불균형 데이터와 설명 가능성을 고려한 게임 재화 어뷰징 검출을 위한 지도, 자기지도 및 준지도 학습 모델

Update date	2023/11/22
Tag	**#AbuseDetection #MultipleLearningModel #ImblancedData #Explainability**
Editor	민희수

▼ Table of Contents

Table of Contents
들어가며
게임 어뷰징(Game Abuse)이란?
분석 목표
통계 및 탐색적 데이터 분석
  데이터 확인
  EDA
  Feature Engineering
1. 잠재 표현 추출
  Autoencoder를 이용한 잠재 표현 찾기
  Process
2. 지도 학습 모델
  Prepation
  Baseline Model
3. 자기지도 학습 모델
  Prepation
  TabNet
4. 준지도 학습 모델
  Prepation
  Label Propagation
  AutoEncoder
적용 및 결과
  성능 비교
  특징의 중요도
  어뷰저 검출 기준
  개선 방향
마치며

들어가며

현재 특정 온라인 RPG 게임에서는 골드를 부당하게 얻는 어뷰저를 탐지하고 제재하는 업무를 수행하고 있습니다. 이미 일부 유저에 대해서는 정상적인 플레이어 여부를 확인했지만, 아직 어뷰저 여부를 판별할 수 없는 다수의 유저가 남아 있습니다. 이러한 상황에서 모든 유저를 개별적으로 조사하는 것은 비효율적이므로 데이터 분석을 통해 어뷰저를 자동으로 식별하는 로직을 개발하고자 합니다.

게임 어뷰징(Game Abuse)이란?

게임 어뷰징(Game Abuse)은 온라인 게임 환경에서 발생하는 부정한 행위로 정의됩니다. 이는 게임의 규칙을 이용하여 부정한 이득을 얻거나 다른 플레이어들에게 불이익을 주는 행동을 포함합니다. 게임 어뷰징은 다양한 형태로 나타날 수 있으며, 이는 게임 업계에서 주의를 기울여야 할 중요한 문제 중 하나입니다.

대표적인 어뷰징 형태에는 타인의 계정으로 등급을 올리는 대리랭, 고의로 연거푸 패배하는 패작 행위가 있습니다. 또한, 게임 어뷰징의 한 형태로 '재화 어뷰징' 이 있습니다. 이는 게임 내에서 사용되는 가상 아이템 또는 화폐와 관련된 것으로, 부정한 방법을 통해 게임 내에서 어떤 형태의 가치를 얻는 행위를 나타냅니다. 예를 들어, 게임 내 아이템의 가치를 인위적으로 조작하거나, 부정한 방법으로 게임 화폐를 적절한 규칙을 따르지 않고 얻는 것이 여기에 속합니다.

게임 어뷰징은 게임의 공정성과 경제적인 안정성에 부정적인 영향을 미칠 수 있으며, 이로 인해 다양한 문제가 발생할 수 있습니다. 때문에 이를 방지하고 검출하는 것은 게임 운영자 및 개발자들에게 중요한 작업으로 간주되고 있습니다.

분석 목표

분석 목표는 게임 내에서 발생하는 재화 어뷰징을 효과적으로 감지하고 예방하기 위한 방법을 개발하는 것입니다.

- **어뷰징 감지 정확도 향상** : 재화 어뷰징은 다양한 형태로 나타날 수 있으며, 이를 효과적으로 감지하는 정확한 모델을 개발합니다. 이를 통해 어뷰징 행위를 빠르게 식별하고 대응할 수 있습니다.
- **데이터 불균형 다루기** : 게임 데이터는 종종 어뷰징 행위가 드물게 발생하는 불균형한 분포를 가집니다. 이는 학습의 편향을 줄 수 있습니다. 이러한 불균형한 데이터를 해결하기 위해 잠재표현과 label propagation을 추가하여 모델을 훈련시키고 검증합니다.
- **설명 가능한 모델 구축** : 모델이 어뷰징을 감지한 이유를 설명할 수 있는 모델(tabnet)을 추가하였습니다. 이는 게임 운영자나 관리자가 모델의 의사결정을 이해하고 신뢰할 수 있도록 도움을 줍니다.
- **지도/자기지도/준지도 학습 사용** : 복잡한 고급 알고리즘 대신 학습의 결과를 이해하기 쉬운 지도 학습을 사용합니다. 또한 데이터 양이나 레이블이 부족한 상황에서도 효과적으로 어뷰징을 감지하도록 자기지도 학습과 준지도 학습 모델을 사용합니다.

이를 통해 최종적으로 어뷰징에 대한 대응을 강화하고 게임의 공정성과 안정성을 유지하는 데 기여할 수 있습니다.

통계 및 탐색적 데이터 분석

데이터 확인

```
[Feature]
- newID : 캐릭터 식별자(ID), 범주형
- char_jobcode : 캐릭터 직업, 범주형
- char_level : 캐릭터의 레벨, 수치형
- logging_timestamp : 해당 스냅샷 로그가 찍힌 시점
- charStatA~G : 캐릭터의 스탯 정보 (예: 캐릭터의 매력도), 수치형
- socialAmountA : 타 캐릭터와의 인터렉션 정보로 그 양을 의미 (예: 등록된 친구 수), 수치형
- socialBooleanA~B : 타 캐릭터와의 인터렉션 정보로 발생 여부를 의미 (0 또는 1, 예: 길드 가입 여부), 범주형
- socialSessionAmountA~B : 타 캐릭터의 인터렉션 정보로 스냅샷 세션 내 발생한 양을 의미 (예: 세션 중 발생시킨 채팅 횟수), 수치형
- activityCumulativeAmountA~B : 캐릭터의 액션 정보로 생성시점부터 누적되는 값 (예: 누적 이벤트X 발생횟수), 수치형
- accountMetaAmountA : 캐릭터가 속한 계정의 메타 정보로 세션에 관계없이 갱신 (예: 캐릭터가 소속된 계정상의 정보), 범주형
- charSessionAmountA~D : 캐릭터가 세션 중 발생시킨 양 (예: 세션 중 특정맵A 방문 횟수), 수치형
- actionSessionAmountA~N : 캐릭터가 세션 중 이동한 양, 수치형
- tradeSesstionAmountA~N : 캐릭터가 세션 중 발생시킨 액션의 양 (예: 세션 중 사냥 횟수), 수치형
- tradeSessionAmountA~E : 캐릭터가 세션 중 발생시킨 거래의 양 (예: 세션 중 개인 간 거래 횟수), 수치형

[Target label]
- isAbuser : 어뷰저 여부 (0: 어뷰저X, 1:어뷰저, -1:아직 조사X), 범주형
```

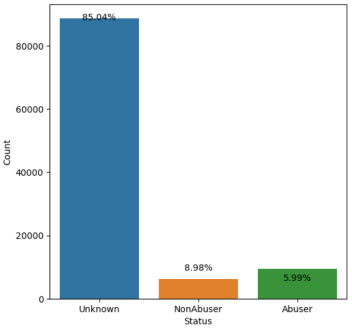
EDA

1. **Describe을 통한 다양한 통계량 요약**

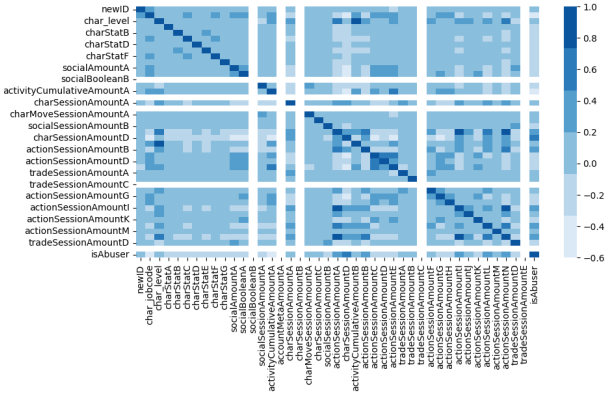
	count	unique	top	freq	mean	std	min
newID	104399.0	NaN	NaN	NaN	NaN	7484.51215	4318.413739
char_jobcode	104399.0	NaN	NaN	NaN	NaN	35.396651	21.345751
char_level	104399.0	NaN	NaN	NaN	NaN	48.324553	22.407708
logging_timestamp	104399	99223	2022-06-13 01:59:12	5	NaN	NaN	
…							
actionSessionAmountN	104399.0	NaN	NaN	NaN	NaN	1219.530082	1095.172136
tradeSessionAmountD	104399.0	NaN	NaN	NaN	NaN	0.054972	0.44824
tradeSessionAmountE	104399.0	NaN	NaN	NaN	NaN	0.0	0.0
isAbuser	104399.0	NaN	NaN	NaN	NaN	-0.760639	0.60131

2. **결측치 확인**
    - 결측치를 확인하였고, 결측값이 존재하지 않아 별도의 전처리 과정은 추가하지 않았습니다.

3. **Target label 확인**
    - 레이블이 없는 데이터(Unkown)의 비율이 85% 이상이므로 이를 해결하기 위해 **자기지도, 준지도 학습 모델을 추가**하였습니다.
    - 어뷰저(Abuser)는 9,370개 (59.99%), 어뷰저 아님(Non Abuser)는 6,249개 (40.01%) 이므로 약간의 데이터 편향/불균형이 존재하여 이를 해결하기 위해 **잠재표현 추출과 label propagation을 추가**하였습니다.



4. **Featurer간 상관관계 (with isAbuser)**
    - 강한 상관관계를 가진 변수는 charSessionAmountD (0.491991), actionSessionAmountN (0.349754), char_jobcode (-0.340932) 등이 있습니다.
    - 약한 상관관계를 가진 변수는 socialBooleanB, accountMetaAmountA, charSessionAmountB, tradeSessionAmountC, tradeSessionAmountE 등이 있습니다.



5. **다중공선성**
    - 다중공선성을 통해 독립 변수 간의 상관관계를 확인할 수 있습니다.
    - Condition number가 2.43e+04이며 다중공선성의 문제가 존재하므로, 이를 해결하기 위해 **데이터 정규화 과정을 추가**하였습니다.

OLS Regression Results

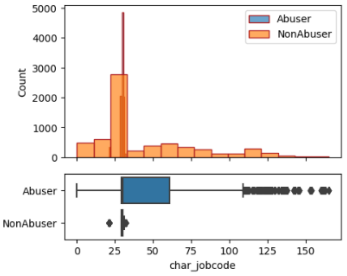
Dep. Variable:	isAbuser	R-squared:	0.301
Model:	OLS	Adj. R-squared:	0.301
Method:	Least Squares	F-statistic:	840.0
Date:	Wed, 22 Nov 2023	Prob (F-statistic):	0.00
Time:	03:56:55	Log-Likelihood:	-8222.1
No. Observations:	15619	AIC:	1.646e+04
Df Residuals:	15610	BIC:	1.653e+04
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.1769	0.027	6.525	0.000	0.124	0.230
charSessionAmountD	0.0650	0.002	27.582	0.000	0.060	0.070
actionSessionAmountN	9.846e-05	1.87e-05	5.273	0.000	6.19e-05	0.000
actionSessionAmountA	-2.031e-05	1.26e-05	-1.607	0.108	-4.51e-05	4.47e-06
char_jobcode	-0.0024	0.000	-13.935	0.000	-0.003	-0.002
activityCumulativeAmountA	-0.0011	0.000	-4.773	0.000	-0.002	-0.001
actionSessionAmountG	-0.1374	0.011	-12.409	0.000	-0.159	-0.116
actionSessionAmountE	0.0020	0.003	0.782	0.434	-0.003	0.007
actionSessionAmountC	-0.0065	0.001	-11.241	0.000	-0.008	-0.005

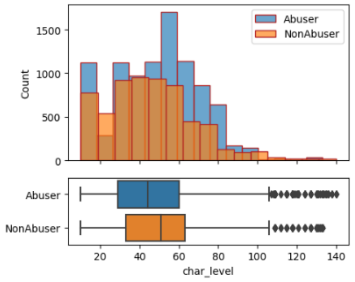
Omnibus:	2161.644	Durbin-Watson:	0.287
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1990.540
Skew:	-0.800	Prob(JB):	0.00
Kurtosis:	2.292	Cond. No.	2.43e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.43e+04. This might indicate that there are strong multicollinearity or other numerical problems.

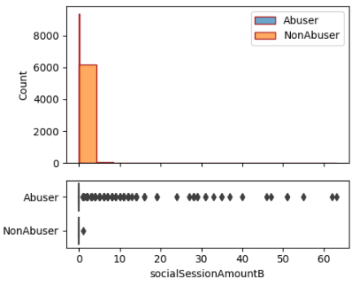
6. **Abuser VS Non Abuser 별 분포**
   - char_jobcode
     - 어뷰저의 jobcode는 균일하지 않고, 한 곳에 밀집되어 있으며 분명한 차이를 보입니다. 이는 어뷰저의 경우, 재화 획득을 위한 일부 직업을 선택하는 것으로 보입니다.



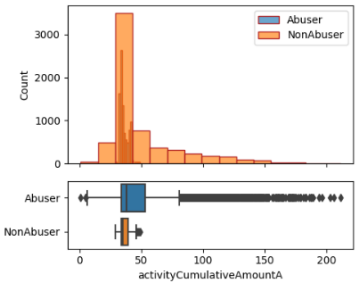
   - char_level
     - 어뷰저의 꼬리가 약간 더 앞서 있으나, 전체적으로 두 분포는 비슷합니다.



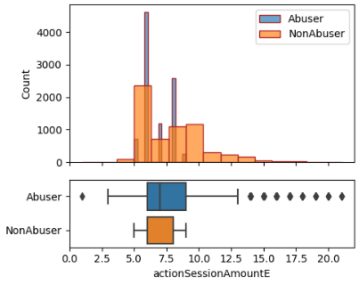
- socialSessionAmountB
  - 어뷰저는 타 캐릭터와 인터랙션 양이 눈에 띄게 많거나 적습니다. 이는 길드, 교환 및 일부로 져주기 등 비정상적으로 재화를 얻기 위해 타 캐릭터와 채팅 등을 활발히 하거나 인터렉션을 거의 하지 않는 것으로 보입니다.



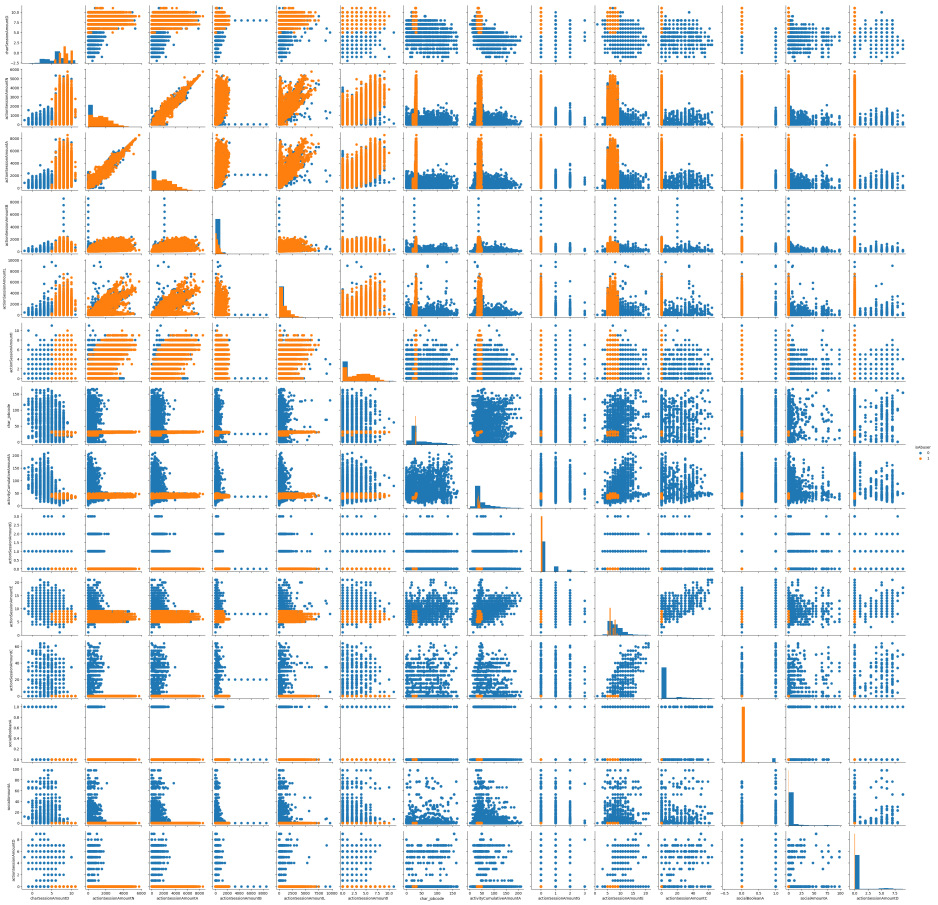
- activityCumulativeAmountA
  - 어뷰저는 액션 정보의 누적값이 눈에 띄게 많거나 적습니다. 이는 전문 작업장, 이벤트, 퀘스트 등을 통해 비정상적으로 재화를 얻는 것으로 보입니다.



- actionSessionAmountE
  - 어뷰저는 세션 중 발생시킨 거래의 양이 눈에 띄게 많거나 적습니다. 이는 타 캐릭터와의 아이템 교환 등을 통해 비정상적으로 재화를 얻거나 거의 거래를 하지 않는 것으로 보입니다.



7. **Abuser와 Non Abuser 데이터 간의 상관관계**



Feature Engineering

1. **Feature selection**

- 의미 있는 특성을 선택할 때, 일반적으로 p-value가 유의미하게 작은 특성들을 선택하는 것이 좋습니다. 즉, p-value가 0.05보다 작은 특성들을 우선적으로 고려하여 특성을 선택하였고, **tradeSessionAmountE 포함 8개 항목을 제외**하였습니다. 해당 내용은 아래와 같습니다.

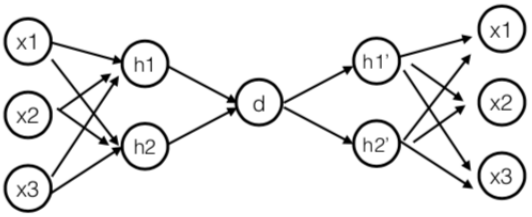
	Variable	Chi-square	p-value
3	logging_timestamp	15554.421488	4.189080e-01
30	tradeSessionAmountB	0.041592	8.384003e-01
39	actionSessionAmountM	0.041592	8.384003e-01
31	tradeSessionAmountC	0.000000	1.000000e+00
18	charSessionAmountB	0.000000	1.000000e+00
16	accountMetaAmountA	0.000000	1.000000e+00
13	socialBooleanB	0.000000	1.000000e+00
42	tradeSessionAmountE	0.000000	1.000000e+00

1. 잠재 표현 추출

Autoencoder를 이용한 잠재 표현 찾기

Autoencoder는 비지도 학습 방법으로, 입력 데이터를 더 낮은 차원의 잠재 표현으로 인코딩하고 디코딩하여 원래의 입력을 재구성하는 신경망입니다. 이 모델을 사용하면 데이터의 차원을 축소하면서 중요한 특성을 보존할 수 있습니다. 이를 통해 레이블이 있는 데이터에 대한 특성을 학습하는 데 도움이 되며, 학습된 잠재 표현은 더 낮은 차원에서 데이터의 특성을 잘 나타냅니다.

레이블이 없는 데이터의 비율이 높고, 데이터 불균형이 있다는 점을 고려하여 잠재 표현을 활용하여 분류 모델을 학습하였습니다. 이로써 모델의 일반화 성능을 향상시킬 수 있었습니다.



Process

1. Autoencoder를 설계하고 레이블이 있는 데이터를 사용하여 학습합니다.
2. 학습된 모델에서 Encoder 부분을 추출합니다.
3. 레이블이 있는 데이터를 Encoder 모델을 통해 특성으로 변환합니다.
4. 최종적으로 추출된 특성을 사용하여 분류 모델을 학습합니다.

```
# Representation 추출
rep_x = np.append(norm_hid_rep, fraud_hid_rep, axis = 0)
y_n = np.zeros(norm_hid_rep.shape[0])
y_f = np.ones(fraud_hid_rep.shape[0])
rep_y = np.append(y_n, y_f)

# Train/Test Dataset Split
train_X, test_X, train_y, test_y = train_test_split(rep_x, rep_y, test_size=0.20)
```

2. 지도 학습 모델

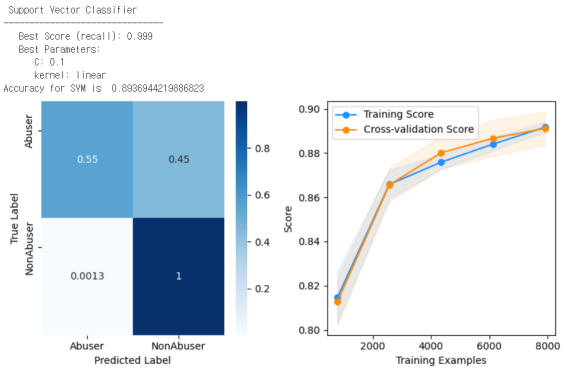
💡 Before score는 일반적인 모델을 이용하여 학습된 결과, Best score는 Prepation이 적용된 결과(3번 실행)입니다.

Prepation

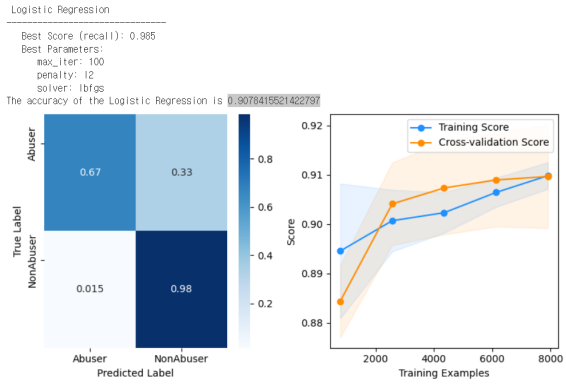
- Feature Selection
- Data Scaling, Data Normalization
- Latent Representation
- Cross Validation
- Hyper-Parameters Tuning

Baseline Model

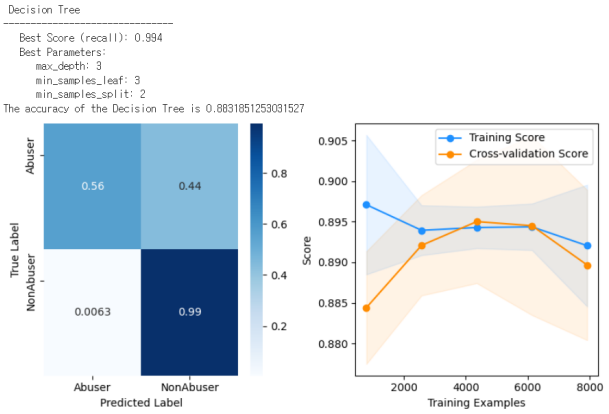
1. **Support Vector Classifier (Linear)**
    - Before score : 0.764
    - Best score : 0.915



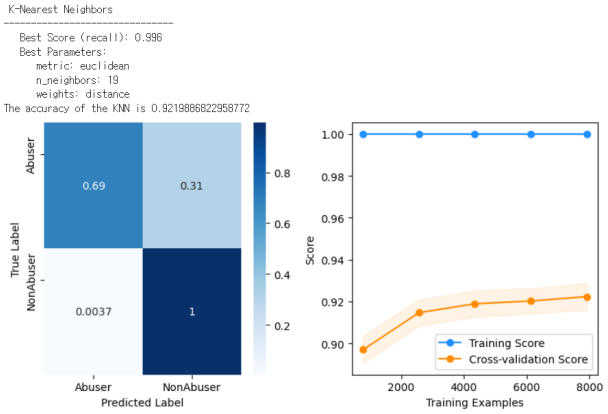
2. **Logistic Regression**
    - Before score : 0.805
    - Best score : 0.907



3. **Decision Tree**
    - Before score : 0.854
    - Best score : 0.918

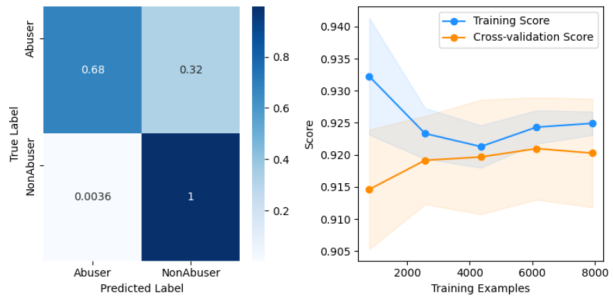


4. **K-Nearest Neighbors(KNN)**
   - Before score : 0.902
   - Best score : 0.930



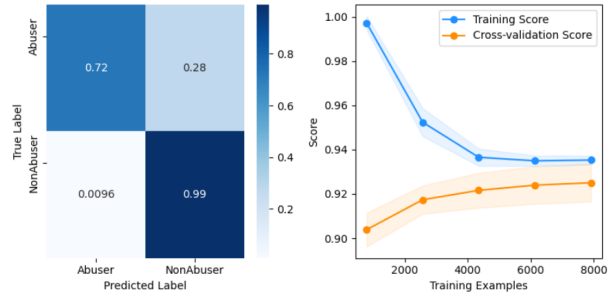
5. **Random Forest Classifier**
   - Before score : 0.852
   - Best score : 0.931

```
Random Forest
--------------------------------
  Best Score (recall): 0.996
  Best Parameters:
    bootstrap: False
    criterion: entropy
    max_depth: 6
    max_features: auto
    min_samples_leaf: 2
    min_samples_split: 2
    n_estimators: 100
The accuracy of the Random Forests is 0.9187550525464834
```



6. **Light Gradient Boosting Machine**
   - Before score : 0.775
   - Best score : 0.922

```
The accuracy of the LGBMClassifier is 0.9227970897332255
```



7. **Gaussian Naive Bayes**
   - Before score : 0.831
   - Best score : 0.881

```
gnb = GaussianNB()
gnb.fit(train_X, train_y)
gnb_prediction = gnb.predict(test_X)
print('The accuracy of the Gaussian Naive Bayes is', accuracy_score(gnb_prediction, test_y))
```

8. **Gradient Boosting Classifier**
   - Before score : 0.876
   - Best score : 0.931

```
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier(n_estimators=500,random_state=0,learning_rate=0.1)
gb.fit(train_X, train_y)
prediction=gb.predict(test_X)
print('The accuracy for Gradient Boostin is:', accuracy_score(prediction, test_y))
```

9. **Xtreme Gradient Boosting Classifier**
    - Before score : 0.880
    - Best score : 0.930

```
from xgboost import XGBClassifier
xgb = XGBClassifier(n_estimators=900,learning_rate=0.1)
xgb.fit(train_X, train_y)
prediction = xgb.predict(test_X)
print('The accuracy for XGBoost is:', accuracy_score(prediction, test_y))
```

10. **Bagging Classifier (KNeighborsClassifier)**

- Before score : 0.812
- Best score : 0.924

```
bc_kn = BaggingClassifier(base_estimator=KNeighborsClassifier(n_neighbors=3), random_state=0, n_estimators=700)
bc_kn.fit(train_X, train_y)
prediction = bc_kn.predict(test_X)
print('The accuracy for bagged KNN is:', accuracy_score(prediction, test_y))
```

11. **BaggingClassifier (DecisionTreeClassifier)**
    - Before score : 0.878
    - Best score : 0.929

```
bc_dt = BaggingClassifier(base_estimator=DecisionTreeClassifier(), random_state=0, n_estimators=100)
bc_dt.fit(train_X, train_y)
prediction = bc_dt.predict(test_X)
print('The accuracy for bagged Decision Tree is:', accuracy_score(prediction, test_y))
```

12. **AdaBoostClassifier**
    - Before score : 0.837
    - Best score : 0.920

```
from sklearn.ensemble import AdaBoostClassifier
abc = AdaBoostClassifier(n_estimators=200,random_state=0,learning_rate=0.1)
abc.fit(train_X, train_y)
prediction = abc.predict(test_X)
print('The accuracy for AdaBoost is:', accuracy_score(prediction, test_y))
```

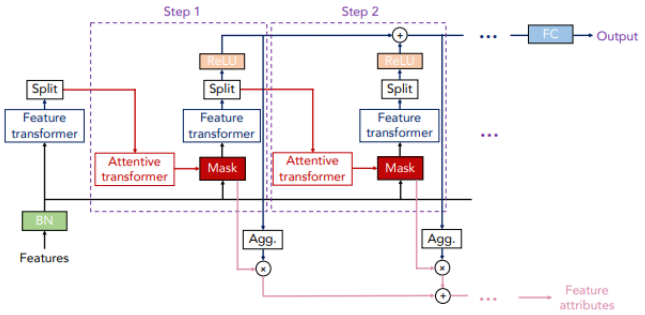
3. 자기지도 학습 모델

Prepation

- Feature Selection
- Data Scaling, Data Normalization
- Latent Representation
- Cross Validation
- Hyper-Parameters Tuning

TabNet

- 전처리 과정을 거치지 않은 raw data로도 end-to-end 학습이 가능합니다.
- Sequential attention 과정에서는 각 스텝마다 중요한 feature를 선별하여 모델의 해석과 성능을 향상시킬 수 있습니다.
- 다양한 도메인의 데이터에서 다른 테이블 학습 모델과 비교했을 때, 분류 및 회귀 문제에서 우수한 성능을 보입니다. 또한, masking된 feature를 예측하는 TabNet decoder를 활용한 자기지도 학습을 통해 우수한 성능을 나타내었습니다



(a) TabNet architecture

- Result
  - ROC AUC 값이 0.955로 가장 좋은 성능을 보였습니다.

```
preds = np.array(TabNetClassifier.predict_proba(test_X))
valid_auc = roc_auc_score(y_score=preds[:,1], y_true=test_y)
print(valid_auc)
```

4. 준지도 학습 모델

Prepation

- Feature Selection
- Data Scaling, Data Normalization
- Latent Representation
- Cross Validation
- Hyper-Parameters Tuning

Label Propagation

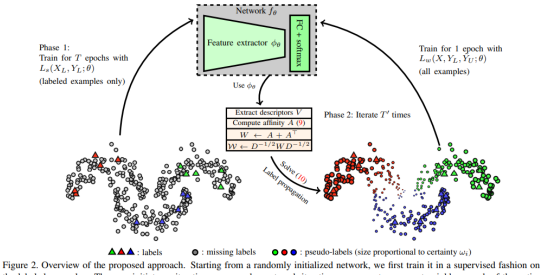


Figure 2. Overview of the proposed approach. Starting from a randomly initialized network, we first train it in a supervised fashion on the labeled examples. Then we initiate an iterative process where at each iteration we compute a nearest neighbor graph of the entire training set in the feature space of the current network, we propagate labels by transductive learning, and then we train the network on the entire training set, with true labels or pseudo-labels on the labeled or unlabeled examples respectively. The pseudo-labels are weighted per example and per class according to prediction certainty and inverse class population, respectively.

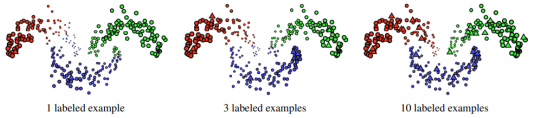
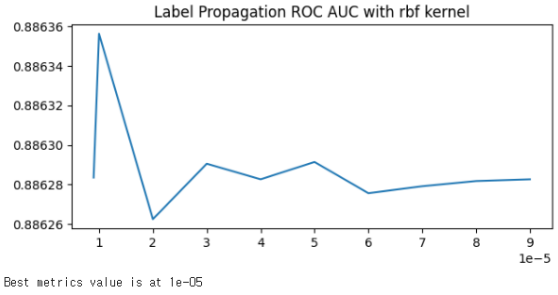


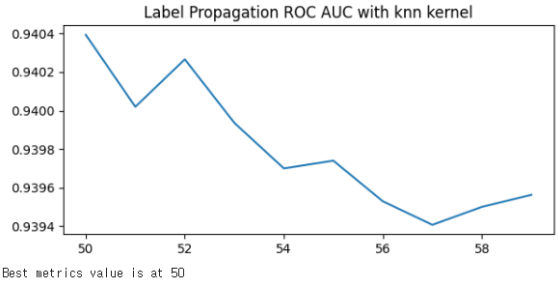
Figure 3. Toy example with 300 examples demonstrating label propagation for different number of labeled examples. Triangle markers correspond the labeled examples and circles to the unlabeled ones which are finally pseudo-labeled by label propagation. The class is color-coded and the size of the circles corresponds to weight ω_i. The true labels are the same as the example of Figure 1 (top).

- Label Propagation(레이블 전파)은 주어진 데이터에서 일부 샘플에 대한 레이블을 알고 있는 경우, 그 레이블 정보를 가지지 않은 다른 샘플에 전파하여 레이블을 예측하는 방법입니다. Label Propagation은 레이블이 있는 데이터에서 시작하여 레이블이 없는 데이터의 예측을 개선하는 데 사용될 수 있습니다.

1. **Label Propagation RBF**



2. **Label Propagation KNN**



3. **Result**
   - Label Propagation의 경우 regularization 성질을 가진 loss function을 사용하여 보다 generalization적인 성능을 나타냈습니다 (noise-robust).
   - 가장 높은 ROC AUC 값은 Label Propagation KNN이며, 0.940으로 나타났습니다.

Algorithm	ROC AUC
Label Propagation RBF	0.886356
Label Propagation KNN	**0.940393**
Label Spreading RBF	0.886356
Label Spreading KNN	0.939455

AutoEncoder

- Autoencoder를 활용하여 레이블이 없는 데이터를 학습하고, 예측하였습니다.

```
# Using AutoEncoder
test_ae = pd.read_csv('/content/gdrive/My Drive/data/data.csv')
test_ae = test_ae[test_ae['isAbuser'] == -1]

test_rep_x = hidden_representation.predict(test_x_scale)
test_ae['isAbuser'] = [int(x) for x in clf.predict(test_rep_x)]
test_ae.head()
```

- **Result (Unlabeld Data)**

newID	isAbuser
0	0
0	0
0	0
0	0
0	0

적용 및 결과

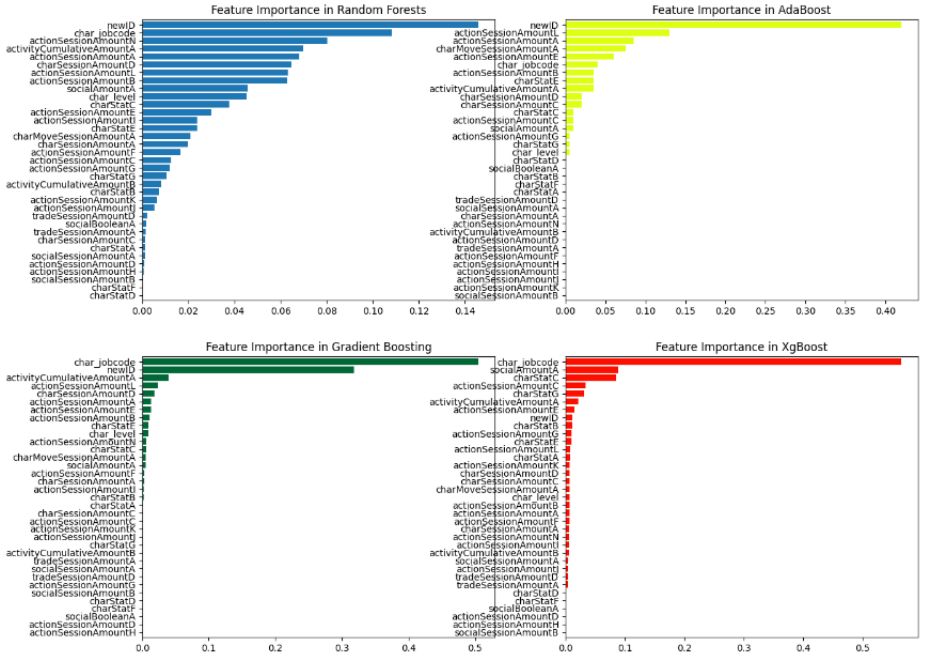
성능 비교

- 지도 모델 중에서는 LGBMC가 Accuracy 0.926, Precision 0.918, AUC 0.856으로 좋은 성과를 보였습니다.
  - SVM은 Recall 0.998에서 우수한 성능을 나타냈습니다.
- 자기지도 모델 중에서는 Tabnet이 ROC AUC 0.955로 우수한 성능을 나타냈습니다.
- 준지도 모델 중에서는 Label Propagation KNN이 ROC AUC 0.940으로 뛰어난 성능을 보였습니다.

	Accuracy	Precision	Recall	AUC
LR	0.909660	0.904785	0.984552	0.829310
DT	0.889652	0.877160	0.993741	0.777976
KNN	0.922292	0.909876	0.996271	0.842920
RF	0.920271	0.907569	0.996404	0.838588
LGBMC	0.926031	0.918488	0.990412	0.856957
SVC	0.891168	0.875438	0.998668	0.775832

특징의 중요도

- 학습된 모델은 feature importance를 가지며, 모델이 어떤 feature에 영향을 많이 받았는지 확인할 수 있습니다.
- **Random Forests**
  - newID (0.145645), char_jobcode (0.108352), actionSessionAmountN (0.080324), activityCumulativeAmountA (0.069946), actionSessionAmountA (0.068001)
- **AdaBoost**
  - newID (0.420), actionSessionAmountL (0.130), actionSessionAmountA (0.085), charMoveSessionAmountA (0.075), actionSessionAmountE (0.060), char_jobcode (0.040)
- **Gradient Boosting**
  - char_jobcode (0.504425), newID (0.318069), activityCumulativeAmountA (0.039651), actionSessionAmountL (0.023228), charSessionAmountD (0.018300)
- **XgBoost**
  - char_jobcode (0.565655), socialAmountA (0.088741), charStatC (0.085682), actionSessionAmountC (0.033504), charStatG (0.031296)



어뷰저 검출 기준

- 앞선 분석을 통해 newID, char_jobcode, actionSessionAmount, activityCumulativeAmount, charMoveSessionAmount, socialAmount 등을 고려하여 어뷰저를 검출합니다.
- 이러한 결과는 캐릭터 직업, 캐릭터가 세션 중 발생시킨 액션의 양(예: 사냥 횟수), 액션 정보로 발생 시점부터 누적되는 값, 캐릭터가 세션 중 이동한 양, 타 캐릭터와의 인터렉션 정보를 의미합니다.
- 결과적으로, 특정 맵이나 이벤트를 반복 플레이하거나 컨텐츠에 플레이가 지나치게 많거나 적고, 소셜 활동이 지나치게 많거나 적은 행동을 하는 유저에 대해 집중적으로 살펴보면 좋을 것 같습니다.

개선 방향

- 기존 모델의 속도와 성능, 확장성 개선 방안 모색
  - 현재 모델의 속도와 성능을 향상시키기 위해 최적화된 알고리즘 및 병렬처리 기술 도입
  - 모델 파라미터 튜닝 및 경량화를 통한 성능 향상 방안 고려
- 유저의 플레이 시간 정보를 활용하는 방안
  - 플레이 시간 정보를 통해 유저의 행동 변화를 감지하여 어뷰징 가능성을 추정
- Scoring 모형을 통해 개별 유저의 어뷰징 위험도를 사전에 미리 추정하는 방안
  - 유저의 행동 패턴, 이력, 플레이 스타일 등을 종합적으로 평가하여 어뷰징 위험도 점수 부여
  - 높은 위험도를 가진 유저에 대한 추가 검증 또는 강화된 모니터링 시스템 도입
- 플레이 시간, 아이템 획득, 로그인 수, 전투 수 등 변수 간의 복잡한 관계를 학습하여 게임의 전반적인 위험 노출 수준을 관리 및 탐지
  - 다변량 시계열 모델 또는 심층 학습 알고리즘을 사용하여 다양한 변수 간의 복잡한 상호작용 학습

마치며

EDA를 통해 게임 재화 어뷰징에 영향을 미치는 특징을 확인했으며, 특징 중요도 분석을 통해 어뷰징 가능성을 예측하는 데 각 특징의 중요성을 정량화할 수 있었습니다. 결과에 따르면, 가장 중요한 특징은 캐릭터 직업, 캐릭터가 세션 중 발생시킨 액션의 양(예: 사냥 횟수), 액션 정보로 발생 시점부터 누적되는 값, 캐릭터의 세션 중 이동 거리, 그리고 타 캐릭터와의 인터랙션 정보입니다. 이러한 결과를 기반으로 게임 회사에서는 어뷰징 가능성이 높은 고객에게 경고나 알림을 주는 방식으로 서비스를 조정하고 개선할 수 있습니다.

게임 재화 어뷰징 검출을 위해 지도, 자기지도, 준지도 모델 등 총 15개의 모델을 사용했습니다. AUC 기준으로, 지도 모델은 평균 81%, 자기지도 모델은 95%, 준지도 모델은 94%의 정확도를 보였으며, 특히 자기지도와 준지도 모델에서 높은 성과를 나타냈습니다. 더 많은 기능과 데이터를 추가하면 예측 성능을 향상시킬 수 있을 것으로 기대됩니다.

그러나 실시간으로 적용되는 서비스 특성상 예측 성능 뿐만 아니라 예측 시간과 확장성 등을 고려하여 분석 및 로직을 구현해야 할 것으로 생각됩니다.