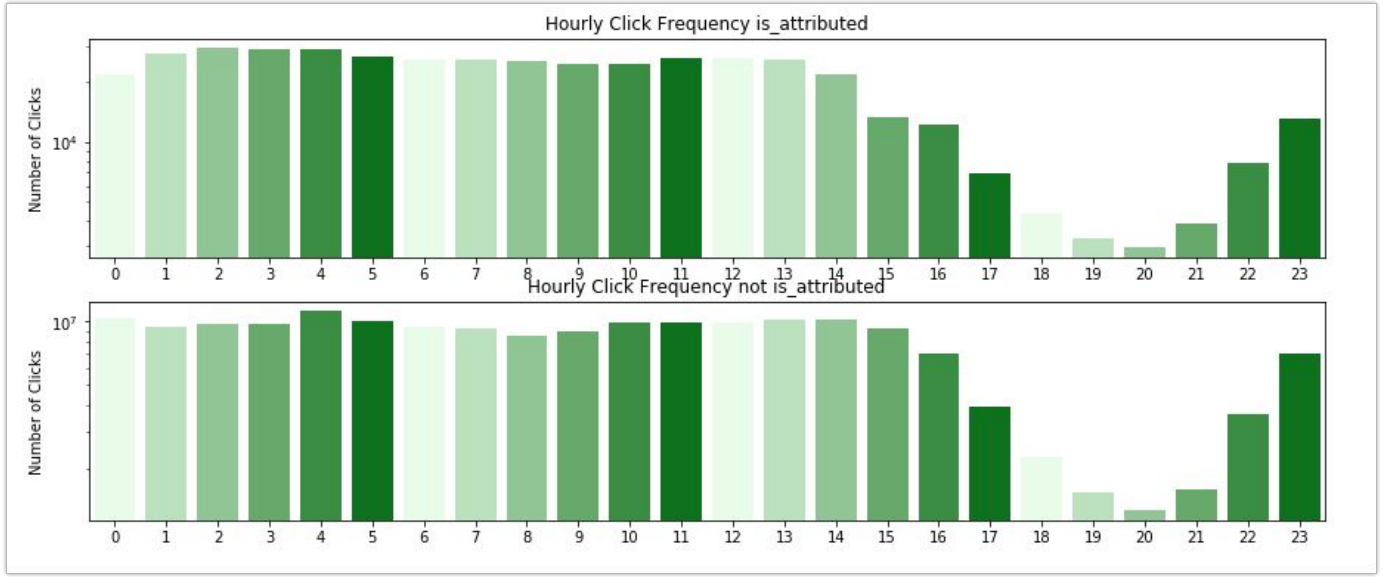
- 가정1- 정상적인 유저건 fraud를 주는 작업장이던 특정 시간대에 다운로드로 전환되는 비율이 높을 것이다.
- 가정2- app별로도 클릭과 전환율이 높은 특정 시간대가 있을 것이다.

실제로 시간별로 다운로드 전환율을 확인해보고, 앱의 시간별 누적 클릭에 따른 수도 확인해 본다.

Time에 따른 데이터 분석

시간대별로 다운로드된 패턴과 다운로드 되지 않은 패턴에 차이가 있는지 확인하기

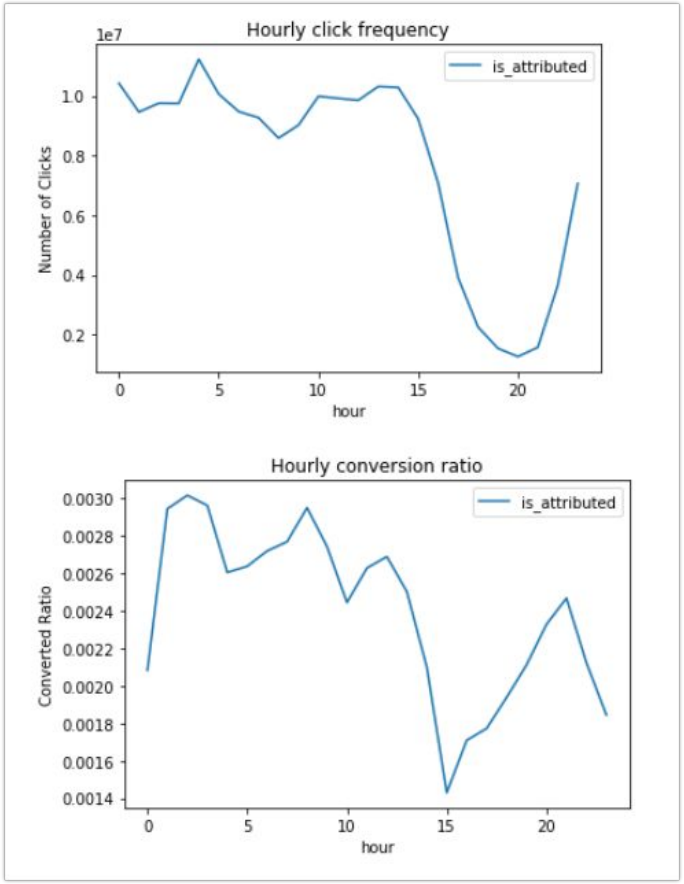
-> 차이를 보기 힘들었다



Time에 따른 데이터 분석

시간별 클릭 빈도와
다운로드로 전환되는 비율
확인

- 특정 시간대에는 확연히
전환율이 떨어지는 구간이
존재하는 것을 확인



Time에 따른 데이터 분석

시간별 클릭 빈도와
다운로드로 전환되는 비율
확인

- Bar 그래프에서 전환율이
0.0020이 되는 지점을
기점으로 삼아 구간을
다섯구간으로 나눈다

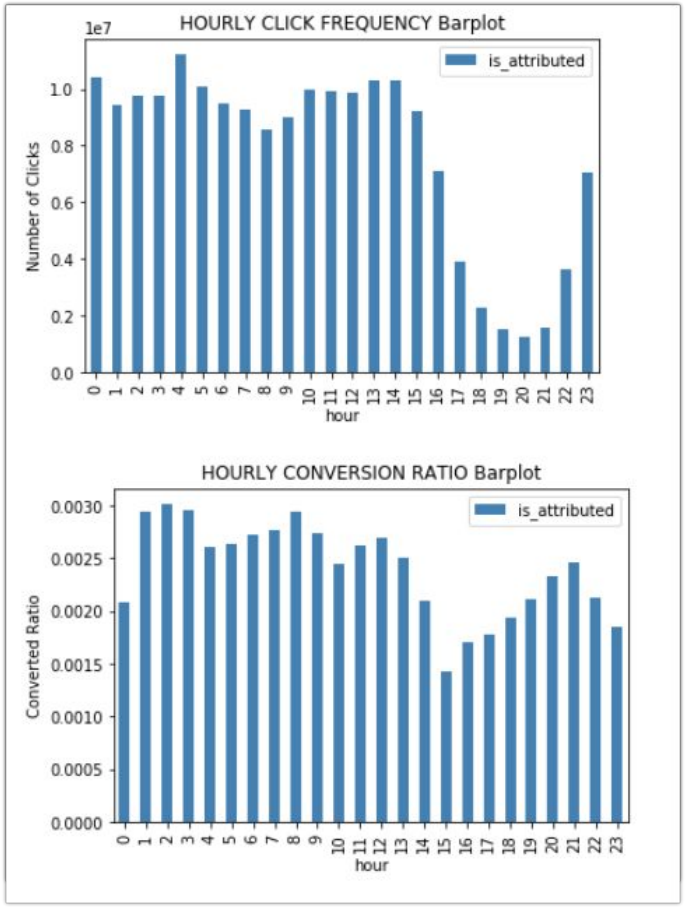
1시~7시 -> 0

8시~13시 -> 1

14시~19시 -> 2

20시~21시 -> 3

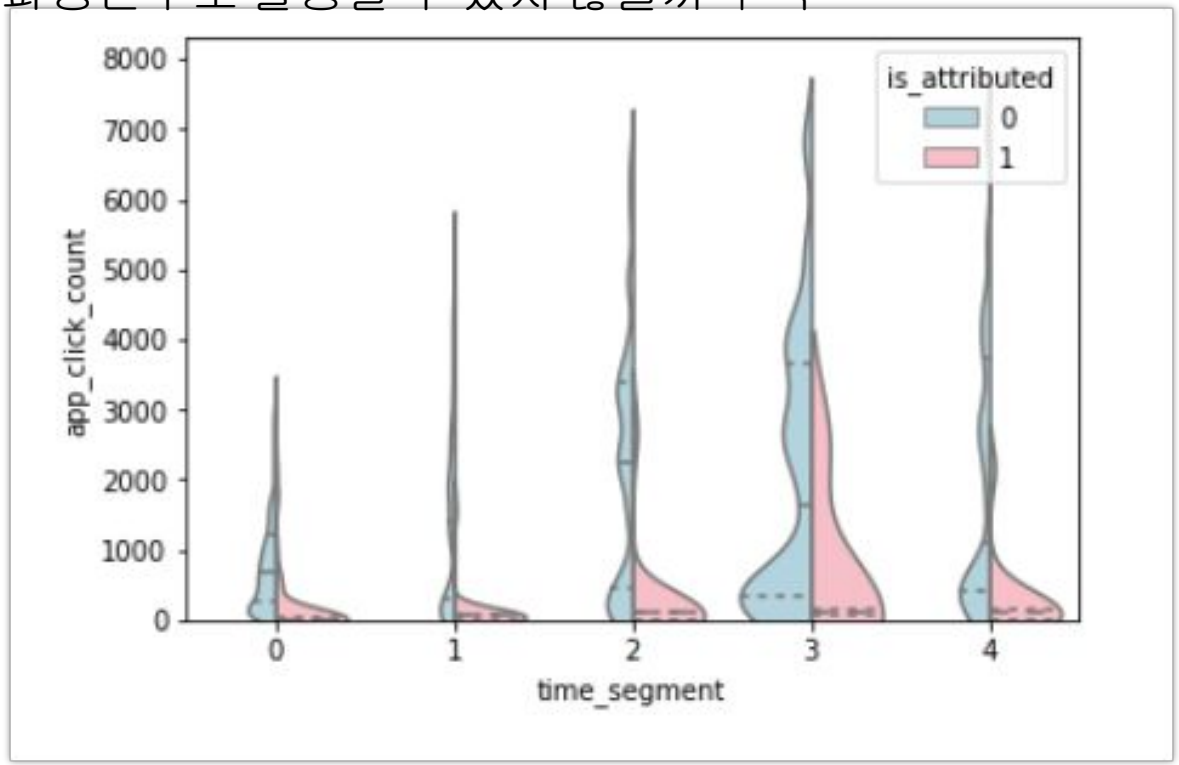
22시~0시 -> 4



Time에 따른 데이터 분석

나눈 구간마다 앱별로 클릭 수를 카운팅해서 이를 그래프로 만듦

-> 파생변수로 활용할 수 있지 않을까 추측



IP, OS, Device에 따른 데이터 분석

User-agent (OS, Device) 와 IP 는,

사용자가 HTTP Header, Proxy 를 통해 다소 쉽게 변조 가능함.

앱다운로드까지 진행한 사용자(예측 대상)가 이러한 변조 가능성이

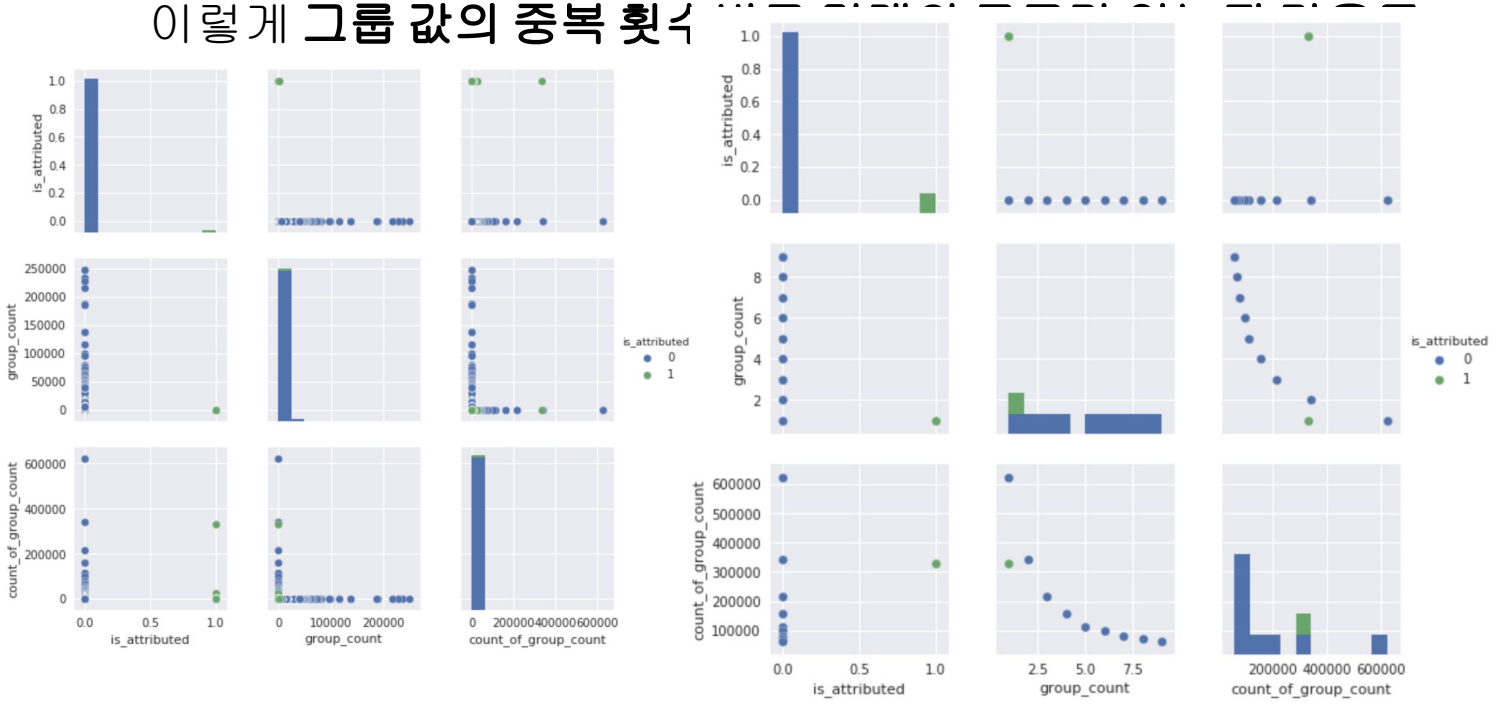
높다고 예상 가능하므로 전체 데이터셋에서 특정 사용자 로그의

“ (IP, OS, Device) 값의 유니크하거나 그에 가까울 수록 앱다운로드 사용자일 가능성이 높다. ” 고 가정

IP, OS, Device에 따른 데이터 분석

ip, device, os 그룹의 값이 전체 데이터셋에서 유니크한 것은 얼마나 존재하고, 한번 중복된 것은 얼마나 존재하고 두번 중복된것은...

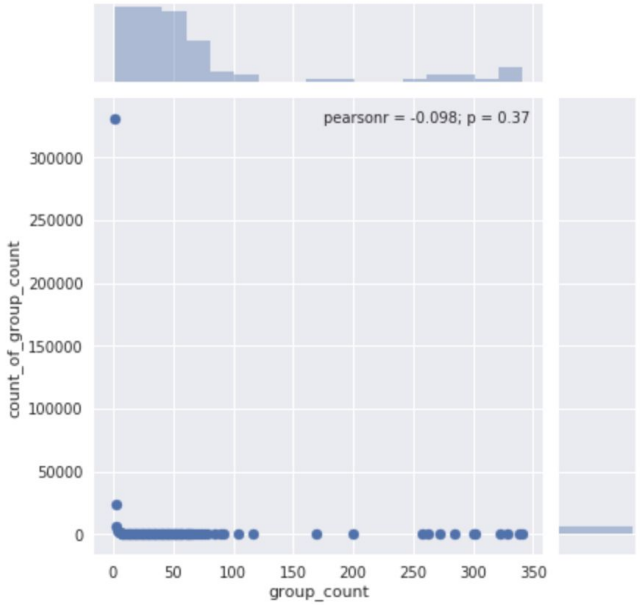
이렇게 **그룹 값의 중복 횟수**



IP, OS, Device에 따른 데이터 분석

ip, device, os 그룹의 값이 전체 데이터셋에서 유일한 경우는,

다운로드 사용자의 약 90% 를 차지 (한번 중복 포함시 약 **96%)**



count_of_group_count 368695
dtype: int64

	group_count	count_of_group_count
0	1	330615

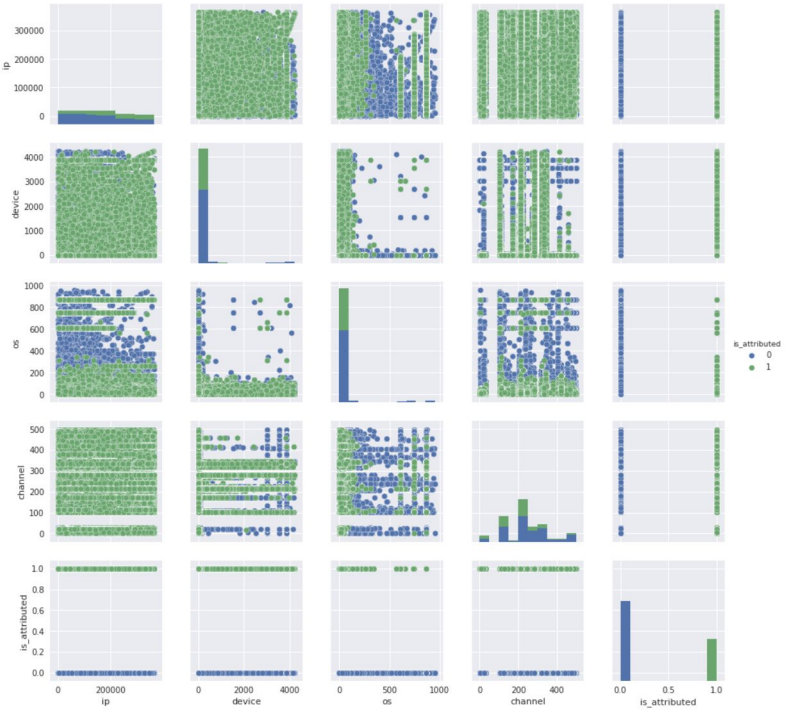
count_of_group_count 38080
dtype: int64

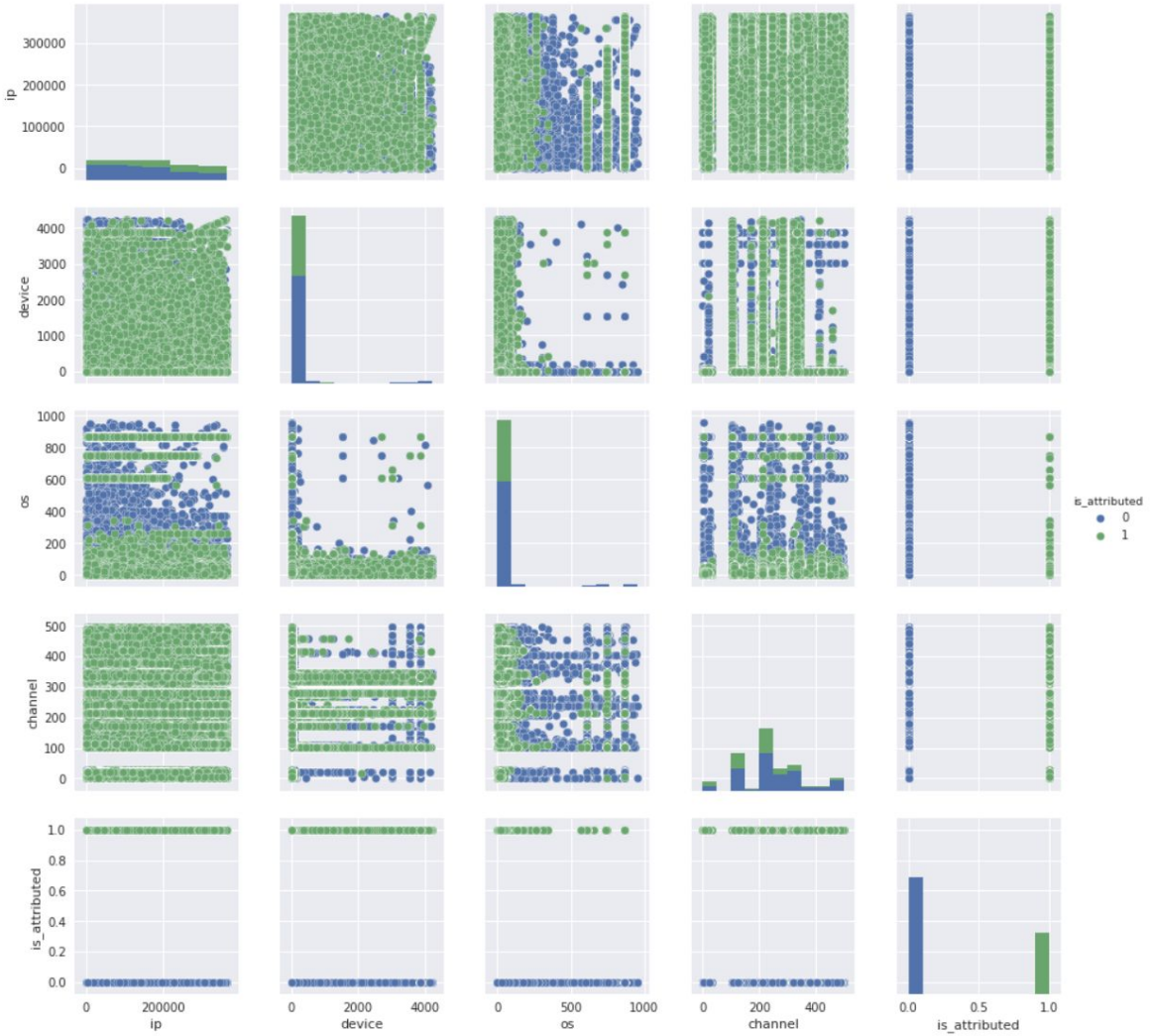
	group_count	count_of_group_count
1	2	23495

IP, OS, Device에 따른 데이터 분석

ip, device, os 그룹의 값이 전체 데이터셋에서 유일한 경우는,

다운로드 받지 않은 클릭도 역시 많기 때문에 추가적으로 둘을 구분할 수 있는 feature 를 찾자.





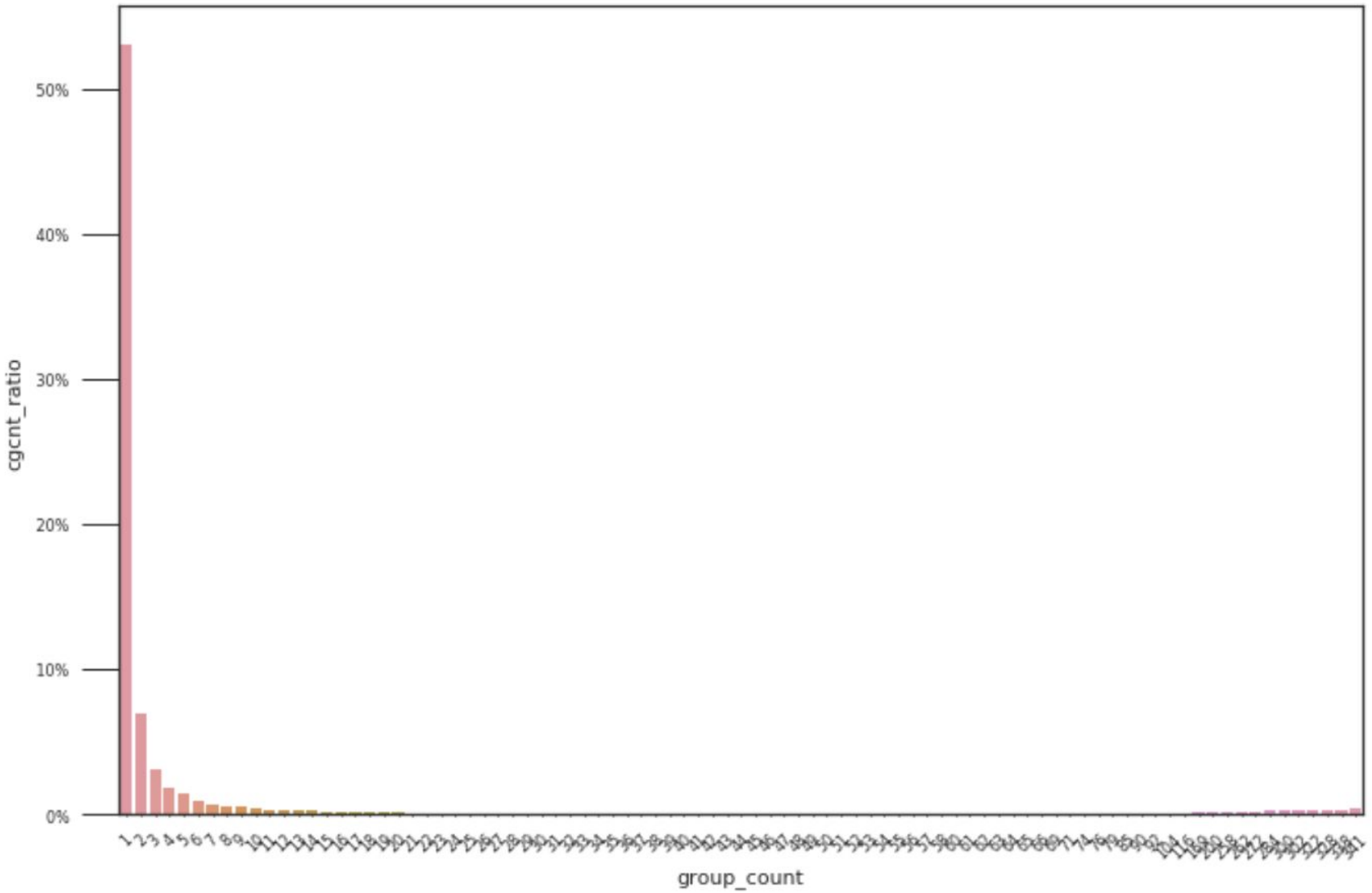
ip, device, os 그룹의 값이 전체 데이터셋에서 유일한 로그들의 Pair Plot

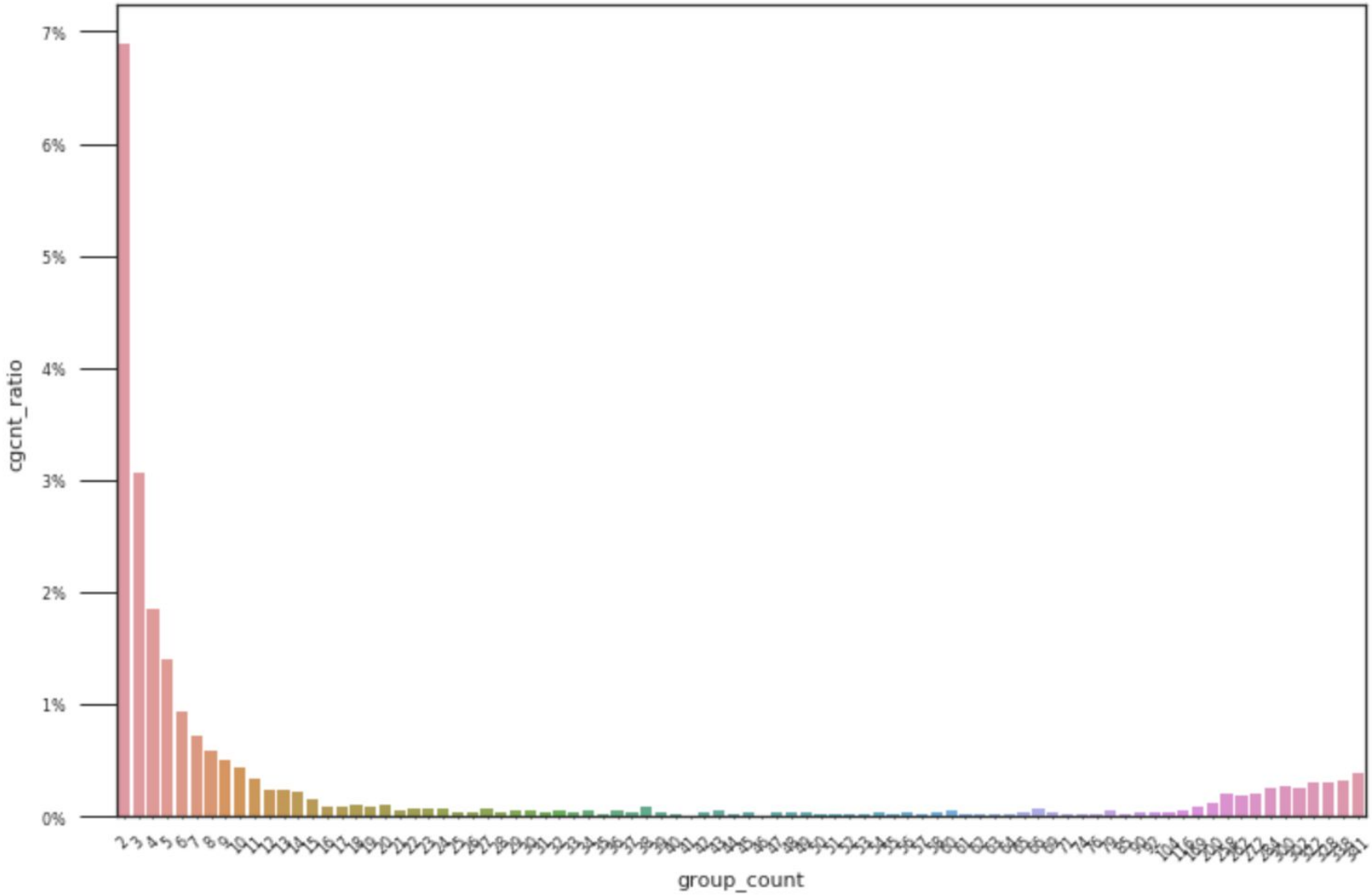
IP, OS, Device에 따른 데이터 분석

다운로드 받지 않은 클릭을 더 잘 구분할 feature 는 없으니,

다운로드 받지 않은 클릭과 다운로드 받은 클릭의 비율을 사용하자.

	group_count	is_down_cgcnt	no_down_cgcnt
0	1	330615	623589
1	2	23495	340932
2	3	6656	216894
3	4	2921	158405
4	5	1587	113827
5	6	938	100716
6	7	580	80693
7	8	418	71887
8	9	312	62364
9	10	247	55740


50%
40%
30%
20%
10%
0%
cgcnt_ratio
group_count


7%
6%
5%
4%
3%
2%
1%
0%
cgcnt_ratio
group_count

Channel에 따른 데이터 분석

왜 Channel을 선택했는가?

- 가정1- channel 광고 전략에 따른 Fraudulent click 발생
- 가정2- 광고 비용의 고의적인 증폭 시도는 channel 단위 수행

수행 작업

- Channel에 따른 download/click ratio(d/c ratio) 분석
  - Channel별 Click count 및 d/c ratio histogram
  - download/click regression plot
  - d/c ratio 및 click count를 이용한

Channel에 따른 데이터 분석

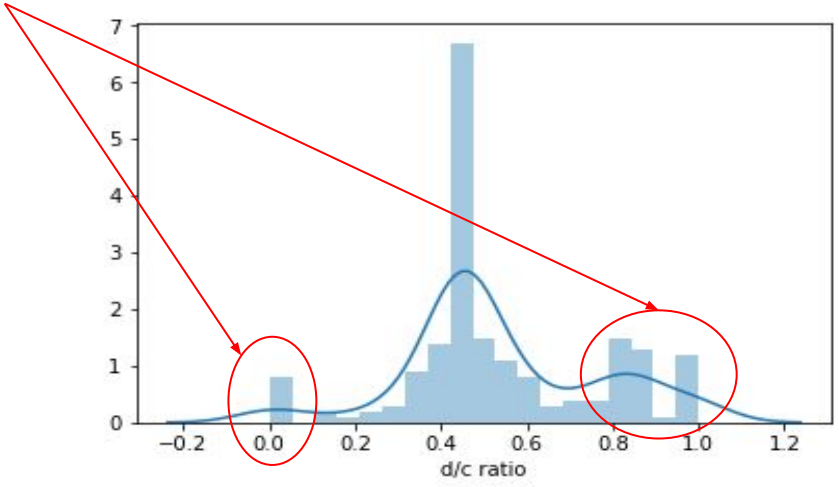
Channel별 d/c ratio 및 click count의 분포 확인

- 중위값 약 0.5, 중위값에 대부분의 채널 집중
- 전체 클릭수가 작은 channel(Low click volume channel)의 경우 0 및 1 근처에 존재하기도 함

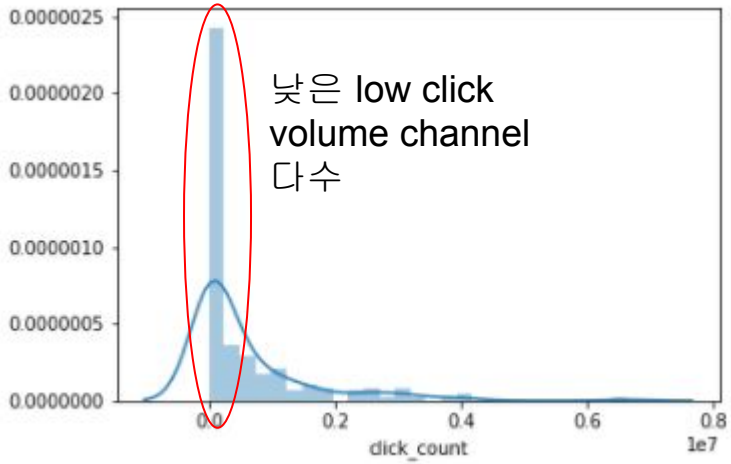
“Channel을 이용한 파생 변수 생성시 click volume 고려 필요”

양끝에 Low click volume 채널 많음

Distribution of d/c ratio



Distribution of click_count

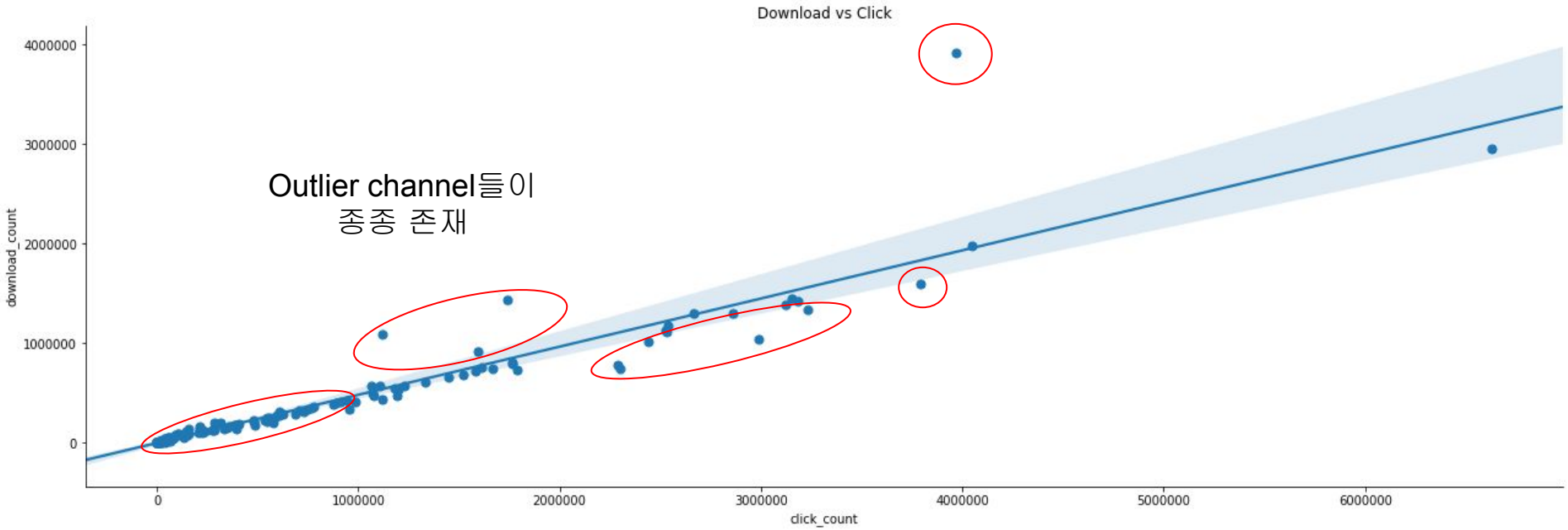


Channel에 따른 데이터 분석

Channel 별 Download(y축) / Click Regression(x축) plot

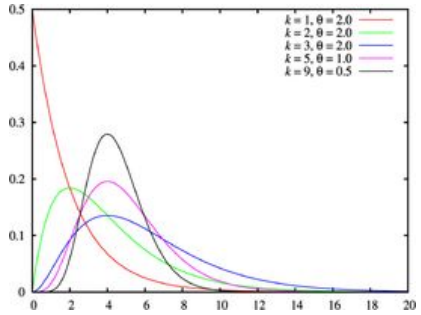
Regression Model에 맞지 않는 Outlier 채널들이 존재

“d/c ratio를 이용하여 Outlier 채널들의 구분 가능”

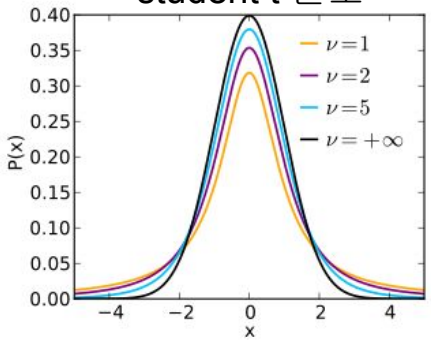


Channel에 따른 데이터 분석

Gamma 분포



student t 분포



Channel의 특징

- Channel 별 d/c ratio 평균 약 0.5
- Low click volume channel의 극단적인 d/c ratio
- d/c ratio의 Regression model에서 벗어나는 채널 존재

“click_count 및 d/c ratio 고려”

파생변수 제안

- Channel abnormality: d/c ratio와 channel size를 조합
  - d/c ratio ~ student t 분포
  - click_count ~ 감마 분포
  - 두 변수는 독립으로 가정(피어슨 상관 계수 -0.15)
  - Channal abnormality = d/c ratio 의 PDF * click_count 의 PDF

*PDF: 확률밀도함수

모델링

EDA 분석 결과

Time:

다운로드 전환율을 통해 시간을 구간지어 나눈 파생변수 제안

팀원 임원균

Channel:

d/c ratio 및 click_count 확률 모델링을 통한 새로운 파생변수 제안

팀원 신종환

IP, User-agent:

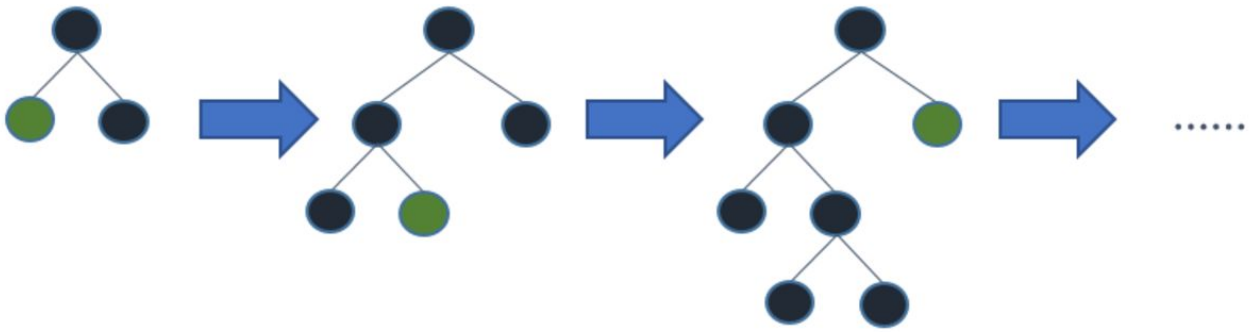
사용자를 특정하는 IP, User-agent 그룹 빈도수 별 다운로드 사용자 비율

팀원 박종민

분석 모델: LightGBM

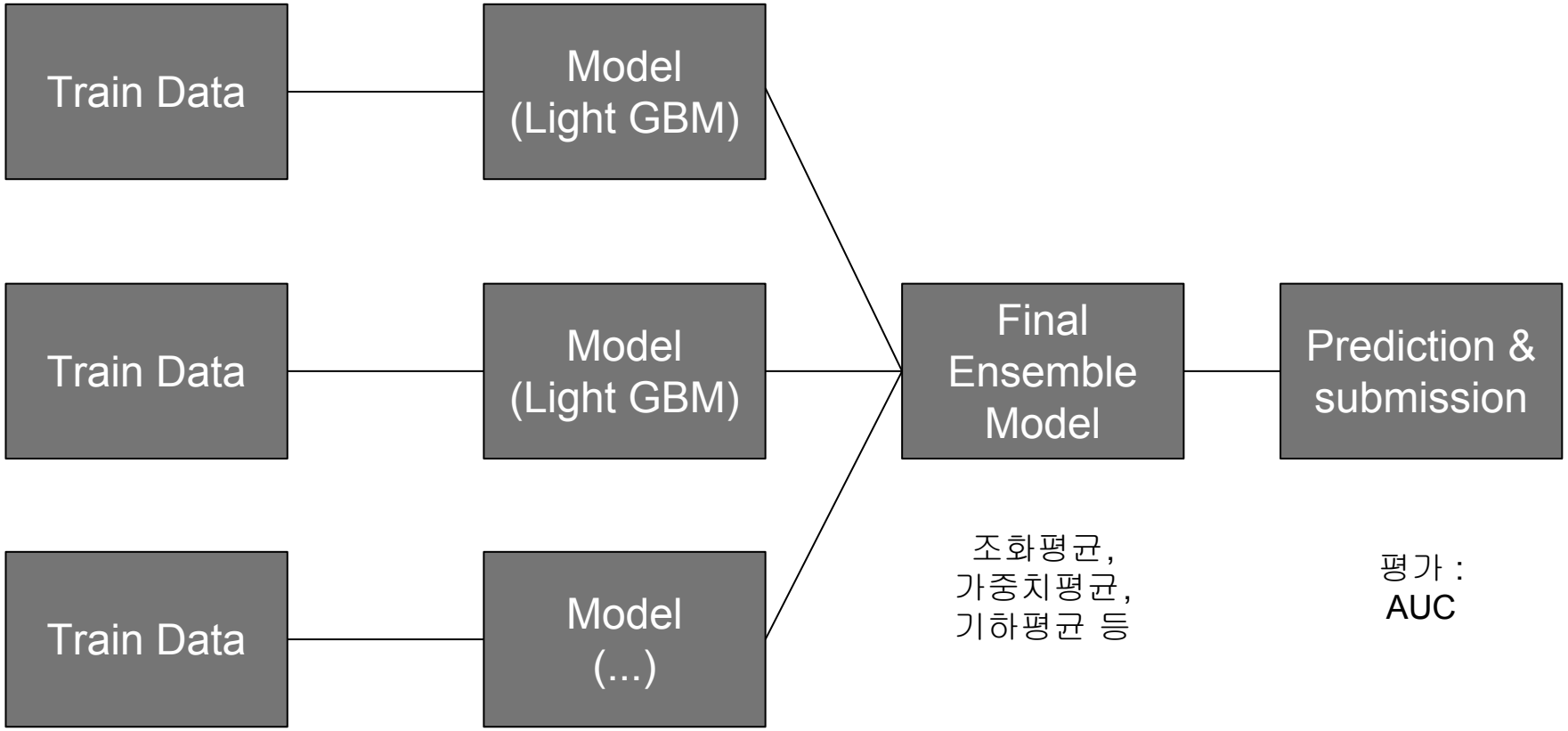
선정 이유:

- XGboost에 비해서 범주형 Value에 최적화되어 있음
- One Hot Encoding을 쓸 수 있는 형태의 데이터가 아니었음



Leaf-wise tree growth

분석 모델: LightGBM



```
def lgb_modelfit_nocv(params, dtrain, dvalid, predictors, target='target', objective='binary', metrics='auc',
                feval=None, early_stopping_rounds=20, num_boost_round=3000, verbose_eval=10, categorical_features=None):
    lgb_params = {
        'boosting_type': 'gbdt',
        'objective': objective,
        'metric':metrics,
        'learning_rate': 0.01,
        #'is_unbalance': 'true',  #because training data is unbalance (replaced with scale_pos_weight)
        'num_leaves': 31,  # we should let it be smaller than 2^(max_depth)
        'max_depth': -1,  # -1 means no limit
        'min_child_samples': 20,  # Minimum number of data need in a child(min_data_in_leaf)
        'max_bin': 255,  # Number of bucketed bin for feature values
        'subsample': 0.7,  # Subsample ratio of the training instance.
        'subsample_freq': 0,  # frequence of subsample, <=0 means no enable
        'colsample_bytree': 0.3,  # Subsample ratio of columns when constructing each tree.
        'min_child_weight': 5,  # Minimum sum of instance weight(hessian) needed in a child(leaf)
        'subsample_for_bin': 200000,  # Number of samples for constructing bin
        'min_split_gain': 0,  # lambda_l1, lambda_l2 and min_gain_to_split to regularization
        'reg_alpha': 0,  # L1 regularization term on weights
        'reg_lambda': 0,  # L2 regularization term on weights
        'nthread': 4,
        'verbose': 0,
        'metric':metrics
    }

    lgb_params.update(params)

    print("preparing validation datasets")

    xgtrain = lgb.Dataset(dtrain[predictors].values, label=dtrain[target].values,
                          feature_name=predictors,
                          categorical_feature=categorical_features
                          )
    xgvalid = lgb.Dataset(dvalid[predictors].values, label=dvalid[target].values,
                          feature_name=predictors,
                          categorical_feature=categorical_features
                          )

    evals_results = {}
```

```
    bst1 = lgb.train(lgb_params,
                     xgtrain,
                     valid_sets=[xgtrain, xgvalid],
                     valid_names=['train','valid'],
                     evals_result=evals_results,
                     num_boost_round=num_boost_round,
                     early_stopping_rounds=early_stopping_rounds,
                     verbose_eval=10,
                     feval=feval)

    n_estimators = bst1.best_iteration
    print("\nModel Report")
    print("n_estimators : ", n_estimators)
    print(metrics+":", evals_results['valid'][metrics][n_estimators-1])

    return bst1

path = './data/'

dtypes = {
        'ip'            : 'uint32',
        'app'           : 'uint16',
        'device'        : 'uint16',
        'os'            : 'uint16',
        'channel'       : 'uint16',
        'is_attributed' : 'uint8',
        'click_id'      : 'uint32'
        }

print('loading train data...')
train_df = pd.read_csv(path+"train.csv", dtype=dtypes, usecols=['ip','app','device','os', 'channel', 'click_time', 'is_attr

print('loading test data...')
test_df = pd.read_csv(path+"test.csv", dtype=dtypes, usecols=['ip','app','device','os', 'channel', 'click_time', 'click_id'

len_train = len(train_df)
train_df=train_df.append(test_df)

del test_df
gc.collect()
```

```
print('Extracting new features...')
train_df['hour'] = pd.to_datetime(train_df.click_time).dt.hour.astype('uint8')
train_df['day'] = pd.to_datetime(train_df.click_time).dt.day.astype('uint8')

gc.collect()

print('grouping by : ip-day-hour combination...')
gp = train_df[['ip','day','hour','channel']].groupby(by=['ip','day','hour'])[['channel']].count().reset_index().rename(inde
train_df = train_df.merge(gp, on=['ip','day','hour'], how='left')
del gp
gc.collect()

print('grouping by : ip-app combination...')
gp = train_df[['ip', 'app', 'channel']].groupby(by=['ip', 'app'])[['channel']].count().reset_index().rename(index=str, colu
train_df = train_df.merge(gp, on=['ip','app'], how='left')
del gp
gc.collect()

print('grouping by : ip-app-device combination...')
gp = train_df[['ip', 'app', 'device', 'channel']].groupby(by=['ip', 'app', 'device'])[['channel']].count().reset_index().re
train_df = train_df.merge(gp, on=['ip','app', 'device'], how='left')
del gp
gc.collect()

print('grouping by : ip-app-os combination...')
gp = train_df[['ip','app', 'os', 'channel']].groupby(by=['ip', 'app', 'os'])[['channel']].count().reset_index().rename(inde
train_df = train_df.merge(gp, on=['ip','app', 'os'], how='left')
del gp
gc.collect()

print("vars and data type: ")
train_df.info()
train_df['ip_tcount'] = train_df['ip_tcount'].astype('uint16')
train_df['ip_app_count'] = train_df['ip_app_count'].astype('uint16')
train_df['ip_app_os_count'] = train_df['ip_app_os_count'].astype('uint16')

test_df = train_df[len_train:]
val_df = train_df[(len_train-9000000):len_train]
train_df = train_df[:(len_train-9000000)]
```

```
print("train size: ", len(train_df))
print("valid size: ", len(val_df))
print("test size : ", len(test_df))

target = 'is_attributed'
predictors = ['app','device','os', 'channel', 'hour', 'day',
              'ip_tcount', 'ip_app_count', 'ip_app_device_count',
              'ip_app_os_count']
categorical = ['app', 'device', 'os', 'channel', 'hour', 'day']

sub = pd.DataFrame()
sub['click_id'] = test_df['click_id'].astype('int')

gc.collect()

print("Training...")
start_time = time.time()


params = {
    'learning_rate': 0.15,
    #'is_unbalance': 'true', # replaced with scale_pos_weight argument
    'num_leaves': 7,  # 2^max_depth - 1
    'max_depth': 3,  # -1 means no limit
    'min_child_samples': 100,  # Minimum number of data need in a child(min_data_in_leaf)
    'max_bin': 100,  # Number of bucketed bin for feature values
    'subsample': 0.7,  # Subsample ratio of the training instance.
    'subsample_freq': 1,  # frequence of subsample, <=0 means no enable
    'colsample_bytree': 0.9,  # Subsample ratio of columns when constructing each tree.
    'min_child_weight': 0,  # Minimum sum of instance weight(hessian) needed in a child(leaf)
    'scale_pos_weight':99 # because training data is extremely unbalanced
}
bst = lgb_modelfit_nocv(params,
                        train_df,
                        val_df,
                        predictors,
                        target,
                        objective='binary',
                        metrics='auc',
                        early_stopping_rounds=30,
                        verbose_eval=True,
                        num_boost_round=500,
                        categorical_features=categorical)
```

```
print('[{}]: model training time'.format(time.time() - start_time))
del train_df
del val_df
gc.collect()

print("Predicting...")
sub['is_attributed'] = bst.predict(test_df[predictors])
print("writing...")
sub.to_csv('sub3.csv',index=False)
print("done...")
print(time.time())
```

```
loading train data...
loading test data...
Extracting new features...
grouping by : ip-day-hour combination...
grouping by : ip-app combination...
grouping by : ip-app-device combination...
grouping by : ip-app-os combination...
vars and data type:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 242441395 entries, 0 to 242441394
Data columns (total 14 columns):
app                     uint16
channel                 uint16
click_id                float64
click_time              object
device                  uint16
ip                      uint32
is_attributed           float64
os                      uint16
hour                    uint8
day                     uint8
ip_tcount               int64
ip_app_count            int64
ip_app_device_count     int64
ip_app_os_count         int64
dtypes: float64(2), int64(4), object(1), uint16(4), uint32(1), uint8(2)
memory usage: 17.6+ GB
train size:  175903890
valid size:  9000000
test size :  57537505
Training...
```

```
loading train data...
loading test data...
Extracting new features...
grouping by : ip-day-hour combination...
grouping by : ip-app combination...
grouping by : ip-app-device combination...
grouping by : ip-app-os combination...
vars and data type:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 242441395 entries, 0 to 242441394
Data columns (total 14 columns):
app                     uint16
channel                 uint16
click_id                float64
click_time              object
device                  uint16
ip                      uint32
is_attributed           float64
os                      uint16
hour                    uint8
day                     uint8
ip_tcount               int64
ip_app_count            int64
ip_app_device_count     int64
ip_app_os_count         int64
dtypes: float64(2), int64(4), object(1), uint16(4), uint32(1), uint8(2)
memory usage: 17.6+ GB
train size:  175903890
valid size:  9000000
test size :  57537505
Training...
preparing validation datasets
/usr/local/lib/python3.6/dist-packages/lightgbm/basic.py:1036: UserWarning: Using categorical_feature in Dataset.
  warnings.warn('Using categorical_feature in Dataset.')
/usr/local/lib/python3.6/dist-packages/lightgbm/basic.py:681: UserWarning: categorical_feature in param dict is overridden.
  warnings.warn('categorical_feature in param dict is overridden.')
Training until validation scores don't improve for 30 rounds.
[10]    train's auc: 0.959052   valid's auc: 0.966735
[20]    train's auc: 0.963849   valid's auc: 0.969233
[30]    train's auc: 0.968648   valid's auc: 0.973559
[40]    train's auc: 0.970148   valid's auc: 0.975247
[50]    train's auc: 0.971168   valid's auc: 0.976422
[60]    train's auc: 0.971866   valid's auc: 0.977454
```

Thank you!!