web programming team project

신용카드고객 세그먼트분류 AI경진대회


```
<p style="text-align:left;">Left</p>
<p style="text-align:center;">Center</p>
<p style="text-align:right;">Right</p>
-
-
-
```

프로젝트 개발 7조

김민수,신승준, 장민성, 조형일

web programming team project

목차

1. 데이터 소개
2. EDA
3. 전처리
4. 모델링
5. 질문(Q&A)

PyCharm 파이프라인

web programming team project

PyCharm 기반 파이프라인 구성

1. 전체 구조 강조: “ML 파이프라인 전체를 직접 구현 및 실행”

"데이터 전처리부터 모델 학습, 캐스케이드 구조 적용, 테스트 예측, 최종 submission 생성까지 모든 과정을 PyCharm 환경 내에서 직접 구성하고, 각 단계의 모듈을 분리하여 관리했습니다. “직접 설계 → 실행 → 에러 핸들링 → 리팩토링” 과정을 반복하며 점진적으로 성능을 개선

PyCharm 프로젝트 구조

```
card_segment
├ config
|  ├ train.py      ← 모델 학습 및 저장
|  ├ infer.py      ← 추론 및 submission 생성
├ data
|  └ preprocessing.csv
├ models
|  └ Stage_A.pkl
|  └ ...
├ requirements.txt
└ README.md
```

infer.py | submission.csv | card_test.csv | train.py | temp.py | preprocessing.csv | preprocessing.py

sklearn(으)로 효율성 극대화

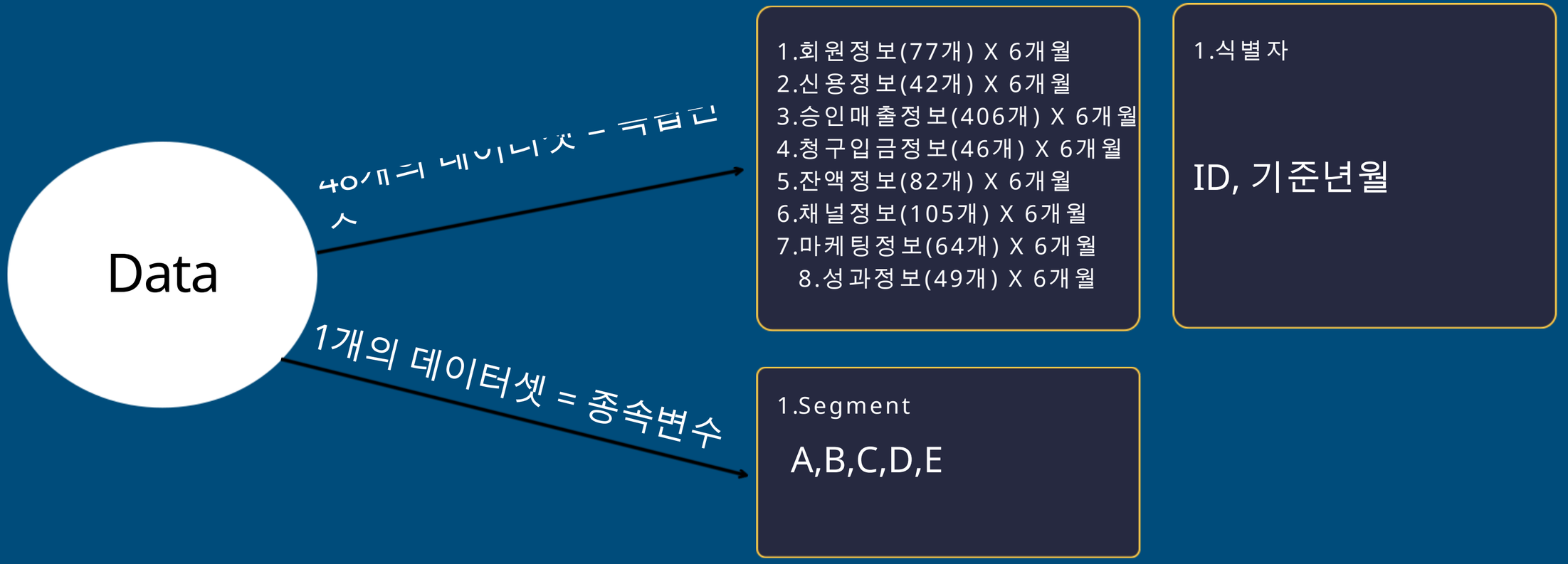
```python
import pandas as pd, numpy as np, joblib, os, xgboost as xgb
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import f1_score, classification_report
from pathlib import Path
from imblearn.over_sampling import RandomOverSampler

# 경로
BASE_DIR   = Path(f"./")
DATA_PATH  = "C:/Users/mstot/PycharmProjects/PythonProject1/card_segment/data/preprocessing.csv"
# ENC_PATH   = BASE_DIR / "cat_encoder.pkl"
MODEL_DIR  = Path(f"./")
TARGET, ID_COL = "Segment.1", "ID"

MODEL_DIR.mkdir(parents=True, exist_ok=True)

# 데이터 로드 (이미 인코딩·캐스팅 완료 상태)
df = pd.read_csv(DATA_PATH)

# 학습 피처 / 레이블
X_full = df.drop(columns=[TARGET, ID_COL]).astype(np.float32)   # float32 통일
# ── 숫자형만 선택 (문자형 변환 방지) ──────
X_full = X_full.select_dtypes(include=[np.number]).astype(np.float32)
y_full = df[TARGET].str.upper()                                 # 'A'~'E'

class DummyModel:  1개의 사용 위치
    def __init__(self, constant: float = 0.0):
        self.constant = constant

    def predict(self, X):  3개의 사용 위치(3개의 동적)
        return np.full(len(X), self.constant, dtype=np.float32)

    def predict_proba(self, X):  1개의 사용 위치(1개의 동적)
        return np.full( shape: (len(X), 2),  fill_value: [1 - self.constant, self.constant], dtype=np.float32)
```

데이터 소개

web programming team project

1. 금융 활동 데이터

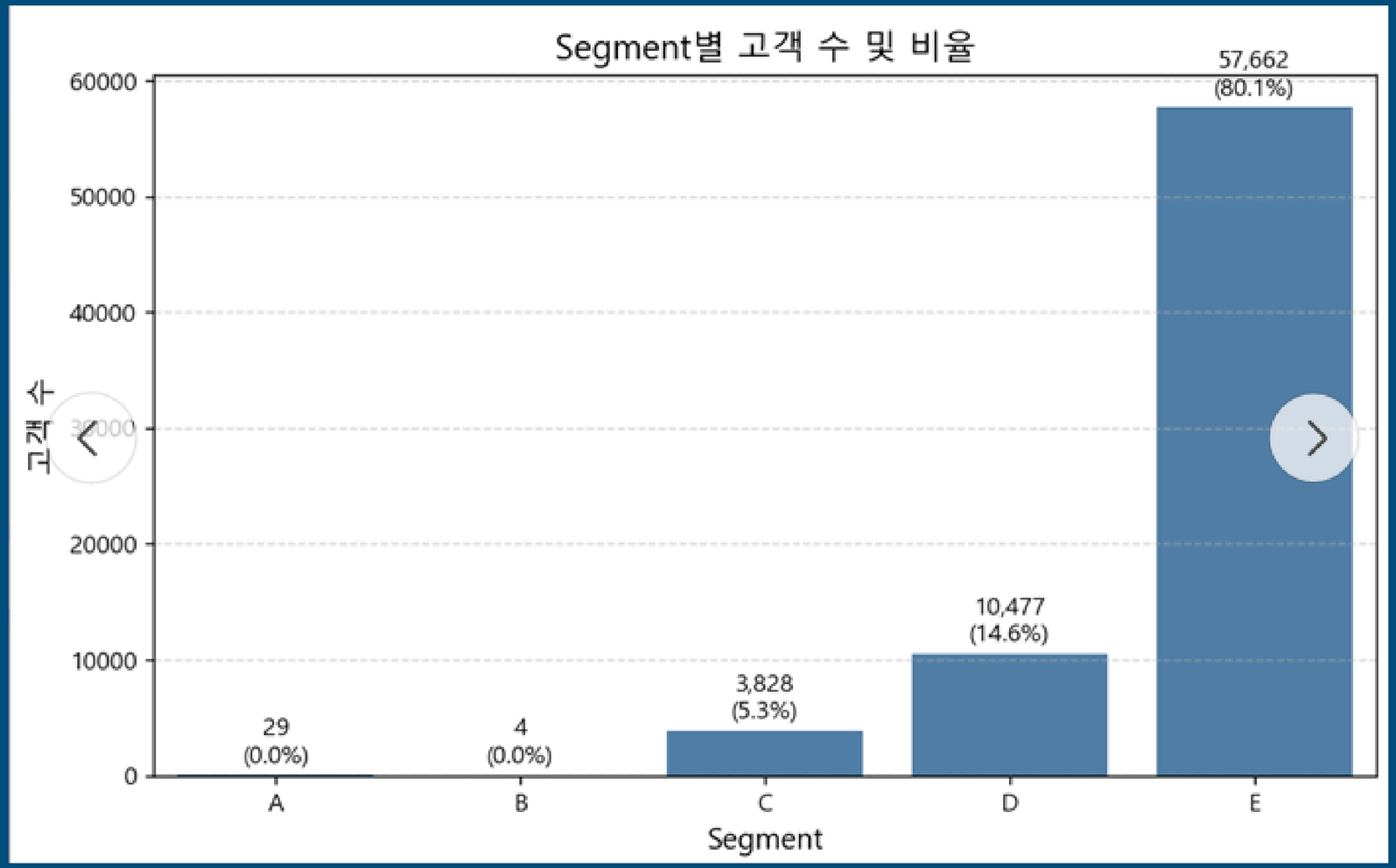
고객의 신용정보, 승인매출, 청구, 잔액, 채널, 마케팅, 성과 등 다양한 데이터 포함



EDA

web programming team project

2. EDA(Segment 비율)



Segment
A 29
B 4
C 3,828
D 10,447
E 57,667
Name: count, dtype: int64

전처리

web programming team project

3. 주요 전처리

1) 데이터 통합

2) 이상치 처리

3) 날짜 변수 처리

4) 문자형 변수 처리

5) 불필요 변수 정리

6) 인코딩

7) 파생 변수 생성

전처리

web programming team project

데이터 통합

원천 데이터는 2018년 7~12월, 총 8개 카테고리의 월별 parquet 파일을 통합하여 240만 건의 train, 60만건의 test 데이터로 구성되었고, 최종적으로 각각 858개, 857개의 feature를 갖는 train_최종, test_최종 데이터셋이 생성. 이후 두 데이터를 통합하여 preprocessing.csv 파일로 저장하였습니다.해당 통합 데이터셋의 feature 수는 708개로, 전처리를 통해 열 수를 효과적으로 축소한 결과입니다.

이상치 처리		
구분	**처리 대상 컬럼**	**특수값 → 처리 방식**
1. 연체일수 정제	연체일수_B1M, 연체일수_B2M, 연체일수_최근	-999999 → NaN
2. 연체회차 정제	최종연체회차	-99 → NaN
3. 경과일 정제	rv최초시작후경과일	99999999 → NaN
4. 카드론 경과월 정제	최초카드론이용경과월, 최종카드론이용경과월	999 → NaN
5. 일자 컬럼 정제	날짜/년월 관련 컬럼(예: 최종이용일자_*)	소수점 반올림 후 → Int64 형 변환

전처리

web programming team project

날짜 변수 처리	
단계	**처리 내용**
1. 날짜 추정	숫자형 컬럼 중 길이 8자리 → YYYYMM 형식으로 인식
2. 타입 변환	int64 → 문자열(str) → pd.to_datetime()으로 날짜 형식 변환 (에러는 무시)
3. 날짜 분해	변환된 날짜 컬럼 → year, month, day 컬럼으로 분리 생성
4. 원본 제거	변환된 날짜 컬럼은 drop() 처리(원본 삭제)
5. 예외 처리	변환 실패 시, 오류 메시지 출력 후 다음 컬럼으로 진행

문자형 변수 처리	
처리 대상	**처리 방식**
1. 이용금액대	사전 정의된 문자열 → 정수 값 매핑(예: “05.10만원-” → 10)
2. 연령	“20대” → 20, “50대이상” → 50
3. ‘만원’포함 문자열	“30만원” → 30.0, “-10만원” → -10.0
4.‘회’포함 문자열	“2회” → 2
5.‘대’ 포함 문자열	“2대” → 2

전처리

web programming team project

불필요한 열 제거전, 종합방문횟수 계산(파생변수 생성)

1. 방문횟수_PC_R6M, 방문일수PC_R6M, 방문횟수_앱_R6M

2. 방문일수_앱_R6M, 방문횟수_모바일웹_R6M, 방문일수_모바일웹_R6M

3. 방문횟수_PC_B0M, 방문일수_PC_B0M, 방문횟수_앱_B0M, 방문일수_앱_B0M

4. 방문횟수_모바일웹_B0M, 방문일수_모바일웹_B0M

전처리

web programming team project

불필요한 열 제거	
1. 대표 결제일	7. _1순위납부업종, _2순위납부업종, _3순위납부업종
2. 통합_납부_업종	8. 통합_기본업종, 통합_쇼핑업종, 통합_교통업종
3. 연체일자_B0M	9. 통합_여유업종, 통합_납부업종
4. _1순위업종, _2순위업종, _3순위업종	10. 상품관련면제카드수_B0M, 캠페인접촉일수_R12M
5. _1순위교통업종, _2순위교통업종, _3순위교통업종	11. 시장단기연체여부_R6M

전처리

web programming team project

파생변수	계산 방식
모멘텀갭_B0M_B2M	(연체입금_B0M – 연체입금_B2M) – (정상청구_B0M –정상청구_B2M)
모멘텀갭_B0M_B2M	(연체입금_B2M – 연체입금_B5M) – (정상청구_B2M – 정상청구_B5M)
모멘텀갭_B0M_B2M	(연체입금_B0M – 연체입금_B5M) – (정상청구_B0M – 정상청구_B5M)
모멘텀갭_B0M_B2M	sigm x log1p(abs(모멘텀갭))

해석
1. 회수 속도 vs 청구 속도 차이 (최근 2개월)
2. 회수 청구 속도 차이
3. 전체 5개월 누적 갭
4.로그 스케일 버전

인코딩

web programming team project

범주형 라벨 인코딩 → 실제로는 2가지 유효값만 가지는 컬럼들 0/1로 매핑처리

1. 대표결제방법코드, 가입통신회사코드, _1순위신용체크구분, _2순위신용체크구분

2. RV전환가능여부, OS구분코드, Segment, Life_Stage

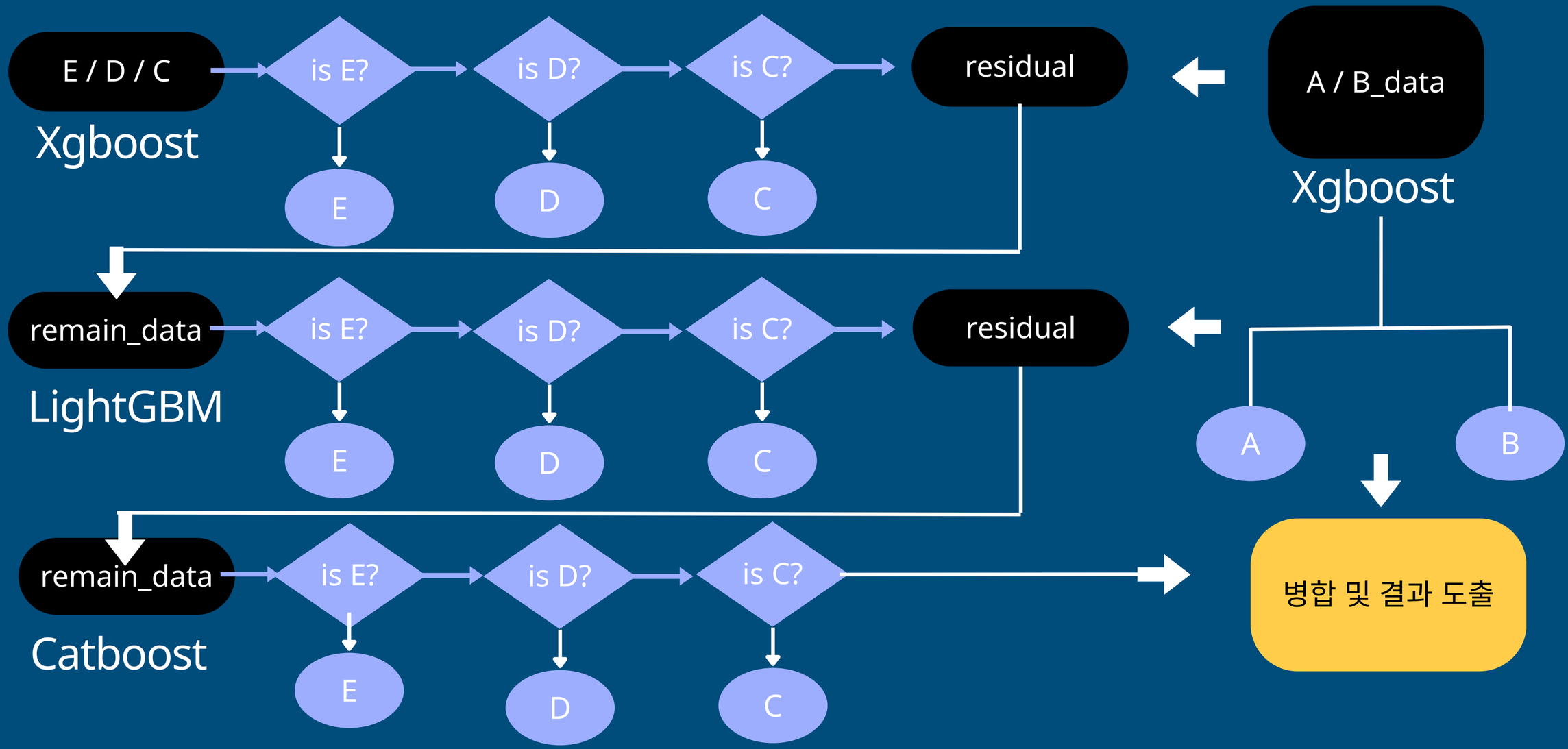
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
24
25
26

4. 모델링

모델링

다중 이진 분류 모델 (C,D,E 분류)

A / B 이진 분류 모델



모델링

web programming team project

XGBoost	과적합을 방지하기 위한 정규화 기법과 병렬 처리를 지원하며, 각 단계에서 가장 큰 손실 감소를 보이는 노드를 깊이 우선 방식으로 분할
CatBoost	범주형 데이터를 자동으로 처리하여 전처리 비용을 줄이고 빠른 학습 속도를 제공하며, 순열 기반 처리로 과적합을 줄이는 Category Boosting 방식
LightGBM	Gradient 기반의 Leaf-wise 분할 방식을 통해 손실 감소가 큰 리프 노드를 우선적으로 분할함으로써 학습 속도는 물론 예측 성능도 향상시키며, 히스토그램 기반 처리로 메모리 사용함을 최소화하여 대규모 데이터셋에 효과적

성능 기준

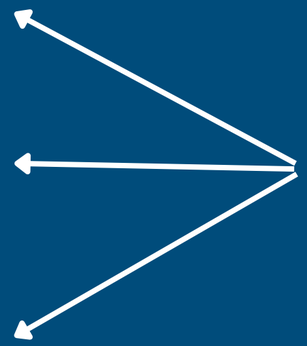
web programming team project

정확도를 높히기 위한 Prec_TGT 값 조정 반복

모델	E_Precision	D_Precision	C_Precision
XGBoost	0.998	0.98	0.98
LightGBM	0.95	0.95	0.95

모델	E_Precision	D_Precision	C_Precision
XGBoost	0.9	0.85	0.83
LightGBM	0.9	0.9	0.9

Stage 1	매우 엄격한 기준 설정→ 초기 검증 단계에서의 정확성 극대화 목적
Stage 2	기준 완화→ 현실 적용 가능성과 일반화 성능 고려
Stage 3	남은 클래스의 분류를 위해 F1 최대화



각 Stage 마다 AB분류 모델이 병렬적으로 실행하여 결과 덮어쓰기

최종 다중 이진분류 모델 + A,B클래스 전용 모델 앙상블

기존의 다중 이진분류 모델에 A,B클래스만 분류하는 모델을 추가하여 낮았던 A,B정확도를 끌어 올림. 설계한 파이프라인: 3단계 Cascade 구조 + A/B 오버샘플링을 통한 전용 모델

1단계: XGBoost 기반 1차 Cascade (E → D → C)

(1) 이전과 동일, but 여기서는 A,B자체를 분류하지 않음.

2단계: LightGBM 기반 2차 Cascade

(1) 개선 포인트

1차보다 더 많은 Positive 샘플 확보 가능

recall 회복

(2) 특징

다양한 이전 단계 정보를 활용한 최종 필터

극단적으로 적은 샘플(A/B 포함)도 적절히 분류

3단계: CatBoost 기반 최종 전환기 (잔여 샘플 처리)

대상: 1, 2차 모두 분류되지 않은 샘플 (약 3~5%)

입력: 기존 피처 + Stage1/Stage2 예측 확률값 10개

3단계: CatBoost 기반 최종 전환기 (잔여 샘플 처리)

대상: 1, 2차 모두 분류되지 않은 샘플 (약 3~5%)

입력: 기존 피처 + Stage1/Stage2 예측 확률값 10개

모델

CatBoostClassifier (GPU 사용)

각 클래스별 Binary 분류, Precision 기반 cutoff

fallback 기준은 F1-score

A/B 전용 모델 (XGBoost + SMOTE + 결측치 처리)

문제: A, B 클래스가 너무 적어 일반 모델로는 절대 학습 불가

해결

(1) A/B 데이터만 따로 추출

(2) 결측치 → SimpleImputer 처리

(4) 소수 클래스 → SMOTE로 증식 (k_neighbors 자동 조정)

(5) F1 기준으로 threshold 설정

결과: A/B 정확도, precision, recall 모두 100%

시행착오

web programming team project

문제점	해결방안
1.데이터 분리 문제	preprocessing.csv에 train과 test가 함께 섞여 있어 Segment.1이 NaN인 샘플만 추출하는 로직 필요
2. 오버샘플링 문제	처음에는 각 단계에서 오버샘플링을 했지만, 데이터 증식이 너무 커져 → **전체 데이터 그대로 사용**으로 전략 수정
3. Dummy Model 문제	특정 클래스에 샘플이 없으면 DummyModel이 생성됨 → joblib.load() 시 오류 발생하여 sys.modules로 수동 등록해 해결
4. 모델 전이 오류	1차에서 완벽하게 분류돼서 2차에 샘플이 거의 없음 → Precision cut-off를 낮춰 샘플 잔여 유도
5. SettingWithCopyWarning	df_test에 직접 값 할당 시 pandas의 경고 → .loc[] 방식으로 수정 유도
6. KeyError 문제	'prob1' 컬럼 없음, lbl 변수 미정의 등 변수 스코프 문제 해결
7. 열 길이 불일치	df와 prob 길이 차이로 인한 오류 발생 → df_test를 기준으로 예측 수행하도록 코드 수정

시행착오

web programming team project

시행착오

문제점	해결방안
1.단일 모델로 A/B 예측 문제	A/B 전용 모델 생성
2. ThresholdThreshold가 너무 높아 Recall = 0 되는 문제	Precision 목표치를 완화, fallback 사용
3.LightGBM 학습 중 컬럼명 오류 발생	특수문자 제거 전처리 추가
4. 너무 보수적으로 하면 대부분 E로 예측됨	다단계 구조 도입 + 잔여 샘플 처리 로직 개발
5.CatBoost GPU 미지원 환경 문제	Precision 목표치를 완화, fallback 사용

최종 성능 요약

web programming team project

```
============================================================
📊 모델 성능 요약
================================+===========================

1️⃣ 1차 Cascade (XGBoost)
        TP      FP   FN       TN  precision  recall      F1
E  56505.0  4253.0  0.0  11242.0     0.9300     1.0  0.9637
D   7473.0   830.0  0.0   2939.0     0.9000     1.0  0.9474
C   2831.0   107.0  0.0      1.0     0.9636     1.0  0.9815

2️⃣ 2차 Cascade (LightGBM)

3️⃣ A, B 전용 모델
Precision: 1.0000
Recall: 1.0000
F1: 1.0000

4️⃣ 최종 통합 성능
전체 정확도: 0.9283
클래스별 정확도: {'A': 1.0, 'B': 1.0, 'C': 0.7543298694377831, 'D': 0.7276533592989289, 'E': 1.0}
```

전체 정확도: 92.8%

- A, B 극소수 클래스 예측 정확도 100%
- 미분류 샘플 없음
- precision 기준 threshold를 통해 신뢰성 높은 분류 가능

한계점 및 개선 여지

web programming team project

1. 데이터 분할 방식의 구조적 한계

- **Cascade 구조에서는 이전 단계에서 False Negative로 잘못 걸러지면 다음 단계에서 복구 불가능**
- **예: D 고객이 Stage1에서 not-D로 분류되면, 이후에는 어떤 Stage에서도 D로 예측될 기회가 없음**

2. 개선 방향

1. 샘플링 전략 재설계
   - Stage마다 Drop 대신 Soft Voting 방식으로 잔여 샘플의 확률 누적 고려
   - 또는 전 단계를 통합한 meta-ensemble 구조 도입 (e.g., stacking with all outputs)
2. 통합 시 클래스 간 확률 비교 불가 문제 해결
   - A/B도 cascade 구조에 참여시키되, 전용 모델의 확률을 후속 모델에서 feature로 활용
3. Feature Engineering 개선
   - → 업종별 이용 패턴, 고객 행동군 유사도 등 새로운 feature 생성
   - → 혹은 B 고객의 전환 유인 요소 등 외부 도메인 지식 도입 필요

```
💾 모델 · threshold · OOF 확률 저장 완료

=== Stage-wise Confusion / Metrics ===
         TP     FP   FN       TN  precision  recall      F1
E  56505.0  570.0  0.0  14925.0     0.9900     1.0  0.9950
D  10232.0  316.0  0.0   4377.0     0.9700     1.0  0.9848
C   3637.0  191.0  0.0    549.0     0.9501     1.0  0.9744
A      0.0    0.0  0.0    549.0     0.0000     0.0  0.0000
B      0.0    0.0  0.0    549.0     0.0000     0.0  0.0000
```

web programming team project

팀 내 지식 공유 & 협업 방식

프로젝트 진행 중 발생한 이슈나 모델 개선 아이디어에 대해 팀원들과지속적으로 **노션**을 통해 기록하고 공유함.
또한, 외부 멘토링을 받은 내용이나 각자 서치한 자료들도 노션에 정리하여, 피드백을 빠르게 반영하고 전체적인 이해도 향상에 기여함.

i. “기술적인 시행착오나 의문점은 노션에 문서화하여 팀 전원이 쉽게 이해하고 접근성 편의 제공 ”
ii. “단순 분업이 아닌, 서로의 내용을 함께 리뷰하고 성장하는 팀 환경을 만들고자 노력”

(1) Notion 기반 기록 공유

- 전처리 이슈 해결법 공유 (ex: 날짜형 NaN 변환법 등)
- 캐스케이드 모델 구조 이해를 위한 자료 축적
- 각자의 실험 결과 및 그래프 캡처 공유

(2) 외부 멘토링 피드백 적용

- 모델별 precision threshold 조정
- 이진 분류 모델에 대한 파이프라인 대폭 조정

(3) 회고 및 개선 논의

- 문제 발생 시 실시간 공유 -> 수정사항 적용 -> 재실험 후 리뷰

(4) 우리가 배운 점

다단계 구조가 실제 현업 문제에 유용하게 적용될 수 있다는 점

데이터 불균형 문제는 모델 구조 자체로 해결 가능

Precision 목표치를 조정하는 것이 전체 성능에 큰 영향을 준다는 점

성능을 맞추기 위해선 “모두를 맞히려 하기보다, 틀릴 가능성을 줄이는 설계”가 더 현실적이라는 점

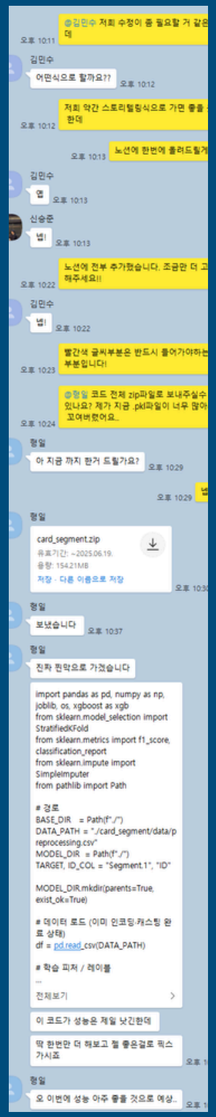
(5) 결론

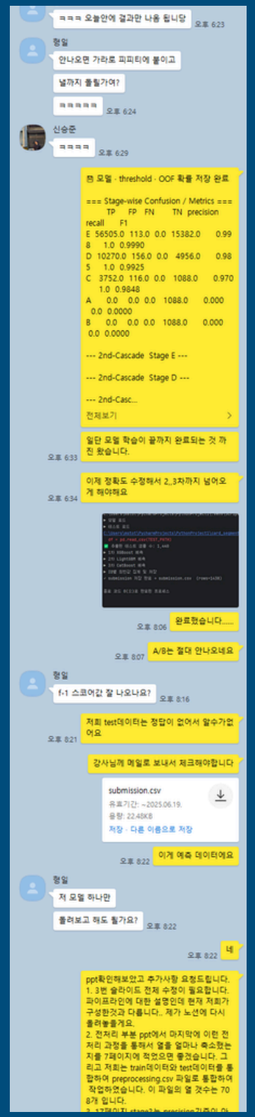
단순한 예측 모델이 아니라, 설계 철학과 실험 결과를 바탕으로 점진적으로 정제된 모델이 만들어짐.

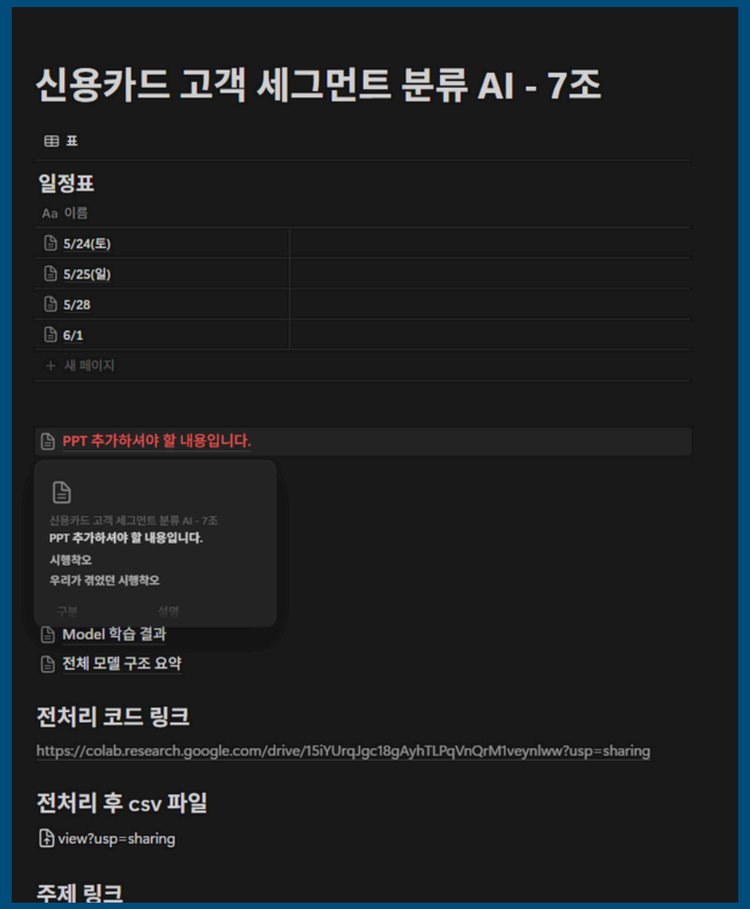
이 구조는 향후 고객 등급 외에 불균형 다중 분류 문제에도 매우 유용하게 확장될 수 있음.

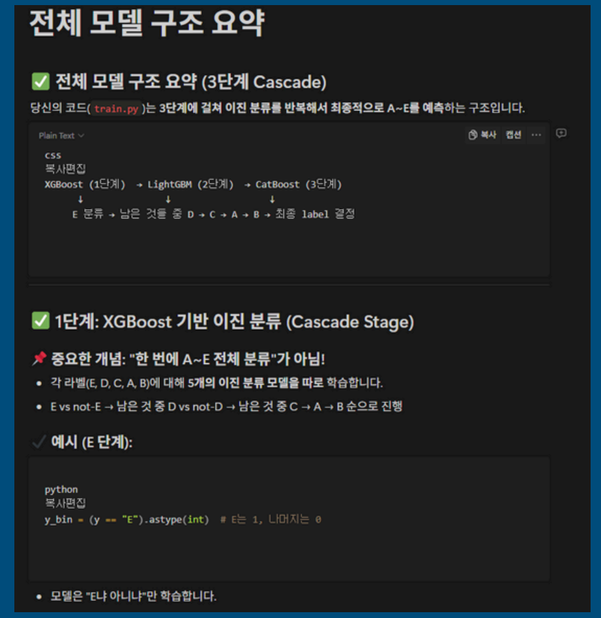
협업

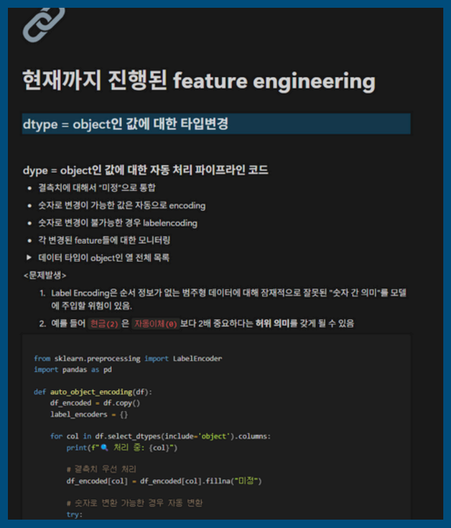
노션과 메신저 등을 이용한 협업 기록











감사합니다

web programming team project

Q&A

자유롭게 질문해주세요!