(민희수) 불균형 데이터와 결측치를 고려한 이진 분류 예측: 지도 학습 모델

Update date	2023/11/30
Tag	**#BinaryClassification #ImblancedData #MissingValue #SupervisedLearning**
Editor	민희수

▼ Table of Contents

들어가며

train 및 test 데이터에는 'target' 변수의 불균형과 결측치 등 몇 가지 문제가 있습니다. 이러한 데이터 이슈를 효과적으로 해결하고 모델링을 수행하기 위한 로직을 개발하고자 합니다.

목표

목표는 데이터의 특성을 고려하여 이진 분류를 효과적으로 예측하기 위한 방법 개발하는 것입니다.

- **예측 정확도 향상:**
  EDA를 통해 Feature Selection, Data Normalization, Hyper-Parameters Tuning 등을 활용하여 효과적이고 정확한 모델링을 구현합니다.
- **데이터 불균형 다루기:**
  Train 데이터셋의 'target' 변수는 불균형한 분포를 가집니다. 이를 해결하기 위해 over-sampling, class weights, label propagation을 활용하여 모델을 훈련하고 검증합니다.
- **결측치 처리:**
  일부 features는 최대 1/2배의 결측값이 존재합니다. 이를 해결하기 위해 Listwise Deletion, Mean/Median Imputation, Model-based Imputation을 활용하여 데이터셋을 처리합니다.
- **지도 학습 사용:**
  복잡한 고급 알고리즘 대신 결과를 이해하기 쉬운 지도 학습을 채택합니다. 데이터셋의 특성, 크기, 복잡성, 계산 리소스를 고려하여 11개의 모델을 선택하였습니다.

통계 및 탐색적 데이터 분석

데이터 확인

```
[Feature]
- Unnamed: 0 : int64, 연속형
- KEY_ID : int64, 연속형
- YEAR : int64, 범주형
- MONTH : object, 범주형
- DAY : int64, 범주형
- ID_1 : int64, 범주형
- ID_2 : object, 범주형
- ID_3 : object, 범주형
- VAR_7~11 : int64, 범주형
- VAR_12~22 : object, 범주형
- VAR_23 : int64, 범주형
- VAR_24 : object, 범주형
- VAR_25 : float64, 연속형
- VAR_26 : int64, 범주형
- VAR_27~32 : float64, 연속형
- VAR_33 : int64, 범주형
- VAR_34~53 : float64, 연속형
- VAR_54 : int64, 범주형

[Target label]
- target : float64, 범주형 (0.0 또는 1.0)
```

EDA

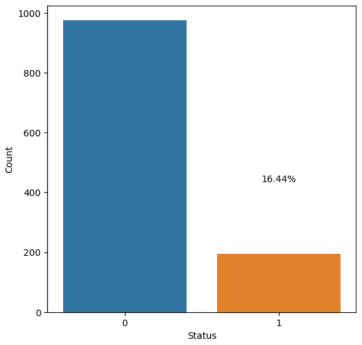
1. **Describe을 통한 다양한 통계량 요약**

count	unique	top	freq	mean	std	min	25%
Unnamed: 0	1180	NaN	NaN	NaN	589.5	340.780966	0
KEY_ID	1180	NaN	NaN	NaN	5.33839E+11	1112609491	5.31
YEAR	1180	NaN	NaN	NaN	2021	0	2021
MONTH	1180	4	12	1110	NaN	NaN	NaN
DAY	1180	NaN	NaN	NaN	12.191525	5.714193	1
…							
VAR_52	1180	NaN	NaN	NaN	273.842797	2.31659	260
VAR_53	1180	NaN	NaN	NaN	1095.370508	9.264117	1039
VAR_54	1180	NaN	NaN	NaN	70.059322	0.768246	70
target	1170	NaN	NaN	NaN	0.165812	0.372071	0

2. **결측치 확인**
   - Train과 Test 데이터셋의 결측치를 확인하였고, 이를 보간하는 전처리 과정을 추가하였습니다.
   - Train 데이터셋 : VAR_12, VAR_13, VAR_16, VAR_35, VAR_38, VAR_42, VAR_49
   - Test 데이터셋 : VAR_35, VAR_38, VAR_42, VAR_49

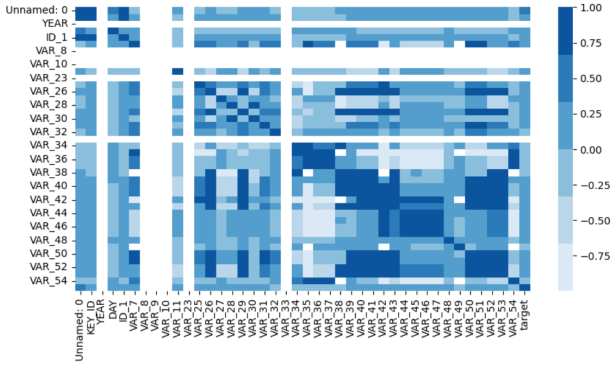
3. **Target label 확인**

- 0는 976개 (83.56%), 1는 194개 (16.44%) 이므로 데이터 편향/불균형이 존재하여 이를 해결하기 위해 **over-sampling, class weights**과 **준지도 학습 모델(**label propagation)**을 추가**하였습니다.



4. **Featurer간 상관관계**

- 강한 상관관계를 가진 변수는 Unnamed: 0 (0 0.350), KEY_ID (0.267), ID_1 (0.267) 등이 있습니다.
- 약한 상관관계를 가진 변수는 YEAR, VAR_8, VAR_9, VAR_10, VAR_23, VAR_33등이 있습니다.



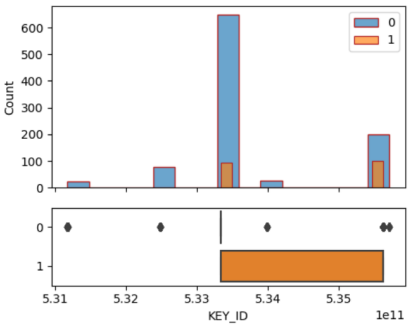
5. **다중공선성**

- 다중공선성을 통해 독립 변수 간의 상관관계를 확인할 수 있습니다.
- Condition number가 8.85e+14이며 다중공선성의 문제가 존재하므로, 이를 해결하기 위해 **데이터 정규화 과정을 추가**하였습니다.

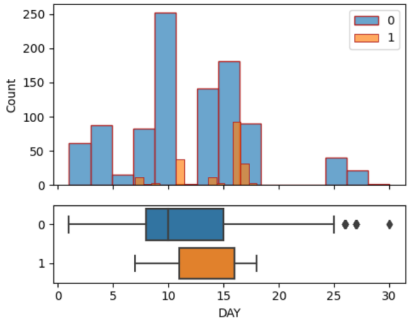
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 target   R-squared:                       0.087
Model:                            OLS   Adj. R-squared:                  0.078
Method:                 Least Squares   F-statistic:                     10.01
Date:                Wed, 29 Nov 2023   Prob (F-statistic):           9.57e-08
Time:                        06:38:53   Log-Likelihood:                -109.54
No. Observations:                 425   AIC:                             229.1
Df Residuals:                     420   BIC:                             249.3
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     11.2864     25.356      0.445      0.656     -38.554      61.127
KEY_ID         0.0001   2.62e-05      5.251      0.000    8.62e-05       0.000
ID_1         -13.7773      2.623     -5.251      0.000     -18.934      -8.620
DAY           -0.0005      0.003     -0.159      0.874      -0.007       0.006
VAR_38        -0.0431      0.059     -0.733      0.464      -0.159       0.072
VAR_34         0.0168      0.029      0.572      0.568      -0.041       0.074
==============================================================================
Omnibus:                      159.984   Durbin-Watson:                   0.301
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              387.883
Skew:                           1.976   Prob(JB):                     5.92e-85
Kurtosis:                       5.507   Cond. No.                     8.85e+14
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 8.85e+14. This might indicate that there are
strong multicollinearity or other numerical problems.
```

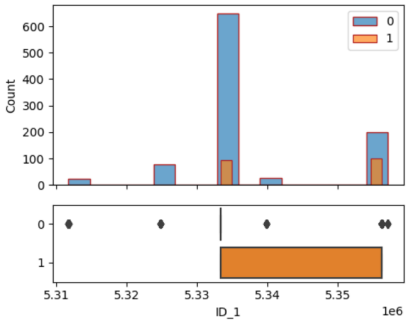
6. **Target 0 과 1 의 분포**
   - KEY_ID
     - 1의 KEY_ID는 균일하지 않고, 5.33 위로 밀집되어 있으며 분명한 차이를 보입니다.



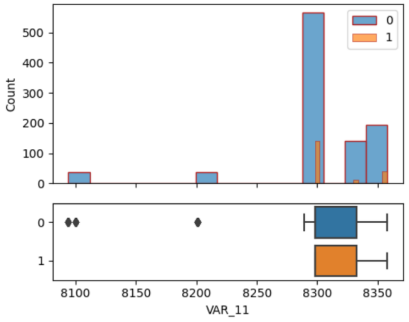
   - DAY
     - 0의 꼬리가 약간 더 앞서 있으나, 전체적으로 두 분포는 비슷합니다.



- ID_2
  - 0의 ID_2는 균일하게 분포되어 있으나 1은 5.33 이후로 분포되어 있으며 분명한 차이를 보입니다.



- VAR_11
  - 1의 VAR_11는 8300 이후로 밀집되어 분포되어 있습니다.



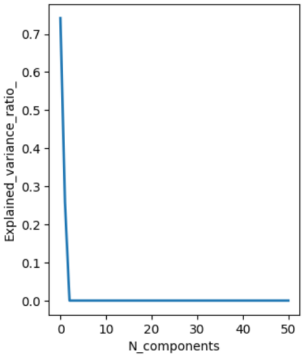
Feature Engineering

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 **DAY 포함 12개 특성 항목을 선택**하였습니다. 해당 내용은 아래와 같습니다.

Variable	Chi-square	p-value
DAY	517.658289	1.10E-97
ID_1	243.310328	3.36E-36
ID_2	64.808614	1.57E-10
VAR_11	34.514421	1.38E-05
VAR_34	32.837965	1.80E-03
VAR_50	21.3074	1.14E-02
VAR_41	17.253207	4.49E-02
MONTH	13.460777	3.74E-03
VAR_38	12.726714	2.61E-02
VAR_13	11.405742	3.34E-03
VAR_7	10.969801	4.15E-03
ID_3	8.627026	3.31E-03

2. **PCA (Principal Component Analysis, 주성분 분석)**
    - PCA는 데이터의 차원을 축소하거나 주요한 정보를 유지하고 불필요한 정보를 제거하는 기법으로, 상관계수가 높은 특성으로 PCA 변환을 진행하였으나 설명 가능성과 상관도가 낮아서 사용하지 않았습니다.

```
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(X_train)
```



1. 어떤 부분을 고려하여 어떻게 처리하시겠습니까?

데이터 축소

- 불필요한 column은 제거하였습니다.
    - Unnamed: 0 는 분류에서 사용되지 않는 feature
    - VAR_14 등 하나의 값만 포함하고 있는 feature는 분석 단계에서 필요 없을 것이라 판단
    - VAR_55는 test 데이터셋에만 포함되어 있음

```
# 불필요한 column 제거
train = train.drop(['Unnamed: 0','VAR_14','VAR_15','VAR_17','VAR_18','VAR_19','VAR_20','VAR_21','VAR_22'], axis=1)
test = test.drop(['Unnamed: 0','VAR_14','VAR_15','VAR_17','VAR_18','VAR_19','VAR_20','VAR_21','VAR_22','VAR_55'], axis=1) # 'VAR_55'는
```

빈 값이 포함된 변수의 경우

- 열의 값은 숫자로, 빈 값은 NaN 값으로 변환하였습니다.

```
features = ['VAR_12','VAR_13','VAR_16']

for feature in features:
  train[feature] = pd.to_numeric(train[feature], errors='coerce')
  test[feature] = pd.to_numeric(test[feature], errors='coerce')
```

숫자형과 문자형 변수가 섞여있는 경우

- 숫자형과 문자형 변수가 섞여있는 경우, 숫자형(정수)으로 변환하였습니다.

```
# MONTH
train.loc[train['MONTH'] == 'December', 'MONTH'] = 12
train.loc[train['MONTH'] == 'November', 'MONTH'] = 11
```

```
train['MONTH'] = train['MONTH'].astype(int)

# ID_3,VAR_24'
from sklearn.preprocessing import LabelEncoder

features = ['ID_3','VAR_24']

for feature in features:
  train[feature] = LabelEncoder().fit_transform(train[feature])
```

하나의 값에 숫자와 문자가 섞여있는 경우

- 숫자와 문자가 혼합된 변수의 경우, 숫자와 문자를 따로 추출하고 문자열 부분의 경우 원-핫 인코딩을 수행하였습니다.

```
def split_string(data, col):
  df_ID = pd.DataFrame(data[col], columns=[col])

  df_ID[col+'_n1'] = df_ID[col].str.extract('(\d+)(?=[A-Za-z])', expand=False)
  df_ID[col+'_n1'] = pd.to_numeric(df_ID[col+'_n1'])

  df_ID[col+'_n2'] = df_ID[col].str.extract('(?<=[A-Za-z])(\d+)', expand=False)
  df_ID[col+'_n2'] = pd.to_numeric(df_ID[col+'_n2'])

  df_ID[col+'_alph'] = df_ID[col].str.extract('([A-Za-z]+)', expand=False)

  # 원-핫 인코딩
  df_ID = pd.get_dummies(df_ID, columns=[col+'_alph'], prefix=col+'_alph')

  data = data.drop([col],axis=1)
  df_ID = df_ID.drop([col],axis=1)
  data = pd.concat([data, df_ID], axis = 1)

  return data

  data.head()

train = split_string(train, 'ID_2')
```

결측치 비율이 높은 변수의 경우

- Listwise Deletion와 Mean / Median imputation, Model-based imputation을 진행한 결과, Mode와 Model-based 방법의 baseline 성능이 동일했습니다. 따라서, **Model-based imputation** 방법을 선택하였습니다.

1. **Listwise Deletion**
   - NaN이 포함된 행을 삭제하는 방법

   ```
   train_imp1 = train_imp1.dropna(axis=1)
   ```

2. **Mode / Mean / Median imputation**
   - 행의 최빈값, 평균값 또는 중간값으로 대체하는 방법

   ```
   features = train_imp2.select_dtypes(include=np.number).columns.tolist()

   for feature in features:
     mode = train_imp2[feature].mode()[0]

     null_idx = train_imp2[feature].isnull()
     train_imp2.loc[null_idx, feature] = mode
   ```

3. **Model-based imputation**
   - 모델 기반 대체하는 방법 (KNN, EM, ML, 회귀, randomForest 등)

   ```
   import miceforest as mf

   features = train_imp3.select_dtypes(exclude='number').columns
   ```

```
for feature in features:
  train_imp3[feature] = train_imp3[feature].astype('category')
  test_imp3[feature] = test_imp3[feature].astype('category')

kds = mf.ImputationKernel(
    train_imp3,
    random_state=random_state
)
kds.mice(5)
train_imp3 = kds.complete_data()
```

Target의 데이터 불균형

- Over-sampling의 Random과 SMOTE, Configure class weight를 진행한 결과, Configure class weight 방법의 baseline 성능이 높았습니다. 따라서, **Configure class weight** 방법을 선택하였습니다.

1. **Over-sampling (Random)**

```
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(random_state=random_state)
X_train_ros, y_train_ros = ros.fit_resample(X_train, y_train)
```

2. **Over-sampling (SMOTE)**

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=random_state)
X_train_sm, y_train_sm = smote.fit_resample(X_train, y_train)
```

3. **Configure class weight**

```
from sklearn.model_selection import GridSearchCV

n_samples = train.shape[0]
n_features = train.shape[1]

param_grid = {'base_estimator': [None, LogisticRegression(), KNeighborsClassifier()],
          'n_estimators': [20,50,100,700],
          'max_samples': [0.5, 1.0, n_samples//2, ],
          'max_features': [0.5, 1.0, n_features//2, ],
          'bootstrap': [True, False],
          'bootstrap_features': [True, False]}

grid = GridSearchCV(BaggingClassifier(random_state=random_state, n_jobs=-1), param_grid)

grid_result = grid.fit(X_train, y_train)
```

데이터 표준화

- 연속형 변수에 대해 데이터 표준화(standardization)를 진행하였습니다. (평균: 0, 표준편차: 1)

```
from sklearn.preprocessing import StandardScaler

sca_columns = ['VAR_25','VAR_31','VAR_32','VAR_27','VAR_28','VAR_29','VAR_30','VAR_31','VAR_32']
scaler = StandardScaler()
X_train[sca_columns] = scaler.fit_transform(X_train[sca_columns])
X_test[sca_columns] = scaler.fit_transform(X_test[sca_columns])
```

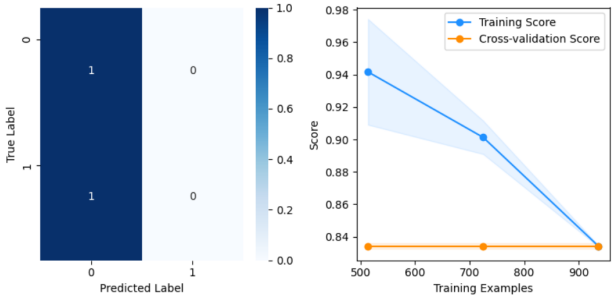
2. 모델링

Prepation

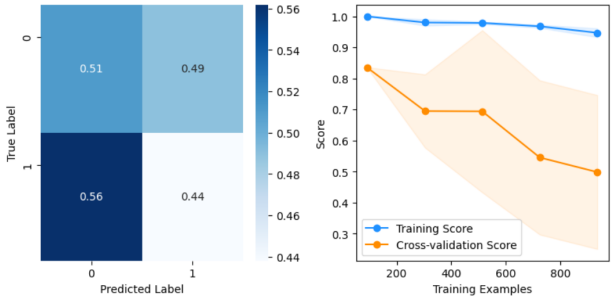
- Feature Selection
- Data Scaling, Data Normalization
- Cross Validation
- Hyper-Parameters Tuning

지도 학습 모델

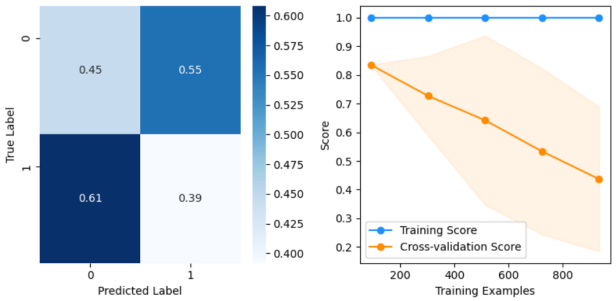
1. **Logistic Regression**
   - Best score : 0.968



2. **Decision Tree**
   - Best score : 0.668

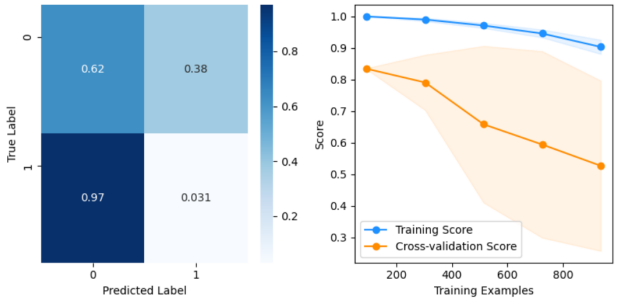


3. **K-Nearest Neighbors(KNN)**
   - Best score : 0.968



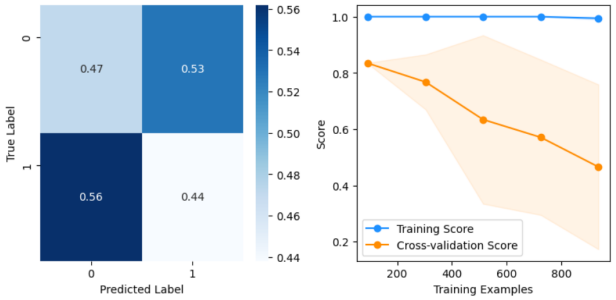
4. **Random Forest Classifier**

- Best score : 0.968



5. **Light Gradient Boosting Machine**

- Best score : 0.922



6. **Support Vector Classifier (Linear)**

- Best score : 0.915

```
svm = svm.SVC(C=1, gamma=0.1, probability=True, random_state=random_state)
svm.fit(X_train, y_train)
prediction = svm.predict(X_test)
print('Accuracy for SVM is ', accuracy_score(svc_prediction, y_test))
```

7. **Gradient Boosting Classifier**

- Best score : 0.861

```
gb = GradientBoostingClassifier(n_estimators=500,random_state=0,learning_rate=0.1)
gb.fit(X_train, y_train)
prediction=gb.predict(X_test)
print('The accuracy for Gradient Boostin is:', accuracy_score(prediction, y_test))
```

8. **Xtreme Gradient Boosting Classifier**
   - Best score : 0.831

```
from xgboost import XGBClassifier
xgb = XGBClassifier(n_estimators=900,learning_rate=0.1)
xgb.fit(X_train, y_train)
prediction = xgb.predict(X_test)
print('The accuracy for XGBoost is:', accuracy_score(prediction, y_test))
```

9. **Bagging Classifier (KNeighborsClassifier)**
   - Best score : 0.930

```
bc_kn = BaggingClassifier(base_estimator=KNeighborsClassifier(n_neighbors=3), random_state=0, n_estimators=700)
bc_kn.fit(X_train, y_train)
prediction = bc_kn.predict(X_test)
print('The accuracy for bagged KNN is:', accuracy_score(prediction, y_test))
```

10. **BaggingClassifier (DecisionTreeClassifier)**
    - Best score : 0.894

```
bc_dt = BaggingClassifier(base_estimator=DecisionTreeClassifier(), random_state=0, n_estimators=100)
bc_dt.fit(X_train, y_train)
prediction = bc_dt.predict(X_test)
```

11. **AdaBoostClassifier**
    - Best score : 0.855

```
from sklearn.ensemble import AdaBoostClassifier
abc = AdaBoostClassifier(n_estimators=200,random_state=0,learning_rate=0.1)
abc.fit(X_train, y_train)
prediction = abc.predict(X_test)
print('The accuracy for AdaBoost is:', accuracy_score(prediction, y_test))
```

예측 및 평가 방법

평가 방법

1. **Confusion Matrix 오차행렬**
   - 모델이 어떻게 예측 했는지를 시각적으로 확인하고, 실제 target과 모델의 예측 target에 대한 정보를 통해 모델의 성능을 전반적으로 이해

		Predicted		
		Positive	Negative	
Observed	Positive	TP	FN	P
	Negative	FP	TN	N

2. **Accuracy 정확도**
   - 모델이 전체 샘플 중에서 얼마나 정확히 예측했는지를 측정하고, 모델의 전반적인 성능을 나타내는 지표로 사용

$$\text{Accuracy} = \frac{correct\ predictions}{all\ predictions} = \frac{TP + TN}{TP + TN + FP + FN}$$

3. **Precision 정밀도**
   - 모델이 양성이라고 예측한 경우 중에서 실제로 양성인 비율을 측정하고, FP를 줄이는 데 사용

$$Precision = \frac{TP}{TP + FP} = \frac{\text{positive predicted correctly}}{\text{all positive predictions}}$$

4. **Recall 재현율**
   - 실제 양성 중에서 모델이 얼마나 많은 것을 찾아냈는지를 측정하고, FN를 줄이는 데 사용

$$Recall = TPR = \frac{TP}{TP + FN} = \frac{TP}{P} = \frac{\text{predicted to be positive}}{\text{all positive observations}}$$

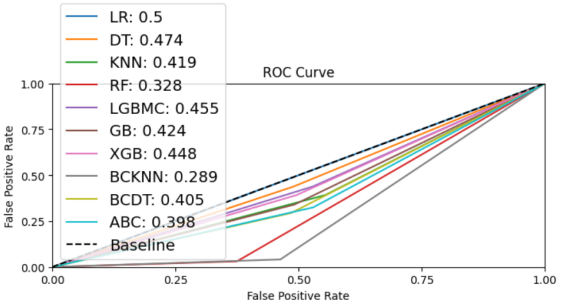
5. **ROC AUC**
   - 모델의 민감성과 특이성 간의 균형을 측정하며, 분류 모델의 성능을 종합적으로 이해

성능 비교

- 지도 모델 중에서는 Linear Regression이 **Accuracy 0.834**으로 좋은 성과를 보였습니다.

	Accuracy	Precision	Recall	AUC
LR	0.834188	0.000000	0.000000	0.500000
DT	0.498291	0.150977	0.438144	0.474195
KNN	0.436752	0.123177	0.391753	0.418725
RF	0.526496	0.016129	0.030928	0.327964
LGBMC	0.465812	0.141431	0.438144	0.454728
GB	0.480342	0.120879	0.340206	0.424201
XGB	0.485470	0.135714	0.391753	0.447925
BCKNN	0.454701	0.017391	0.041237	0.289061
BCDT	0.478632	0.107547	0.293814	0.404592
ABC	0.446154	0.108621	0.324742	0.397515

- **ROC Curve**



마치며

탐색적 데이터 분석 (EDA)를 통해 데이터셋의 특성을 확인하고, 숫자형과 문자형 데이터 혼합, 결측치, 그리고 데이터 불균형과 같은 문제들을 파악했습니다. 이들 문제를 해결하여 예측 결과의 정확도를 향상 시켰습니다.

뿐만 아니라, 예측하고자 하는 target에 미치는 영향을 확인하고, P-value 및 PCA를 사용하여 특징 중요도를 분석하여 각 특징의 중요성을 정량화할 수 있었습니다. 분석 결과, DAY, ID_1, ID_2 등이 가장 중요한 특징으로 나타났습니다.

마지막으로 11개의 지도 모델을 사용하여 이진 분류 예측을 수행했고, accuracy 기준으로 최대 83%의 정확도를 달성했습니다. 특히 Linear Regression 모델에서 뛰어난 성과를 얻었습니다. 더 많은 기능과 데이터를 추가하면 모델의 예측 성능을 더 향상시킬 수 있을 것으로 기대됩니다.