번개장터 CTR을 높이기 위한 광고 추천 알고리즘 개발

SSAC 6조
문영주, 백승재, 손희서, 조지민

목차 Table of Contents

1 프로젝트 개요

- 문제정의
- 프로젝트 진행과정

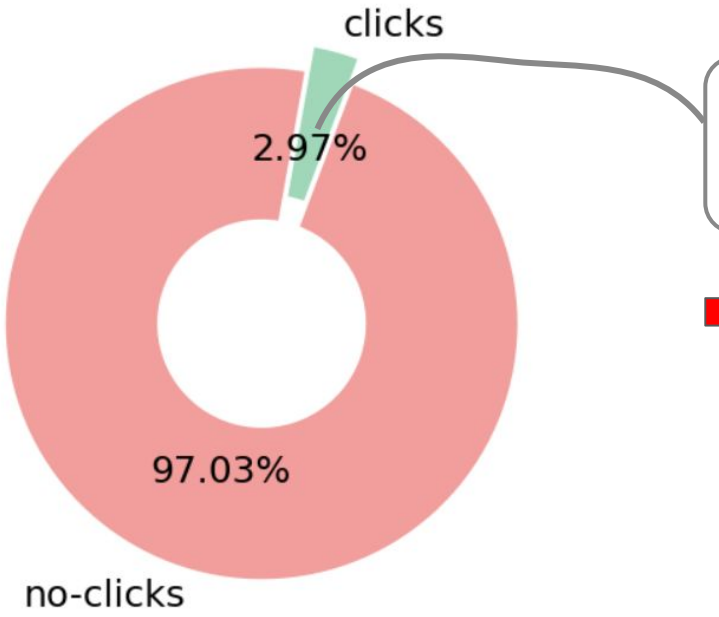
1 프로젝트 개요

문제 정의

CTR(Click-through rate, 클릭률)

- 온라인 광고의 노출횟수 대비 클릭 수를 의미
- 광고가 노출된 횟수(Impression) 중, 실제 클릭을 통해 이동한 경우의 비율
- CTR = (클릭 수 / 노출된 횟수) × 100
- 온라인 광고 효과를 측정하는 데 있어 CTR은 중요 지표
- 온라인 광고시장 평균 CTR 약 0.3%

출처 : [네이버 지식백과] 클릭률 [Click-through rate] (ICT 시사상식 2015, 2014.12.31)



2021년 8월 31일 impression_log의
imp_id 중복 값을 제거한 후
impression(노출) 대비 view(광고클릭) 비율

**CTR을 예측하고,
CTR 비율을 높이는데 도움 되도록
알맞는 viewer에게 알맞은 content를
추천하는 알고리즘 모델링 프로젝트 진행**

1 프로젝트 개요

프로젝트 진행 과정

2 데이터 전처리

- 데이터 정의
- 데이터셋 만들기
- 이상치 처리
- 데이터 EDA
- 결측치 처리

2 데이터 전처리

데이터 정의

{ 개인고객 정보 }

	col_name	설명
1	'user_id'	광고에 노출된 대상의 아이디
2	'gender'	상품 노출 대상의 성별
3	'age'	상품 노출 대상의 나이
4	'following_cnt'	상품 노출 대상의 팔로잉 수
5	'pay_count'	상품 노출 대상의 거래수
6	'parcel_post_count'	상품 노출 대상의 택배 거래수
7	'transfer_count'	상품 노출 대상의 송금 거래수
8	'chat_count'	상품 노출 대상의 채팅수

{ 광고상품 정보 }

	col_name	설명
1	'content_id'	광고 상품 아이디
2	'user_id'	광고주 아이디
3	'name'	광고 상품 이름
4	'keyword'	광고 상품 태그
5	'price'	광고 상품 가격
6	'flag_used'	광고 상품 중고 여부
7	'category_id_1'	광고 상품 1차 카테고리
8	'category_id_2'	광고 상품 2차 카테고리
9	'category_id_3'	광고 상품 3차 카테고리
10	'emergency_cnt'	광고 상품 신고수
11	'comment_cnt'	광고 상품에 달린 코멘트수
12	'interest'	광고 상품 클릭수
13	'pfavcnt'	광고 상품 좋아요 수

{ 광고주 정보 }

	col_name	설명
1	'user_id'	광고주 아이디
2	'favorite_count'	광고주 선호수
3	'grade'	광고주 등급
4	'item_count'	광고주가 가지고 있는 상품수
5	'interest'	광고주가 받은 클릭수
6	'review_count'	광고주가 받은 리뷰수
7	comment_count	광고주가 작성한 코멘트 수
8	'follower_count	광고주의 팔로워수
9	'pay_count'	광고주의 결제거래시스템 거래수
10	'parcel_post_count''	광고주의 택배 거래수
11	'transfer_count'	광고주의 거래수
12	'chat_count'	광고주의 채팅수

{ 광고노출 정보 }

	col_ame	설명
1	'imp_id'	노출 아이디
2	'server_time_kst'	로그 생성 시간
3	'bid_price'	클릭된 광고 상품 입찰가(PPC)
4	'device_type'	클릭 대상 디바이스 타입

{ 광고클릭 정보 }

	col_ame	설명
1	'imp_id'	노출 아이디
2	'server_time_kst'	뷰 생성 시간
3	'bid_price'	클릭된 광고 상품 입찰가(PPC)
4	'device_type'	클릭 대상 디바이스 타입

2 데이터 전처리

데이터 정의

1. 주어진 데이터 테이블들을 병합하여 만들 수 있는 가장 큰 데이터셋 구축

2. 전체 데이터 셋을 고려하여 isolation forest를 통해 이상치 제거

3. 모델링에 사용할 컬럼을 선정하여 EDA진행

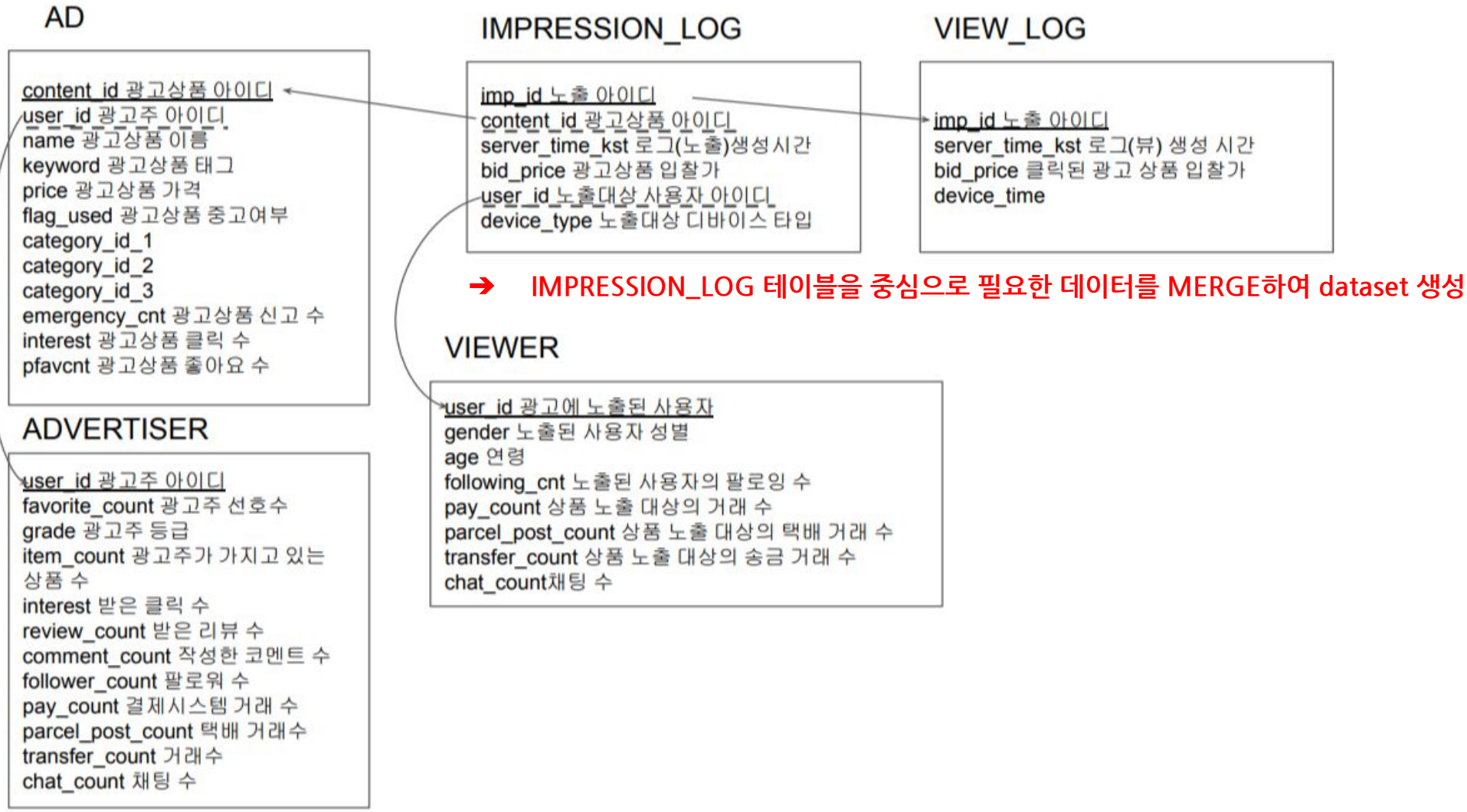
4. EDA를 바탕으로 결측치 제거

5. SMOTE로 불균형 데이터 처리

2 데이터 전처리

데이터셋 만들기

데이터 관계도

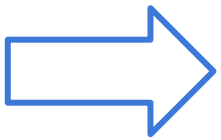


➔ IMPRESSION_LOG 테이블을 중심으로 필요한 데이터를 MERGE하여 dataset 생성

2 데이터 전처리

데이터셋 만들기

#	Column	Non-Null Count	Dtype
0	imp_id	842463 non-null	object
1	content_id	842463 non-null	int64
2	server_time_kst_x	842463 non-null	object
3	imp_hour	842463 non-null	int64
4	bid_price_x	842463 non-null	int64
5	click_label	842463 non-null	int64
6	user_gender	842463 non-null	object
7	user_age_group	842463 non-null	int64
8	user_age	842463 non-null	int64
9	user_following_cnt	842463 non-null	int64
10	user_pay_count	842463 non-null	int64
11	user_parcel_post_count	842463 non-null	int64
12	user_transfer_count	842463 non-null	int64
13	user_chat_count	842463 non-null	int64
14	name	772222 non-null	object
15	keyword	707739 non-null	object
16	price	842463 non-null	float64
17	flag_used	842463 non-null	float64
18	category1	842463 non-null	int64
19	emergency_cnt	842463 non-null	float64
20	comment_cnt	842463 non-null	float64
21	ad_interest	842463 non-null	float64
22	ad_pfavcnt	842463 non-null	float64
23	adver_favorite_count	842463 non-null	float64
24	adver_grade	842463 non-null	float64
25	adver_item_count	842463 non-null	float64
26	adver_interest	842463 non-null	float64
27	adver_review_count	842463 non-null	float64
28	adver_comment_count	842463 non-null	float64
29	adver_pay_count	842463 non-null	float64
30	adver_parcel_post_count	842463 non-null	float64
31	adver_transfer_count	842463 non-null	float64
32	adver_chat_count	842463 non-null	float64
33	m_time	842463 non-null	int64
34	content_img_url	842463 non-null	object



#	Column	Non-Null Count	Dtype
0	user_gender	842463 non-null	object
1	imp_hour	842463 non-null	int64
2	click_label	842463 non-null	int64
3	user_age_group	842463 non-null	int64
4	flag_used	842463 non-null	float64
5	category1	842463 non-null	int64
6	emergency_cnt	842463 non-null	float64
7	user_age	842463 non-null	int64
8	user_following_cnt	842463 non-null	int64
9	user_pay_count	842463 non-null	int64
10	user_parcel_post_count	842463 non-null	int64
11	user_transfer_count	842463 non-null	int64
12	user_chat_count	842463 non-null	int64
13	price	842463 non-null	float64
14	comment_cnt	842463 non-null	float64
15	ad_interest	842463 non-null	float64
16	ad_pfavcnt	842463 non-null	float64
17	adver_favorite_count	842463 non-null	float64
18	adver_grade	842463 non-null	float64
19	adver_item_count	842463 non-null	float64
20	adver_interest	842463 non-null	float64
21	adver_review_count	842463 non-null	float64
22	adver_comment_count	842463 non-null	float64
23	adver_pay_count	842463 non-null	float64
24	adver_parcel_post_count	842463 non-null	float64
25	adver_transfer_count	842463 non-null	float64
26	adver_chat_count	842463 non-null	float64

2 데이터 전처리

이상치 처리

Outlier preprocessing

- isolation forest & grid search로 찾은 1000여 개 이상값 삭제

```
# GridSearchCV(모델, param_grid, cv, scoring, n_jobs, verbose)
param_grid = {
    'n_estimators':[50, 100, 150],
    'max_samples':[50, 100, 150, 200],
    'contamination':[float(0.004), float(0.1), float(0.5), 'auto']
}

# 전처리를 거친 x_train으로 grid search cross validation 수행
grid_search = GridSearchCV(IsolationForest(max_features=1.0, bootstrap=False, random_state=42,),
                           param_grid=param_grid, cv=10, n_jobs=-1, verbose=0, scoring = 'accuracy')
grid_search.fit(dataset)

# 가장 좋은 성능을 보인 파라미터 조합의 estimator을 model에 저장
model = grid_search.best_estimator_

# dataset 데이터를 예측 및 성능 평가
data_pred = model.predict(dataset)
# print("train RMSE: {:.2f}".format(np.sqrt(mean_squared_error(dataset, data_pred))))
```

```
#이상치 값 개수
-data_pred[data_pred == -1].sum()
```

```
1038
```

```
model
```

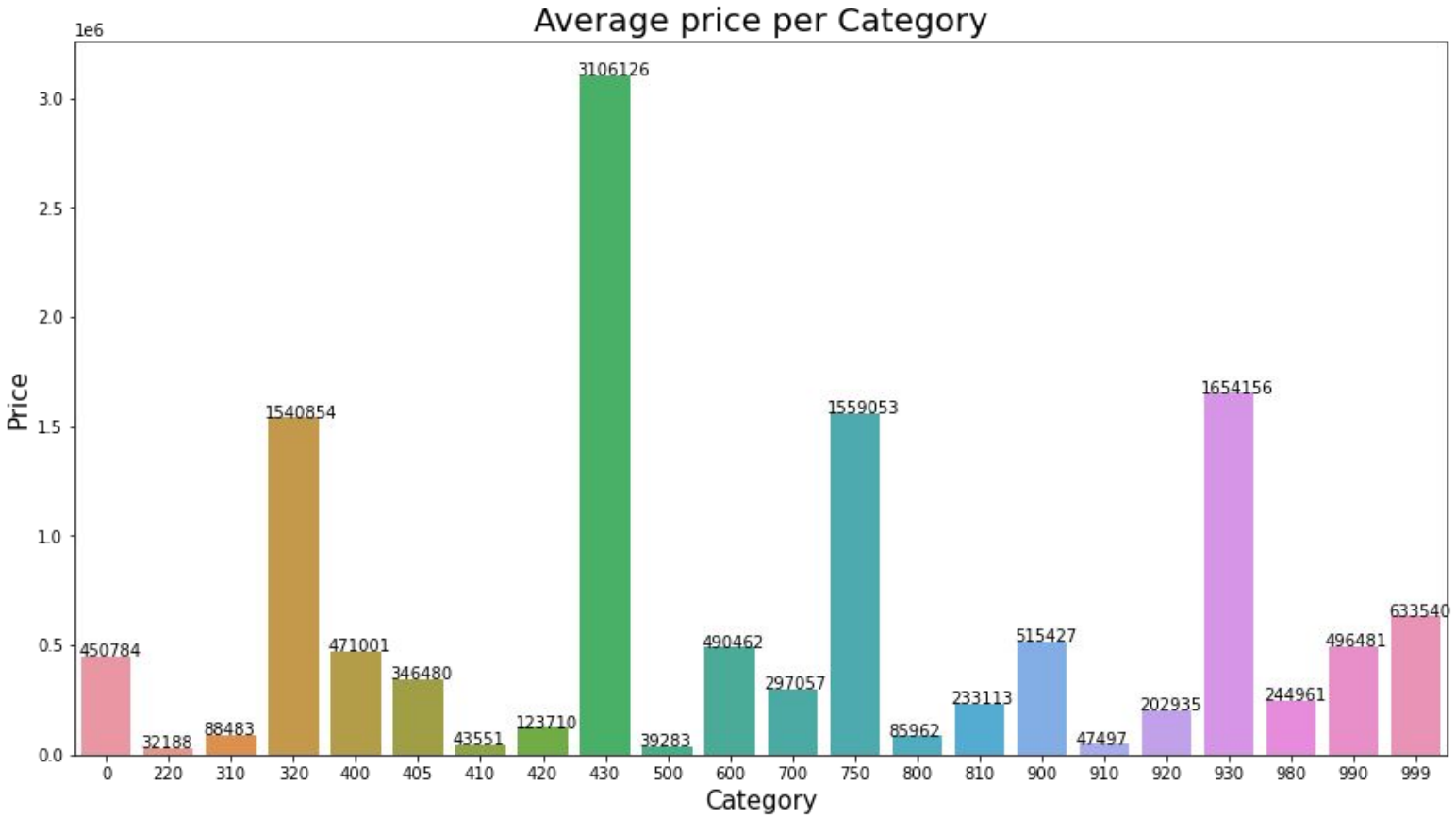
```
IsolationForest(contamination=0.004, max_samples=50, n_estimators=50,
                random_state=42)
```

2 데이터 전처리

데이터 EDA

category_id_1	name	impcnt	unique	click	ctr
0	알수없음	70241	112	911	1.296963
220	지역서비스	32	3	1	3.125000
310	여성의류	63501	112	589	0.927544
320	남성의류	53095	43	445	0.838120
400	패션액세서리	5107	15	100	1.958097
405	신발	4314	20	31	0.718591
410	뷰티/미용	12689	40	247	1.946568
420	시계/쥬얼리	30121	67	476	1.580293
430	가방	51202	35	402	0.785126
500	유아동/출산	1781	5	8	0.449186
600	디지털/가전	449329	854	19554	4.351822
700	스포츠/레저	34256	30	716	2.090145
750	차량/오토바이	24887	75	479	1.924700
800	생활/가공식품	11916	51	272	2.282645
810	가구/인테리어	9894	30	166	1.677785
900	도서/티켓	11734	8	206	1.755582
910	스타굿즈	2313	4	27	1.167315
920	음반/악기	4250	14	73	1.717647
930	키덜트	1504	3	45	2.992021
980	반려동물용품	129	3	2	1.550388
990	예술/희귀품/수집품	7824	10	76	0.971370
999	기타	904	5	17	1.880531

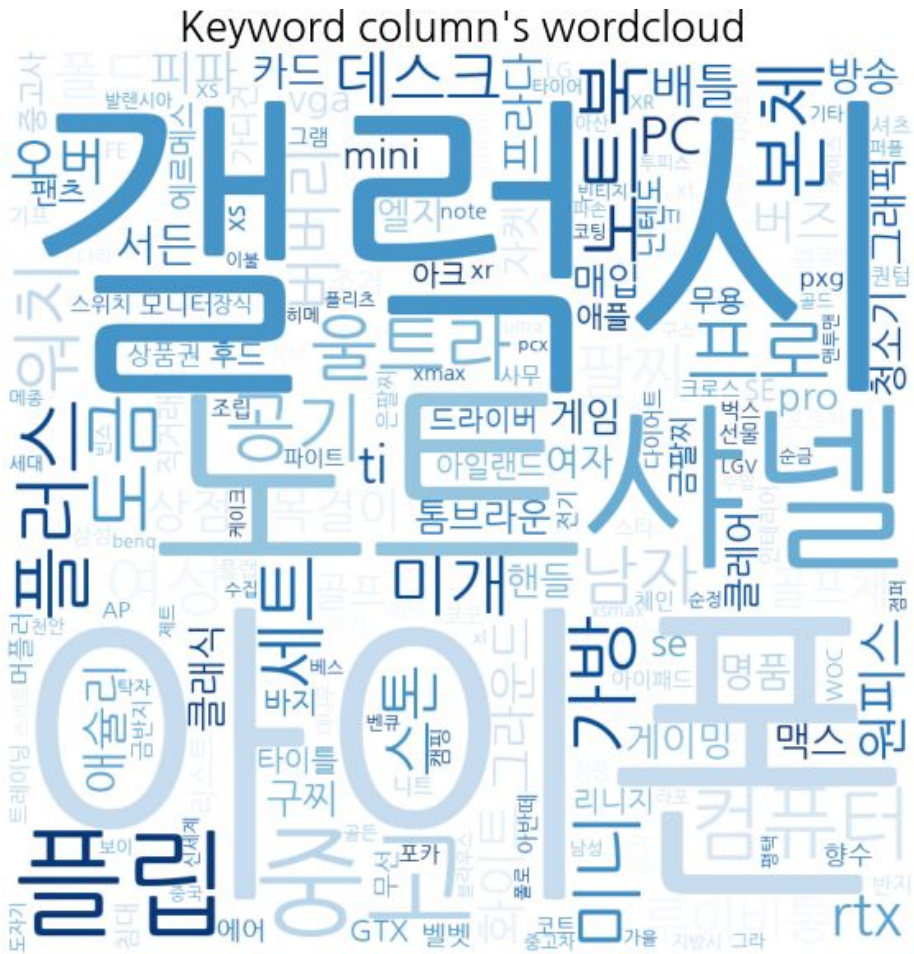
카테고리 대분류 EDA



2 데이터 전처리

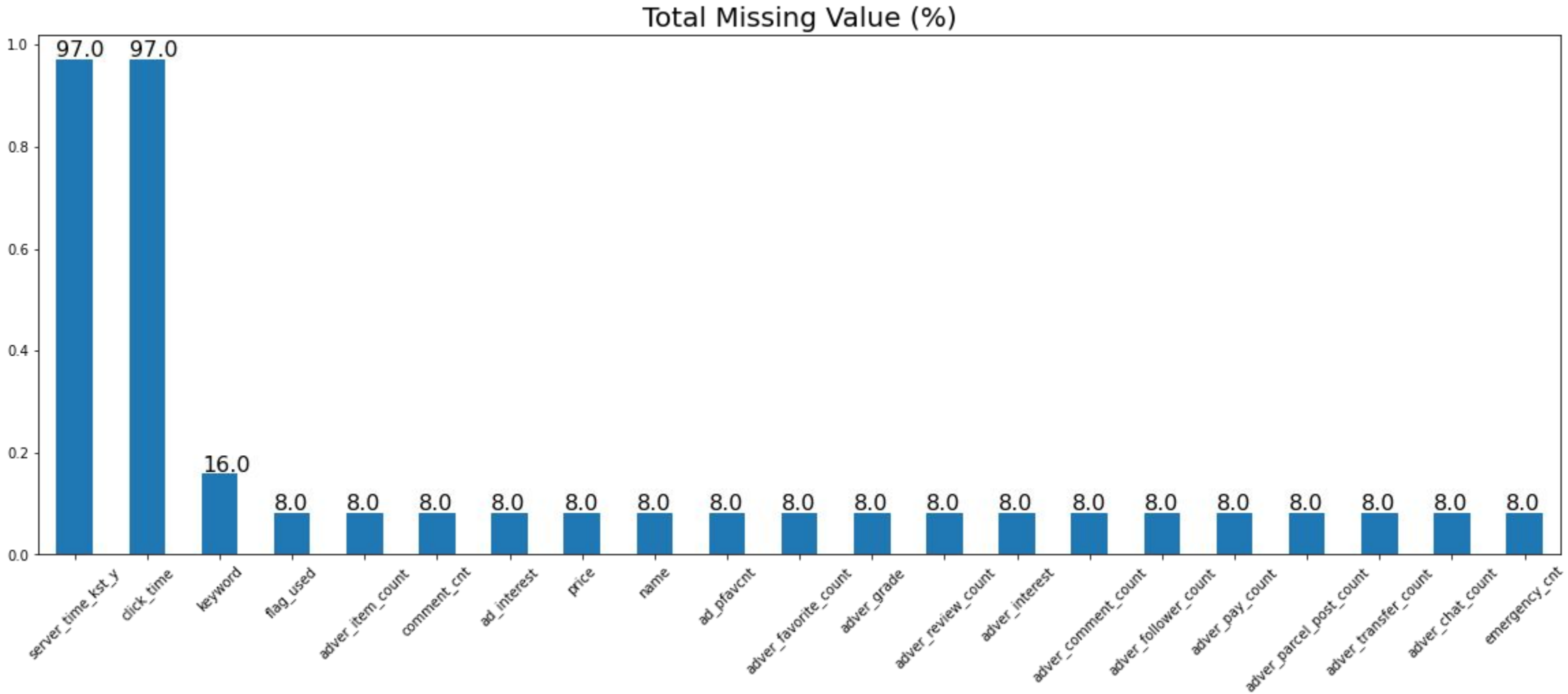
데이터 EDA

상품명, 키워드 EDA



2 데이터 전처리

결측치 처리



2 데이터 전처리

결측치 처리

KNN Imputer

- 광고 & 광고주 데이터 결측치 처리에 이용

```
from sklearn.impute import KNNImputer
```

```
param_grid = [
    {'n_neighbors': range(3, 11, 2)}
]

imputer = KNNImputer(missing_values=np.nan, add_indicator=True)
gs = GridSearchCV(imputer, param_grid = param_grid, cv=10, n_jobs = -1, scoring = 'f1')#score: r2, f1
gs.fit(df_impute)

# 가장 좋은 성능을 보인 파라미터 조합의 estimator을 model에 저장
model = gs.best_estimator_

# df_impute[(df_impute.category_id_1 == 0)]
```

```
model #f1 scoring일 때 최적 모델 값
```

```
KNNImputer(add_indicator=True, n_neighbors=3)
```

2 데이터 전처리

결측치 처리

KNN Imputer

```
df_impute[['price', 'flag_used', 'emergency_cnt', 'comment_cnt', 'ad_interest', 'ad_pfavcnt']] = ₩
    model.fit_transform(df_impute[['price', 'flag_used', 'emergency_cnt', 'comment_cnt', 'ad_interest', 'ad_pfavcnt']])
df_impute[(df_impute.category_id_1 == 0)]
```

	content_id	price	flag_used	category_id_1	emergency_cnt	comment_cnt	ad_interest	ad_pfavcnt
778962	139971377.0	259000.0	1.0	0.0	6.0	11.0	96247.0	2097.0
778963	158578923.0	48000.0	2.0	0.0	0.0	0.0	211.0	7.0
778964	159418585.0	105000.0	1.0	0.0	0.0	0.0	162.0	7.0
778965	158578923.0	48000.0	2.0	0.0	0.0	0.0	211.0	7.0
778966	161617931.0	290000.0	1.0	0.0	0.0	0.0	666.0	35.0
...	...	...	...	...	...	...	...	...
849198	113522115.0	89000.0	2.0	0.0	4.0	4.0	16474.0	282.0
849199	156709232.0	360000.0	1.0	0.0	0.0	0.0	263.0	4.0
849200	131022694.0	2000000.0	1.0	0.0	1.0	36.0	20319.0	104.0
849201	156709232.0	360000.0	1.0	0.0	0.0	0.0	263.0	4.0
849202	131022694.0	2000000.0	1.0	0.0	1.0	36.0	20319.0	104.0

70241 rows × 8 columns

2 데이터 전처리

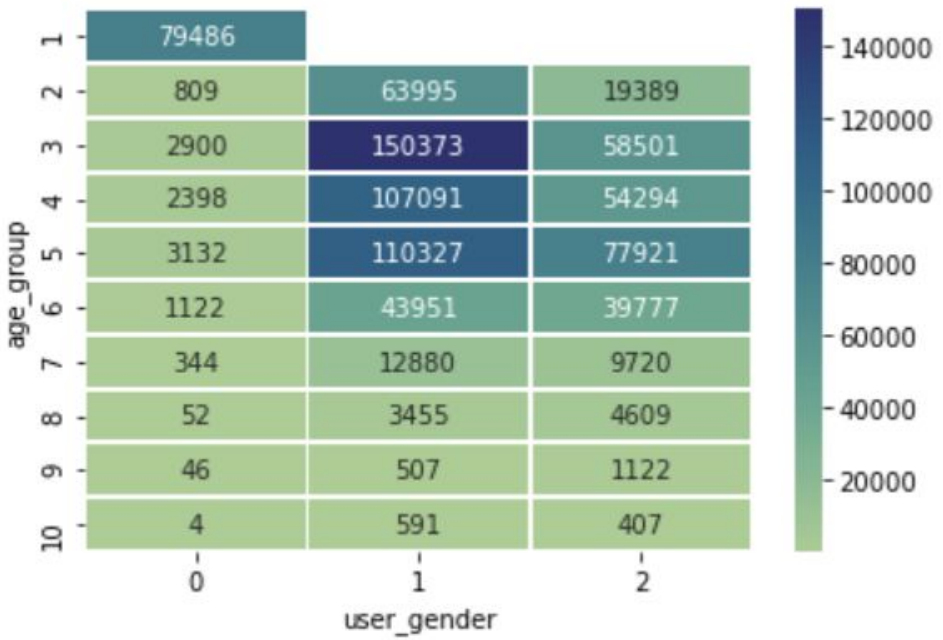
결측치 처리: user_gender & user_age

1. user_age_group 변수 생성

- user_age를 0~9세, 10~19세, …,80~89세, 90세 이상을 각각 1, 2, …, 10 범주로 구분.

2. user_age_group과 user_gender 분포를 시각화

- K-Means Clustering을 하기 위해 user_gender = 0 or user_age_group = 1 or 10 인 데이터 삭제.



	user_age	user_gender_2
0	49.974285	0.465799
1	17.435060	0.249807
2	36.261691	0.351255
3	72.502809	0.514700
4	42.860744	0.401423
5	29.270535	0.296552
6	58.835816	0.458696
7	23.575416	0.266775

	user_age	user_gender_2
0	50	1
1	17	1
2	36	1
3	73	2
4	43	1
5	29	1
6	59	1
7	24	1

3. K-Means Clustering

- 8개의 군집으로 나머지 데이터를 구분하여 각 군집의 평균 나이와 성별 구하기.
- 8개의 군집별로 전체 데이터에서 각 그룹의 데이터가 차지하는 비율을 구하기.

2 데이터 전처리

데이터 처리: user_gender & user_age

4. 8개의 군집별로 평균나이, 성별, 비율 그리고 연령대 값 구하기

- user_gender = 0 or user_age_group = 1 or 10 인 데이터들의 나이와 성별 정보를 군집의 비율대로 채워주기.

	age	gender	percentage	age_group	fill_cnt
0	50	1	0.136587	6	12469
1	17	1	0.133343	2	12173
2	36	1	0.151367	4	13818
3	73	2	0.020195	8	1844
4	43	1	0.184540	5	16847
5	29	1	0.141191	3	12889
6	59	1	0.060135	6	5490
7	24	1	0.172643	3	15761

	imp_id	content_id	server_time_kst_x	imp_hour	bid_price_x	click_label	user_gender	user_age_group	user_age
0	9919612e44382421285c	162261579	2021-09-01 00:01:42.466000+09:00	0	55	0	F	1	16

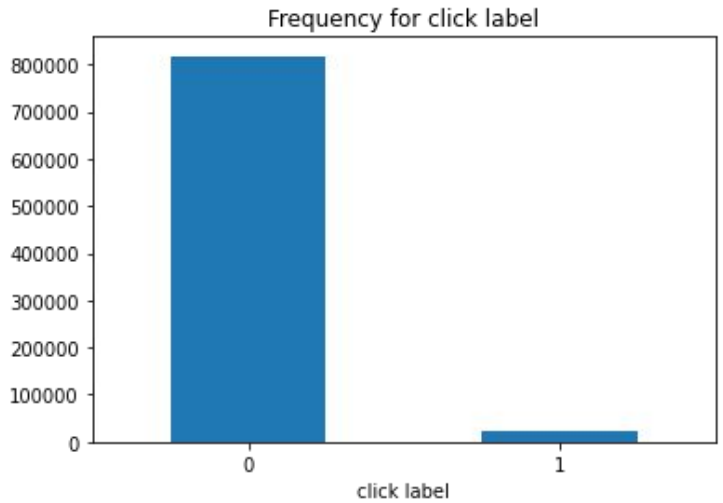
2 데이터 전처리

데이터 처리: Imbalance 처리 방법1

SMOTE

- 딥러닝 분석을 위해서는 많은 데이터 확보가 효과적이므로 오버샘플링 기법을 적용하는게 좋다.
- 오버샘플링 기법 중 합성데이터를 생성하는 방식으로 가장 많이 사용되고 있는 모델로 합성 소수 샘플링 기술로 다수 클래스를 샘플링하고 기존 소수 샘플을 보간하여 새로운 소수 인스턴스를 합성해낸다.
- 일반적인 경우 성공적으로 작동하지만, 소수데이터들 사이를 보간하여 작동하기 때문에 모델링셋의 소수데이터들 사이의 특성만을 반영하고 새로운 사례의 데이터 예측엔 취약할 수 있다.
- similar examples을 사용해 새로운 인스턴스 생성 → boundary가 무난하고 classifier는 general하고 overfitting이 되지 않음.

```
def split_data():

  ### read dataset
    file_name = 'final_data_v2.csv'
    file_path = os.getcwd()+'/drive/MyDrive/Colab Notebooks/'
    df = pd.read_csv(file_path+file_name, encoding='utf-8')
    df.rename(columns={'category_id_1':'category1'}, inplace=True)

    modified_df = preprocessing()

    X = modified_df.values
    y = df['click_label'].values

    # split the train/test data (7:3 ratio)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 2022, stratify = y) #y 비율에 따른 층화추출 및 데이터를 7:3의 비율로 나누기
    print(f"# of train_data's rows: {X_train.shape[0]} ₩n# of test_data's rows: {X_test.shape[0]}")
    print(f'train:test ratio = {round(X_train.shape[0]/(X_train.shape[0]+ X_test.shape[0]),2)}:{round(X_test.shape[0]/(X_train.shape[0]+ X_test.shape[0]), 2)}')

    oversample = SMOTE(random_state=2022) # 불균형 데이터 셋인 번개장터 데이터 셋 불균형 문제 완화
    X_train, y_train = oversample.fit_resample(X_train, y_train)

    ## create train and validation datasets (8:2 ratio)
    x_train, x_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state = 2022, stratify = y_train)
    print(f"# of train_data's rows: {x_train.shape[0]} ₩n# of validation_data's rows: {x_val.shape[0]}")
    print(f'train:test ratio = {round(x_train.shape[0]/(x_train.shape[0]+ x_val.shape[0]),2)}:{round(x_val.shape[0]/(x_train.shape[0]+ x_val.shape[0]), 2)}')
```



2 데이터 전처리

데이터 처리: Imbalance 처리 방법2

▶ **Imbalance한 원본 데이터**

click_label비율 33:1 ⇒ 대부분 click하지 않은 경우 ⇒ 학습이 제대로 되지 않는 문제 발생

⇒ 학습 데이터를 Balance하게 만들기(click_label의 비율이 동일하게)

```
click = data[data['click_label']==1]
click.shape
```

```
(24713, 27)
```

1. click_label=0인 데이터들 중 50%만 random sampling하기

- 817,750개 ⇒ 408,875개
- click_label=1인 데이터 정보를 지나치게 늘리지 않기 위한 과정

```
noclick_sampled = noclick.sample(frac=0.5,replace=False,axis=0)
noclick_sampled.shape
```

```
(408875, 27)
```

2. SMOTE로 click_label=1인 데이터 수 늘리기

- random sampling한 click_label=0인 데이터와 click_label=1인 데이터를 병합하여 비율이 1:1이 되도록 click_label=1인 데이터 추가로 생성하기

```
sm_model = SMOTE(k_neighbors = 3)

X_train = sampled_data.drop('click_label', axis=1)
Y_train = sampled_data['click_label']
X_train.shape, Y_train.shape
```

```
((433588, 26), (433588,))
```

⇛ SMOTE까지 한 결과 click_label의 비율 1:1 (총 817,750개)

```
Y_train_bal.value_counts()
```

```
1    408875
0    408875
Name: click_label, dtype: int64
```

```
X_train_bal , Y_train_bal = sm_model.fit_resample(X_train, Y_train)
X_train_bal = pd.DataFrame(X_train_bal, columns = X_train.columns)
Y_train_bal = pd.Series(Y_train_bal)
```

3 모델링

- 모델링 개요
- 변수파악
- DeepFM
- Wide&Deep
- Autoencoder
- 기타

3 모델링

모델링 개요

Deep FM

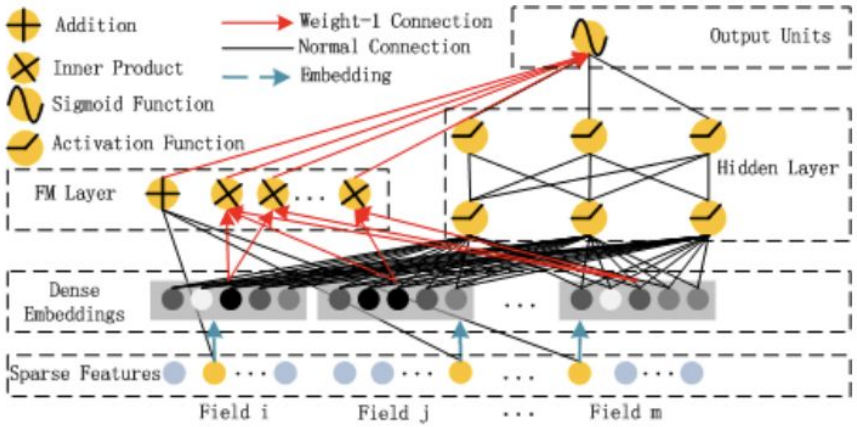
- **Factorization Machine + Deep Learning**

  - Wide & Deep과 다르게 별도의 feature engineering 필요없음

  - FM부분과 Deep 부분이 같은 Input을 공유하여 학습 (효율적인 학습 가능)

- **FM component : low-order feature interaction 학습 (1&2차원 변수 학습)**
- **Deep component: high-order feature interaction 학습 (3차 이상의 변수 학습)**

  - Embedding layer을 통해 high-dimension & sparse한 데이터를 dense한 데이터로 변환



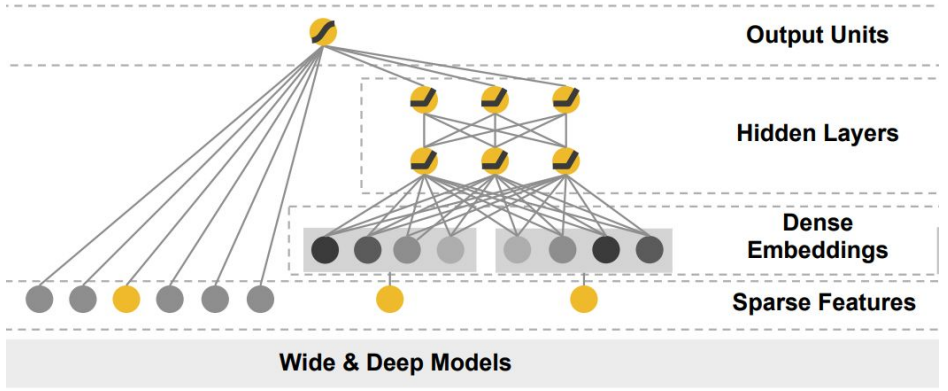
Wide&Deep

- **기억 & 일반화**
- **Wide component :** 자주 등장하는 조합 기억하기 ⇒ **기억**

  feature engineering 역량이 중요. (변수들간 상호작용에 대한 사전 지식 필요)

- **Deep component :** 새로운 조합 발견 ⇒ **다양성 & 일반화**

  Embedding layer을 통해 high dimensional & sparse한 데이터를 low dimentional &dense한 데이터로 변환.



3 모델링

모델링 개요

Autoencoder

- **라벨이 없는 훈련 데이터를 사용한 학습(즉, 지도 학습) 없이도 입력 데이터의 표현을 효율적으로 학습할 수 있는 인공신경망**
- **차원 축소(Dimensionality Reduction)** 목적으로 사용
- 훈련 데이터와 매우 비슷한 새로운 데이터를 생성하는 생성 모델(generative model)로서 사용

DNN

- DL의 가장 기본적인 모델 활용.
- 여러 층의 Dense layer과 Dropout layer을 쌓기. (layer 개수 4, 6, 8, 10)
- 여러가지 hyper-parameter 조합 시도.

Boosting ML

- **LightGBM**
- **CatBoost**

  - GBM의 overfitting문제를 개선하고 XGB와 LGBM보다 빠른 학습성능을 가짐

  - hyper-parameter에 따른 성능 차이를 완화하는데 초점

3 모델링

변수파악

▶ **독립변수 : 26개**

- **범주형: 5개**

  ('imp_hour', 'user_gender', 'user_age_group', 'flag_used', 'category_id_1')

- **연속형: 21개**

  ('bid_price', 'user_age', 'user_following_cnt', 'user_pay_count', 'user_parcel_post_count', 'user_transfer_count', 'user_chat_count', 'price', 'emergency_cnt', 'comment_cnt', 'ad_interest', 'ad_pfavcnt', 'adver_favorite_count', 'adver_item_count', 'adver_interest', 'adver_review_count', 'adver_comment_count', 'adver_pay_count', 'adver_parcel_post_count', 'adver_transfer_count', 'adver_chat_count')

▶ **종속변수 : 'click_label'**

ex.

imp_hour	bid_price_x	user_gender	user_age_group	user_age	user_following_cnt	user_pay_count	user_parcel_post_count	user_transfer_count	user_chat_count	price
0	50	0	2	20	1	2	0	0	0	630000.0

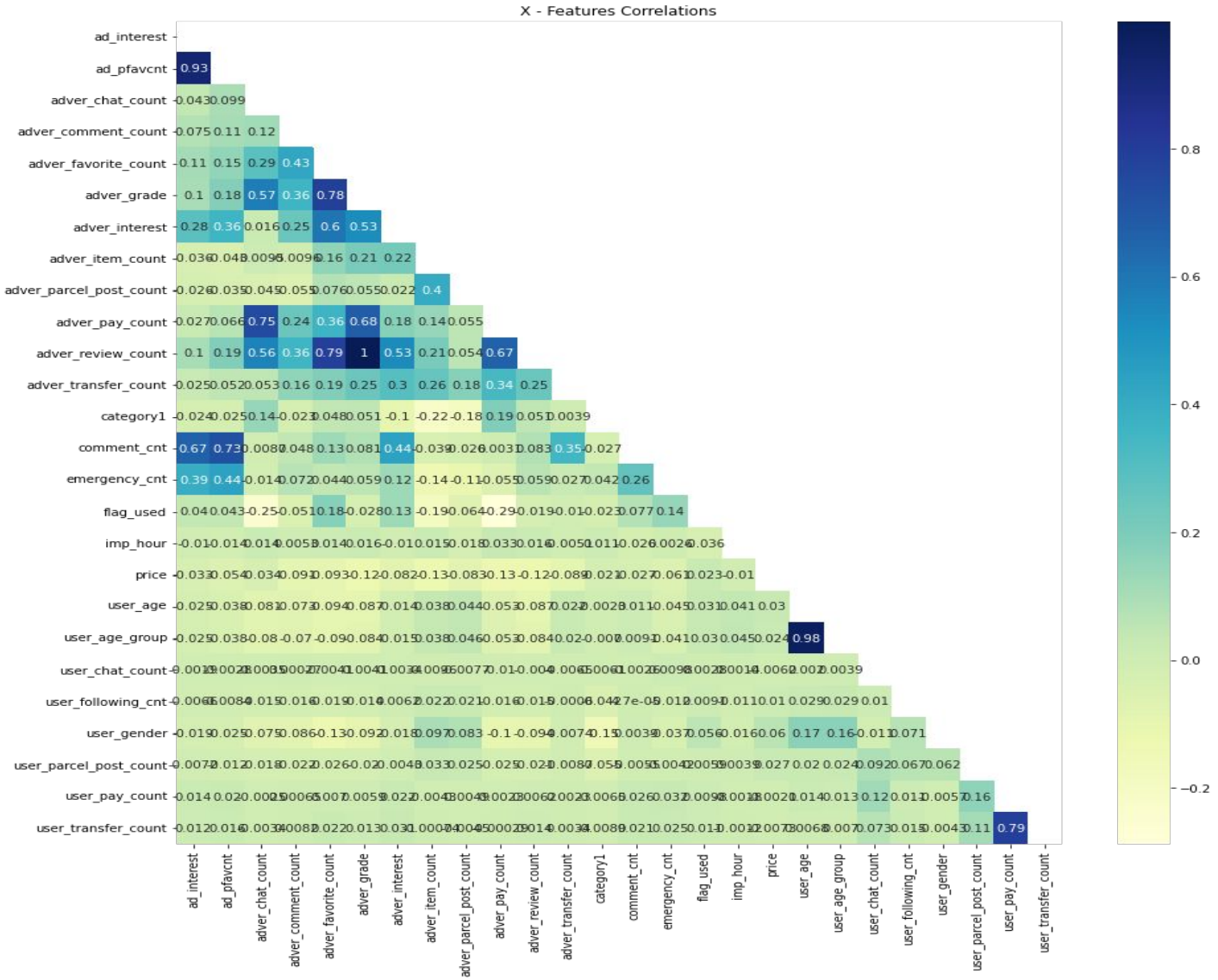
flag_used	category_id_1	emergency_cnt	comment_cnt	ad_interest	ad_pfavcnt	adver_favorite_count	adver_item_count	adver_interest	adver_review_count	adver_comment_count
1	10	0.0	0.0	627.0	9.0	5649.0	595.0	11355.0	2661.0	1056.0

adver_pay_count	adver_parcel_post_count	adver_transfer_count	adver_chat_count
1526.0	0.0	11.0	1022.0

3 모델링

변수파악

독립변수 간 상관관계 (EDA)



3 모델링

DeepFM

FM Component

```
class FM_layer(tf.keras.layers.Layer):
    def __init__(self, num_feature, num_field, embedding_size , field_index):
        super(FM_layer, self).__init__()
        self.embedding_size = embedding_size # k : embedding vector dim
        self.num_feature = num_feature
        self.num_field = num_field
        self.field_index = field_index # col번호, 소속된 col번호

        # Parameters of FM layer
        # w : 1st order interactions
        # V : 2nd order interactions
        ## num_feature = len(field_index)=79, num_field = len(field_dict)=26
        self.w = tf.Variable(tf.random.normal(shape=[num_feature], mean=0, stddev=1),
        self.V = tf.Variable(tf.random.normal(shape=(num_field, embedding_size), mean
```

Deep Component

```
class DeepFM(tf.keras.Model):
    def __init__(self, num_feature, num_field, embedding_size, field_index):
        super(DeepFM, self).__init__()
        self.embedding_size = embedding_size
        self.num_feature = num_feature
        self.num_field = num_field
        self.field_index = field_index

        self.fm_layer = FM_layer(num_feature, num_field, embedding_size, field_index)

        self.layers1 = tf.keras.layers.Dense(units=64, activation='relu')
        self.dropout1 = tf.keras.layers.Dropout(rate=0.2)
        self.layers2 = tf.keras.layers.Dense(units=32, activation='relu')
        self.dropout2 = tf.keras.layers.Dropout(rate=0.2)
        self.layers3 = tf.keras.layers.Dense(units=16, activation='relu')
        self.dropout3 = tf.keras.layers.Dropout(rate=0.2)
        self.layers4 = tf.keras.layers.Dense(units=2, activation='relu')

        self.final = tf.keras.layers.Dense(units=1, activation='sigmoid')
```

Train DeepFM Model

```
def train(epochs):
    train_ds, test_ds, field_dict, field_index = get_data()

    model = DeepFM(embedding_size=EMBEDDING_SIZE, num_feature=len(field_index),
                   num_field=len(field_dict), field_index=field_index)
    #optimizer = tf.keras.optimizers.SGD(learning_rate = 0.01)
    optimizer = tf.keras.optimizers.Adam(learning_rate=0.01)
```

3 모델링

DeepFM

Deep FM

- Embedding Layer 구조에는 두 가지 흥미로운 점이 있다.
- **다른 field의 input vector의 길이는 다를 수 있지만 embedding은 같은 크기(k)**이다.
  - 예시: gender field는 보통 length가 남, 여 2인 반면 국적이나 나이 field의 길이는 더 길다.
    하지만 embedding시에는 똑같이 k=5차원 벡터로 임베딩 된다.
- FM component와 Deep component가 같은 feature embdding을 공유한다는 점이 주목할 만 한데 이 덕분에 두 가지 중요한 장점을 가짐.

  1) raw feature로부터 낮은 차원과 높은 차원의 피쳐 상호작용을 둘 다 학습할 수 있다.

  2) Wide & Deep 모델과 다르게 input의 직접적인 feature engineering이 필요하지 않다.
- **FM에서의 latent feature vector(V)가 이 네트워크의 가중치로 사용되고 input field vector를 압축하는 데 사용되고 학습**된다.

$< FMmodel - based >$

$$\hat{y}(x) := w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle v_i, v_j \rangle x_i x_j \quad (1)$$

$$\langle v_i, v_j \rangle := \sum_{f=1}^{k} v_{i,f} \cdot v_{j,f} \quad (2)$$

$< DeepFMmodel >$

$$\hat{y} = sigmoid(y_{FM} + y_{DNN}) \quad (3)$$

3 모델링

DeepFM

Epoch[1]	Batch Size	Loss	Accuracy	AUC score
10	512	0.3440	0.8401	0.9229
20	512	0.3308	0.8475	0.9291
30	512	0.3245	0.8511	0.9319
40	512	0.3209	0.8535	0.9335
50	512	0.3180	**0.8550**	**0.9347**
Test Data	512	-	**0.7013**	**0.6710**

- train, validation data: 불균형 처리 (SMOTE) 사용

1) Embedding size = 10 일 때의 결과

3 모델링

Wide&Deep

Wide & Deep

► **Wide component**

- 변수들 간 상호작용을 직접 지정해줘야 되기 때문에 도메인 지식이 요구되는 부분.
- 변수들 간 상호작용 개수에 따른 성능 차이 크지 않음

► **Deep component**

- Layer의 개수, dropout 비율에 따른 성능 비교
- Layer의 개수가 많아진다고 성능이 좋아지지는 않음
- dropout 비율이 0.5일 때가 가장 적절해 보임.

```
# columns for Wide model
wide_cols = ['imp_hour','user_gender', 'user_age_group','flag_used','category_id_1']
x_cols = (['imp_hour', 'user_gender'],['imp_hour', 'user_age_group'],['imp_hour', 'flag_used'],
        ['imp_hour', 'category_id_1'],['user_gender', 'user_age_group'], ['user_gender', 'flag_used'],
        ['user_gender', 'category_id_1'],['user_age_group', 'flag_used'],['user_age_group', 'category_id_1'],
        ['flag_used', 'category_id_1'])

# columns for Deep model
embedding_cols = ['imp_hour','user_gender', 'user_age_group','flag_used','category_id_1']
cont_cols = ['bid_price_x', 'user_age', 'user_transfer_count', 'price','ad_interest',
                'adver_interest', 'adver_review_count','adver_pay_count']

target = 'click_label'
```

<Input data columns>

```
def wide_deep(df_train, df_test, wide_cols, x_cols, embedding_cols, cont_cols,method):

    # wide model data
    X_train_wide, y_train_wide, X_test_wide, y_test_wide = wide(df_train, df_test, wide_

    # deep model data
    X_train_deep, y_train_deep, X_test_deep, y_test_deep, deep_inp_embed, deep_inp_layer

    X_train_wide_deep = [X_train_wide] + X_train_deep
    Y_train_wide_deep = y_train_deep # wide or deep the same
    X_test_wide_deep = [X_test_wide] + X_test_deep
    Y_test_wide_deep = y_test_deep # wide or deep the same

    activation, loss , metrics = fit_param[method]

    if metrics:
        metrics = [metrics]

    # Wide
    w = Input(shape=(X_train_wide.shape[1],), dtype='float32', name='Wide')

    # Deep
    d = concatenate(deep_inp_embed)
    d = Flatten()(d)
    d = Dense(100, activation='relu', kernel_regularizer=l1_l2(l1=0.001, l2=0.001))(d)
    d = Dropout(0.5)(d)
    d = Dense(50, activation='relu')(d)
    d = Dropout(0.5)(d)
    d = Dense(20, activation='relu')(d)
    d = Dropout(0.5)(d)
```

<Wide & Deep model code>

3 모델링

Wide&Deep

epochs	batch_size	Deep component 구성 조건	Wide 조합 개수	ACC	AUC	TEST_ACC	TEST_AUC
10	512	dropout = 0.2, layer 2개 l1=0.01, l2=0.01, learning_rate=0.001	2개	0.6752	0.7230	0.5854	0.6450
10	512	dropout = 0.2, layer 2개 l1=0.01, l2=0.01, learning_rate=0.001	5개	0.6813	0.7301	0.5770	0.6469
10	512	dropout = 0.2, layer 2개 l1=0.01, l2=0.01, learning_rate=0.001	10개	0.6834	0.7323	0.5820	0.6484
10	1024	dropout = 0.5, layer 3개 l1=0.001, l2=0.001, learning_rate=0.0001	10개	0.6798	0.7300	0.5947	0.6434
10	1024	dropout = 0.5, layer 3개 l1=0.001, l2=0.001, learning_rate=0.003	10개	0.6837	0.7340	0.5895	0.6431
10	1024	dropout = 0.5, layer 5개 l1=0.001, l2=0.001, learning_rate=0.003	10개	0.6833	0.7318	0.5875	0.6468

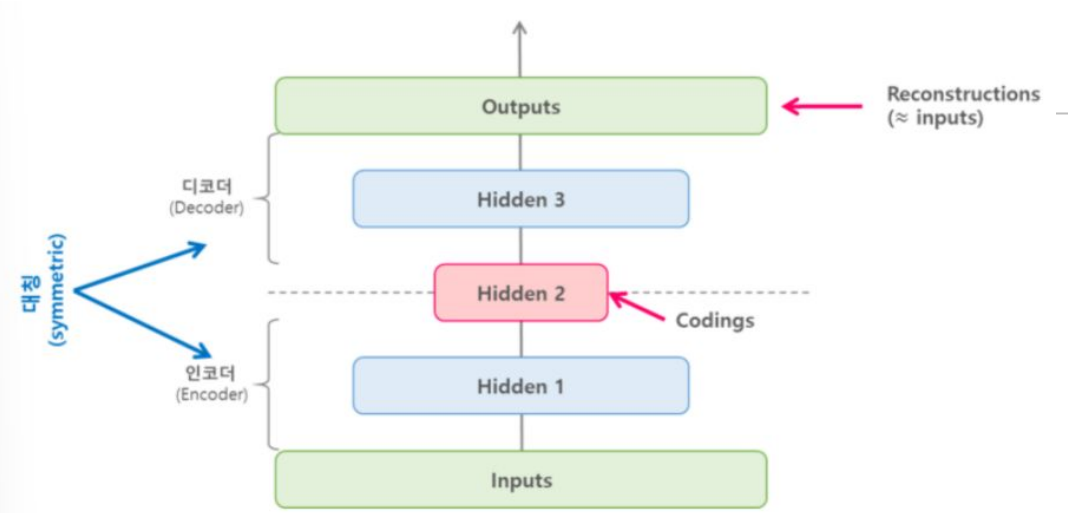
*2개:['imp_hour', 'user_gender'],['imp_hour', 'user_age_group']
*5개:['imp_hour', 'category_id_1'],['user_gender','user_age_group'],['user_age_group', 'category_id_1']

3 모델링

Autoencoder

Stacked Autoencoder

- 추가된 hidden 레이어를 기준으로 인코더와 디코더는 **대칭 구조**
- 완벽하게 대칭 구조를 이룰 때는 일반적으로 인코더와 디코더의 가중치를 묶게 되는데 이렇게 할 경우 **모델의 가중치 수를 절반으로 줄여 훈련속도를 높이고 overfitting 위험을 줄일 수 있다.**
- modeling 조건:
  a. 데이터 셋 기준: Deep FM과 동일
  b. epoch = 200, metrics= binary_accuracy
  c. validation&train data에만 SMOTE 적용
  d. layer: encoding, decoding 각각 1개의 layer (150)
     + latent layer(15)
  e. layer 마다 l2 규제 추가하여 layer 구성하고 학습
  f. batch_size = 512
  g. early_stopping 규제 적용



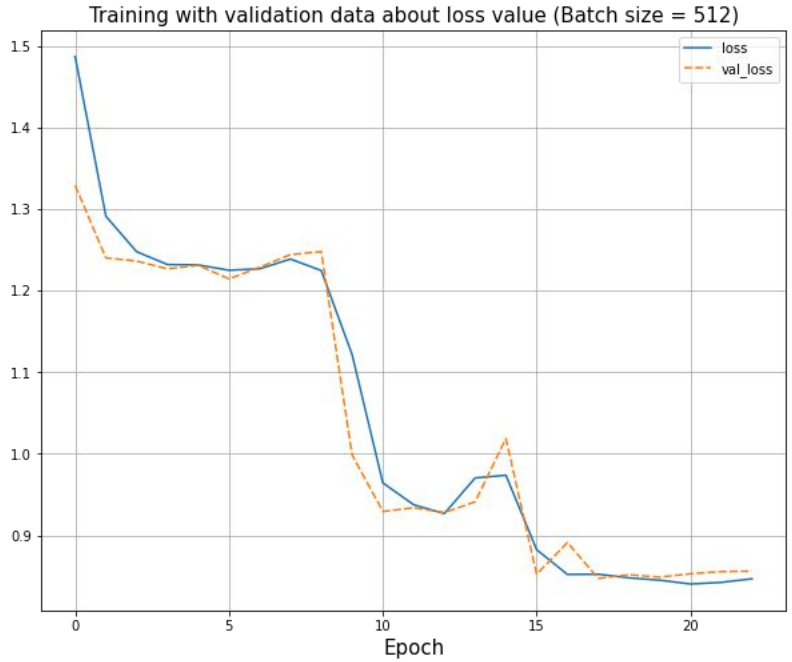

```
# Stacked autoencoder code
## input layer
input_layer = Input(shape=(x_train.shape[1],))
print(f"input layer's shape: {input_layer.shape}")

## encoding architecture
encode_layer = Dense(150, activation='relu', activity_regularizer = 'l2')(input_layer)

## latent view
latent_view  = Dense(10, activation='sigmoid')(encode_layer)

## decoding architecture
decode_layer = Dense(150, activation='relu', activity_regularizer = 'l2')(latent_view)

## output layer
output_layer = Dense(x_train.shape[1],)(decode_layer)
print(f"output layer's shape: {output_layer.shape}")

# layer_loss = tf.losses.binary_crossentropy(input_layer, output_layer)
# print(f"binary_crossentropy of Autoencoder model: {layer_loss}")

model = tf.keras.Model(input_layer, output_layer)
print(model.summary())
```

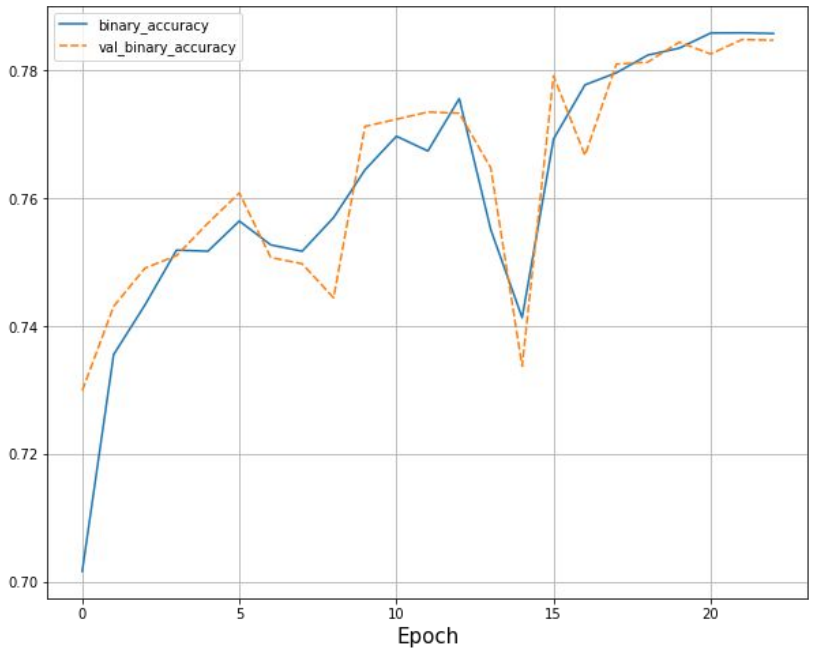
3 모델링

Autoencoder

- model fitting

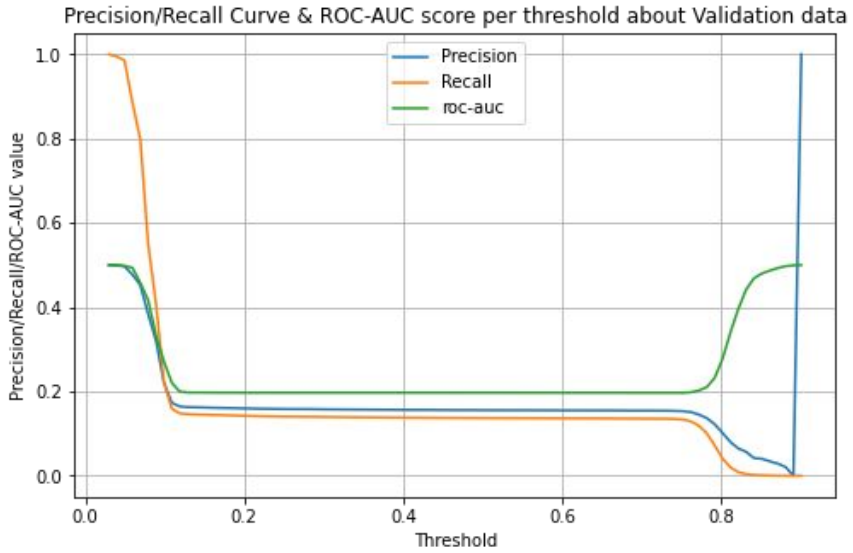
```
# patience: patience 는 성능이 증가하지 않는 epoch 을 몇 번이나 허용할 것인가를 정의
learning_rate = 0.01
optimizer = keras.optimizers.Adam(learning_rate=learning_rate)
model.compile(optimizer=optimizer, loss = 'binary_crossentropy', metrics = ['binary_accuracy'])
early_stopping = EarlyStopping(monitor='val_loss', patience=5, verbose=1, mode='auto')
start = perf_counter()
history = model.fit(x_train, y_train, epochs=200, batch_size=512, validation_data=(x_val, y_val), callbacks=[early_stopping])
print("End of Training")
print("걸린 시간: {:g}분 {:.2f}초".format((perf_counter() - start)//60, round((perf_counter() - start)%60)))
```



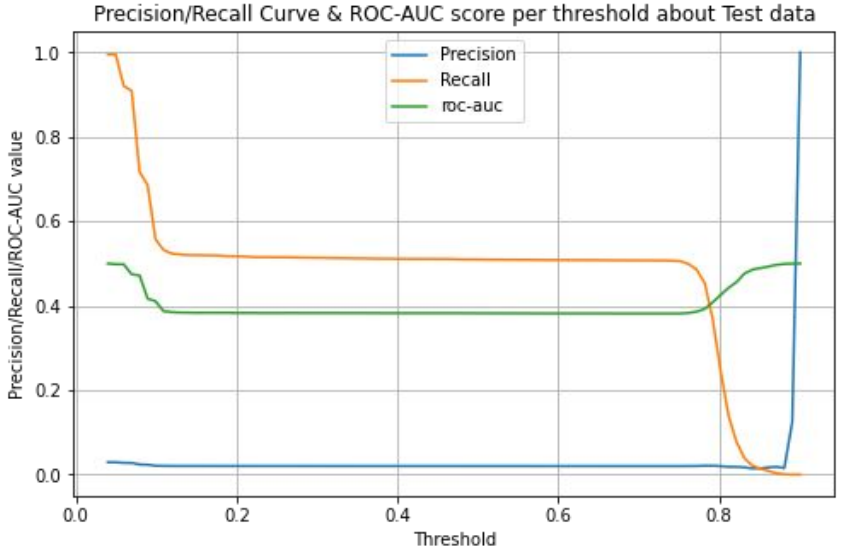


3 모델링

Autoencoder




```
threshold: 0.9009901285171509
[[     0 114485]
 [     9 114476]]
              precision    recall  f1-score   support

           0       0.00      0.00      0.00    114485
           1       0.50      1.00      0.67    114485

    accuracy                           0.50    228970
   macro avg       0.25      0.50      0.33    228970
weighted avg       0.25      0.50      0.33    228970

f1 score:  0.6666317266761005
ROC-AUC: 0.4999606935406385
```




```
threshold: 0.9009901285171509
[[245325      0]
 [  7414      0]]

              precision    recall  f1-score   support

           0       0.97      1.00      0.99    245325
           1       0.00      0.00      0.00      7414

    accuracy                           0.97    252739
   macro avg       0.49      0.50      0.49    252739
weighted avg       0.94      0.97      0.96    252739

f1 score:  0.0
ROC-AUC: 0.5
```

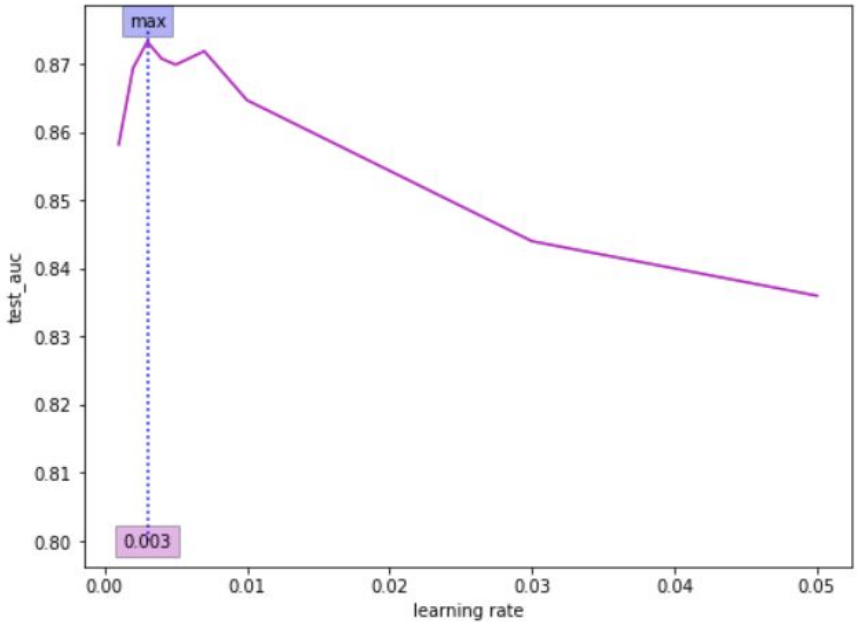
3 모델링

DNN

DNN

▶ 가장 기본적인 DL 모델 구조 설계

- **Layer개수**: 4,6,8,10
- **dropout layer**이 부재하면 overfit되며, 너무 많거나 dropout 비율이 크면 underfit되는 경향을 보임. (dropout 비율 **0.1**로 결정)
- **Optimizer** : sgd,adam,adagrad,rmsprop 중에서 **adam**이 가장 성능이 우수
- **Learning rate** : 가장 높은 성능을 보인 **0.003**로 결정




```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 200)               16000
dropout (Dropout)            (None, 200)               0
dense_1 (Dense)              (None, 100)               20100
dropout_1 (Dropout)          (None, 100)               0
dense_2 (Dense)              (None, 50)                5050
dropout_2 (Dropout)          (None, 50)                0
dense_3 (Dense)              (None, 1)                 51
=================================================================
Total params: 41,201
Trainable params: 41,201
Non-trainable params: 0
```

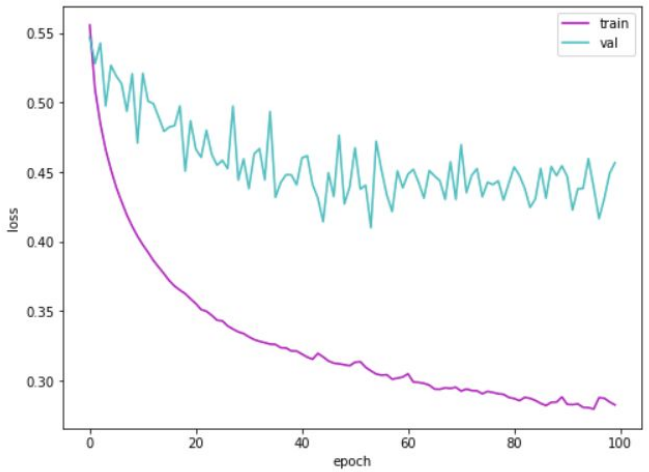
```
model = keras.Sequential()
model.add(keras.layers.Dense(200, activation='relu', input_dim=79))
#model.add(keras.layers.Dropout(0.1))
model.add(keras.layers.Dense(100,activation='relu'))
#model.add(keras.layers.Dropout(0.1))
model.add(keras.layers.Dense(50, activation='relu'))
#model.add(keras.layers.Dropout(0.1))
model.add(keras.layers.Dense(1,activation='sigmoid'))
```

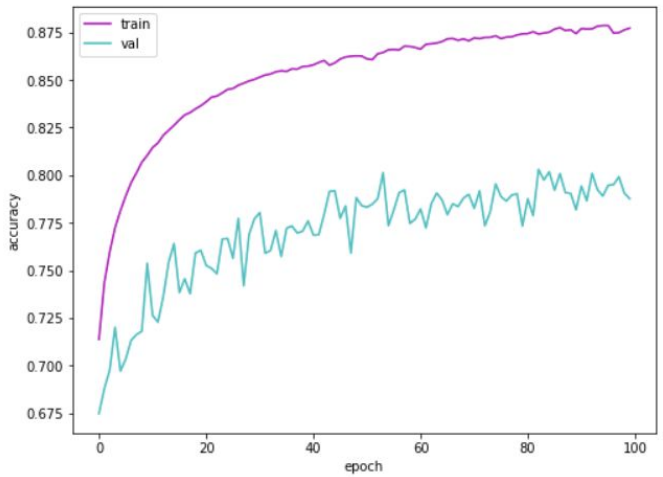
3 모델링

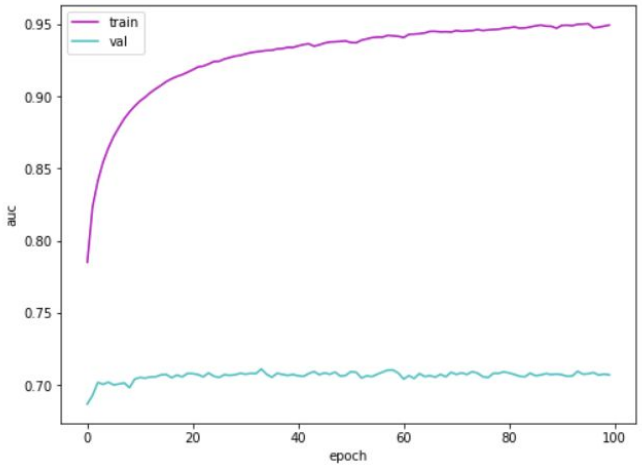
DNN

공통조건 : learning_rate = 0.003 / dropout = 0.1 / adam optimizer / binary crossentropy

epochs	batch_size	Layer	ACC	AUC	TEST_ACC	TEST_AUC
100	1000	Layer 4개	0.8188	0.9005	0.7598	0.7063
100	1000	Layer 6개	0.8427	0.9217	0.7818	0.7075
100	1000	Layer 8개	0.8592	0.9348	0.7832	0.7061
100	1000	Layer 10개	0.8665	0.9421	0.7863	0.7043







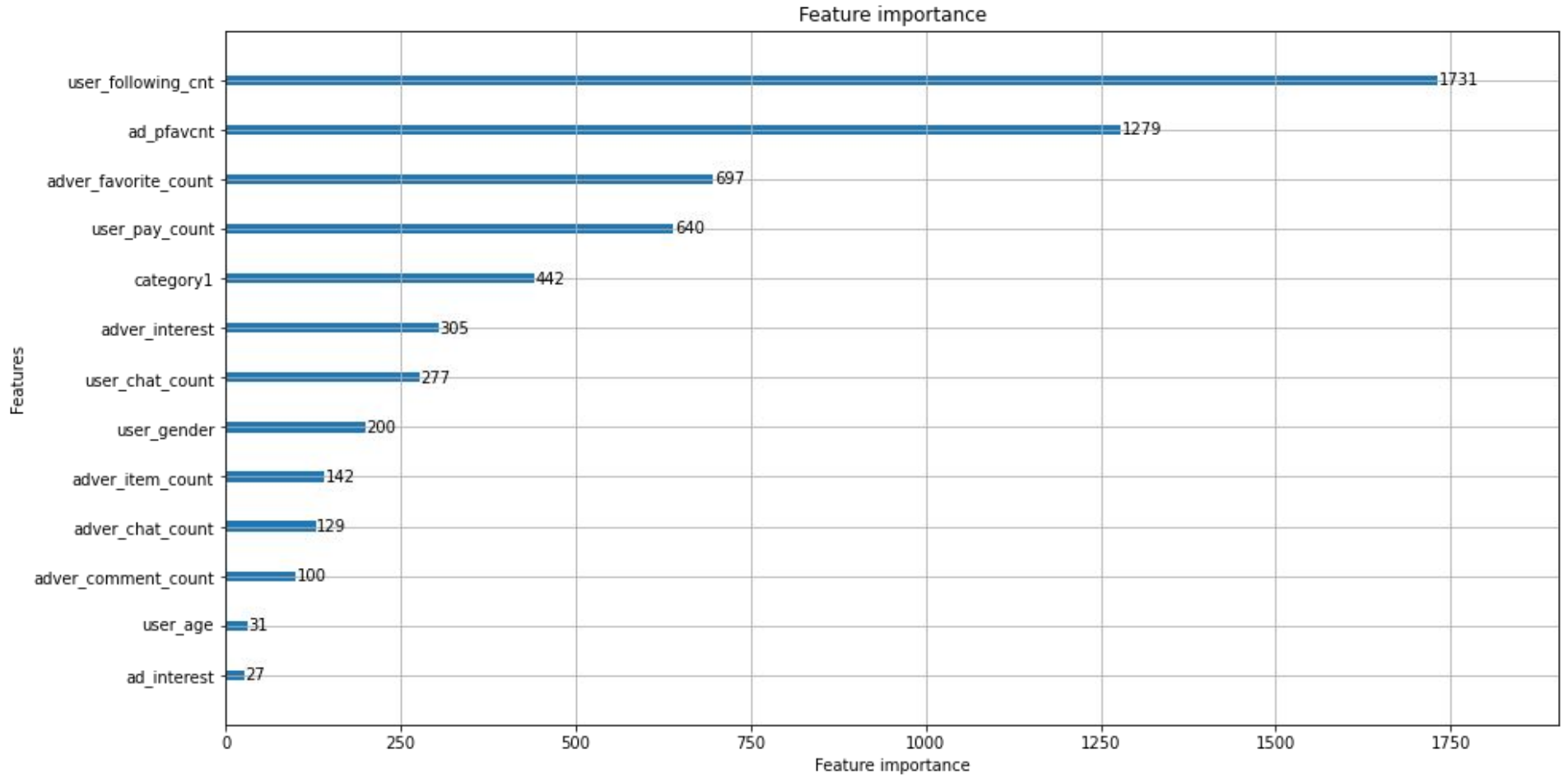
<Layer 10개>

*epoch는 10,50,100 중 100번이 가장 성능이 우수.

3 모델링

Boosting ML

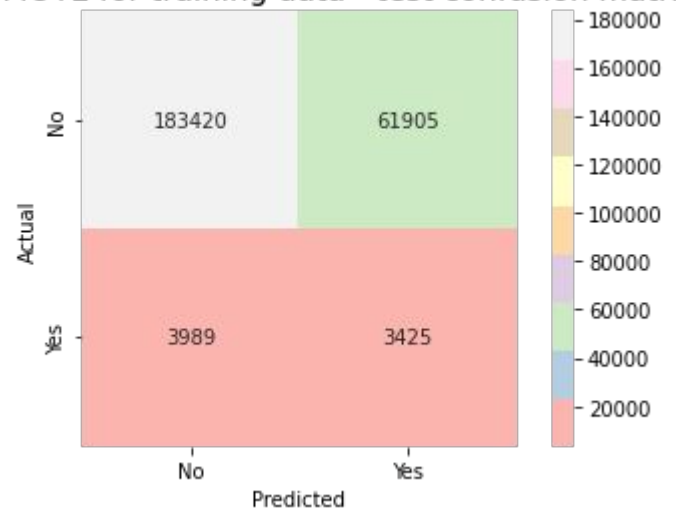
Light GBM



3 모델링

Boosting ML

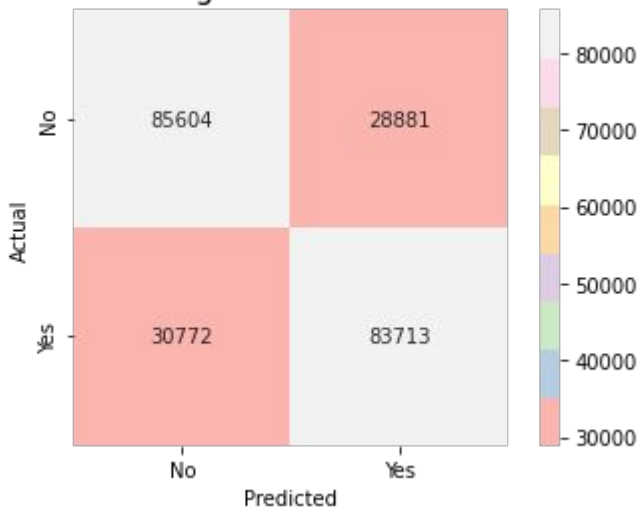
After SMOTE for training data - test confusion matrix



Light GBM 분류 report:

	precision	recall	f1-score	support
0	0.98	0.75	0.85	245325
1	0.05	0.46	0.09	7414
accuracy			0.74	252739
macro avg	0.52	0.60	0.47	252739
weighted avg	0.95	0.74	0.83	252739

After SMOTE for training data - validation confusion matrix



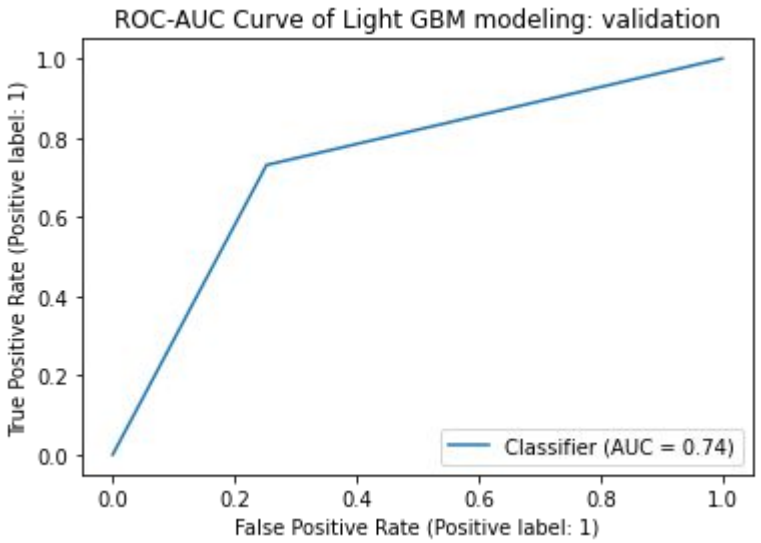
Light GBM 분류 report:

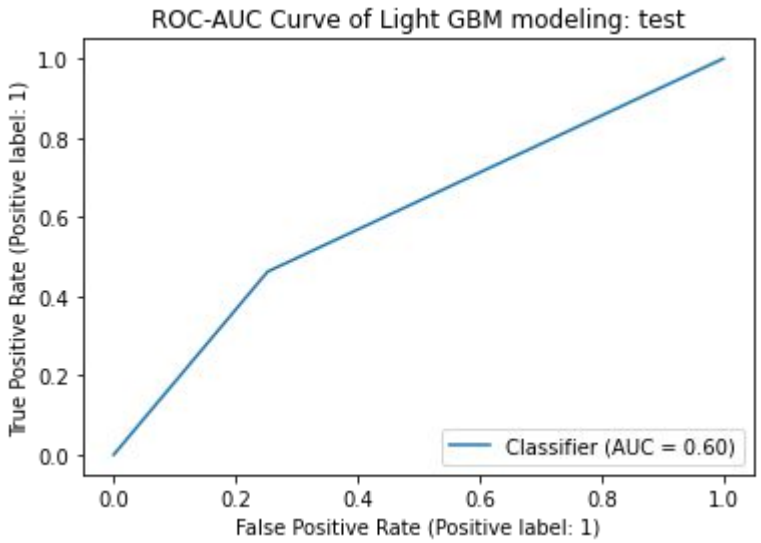
	precision	recall	f1-score	support
0	0.74	0.75	0.74	114485
1	0.74	0.73	0.74	114485
accuracy			0.74	228970
macro avg	0.74	0.74	0.74	228970
weighted avg	0.74	0.74	0.74	228970

3 모델링

Boosting ML

ROC-AUC score for Validation data	ROC-AUC score for Test data
0.7394724199676813	0.6048125589197795





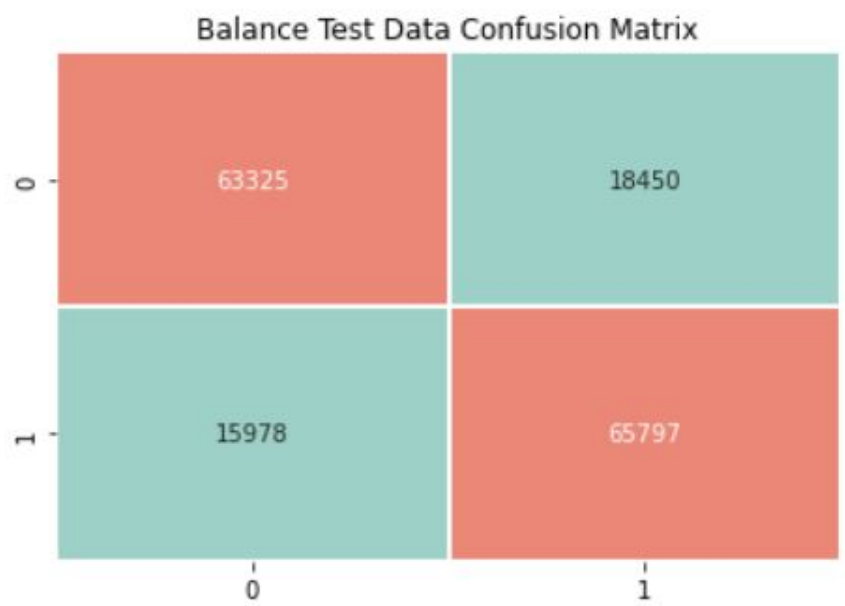
3 모델링

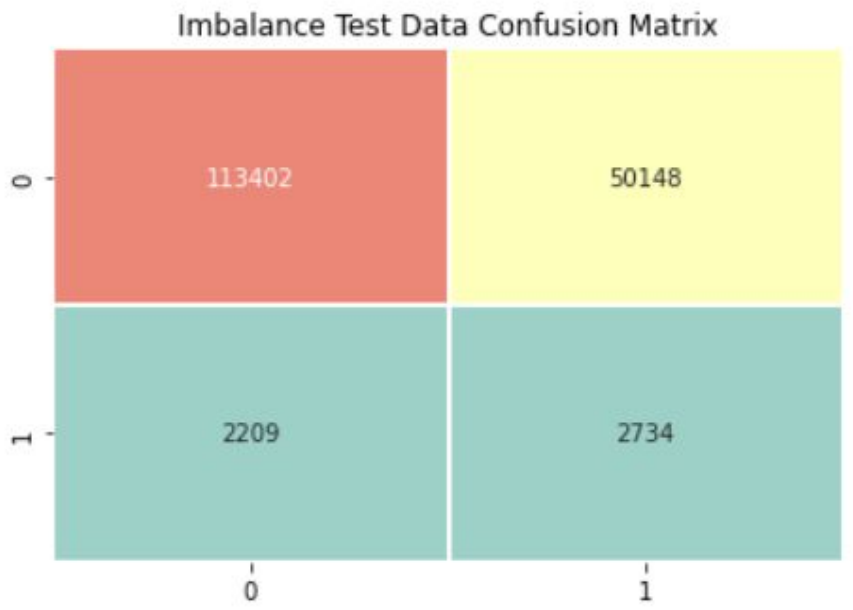
Boosting ML

CatBoost

```
cat = CatBoostClassifier(learning_rate=0.01, depth=10)
cat.fit(X,Y)
```

	TEST_ACC	TEST_AUC
balance test data	0.7895	0.7895
imbalance test data	0.6893	0.6232





4 결론

- 모델링 결과
- 한계점
- 시사점

4 결론

모델링 결과

	ACC	AUC	TEST_ACC	TEST_AUC
DeepFM	**0.8550**	**0.9347**	**0.7013**	**0.6710**
Wide&Deep	0.6798	0.7300	0.5947	0.6434
Autoencoder	0.4999	0.4999	0.9706	0.5
DNN	**0.8665**	**0.9421**	**0.7863**	**0.7043**
LightGBM	0.740	0.7397	0.739	0.6048
CatBoost	0.7897	0.7897	0.6893	0.6232

*ACC/AUC: balance한 데이터
*TEST_ACC/AUC: imbalance한 데이터

4 결론

한계점

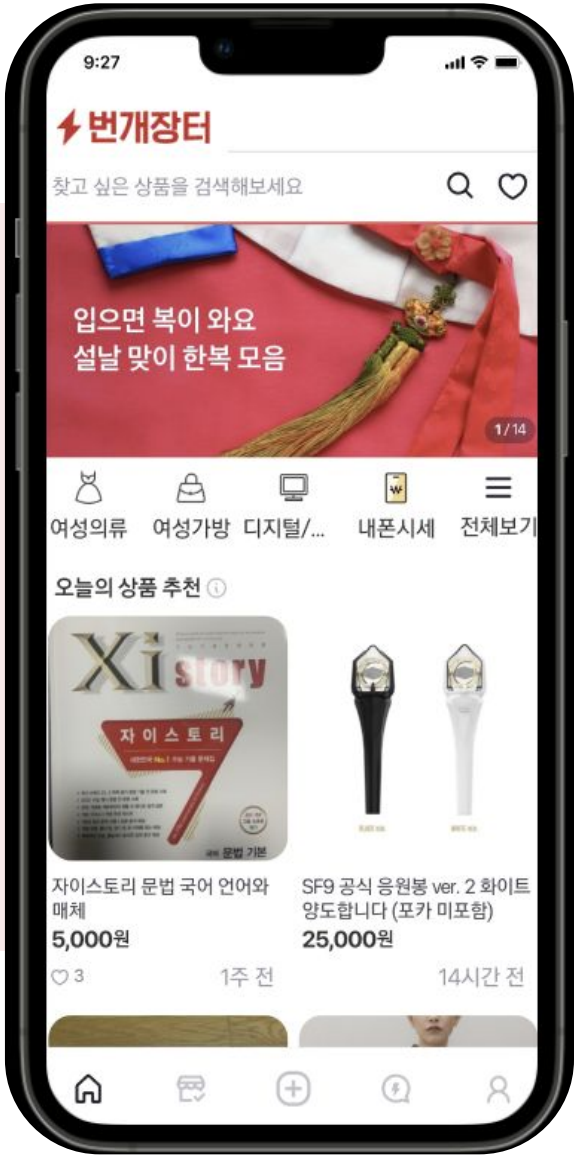
Model	한계점
DeepFM	- **불균형 처리를 한 훈련 데이터에 한해** 성능이 다른 모델들에 비해 가장 나은 편임. - **오버샘플링의 영향력 및 과적합으로 인해 테스트 데이터에 있어서 성능이 낮게** 나온 문제가 있음. - 전반적인 훈련 데이터의 성능은 80~90%, 테스트 데이터는 훈련데이터에 비해 약 20% 낮음.
Wide&Deep	- **전문지식을 바탕으로 한 feature engineering이 요구됨.** - 번개장터 데이터의 경우 상호작용 변수 쌍의 개수가 모델 성능에 큰 영향은 없었음. - **전반적으로 성능의 한계가 있음**(epoch를 늘려도 성능이 개선되지 않음)
Autoencoder	- 임곗값(threshold)가 늘어나면서 테스트 및 검증데이터의 정밀도는 0으로 떨어지다가 급격하게 증가하며 재현율은 0으로 수렴하는 모양을 보여줌. - **성능이 전반적으로 좋다고 보기는 어려움**.
DNN	- 범주형 변수가 존재하여 Sparse한 성격을 갖는 데이터지만 모델링할 때 이를 특별히 고려하지 않음. - **원본 데이터가 imbalance하여 모델 학습이 잘 되지 않음**.
LightGBM	- **데이터의 불균형으로 인해 과적합** 문제가 발생하는 한계점이 존재.
CatBoost	- 기존의 Boosting계열의 ML 모델들보다는 성능이 우수하다고 알려져 있지만 **데이터 자체가 학습하기 어려운 imbalance 데이터였기 때문에 우수한 성능으로 학습을 시키기에는 한계가 존재**.

4 결론

시사점

최고의 시너지가 발생할 수 있도록 광고주와 고객 연결해주기

- 특정 광고를 클릭할 확률이 높을 것으로 예상되는 고객에게 해당 광고를 노출.
- 앱 사용 시간대, 연령대, 성별 등을 반영하여 최적의 조합 제공.
- 광고주에게
  - 시간대별, 성별, 연령대별 클릭률이 높은 상품에 대한 데이터를 실시간 제공.
  - 상품의 카테고리별로 어떤 유형의 고객이 해당 광고를 클릭할 확률이 높은지 정보 제공



감사합니다.