신용카드 고객 세그먼트 분류

AI 경진대회

승리를 위하여

00.
목차

2018.07 ~ 2018.12 고객별 금융활동 데이터

train data

(2400000, 872)

회원 정보 + Segment : 78개 칼럼
신용 정보 : 42개 칼럼
승인매출 정보 : 406개 칼럼
청구입금 정보 : 46개 칼럼
잔액 정보 : 82개 칼럼
채널 정보 : 105개 칼럼
마케팅 정보 : 64개 칼럼
성과 정보 : 49개 칼럼

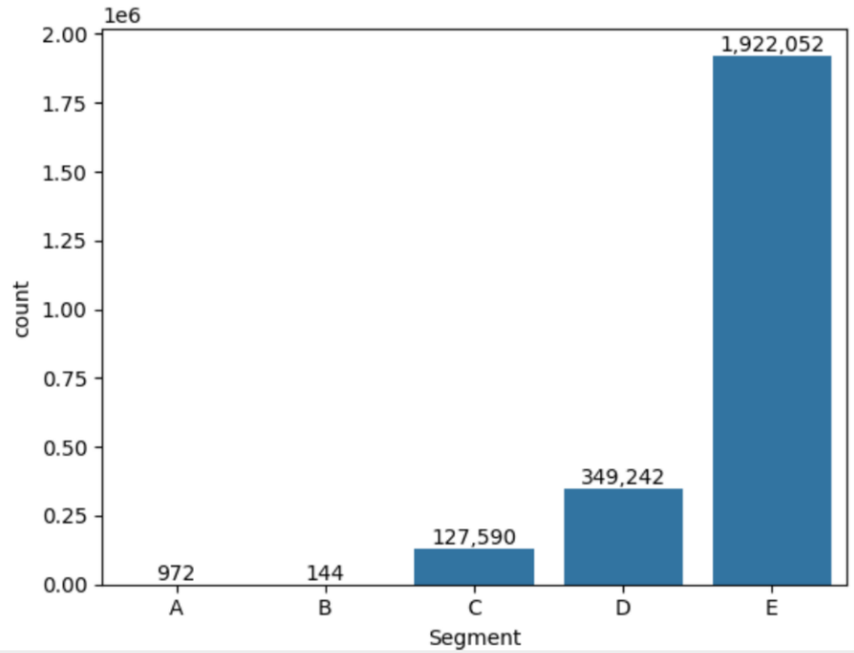
test data

(600000, 871)

회원 정보 : 78개 칼럼
신용 정보 : 42개 칼럼
승인매출 정보 : 406개 칼럼
청구입금 정보 : 46개 칼럼
잔액 정보 : 82개 칼럼
채널 정보 : 105개 칼럼
마케팅 정보 : 64개 칼럼
성과 정보 : 49개 칼럼

EDA 및 전처리

Segment 비율



E가 약 80%를 차지하고, A/B는 매우 적은
불균형 데이터

결측치 처리

- 여러 변수에 결측치 존재
- 결측치가 너무 많은 변수는 삭제
- 변수 특성에 따라 '기타'로 분류하거나 '중앙값'으로 대체하는 등 다르게 처리

자료형 변환

문자열 변수에 대해 숫자로 인코딩해서 변환

ex) 한도심사요청건수 : '0회' → 0, '1회 이상' → 1

ex) 카드론동의여부: 'Y' → 1, 'N' →0

02.

EDA 및 전처리

파생변수 생성

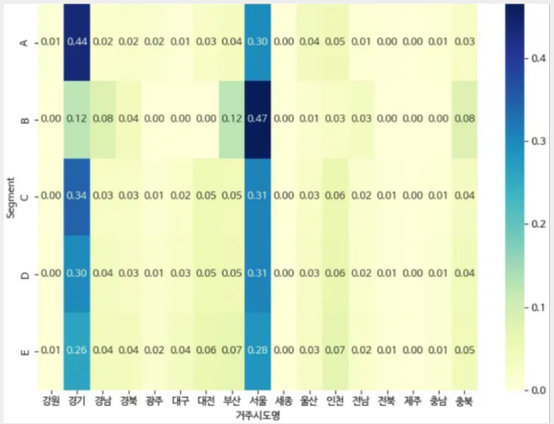
ex) '청구서수령방법'이 '당사멤버십'인 고객의 등급은 C,D,E만 존재

→ 당사멤버십 여부 (0,1)로 파생변수 생성

변수 삭제

- 파생변수에 대한 원천변수 삭제
- 모든 값이 동일한 변수 삭제

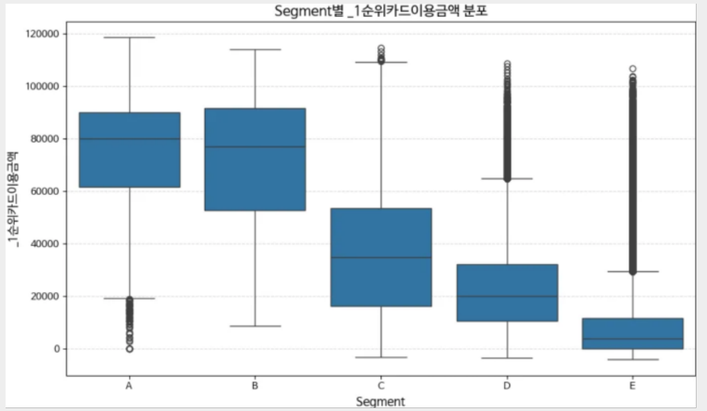
상관관계 분석



상관계수 계산
box plot, heatmap 등

↓

Segment와 다른 변수 간의
상관관계 파악



모델 선정

GPU 지원이 가능하고 대용량 데이터를 효율적으로 처리할 수 있는 모델을 선정하였고, Google Colab T4 GPU를 이용하여 모델링 진행

XGBoost	과적합을 방지하기 위한 정규화 기법과 병렬 처리를 지원하며, 각 단계에서 가장 큰 손실 감소를 보이는 노드를 깊이 우선(Depth-wise) 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 사용량을 최소화하여 대규모 데이터셋에 효과적

1) Feature Importance - 상위 변수 추출

문제	전체 변수 수가 너무 많아 학습 속도 저하 및 과적합 우려
방법	XGBoost / CatBoost / LGBM 로 각각 feature importance 산출 - `feature_importances_` 기반 중요도 기준 상위 100개 / 200개 / 300개 변수 조합별 모델 성능 비교

```
importance_df = pd.DataFrame({
    'feature': X.columns,
    'importance': temp_model.feature_importances_
}).sort_values(by='importance', ascending=False)
```

2) Oversampling - 클래스 불균형 보정

문제	A/B 클래스 데이터 부족 → 모델 성능 저하, 다수 클래스에 편향 발생
방법	SMOTE + 클래스 가중치 적용 - 다양한 비율의 A/B/C 클래스 오버샘플링 조합 실험 - Train set을 8:2 비율로 검증용 데이터 분리 - 학습 시 균형잡힌 클래스 가중치 적용: `class_weight = 'balanced'` - mlogloss, macro F1-score 기준 비교

```
smote = SMOTE(sampling_strategy={0: 30000, 1: 30000, 2: 250000}
              , random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_top300, y)
```

```
# 클래스 weight 계산
classes = np.unique(y_resampled)
weights = compute_class_weight(class_weight='balanced',
                               classes=classes,
                               y=y_resampled)
class_weights = dict(zip(classes, weights))
sample_weights = pd.Series(y_resampled).map(class_weights)
```

3) Hyperparameter tuning

모델 성능 극대화를 위한 파라미터 최적화 수행

모델	튜닝 시도한 파라미터 종류
XGBoost	n_estimators, learning_rate, max_depth, min_child_weight, subsample, colsample_bytree, gamma, reg_alpha, reg_lambda
CatBoost	learning_rate, depth, l2_leaf_reg, bagging_temperature, random_strength, border_count
LightGBM	num_boost_round, learning_rate, num_leaves, max_depth, min_data_in_leaf, bagging_fraction, bagging_freq, feature_fraction, lambda_l1, lambda_l2, min_gain_to_split

4) Soft voting - 단일 모델 대비 안정적이고 높은 성능 확보 시도

- 각 모델에서 predict_proba()로 클래스별 예측 확률 추출
- Soft Voting 방식으로 모델별 확률을 가중 평균: 가중치 비율 다양한 조합 실험
- np.argmax()를 통해 가장 높은 확률의 클래스 선택

최종 모델 선정

Soft voting: XGBoost 40% + LightGBM 30% + CatBoost 30%

XGBoost 기준 중요도 상위 300개 변수 및 GPU 사용

XGBoost

- 오버샘플링
  : A 3만, B 3만, C 25만
- n_estimators = 5000,
  objective = 'multi:softprob',
  num_class = 5,
  eval_metric = 'mlogloss'

LightGBM

- 오버샘플링
  : A 3만, B 3만, C 25만
- objective='multiclass',
  metric = 'multi_logloss',
  num_class = 5
- 파라미터 튜닝 결과 사용

  num_boost_round = 1500,
  learning_rate = 0.16304239034369372,
  num_leaves = 260, max_depth = 6,
  min_data_in_leaf = 1200,
  bagging_fraction = 0.8999999999999999,
  bagging_freq = 4, feature_fraction = 0.8,
  lambda_l1 = 4.4610495762623494e-05,
  lambda_l2 = 0.6898684039866835,
  min_gain_to_split = 0.3944497642389666

CatBoost

- 오버샘플링
  : A 6만, B 12만, C 24만
- iterations = 5000