

MBA Admission Predictor

공공데이터 (정형데이터) 활용 지도학습 모델 구현

최대웅

2024.10.14

목차

1. 프로젝트 개요
2. 데이터 설명
3. 데이터 전처리
4. 데이터 불균형
5. CTGAN을 활용한 데이터 불균형 해결
6. 특성 공학
7. 모델 개발 및 평가
8. 결과
9. 결론

1. 프로젝트 개요

프로젝트 목적과 배경

목적

지원자의 데이터를 기반으로 MBA 입학 가능성을 예측, 분류하는 모델을 개발하고, 이를 통해 입학 과정의 투명성을 높이는 것.

배경

기존의 입학 평가 과정은 시간이 많이 소요되고 평가 기준이 일관되지 않으며, 사회적 배경에 따른 편견이 개입될 가능성도 존재. 이러한 문제를 극복하기 위해, 다양한 지원자들의 데이터를 바탕으로 객관적인 평가 기준을 설정하고, 지원자의 MBA 입학 가능성을 예측하는 동시에 각 요소들이 결과에 미치는 영향을 분석하여, 투명하고 공정한 입학 평가 과정을 만들기 위해 시작.

연구 과제

1. 다양한 지원자들의 데이터를 수집하고 가공하여 입학 평가 과정에 필요한 데이터를 구축.
2. 입학 가능성에 영향을 미치는 주요 특성을 선택, 사회적 배경에 따른 요소와의 관계 분석.
3. 최적의 분류 모델을 선택하기 위해 알고리즘 실험.
4. 모델 성능 평가 및 사회적 배경에 따른 편향성 여부 평가.

2. 데이터 설명

데이터 출처 및 구조

데이터 출처

<https://www.kaggle.com/datasets/taweilo/mba-admission-dataset>

데이터 구조

- 데이터 형태: CSV 파일
- 컬럼 수 및 전체 데이터 수: 10*6195

주요 변수 - 명목형 변수

값들이 서로 순서가 없는 카테고리나 그룹으로 나누어지는 변수.

application_id	gender	international	gpa	major	race	gmat	work_exp	work_industry	admission
1	Female	FALSE	3.3	Business	Asian	620	3	Financial Services	Admit

- application_id : 지원자를 구분하기 위해 부여된 고유 번호
- gender : 성별
- international : 국제 학생 여부
- major : 대학 전공
- race : 인종
- work_industry : 근무했던 산업 분야
- admission : 입학 여부

주요 변수 - 연속형 변수

값이 실수로 표현되며, 측정이나 계산이 가능한 변수.

application_id	gender	international	gpa	major	race	gmat	work_exp	work_industry	admission
1	Female	FALSE	3.3	Business	Asian	620	3	Financial Services	Admit

- gpa : 평균 학점
- gmat : GMAT 시험 점수
- work_exp : 업무 경력 기간

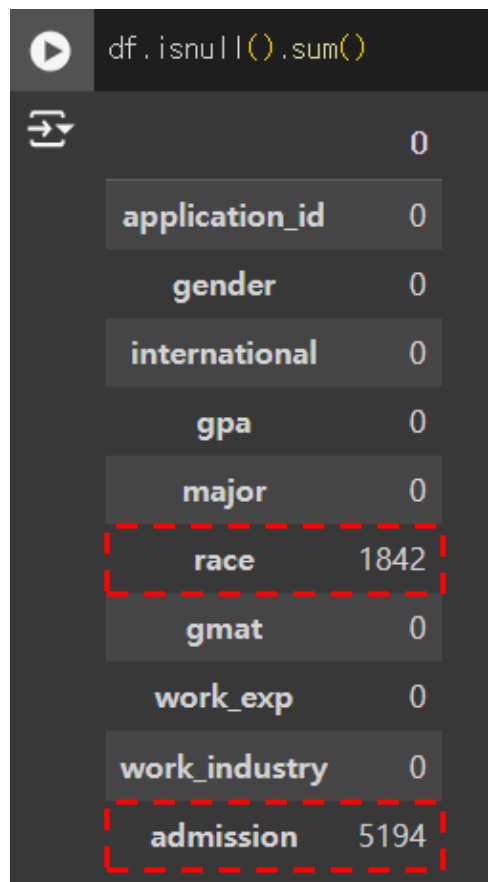
목표 변수

admission : MBA 입학 여부. 이 변수를 목표 변수로 사용하며, 분류 모델을 통해 입학 가능성을 분류하는데 사용.

3. 데이터 전처리

결측치 처리

데이터에 빠진 값이 있는 경우, 해당 값을 대체하거나 삭제.



df.isnull().sum()	
	0
application_id	0
gender	0
international	0
gpa	0
major	0
race	1842
gmat	0
work_exp	0
work_industry	0
admission	5194

race	Admit
Asian	
Black	Admit
Black	

```
df['race'] = df['race'].fillna('unknown')  
df['admission'] = df['admission'].fillna('denied')
```

fillna() 메서드를 사용해서 각 컬럼의 결측값을 채움.

- 'race' 컬럼의 결측값은 'unknown'으로 채움.
- 'admission' 컬럼의 결측값은 'denied'로 채움.

이상치 처리

데이터에 이상 값이 있을 경우, 해당 값을 대체하거나 삭제.

```
df['admission'] = df['admission'].apply(lambda x : 'denied' if x == 'Waitlist' else x)
```

- 합격 여부를 판단하기 위해 apply()를 사용해서 'Waitlist' 값을 'denied'로 변환.

```
df = df.drop('application_id', axis = 1)
```

- 'application_id'는 지원자를 구분하기 위한 번호로 합격 여부 판단에 영향을 주지 않으므로 삭제.

데이터 인코딩

범주형 변수를 Label Encoding을 사용하여 수치형 데이터로 변환.

```
le = LabelEncoder()

categorical_cols = df_augmented.select_dtypes(include=['object']).columns

for col in categorical_cols:
    df_augmented[col] = le.fit_transform(df_augmented[col])
    print(f"Classes (범주) for {col}: {le.classes_}")
    encoded_data = df_augmented[col]
    mapping = dict(zip(le.classes_, range(len(le.classes_))))
    print(f"Mapping (범주 -> 인코딩 값) for {col}: {mapping}\n")
```

- LabelEncoder 객체를 생성한 후, select_dtypes(include=['object'])를 통해 범주형 데이터만 찾아 인코딩.
- fit_transform() 메서드를 사용해 현재 컬럼의 값을 각각의 정수로 변환.
- dict(zip(le.classes_, range(len(le.classes_))))를 사용하여 원래 범주와 인코딩된 값을 매핑한 딕셔너리 생성.

데이터 인코딩

출력 결과

```
Classes (범주) for gender: ['Female' 'Male']  
Mapping (범주 -> 인코딩 값) for gender: {'Female': 0, 'Male': 1}  
  
Classes (범주) for major: ['Business' 'Humanities' 'STEM']  
Mapping (범주 -> 인코딩 값) for major: {'Business': 0, 'Humanities': 1, 'STEM': 2}  
  
Classes (범주) for race: ['Asian' 'Black' 'Hispanic' 'Other' 'White' 'unknown']  
Mapping (범주 -> 인코딩 값) for race: {'Asian': 0, 'Black': 1, 'Hispanic': 2, 'Other': 3, 'White': 4, 'unknown': 5}  
  
Classes (범주) for work_industry: ['CPG' 'Consulting' 'Energy' 'Financial Services' 'Health Care'  
    'Investment Banking' 'Investment Management' 'Media/Entertainment'  
    'Nonprofit/Gov' 'Other' 'PE/VC' 'Real Estate' 'Retail' 'Technology']  
Mapping (범주 -> 인코딩 값) for work_industry: {'CPG': 0, 'Consulting': 1, 'Energy': 2, 'Financial Services': 3, 'Health'
```

데이터 정규화

서로 다른 스케일의 데이터를 일정한 범위로 조정.

```
scaler = StandardScaler()  
X_scaled_augmented = scaler.fit_transform(X)
```

- StandardScaler 객체 생성. (각 피처의 값이 평균 0, 표준편차 1이 되도록 변환해주는 객체)
- fit_transform() 메서드를 사용하여 데이터의 평균과 표준편차를 계산하고, 계산된 표준편차를 이용해 데이터를 변환.

4. 데이터 불균형

Admission 데이터 불균형

그래프에서 볼 수 있듯이, **합격(0)** 클래스는 소수, **불합격(1)** 클래스는 다수로 구성되어 있음

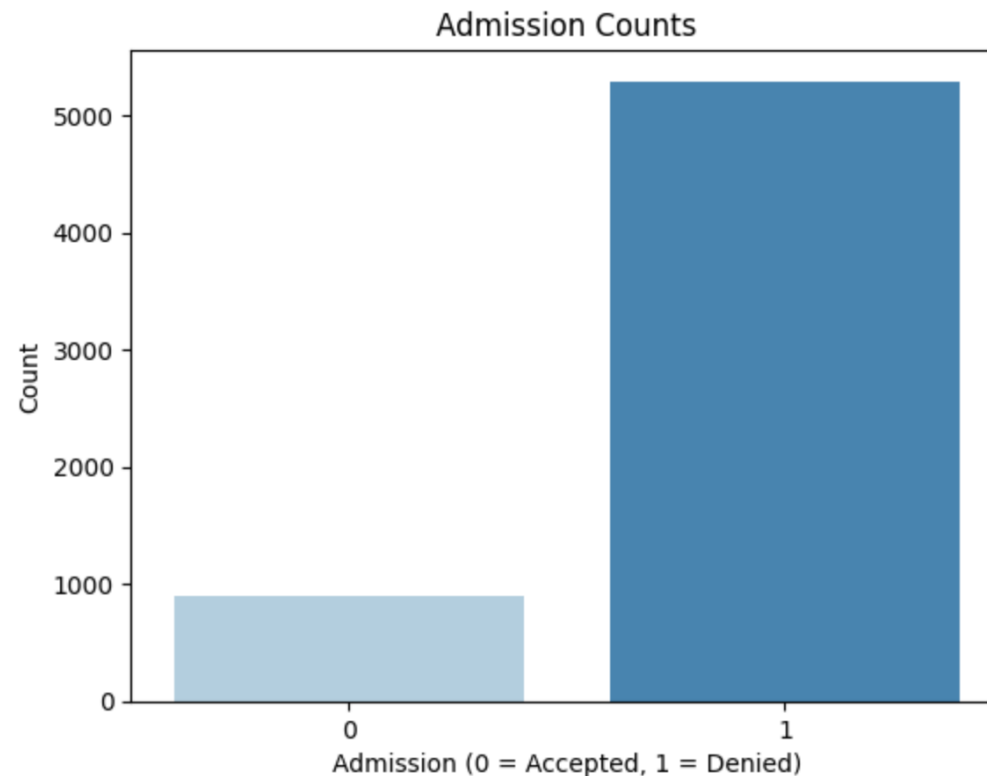
합격 샘플은 약 1,000개, 불합격 샘플은 약 5,000개 이상 존재하는 불균형 상태

문제점:

- 불균형한 데이터는 모델 학습에 부정적인 영향을 미침
- 모델이 다수 클래스에 편향되기 쉽고, 소수 클래스에 대한 예측 성능이 저하될 수 있음
- 이로 인해 모델은 정확도가 높아 보일 수 있지만, 소수 클래스를 제대로 예측하지 못하는 상황이 발생함

해결 방법

1. 언더샘플링(Undersampling)
2. 오버샘플링(Oversampling)
3. SMOTE(Synthetic Minority Over-sampling Technique)
4. CTGAN



언더샘플링(Undersampling)

정의: 다수 클래스의 샘플 수를 줄여서 클래스 간의 균형을 맞추는 기법(Mohammed et al., 2020).

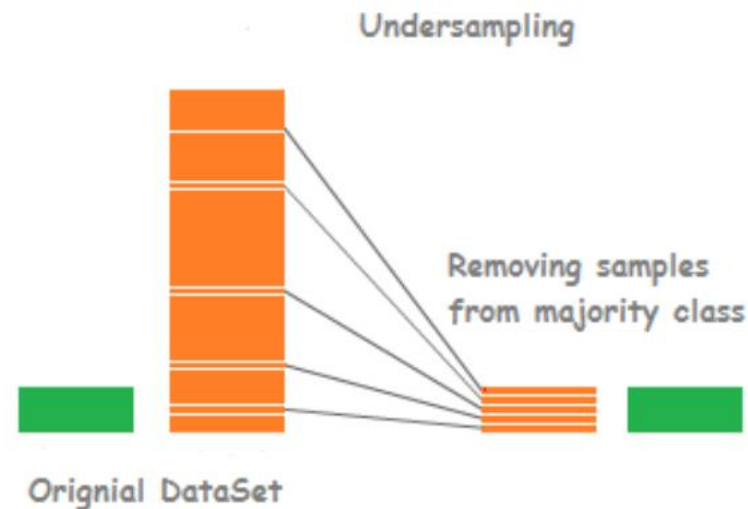
- 다수 클래스에서 일부 데이터를 무작위로 제거하여 균형을 맞춤

장점:

- 처리 시간 단축: 데이터 크기가 감소하므로 모델 학습이 빨라짐.
- 불필요한 데이터 삭제로 과적합을 방지할 수 있음

단점:

- 정보 손실:** 유의미한 데이터까지 삭제될 수 있어 성능 저하 가능성이 있음



```
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler(random_state=42)
X_resampled, y_resampled = rus.fit_resample(X_train, y_train)
```

오버샘플링(Oversampling)

정의: 소수 클래스의 샘플 수를 인위적으로 늘려서 클래스 간의 균형을 맞추는 기법(Mohammed et al., 2020).

- 소수 클래스의 샘플을 복제하거나 새로운 샘플을 생성하여 사용함

장점:

- 정보 보존:** 모든 데이터를 유지하면서 모델을 학습할 수 있어 정보 손실이 적음
- 소수 클래스에 대한 성능을 개선할 수 있음

단점:

- 과적합 위험:** 소수 클래스의 데이터를 반복적으로 사용하면서 학습 데이터에 과적합될 가능성이 있음



```
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(X_train, y_train)
```

SMOTE(Synthetic Minority Over-sampling Technique)

정의: SMOTE는 소수 클래스의 샘플을 단순 복제하는 대신, 기존 샘플의 인접한 데이터 포인트들 사이에서 새로운 데이터를 생성하는 기법 (Chawla et al., 2002).

목적: 데이터 불균형을 해결하고, 모델이 소수 클래스에 대한 일반화 성능을 향상시키도록 도움

작동 방식:

1. 소수 클래스의 각 샘플에 대해 가장 가까운 이웃(K-Nearest Neighbors, K-NN)을 찾음
2. 이웃 샘플들과의 차이를 계산하여 새로운 샘플을 생성
3. 새로운 샘플은 기존의 소수 클래스 샘플과 인접한 공간에 위치하게 되어 데이터의 다양성을 증가시킴

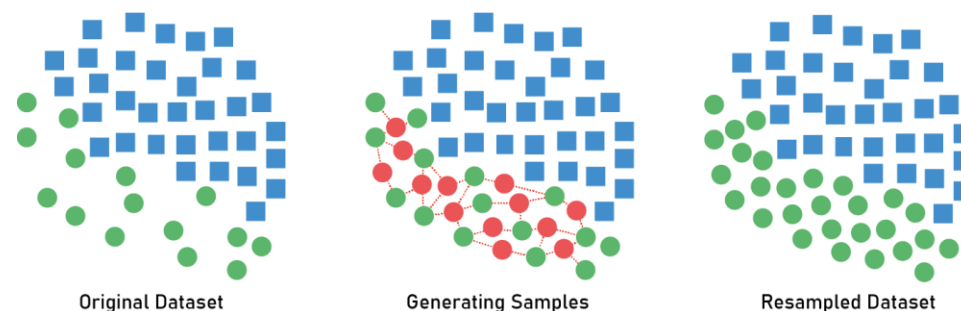
장점:

- 과적합을 방지하면서 소수 클래스의 데이터 다양성을 증가시킴
- 소수 클래스의 데이터가 부족한 상황에서 모델의 성능을 향상시킴

단점:

- SMOTE가 생성한 샘플이 실제 데이터 분포를 충분히 반영하지 못할 수 있으며, 소수 클래스의 경계에서 과적합이 발생할 위험이 있음

Synthetic Minority Oversampling Technique



```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
```

각 기법 적용 후 소수 클래스 성능 결과

UnderSampling

```
pd.Series(y_resampled).value_counts()
```

count	
admission	
0	786
1	786

dtype: int64

Model: Logistic Regression

Accuracy: 71.83%

Classification Report:

	precision	recall	f1-score	support
0	0.35	0.77	0.49	214
1	0.94	0.71	0.81	1025

accuracy			0.72	1239
macro avg	0.65	0.74	0.65	1239
weighted avg	0.84	0.72	0.75	1239

ROC-AUC: 0.82

OverSampling

```
pd.Series(y_resampled).value_counts()
```

count	
admission	
1	4169
0	4169

dtype: int64

Model: Logistic Regression

Accuracy: 72.15%

Classification Report:

	precision	recall	f1-score	support
0	0.36	0.76	0.49	214
1	0.93	0.71	0.81	1025

accuracy			0.72	1239
macro avg	0.65	0.74	0.65	1239
weighted avg	0.83	0.72	0.75	1239

ROC-AUC: 0.82

SMOTE

```
pd.Series(y_resampled).value_counts()
```

count	
admission	
1	4169
0	4169

dtype: int64

Model: Logistic Regression

Accuracy: 72.48%

Classification Report:

	precision	recall	f1-score	support
0	0.36	0.76	0.49	214
1	0.94	0.72	0.81	1025

accuracy			0.72	1239
macro avg	0.65	0.74	0.65	1239
weighted avg	0.84	0.72	0.76	1239

ROC-AUC: 0.82

언더샘플링, 오버샘플링, SMOTE를 활용해 모델을 학습한 결과, 다수 클래스에서는 괜찮은 성능을 보였으나 **소수 클래스의 F1 Score**는 여전히 낮음

5. CTGAN을 활용한 데이터 불균형 해결

CTGAN이란?

정의:

- CTGAN(Conditional Tabular GAN)은 테이블형 데이터의 행 확률 분포를 학습하고, 이를 기반으로 현실적인 합성 데이터를 생성하는 모델 (Xu et al., 2019).

주요 문제

범주형 데이터의 불균형(Discrete Column Imbalance):

- Admission 데이터에서 **Admitted (10%)**와 **Denied(90%)** 같은 불균형이 발생함
- 기존의 데이터 생성 모델들은 이런 불균형을 잘 처리하지 못함

연속형 데이터의 복잡한 분포(Continuous Column Distribution):

- 연속형 데이터는 종종 **multi-modal** (여러 개의 분포 모드)이나 **non-Gaussian** (정규분포가 아님)인 경우가 많음
- 기존 방식으로 이 데이터를 처리하면 모델이 복잡한 분포를 학습하지 못하거나 경사 소실(**vanishing gradient**) 문제가 발생할 수 있음

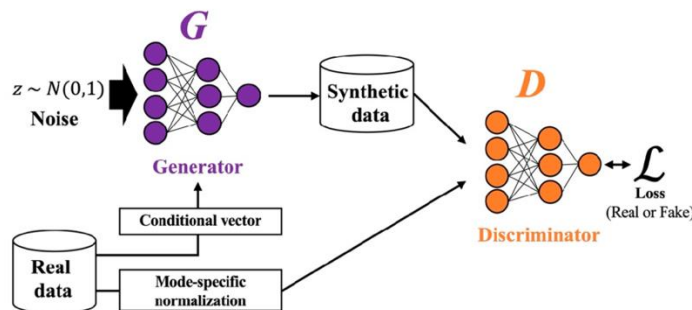
CTGAN이란?

Mode-Specific Normalization:

1. 연속형 데이터의 복잡한 분포 문제를 해결하기 위해, 각 컬럼의 데이터를 여러 개의 Gaussian 모드로 나눠 정규화(normalization) 함
2. 이렇게 하면 단순한 min-max 정규화보다 데이터의 다양한 분포를 더 잘 표현할 수 있음

Conditional Generator & Training-by-Sampling:

1. 범주형 데이터의 불균형 문제를 해결하기 위해, 조건부 생성기(Conditional Generator)를 사용함
2. 범주형 데이터의 특정 클래스를 조건으로 사용해 모델이 소수 클래스를 학습할 수 있도록 함.
3. 이 방식은 **GAN**이 데이터의 불균형을 해결하는 데 도움이 됨



CTGAN이란?

1. 생성기 (Generator, G)

입력:

- 노이즈 벡터 $z \sim N(0,1)$
- 조건부 벡터 (Conditional Vector)로 범주형 데이터 학습
- Mode-Specific Normalization으로 연속형 데이터의 복잡한 분포 처리

출력:

- 합성 데이터(Synthetic Data) 생성
- 판별기가 가짜 데이터를 진짜로 인식하도록 학습

2. 판별기 (Discriminator, D)

입력:

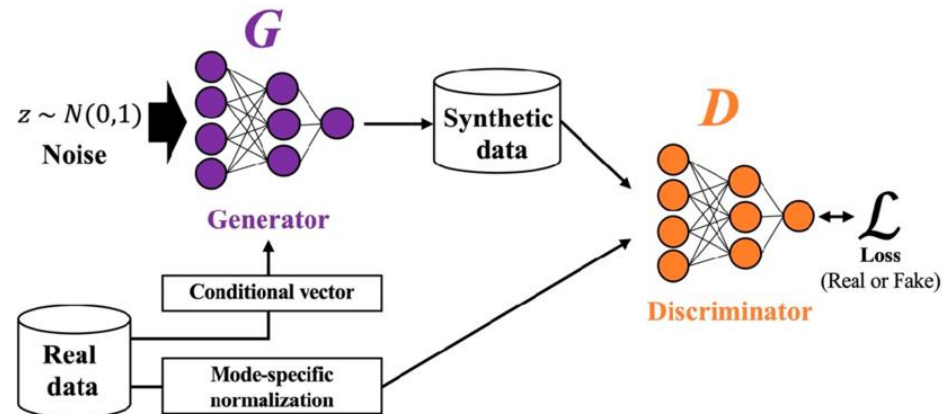
- 생성된 합성 데이터와 실제 데이터 모두 입력 받음

역할:

- 데이터가 진짜(real)인지 가짜(fake)인지 구별
- 생성기가 생성한 데이터를 평가하여 손실(Loss)을 계산

경쟁적 학습:

- 생성기와 상호작용하며 가짜 데이터를 더 정확히 구별하도록 학습



- 생성기와 판별기는 **Adversarial Learning**(경쟁적 학습)을 통해 서로의 성능을 향상시킴.
- 반복적인 학습으로 더 **현실적인 합성 데이터** 생성 가능.

CTGAN 적용

```
# 소수 클래스만 추출 (여기서는 '0'이 소수 클래스라고 가정)
minority_class_data = df[df['admission'] == 0] # admission이 0인 데이터 추출
minority_class_data = minority_class_data.drop(columns=['admission']) # 타겟 열 제외

# CTGAN 모델 정의 및 학습
ctgan = CTGAN(epochs=100) # 학습할 epoch 수는 필요에 따라 조정 가능
categorical_cols = df.select_dtypes(include=['object']).columns.tolist()
ctgan.fit(minority_class_data, discrete_columns=categorical_cols) # 범주형 컬럼 명시

# 새로운 데이터 생성 (예: 4800개의 소수 클래스 데이터 생성)
new_samples = ctgan.sample(4800)

# 생성된 데이터를 원래 소수 클래스의 레이블인 0으로 지정
new_samples['admission'] = 0

# 기존 데이터와 결합
df_augmented = pd.concat([df, new_samples], ignore_index=True)
```

Model: Logistic Regression

Accuracy: 87.90%

Classification Report:

	precision	recall	f1-score	support
0	0.89	0.88	0.88	1152
1	0.87	0.88	0.87	1047

accuracy			0.88	2199
macro avg	0.88	0.88	0.88	2199
weighted avg	0.88	0.88	0.88	2199

ROC-AUC: 0.95

1. 소수 클래스 데이터 추출

- admission 컬럼에서 소수 클래스(여기서는 0)만 선택하여 소수 클래스 데이터를 추출.
- 타겟 열인 admission 컬럼은 제외한 상태로 학습에 사용.

2. CTGAN 모델 정의 및 학습

- CTGAN 모델을 정의하고 epochs=100으로 설정하여 학습.
- 범주형 컬럼(categorical columns)은 별도로 지정하여 모델에 입력.
- 소수 클래스 데이터만 사용해 CTGAN 모델 학습 진행.

3. 새로운 소수 클래스 데이터 생성

- 4800개의 소수 클래스 데이터를 생성하여 원래의 소수 클래스 데이터와 수를 맞춤.
- 이는 다수 클래스와 소수 클래스의 균형을 맞추기 위해 설정.

4. 소수 클래스 레이블 부여 및 데이터 결합

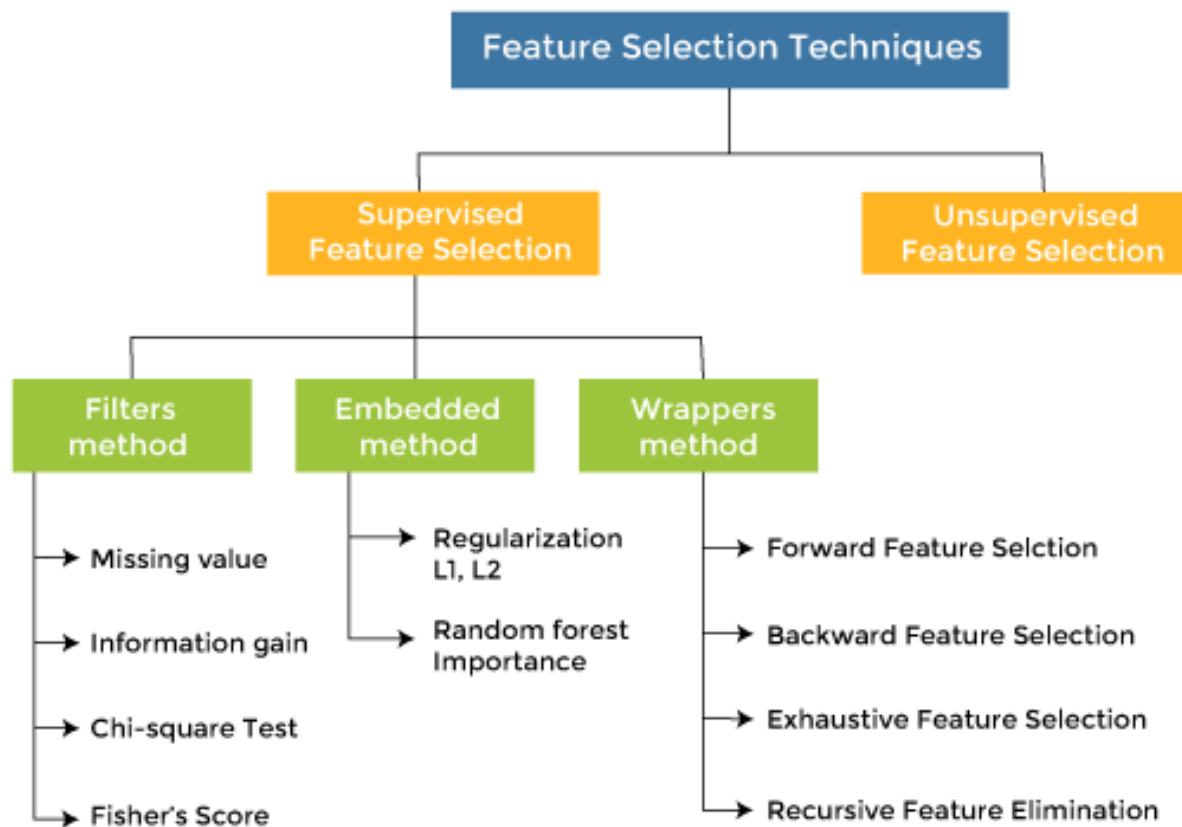
- 생성된 데이터에 admission = 0 으로 소수 클래스 레이블을 부여.
- 기존 데이터와 결합하여 새로운 증강된 데이터셋 완성.

증강된 소수 클래스(Admitted: 0)와 향상된 F1 Score

6. 특성 공학

특성 선택 이란?

- 특성 선택은 **모델 성능을 향상시키기 위해**, 데이터의 여러 특성들 중에서 **중요한 특성만을 선택하는 과정**
- 모든 특성이 모델 학습에 필요하지 않기 때문에, 불필요하거나 상관관계가 낮은 변수를 제거하여 **모델의 복잡도를 낮추고 일반화 성능을 높이는 목적**
- 모델의 성능 향상 뿐만 아니라, 해석 가능성을 높이고, 학습 시간을 단축시키는 역할**을 함



분류와 회귀 문제에서의 특성 선택 차이

분류 문제:

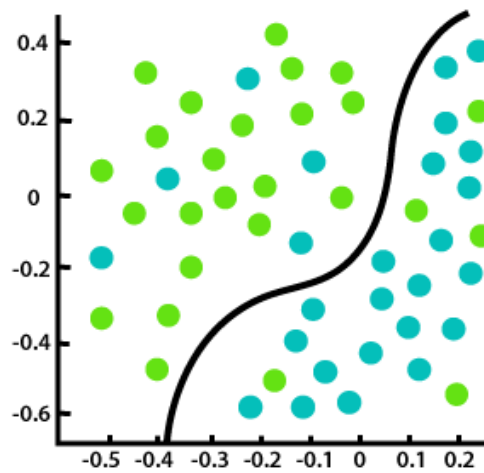
- 범주형 변수에 대한 독립성 검정과 피쳐 중요도 평가가 중요.
- 카이제곱 검정, 랜덤 포레스트, 라쏘와 같은 기법을 주로 사용.

회귀 문제:

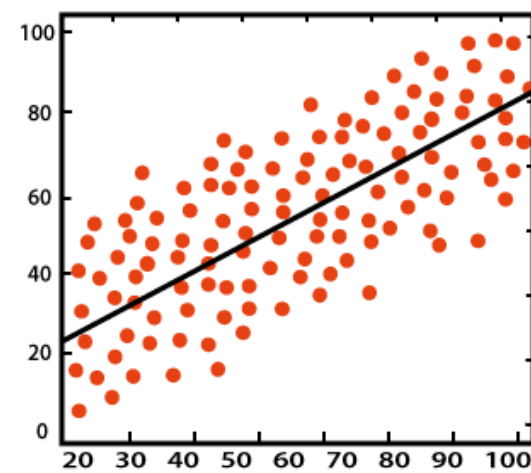
- 수치형 변수 간의 상관관계와 가중치 규제를 통해 중요한 변수를 선택.
- Lasso, RFE, 피어슨 상관계수 등의 기법을 주로 사용.

핵심 차이:

- 분류 문제는 범주형 특성이 많으며, 이를 다루기 위한 특성 선택 기법이 필요.
- 회귀 문제는 수치형 특성의 비중이 높으며, 독립 변수와 종속 변수 간의 관계를 반영한 특성 선택 기법이 주로 사용됨.



Classification



Regression

MBA 프로젝트는 분류 문제이므로, 분류 특성 선택 기법을 사용했습니다.

카이제곱 검정

목적: 각 변수(컬럼)와 종속 변수(입학 여부, 'admission') 간의 독립성을 검정하여, **중요한 변수**를 선택.

카이제곱 검정(Chi-Square Test):

- 범주형 변수와 종속 변수 간의 연관성을 평가하는 통계적 방법.
- 각 변수의 **p-value**를 기준으로, **유의미한 특성**을 선택함.

결과 해석

- **카이제곱 통계량:**
 - 각 변수와 'admission' 간의 **연관성을 측정**.
 - 카이제곱 값이 클수록 변수 간의 관계가 강함을 의미.
- **p-value:**
 - **p-value**는 **유의미성**을 나타내며, **$p < 0.03$** 이면 변수와 종속 변수 간에 **유의미한 상관관계**가 있다고 판단하여, 해당 변수를 선택.
- **자유도(DOF):**
 - 변수의 **자유도**는 변수의 카테고리 수에 따라 결정되며, 통계적 자유도를 나타냄.

```
from scipy.stats import chi2_contingency

selected_columns = []

for element in df.columns:

    contingency_table = pd.crosstab(df[element], df['admission'])

    # 카이제곱 검정 수행
    chi2, p, dof, expected = chi2_contingency(contingency_table)
    if p < 0.03:
        selected_columns.append(element)

print(element)
print("Chi-Square Statistic:", chi2)
print("p-value:", p)
print("Degrees of Freedom:", dof)
print("")
```

```
gender
Chi-Square Statistic: 95.4601654520653
p-value: 1.5089725785759555e-22
Degrees of Freedom: 1

international
Chi-Square Statistic: 0.5839856987918092
p-value: 0.4447542811936849
Degrees of Freedom: 1

gpa
Chi-Square Statistic: 677.0609368349182
p-value: 1.629006490177676e-86
Degrees of Freedom: 100

major
Chi-Square Statistic: 0.28451932190137114
p-value: 0.8673959983722536
Degrees of Freedom: 2

race
Chi-Square Statistic: 51.11047016621665
p-value: 8.209554766013871e-10
Degrees of Freedom: 5

gmat
Chi-Square Statistic: 939.5027943511665
p-value: 2.1223845341083515e-185
Degrees of Freedom: 1

work_exp
Chi-Square Statistic: 10.390943281471268
p-value: 0.2386514807174321
Degrees of Freedom: 8

work_industry
Chi-Square Statistic: 17.649920799709022
p-value: 0.17126972345693742
Degrees of Freedom: 13

admission
Chi-Square Statistic: 6186.61567232219
p-value: 0.0
Degrees of Freedom: 1
```

랜덤 포레스트란?

- 랜덤 포레스트는 여러 개의 ****결정 트리(Decision Tree)****를 사용하는 **앙상블 학습 알고리즘**
- 각 트리는 **훈련 데이터의 일부와 특성의 일부**를 무작위로 선택하여 학습하며, 그 결과를 종합해 **예측 성능을 높임**

특징 중요도(Feature Importance) 추출

1. **특징 중요도**는 랜덤 포레스트에서 **각 특성이 모델의 예측에 기여하는 정도**를 나타냄
2. **특성 중요도 계산 방법**:
 - 각 결정 트리에서 **특성을 사용하여 데이터를 분할할 때, 불순도(Impurity) 감소량**을 계산함
 - 랜덤 포레스트는 모든 트리에서의 **평균 불순도 감소량**을 바탕으로 **각 특성의 중요도**를 추출함

```
# 랜덤 포레스트 모델 학습
model = RandomForestClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

# 특성 중요도 추출
importances = model.feature_importances_
print(len(importances))
# 특성 중요도를 데이터프레임으로 변환
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': importances})

# 중요도 순으로 정렬
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# 결과 출력
print(feature_importance_df)
```

```
8
      Feature  Importance
2         gpa    0.315986
5         gmat    0.290446
7  work_industry    0.133386
6       work_exp    0.093658
4         race    0.063968
3         major    0.059652
0         gender    0.030113
1  international    0.012791
```

Lasso(라쏘 회귀)란?

- Lasso 회귀(Lasso Regression)는 **L1 정규화**를 적용하는 회귀 기법임.
- **L1 정규화**는 특성의 **가중치를 0으로 만들어 불필요한 특성을 자동으로 제거하는** 효과를 가짐.
- 이를 통해, **모델의 복잡도를 줄이면서도 중요한 특성만 남길 수 있는 특성 선택(Feature Selection)** 기법으로 사용됨.

Lasso 특성 선택 결과 해석

- 선택된 특성은 모델의 **예측 성능에 중요한 변수**임
- **Lasso 회귀**는 **선형 모델**에서 특성을 선택하는 데 유용하며, 불필요한 특성을 제거해 **모델을 단순화**하고 **해석 가능성**을 높임.

```
lasso = Lasso(alpha=0.01)
lasso.fit(X_train, y_train)

lasso_coef = pd.Series(lasso.coef_, index=X.columns)
selected_features = lasso_coef[lasso_coef != 0].index

print(f"Selected Features by Lasso: {selected_features}")
```

➡ Selected Features by Lasso: Index(['gender', 'gpa', 'race', 'gmat'], dtype='object')

최종 특성 선택 방법

기법별 선택된 특성의 교집합:

- 카이제곱, 랜덤 포레스트, Lasso 회귀에서 공통으로 선택된 특성을 우선적으로 선택.
- 각 기법이 서로 다른 관점에서 특성을 평가하므로, 중복된 특성은 중요할 가능성이 높음.

모델 실험 결과 기반:

- 성능 실험을 통해 성능을 향상시키는 특성을 추가적으로 선택하여 최적의 특성 조합을 찾음.

도메인 지식 활용:

- 데이터 분석 결과 **성별과 인종**이 중요한 변수로 확인되었으나, **사회적 편견**을 방지하고 **공정한 입학 평가**를 위해 **성별, 인종, 국제학생 여부**와 같은 변수를 모델에서 제외
- 프로젝트 배경에서 제시한 것처럼, **사회적 배경에 따른 편견**을 **최소화**하고, 객관적이고 **공정한 평가**가 이루어지도록 하기 위함.

```
gender
Chi-Square Statistic: 95.4601654520653
p-value: 1.5089725785759555e-22
Degrees of Freedom: 1

international
Chi-Square Statistic: 0.5839856987918092
p-value: 0.4447542811936849
Degrees of Freedom: 1

gpa
Chi-Square Statistic: 677.0609368349182
p-value: 1.629006490177676e-86
Degrees of Freedom: 100

major
Chi-Square Statistic: 0.28451932190137114
p-value: 0.8673959983722536
Degrees of Freedom: 2

race
Chi-Square Statistic: 51.11047016621665
p-value: 8.209554766013871e-10
Degrees of Freedom: 5

gmat
Chi-Square Statistic: 939.5027943511665
p-value: 2.1223845341083515e-185
Degrees of Freedom: 21

work_exp
Chi-Square Statistic: 10.390943281471268
p-value: 0.2386514807174321
Degrees of Freedom: 8

work_industry
Chi-Square Statistic: 17.649920799709022
p-value: 0.17126972345693742
Degrees of Freedom: 13

admission
Chi-Square Statistic: 6186.61567232219
p-value: 0.0
Degrees of Freedom: 1
```

Selected Features by Lasso: Index(['gender', 'gpa', 'race', 'gmat'], dtype='object')

선택된 특성

1. Gmat
2. GPA
3. Work_Exp

7. 모델 개발 및 평가

모델 개발

다양한 모델의 성능을 평가:

	Model	Accuracy (%)	ROC-AUC	F1-Score
1	Logistic Regression	72.99	0.79	0.73
2	K-Nearest Neighbors	71.94	0.79	0.72
3	Decision Tree	85.04	0.88	0.85
4	Random Forest	83.08	0.91	0.83
5	Support Vector Machine	75.4	0.81	0.75
6	XGBOOST	86.86	0.95	0.87
7	GaussianNB	73.17	0.79	0.73
8	GradientBoostingClassifier	83.36	0.83	0.83

가장 성능이 좋은 XGBoost
모델을 선택함

XGBoost 모델

- Gradient Boosted Decision Trees(GBDT)의 효율적 구현.
- 빠른 학습 속도와 높은 성능 제공.
- 대규모 데이터셋 및 복잡한 모델에서도 효과적.

학습 데이터: 소수 클래스의 데이터 불균형을 해결하기 위해 **증강된 데이터**를 사용하여 학습을 진행.

- CTGAN을 사용해 소수 클래스 데이터를 증강함.

테스트 데이터: 성능 평가 시에는 **증강되지 않은 원본 데이터**를 사용.

- 원본 데이터로 테스트해야 모델의 **일반화 성능**을 정확히 평가 가능.

```
models = [
    ("Logistic Regression", LogisticRegression(class_weight='balanced')),
    ("K-Nearest Neighbors", KNeighborsClassifier(n_neighbors=3)),
    ("Decision Tree", DecisionTreeClassifier(class_weight='balanced', random_state=42)),
    ("Random Forest", RandomForestClassifier(class_weight='balanced', random_state=42)),
    ("Support Vector Machine", SVC(probability=True)),
    ("XGBOOST", XGBClassifier(scale_pos_weight=10, random_state=42)),
    ("GaussianNB", GaussianNB()),
    ("GradientBoostingClassifier", GradientBoostingClassifier()),
    ("AdaBoostClassifier", AdaBoostClassifier())
]

for model_name, model in models:
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    print(f"\nModel: {model_name}")
    print(f"Accuracy: {accuracy * 100:.2f}%")

    print("Classification Report:")
    print(classification_report(y_test, y_pred))

    if hasattr(model, "predict_proba"):
        y_probs = model.predict_proba(X_test)[:, 1]
        roc_auc = roc_auc_score(y_test, y_probs)
        print(f"ROC-AUC: {roc_auc:.2f}")
```

하이퍼파라미터 튜닝

- 최적의 하이퍼파라미터 조합을 찾기 위해 다양한 값들을 실험하고 평가하는 과정.
- XGBoost의 하이퍼파라미터

```
param_grid = {  
    'n_estimators': [100, 200, 300],      # 트리 개수  
    'learning_rate': [0.01, 0.1, 0.3],    # 학습률 (스텝 크기)  
    'max_depth': [3, 5, 7],               # 트리의 최대 깊이  
    'subsample': [0.8, 1.0],              # 데이터 샘플링 비율  
    'colsample_bytree': [0.8, 1.0],        # 트리를 만들 때 사용할 컬럼의 샘플링 비율  
    'gamma': [0, 1, 5],                   # 노드를 분할하기 위한 최소 손실 감소  
    'reg_lambda': [1, 10],                # 가중치에 대한 L2 정규화 값  
    'scale_pos_weight': [1, 10]           # 클래스 불균형을 조정하는 가중치  
}
```

- `n_estimators` : 트리의 개수. 많을수록 과적합의 위험이 커짐.
- `learning_rate` : 각 학습 단계에서 가중치 업데이트의 크기를 결정하는 학습률.
- `max_depth` : 트리의 최대 깊이. 깊어질수록 과적합의 위험이 커짐.
- `subsample` : 훈련에 사용할 데이터 샘플 비율. 1.0이면 모든 데이터를 사용.
- `colsample_bytree` : 트리를 구성할 때 사용할 컬럼의 샘플링 비율.
- `gamma` : 새로운 노드를 만들 때 필요한 최소 손실 감소. 값이 크면 모델이 보수적으로 분할을 시도함.
- `reg_lambda` : L2 정규화 값. 모델의 복잡도를 억제하기 위해 사용.
- `scale_pos_weight` : 클래스 불균형이 있을 때 가중치를 조정해 보정함.

그리드 서치

- **GridSearchCV**는 주어진 하이퍼파라미터의 모든 조합을 테스트하여 가장 성능이 좋은 하이퍼파라미터 세트를 찾는 방법.
- 교차 검증을 사용하여 각 하이퍼파라미터 조합을 평가.
- **주요 파라미터**

estimator : 사용하려는 모델.

param_grid : 테스트할 하이퍼파라미터 값.

scoring : 모델 성능을 평가할 지표.

cv : 교차 검증을 위한 fold 개수.

```
# XGBoost 분류기 초기화
xgb = XGBClassifier(random_state=42)

# 5겹 교차 검증을 사용하는 GridSearchCV 설정
grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid, scoring='roc_auc', cv=5, verbose=2, n_jobs=-1)

# GridSearchCV 실행 (학습)
grid_search.fit(X_train, y_train)

# 최적의 하이퍼파라미터 출력
print("최적의 하이퍼파라미터: ", grid_search.best_params_)
```

하이퍼파라미터 튜닝

GridSearch와 XGBoost를 함께 사용하는 이유

XGBoost는 각 하이퍼파라미터의 값에 따라 성능이 크게 달라질 수 있는데, GridSearch와 같은 자동화된 하이퍼파라미터 탐색 기법을 사용하면 모든 하이퍼파라미터 조합을 테스트하고 최적의 값을 찾아주기 때문에 모델 성능을 향상시킬 수 있음.

출력 결과

```
최적의 하이퍼파라미터: {'colsample_bytree': 0.8, 'gamma': 0, 'learning_rate': 0.3, 'max_depth': 3,  
'n_estimators': 200, 'reg_lambda': 1, 'scale_pos_weight': 10, 'subsample': 1.0}
```

모델 평가

클래스 0에 대한 성능

- Precision (정밀도) : 1.00
- Recall (재현율) : 0.76
- F1-score : 0.87

클래스 1에 대한 성능

- Precision (정밀도) : 0.80
- Recall (재현율) : 1.00
- F1-score : 0.89

-> 클래스 0에 대한 Recall이 낮은 부분이 아쉬우나 ROC-AUC 값이 0.96으로 매우 높기 때문에 신뢰할 수 있는 분류 모델로 평가됨.

```
# 예측 및 평가
y_pred_best = best_xgb.predict(X_test)
accuracy_best = accuracy_score(y_test, y_pred_best)
print(f"최적 XGBoost 모델 정확도: {accuracy_best * 100:.2f}%")

print("분류 리포트:")
print(classification_report(y_test, y_pred_best))

# 최적 모델의 ROC-AUC 점수 계산
if hasattr(best_xgb, "predict_proba"):
    y_probs_best = best_xgb.predict_proba(X_test)[:, 1]
    roc_auc_best = roc_auc_score(y_test, y_probs_best)
    print(f"최적 모델 ROC-AUC: {roc_auc_best:.2f}")
```

최적 XGBoost 모델 정확도: 87.86%

분류 리포트:

	precision	recall	f1-score	support
0	1.00	0.76	0.87	1127
1	0.80	1.00	0.89	1072
accuracy			0.88	2199
macro avg	0.90	0.88	0.88	2199
weighted avg	0.90	0.88	0.88	2199

최적 모델 ROC-AUC: 0.96

8. 결과

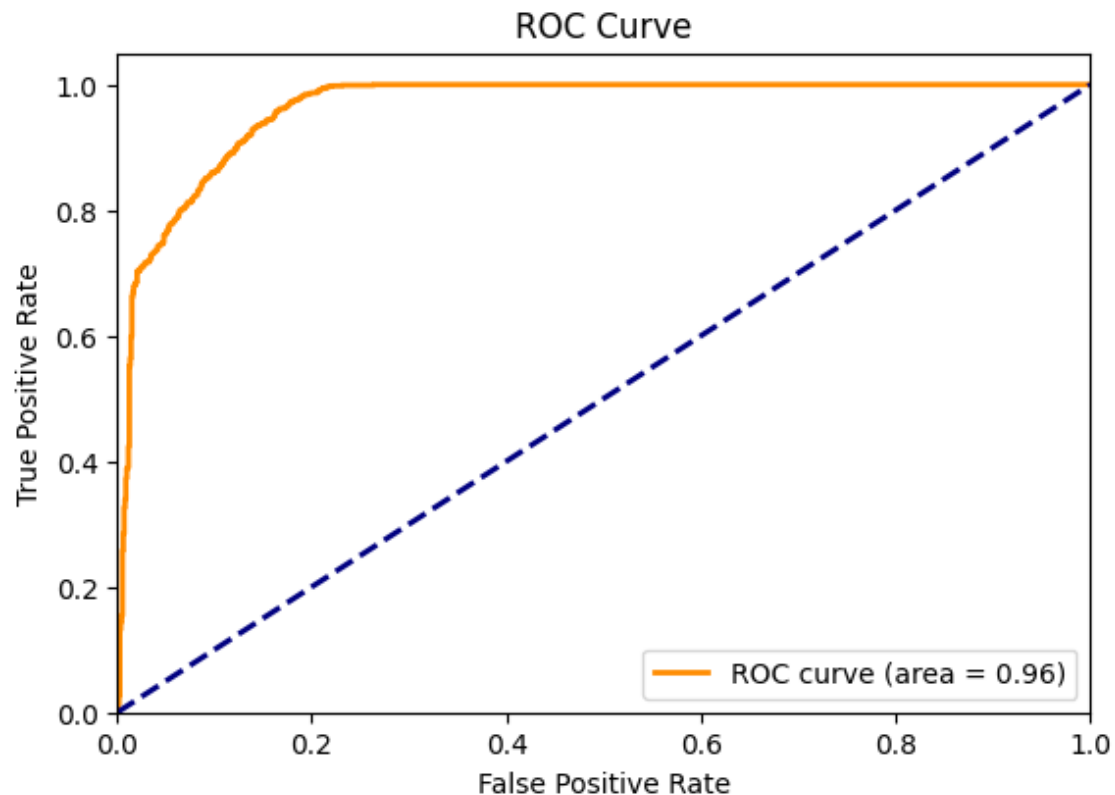
ROC Curve

곡선의 면적을 나타내는 AUC가 모델의 성능을 요약하는 지표로, 모델의 분류 성능을 평가하는데 사용.

AUC 값 : 0.96

-> AUC 값이 1에 가까울수록 완벽한 분류, 0.5는 무작위 추측

-> 곡선의 모양이 왼쪽 상단 모서리에 가까울수록 모델의 성능이 좋음.



Learning Curve

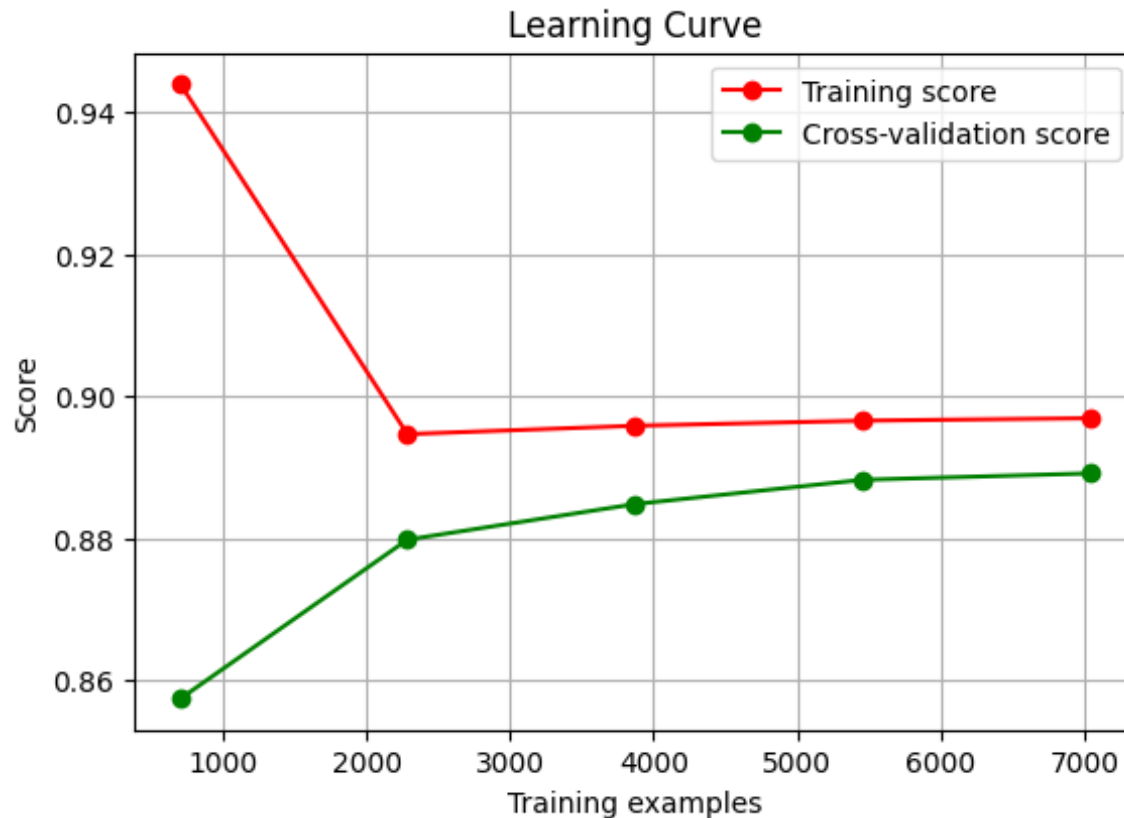
훈련 데이터와 교차 검증 데이터의 성능을 비교하여 과적합 또는 과소적합을 확인.

Training score

데이터가 적을 때 0.94에서 시작하지만, 데이터가 증가하면서 안정적으로 0.90 근처에서 평평해짐.

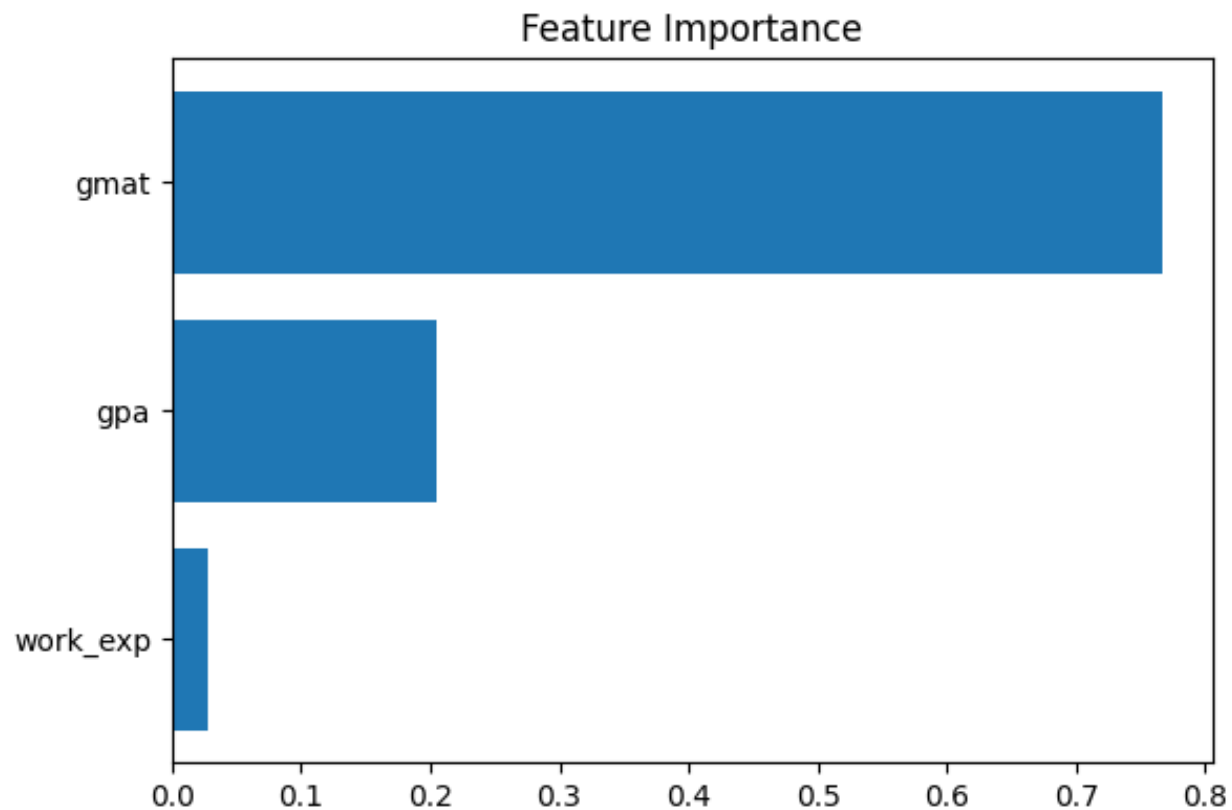
Cross-validation score

초기에는 0.86으로 낮았지만, 데이터가 증가하면서 상승하여 0.89 정도에서 수렴함.



Feature Importance

gmat, gpa, work_exp 순으로 모델의 예측에 가장 큰 영향을 미침.



9. 결론

주요 발견 사항


- 데이터 불균형 문제를 CTGAN으로 해결
- 성별, 국제학생 여부, 인종이 입학 가능성에 영향을 미치지만, 투명성을 위해 해당 변수들을 제외한 모델 개발
- **F1-Score 0.88**이라는 높은 성능을 달성

결론

- 시험 성적(GMAT, GPA)과 경력(work_exp)만을 사용한 모델이 **F1-Score 0.88**의 성능을 기록하며, **성공적으로 입학 가능성을 예측함**
- 성별, 국제학생 여부, 인종이 입학에 영향을 미칠 수 있다는 점을 확인했으나, 투명성을 유지하기 위해 이를 제외한 상태로도 **성공적인 예측이 가능함**

제안

- 향후, 추가적인 데이터를 통해 더 많은 특성을 분석하고 모델을 개선할 수 있음
- 더 다양한 특성을 사용한 모델 성능 평가를 통해, 입학 절차의 객관성을 높이는 방안 고려



Model: XGB00ST				
Accuracy: 89.54%				
Classification Report:				
	precision	recall	f1-score	support
0	0.99	0.81	0.89	1152
1	0.83	0.99	0.90	1047
accuracy			0.90	2199
macro avg	0.91	0.90	0.90	2199
weighted avg	0.91	0.90	0.89	2199
ROC-AUC: 0.96				

성별, 인종을 추가한 모델 성능

참고 문헌

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16, 321-357.

Mohammed, R., Rawashdeh, J., & Abdullah, M. (2020). Machine Learning with Oversampling and Undersampling Techniques: Overview Study and Experimental Results. *Proceedings of the 2020 International Conference on Information and Communication Systems (ICICS)*, 243-248. <https://doi.org/10.1109/ICICS49469.2020.239556>

Xu, L., Skoularidou, M., Cuesta-Infante, A., & Veeramachaneni, K. (2019). Modeling Tabular Data using Conditional GAN. *Advances in Neural Information Processing Systems*, 32, 7335-7345.

감사합니다.