Home Credit - Credit Risk Model Stability

요 약

신용 기록이 부족한 고객에게 대출 서비스를 제공할 기회를 확대하기 위해, 대출 상환 불이행 가능성을 예측하는 모델을 개발하는 것을 목표로 한다. 신용 평가 데이터셋을 활용하여, LightGBM과 CatBoost 모델을 사용한 앙상블 기법을 통해 예측 성능을 극대화하였고, 데이터 전처리 과정에서는 다양한 변환 및 필터링 기법을 적용하여 데이터의 품질을 개선하였으며, Pipeline 클래스를 활용하여 데이터 유형 변환과 날짜 처리 등을 자동화하였다. 주요 특징 추출 과정에서는 수치형, 날짜형, 문자열형 데이터를 효과적으로 처리하기 위해 Aggregator 클래스를 사용하였다. 이를 통해 상위 약 15%의 순위를 기록할 수 있었다.

1. 서론

전통적으로 신용 기록이 거의 없는 사람들은 대출을 거부 받을 가능성이 높다는 문제점이 존재한다. 소비자에게 금융 서비스를 제공하는 기업 입장에서는, 신용 기록이 거의 없는 고객에게 대출 서비스를 제공하는 것이 위험을 부담하는 행위이기 때문이다. 따라서 이번 프로젝트를 통해 어떠한 고객이 대출 상환을 불이행할 가능성이 높은 지 예측함으로써 신용기록이 부족한 사람들에게도 대출 서비스를 제공할 기회를 찾아주는 것이 목표이다.

2. 본론

2.1 데이터 분석

데이터 셋의 주요 열을 살펴보면 다음과 같다:

case_id: 각 신용 케이스에 대한 고유 식별자이다. 관련 테이블을 기본 테이블과 결합할 때 이 ID가 필요하다.

date_decision: 대출 승인 여부에 대한 결정이 내려진 날짜를 나타낸다.

WEEK_NUM: 집계를 위해 사용되는 주 번호이다. 테스트 샘플에서는 WEEK_NUM이 훈련 데이터의 마지막 WEEK_NUM 값부터 연속적으로 이어진다.

MONTH: 집계 목적으로 사용되는 월을 나타내는 열이다.

target: 특정 신용 케이스(대출)에 대해 일정 기간 후 고객이 연체했는지 여부에 따라 결정되는 목표 값이다.

num_group1: depth=1 및 depth=2 테이블에서 case_id의 과거 기록을 위한 인덱싱 열이다.

num_group2: depth=2 테이블의 case_id 과거 기록을 위한 두 번째 인덱싱 열이다. num_group1과 num_group2의 순서는 중요하며, 특징 정의에서 명확히 설명된다.

데이터 셋은 유사한 그룹에 대해 다음과 같은 표기법을 사용한다:

P: DPD(Days past due) 변환

M: 카테고리 마스킹

A: 금액 변환

D: 날짜 변환

T: 미정의 변환

L: 미정의 변환

이러한 표기법을 통해 데이터의 각 열이 어떤 종류의 변환이나 처리를 필요로 하는지 쉽게 파악할 수 있다.

2.2 데이터 전처리

[Pipeline 클래스]

데이터 전처리를 수행

set_table_dtypes(df)

이 함수는 데이터프레임(df)의 각 열을 순회하면서 특정 조건에 따라 데이터 유형을 변환한다.

case_id, WEEK_NUM, num_group1, num_group2 열의 경우: Int64로 변환한다.

date_decision 열의 경우: Date 타입으로 변환한다.

열 이름의 마지막 문자가 "P" 또는 "A"인 경우: Float64로 변환한다.

열 이름의 마지막 문자가 "M"인 경우: String으로 변환한다.

열 이름의 마지막 문자가 "D"인 경우: Date 타입으로 변환한다.

handle_dates(df)

이 함수는 데이터프레임의 날짜 열을 처리한다.

각 열을 순회하며, 열 이름의 마지막 문자가 "D"인 경우 다음 작업을 수행한다:

현재 열의 날짜 값을 "date_decision" 열의 날짜 값과 뺀다.

두 날짜 사이의 총 일수를 계산한다.

처리 후, "date_decision"과 "MONTH" 열을 데이터프레임에서 제거한다.

filter_cols(df)

이 함수는 특정 조건에 따라 열을 필터링하여 제거한다.

각 열을 순회하며, "target", "case_id", "WEEK_NUM" 열을 제외한 나머지 열에 대해 결측치 비율을 계산한다.

결측치 비율이 70%를 초과하는 열을 삭제한다.

다시 각 열을 순회하며, "target", "case_id", "WEEK_NUM" 열을 제외한 나머지 열 중 문자열 타입의 열에 대해 고유 값의 수를 계산한다.

고유 값의 수가 1이거나 200을 초과하는 열을 삭제한다.

[Aggregator 클래스]

데이터프레임에서 특정 유형의 특징을 집계

num_expr(df)

이 함수는 데이터프레임에서 수치형 특징을 추출한다.

이름이 "P" 또는 "A"로 끝나는 열을 선택한다. 이는 특정 수치형 측정을 나타낸다.

각 선택된 열에 대해 다음과 같은 표현식을 생성하고

이름을 붙인다:

최대 값 (max)

마지막 값 (last)

평균 값 (mean)

date_expr(df)

이 함수는 데이터프레임에서 날짜 관련 특징을 추출한다.

이름이 "D"로 끝나는 열을 선택한다. 이는 날짜 열을 나타낸다.

각 선택된 열에 대해 다음과 같은 표현식을 생성하고 이름을 붙인다:

최대 값 (max)

마지막 값 (last)

평균 값 (mean)

str_expr(df)

이 함수는 데이터프레임에서 문자열 특징을 추출한다.

이름이 "M"으로 끝나는 열을 선택한다. 이는 문자열 타입 열을 나타낸다.

각 선택된 열에 대해 다음과 같은 표현식을 생성하고 이름을 붙인다:

최대 값 (max)

마지막 값 (last)

other_expr(df)

이 함수는 데이터프레임에서 기타 특징을 추출한다.

이름이 "T" 또는 "L"로 끝나는 열을 선택한다.

각 선택된 열에 대해 다음과 같은 표현식을 생성하고 이름을 붙인다:

최대 값 (max)

마지막 값 (last)

count_expr(df)

이 함수는 데이터프레임에서 카운트 관련 특징을 추출한다.

열 이름에 "num_group"이 포함된 열을 선택한다.

각 선택된 열에 대해 다음과 같은 표현식을 생성하고 이름을 붙인다:

최대 값 (max)

마지막 값 (last)

get_exprs(df)

이 함수는 이전 함수들로부터 받은 모든 표현식을 집계하여 종합적인 특징 추출 표현식 리스트를 만든다.

모든 개별 특징 추출 메서드를 호출하고, 결과 리스트를 연결한다.

모든 유형의 특징에 대한 종합적인 표현식 리스트를 반환한다.

[기타 처리 함수]

feature_eng(df_base, depth_0, depth_1, depth_2)

이 함수는 기본 데이터프레임(df_base)과 여러 추가 데이터프레임(depth_0, depth_1, depth_2) 세트를 대상으로 피처 엔지니어링을 수행한다. 데이터프레임은 세 가지 깊이로 분류된다. 각 깊이는 데이터의 성격과 연결된 과거 기록의 깊이를 나타낸다.

depth=0: 특정 case_id에 직접적으로 연결된 정적인 특징을 포함한다. 예를 들어, 기본적인 개인 정보나 대출 신청 당시의 정보를 포함한다.

depth=1: 각 case_id에 대해 추가적인 과거 기록을 포함한다. 이 기록은 num_group1을 통해 인덱싱된다. 예를 들어, 이전 대출 기록이나 신용 보고서에서 수집된 데이터를 포함한다.

depth=2: 더욱 깊은 과거 기록을 포함하며, num_group1과 num_group2를 통해 인덱싱된다. 예를 들어, 장기간에 걸친 상세한 거래 기록이나 여러 신용 보고서 기관에서 수집된 데이터를 포함한다.

이러한 depth 구조는 데이터의 다양한 층위를 효과적으로 조직화하고 통합하여, 피처 엔지니어링 및 분석에 유용하게 사용된다.

따라서 이 함수는 새로운 특징을 추가하고, 추가 데이터프레임을 결합하며, Pipeline 클래스를 사용해 날짜를 처리한다.

입력 (Inputs):

df_base: 피처 엔지니어링을 수행할 기본 데이터프레임.

depth_0, depth_1, depth_2: 추가 특징을 나타내는 여러 깊이의 데이터프레임 목록.

출력 (Output):

피처 엔지니어링이 완료된 데이터프레임을 반환한다.

프로세스 (Process):

month_decision: "date_decision" 열에서 월을 추출하여 새로운 열로 추가한다.

weekday_decision: "date_decision" 열에서 요일을 추출하여 새로운 열로 추가한다.

[추가 데이터프레임을 기본 데이터프레임에 결합]

각 추가 데이터프레임(depth_0, depth_1, depth_2)을 순회하며, "case_id" 열을 키로 사용해 왼쪽 조인 방식으로 기본 데이터프레임에 결합한다.

서로 다른 특징 세트를 구별하기 위해 열 이름에 접미사를 추가한다.

[Pipeline 클래스를 사용해 날짜 처리]

Pipeline.handle_dates 메서드를 적용해 날짜 처리를 수행한다.

[피처 엔지니어링이 완료된 데이터프레임 반환]

최종적으로 피처 엔지니어링이 완료된 데이터프레임을 반환한다.

이름 목록을 반환한다.

to_pandas(df_data, cat_cols=None)

이 함수는 데이터프레임을 Pandas 데이터프레임으로 변환하고, 선택적으로 지정된 열을 범주형 데이터 타입으로 변환한다.

입력 (Inputs):

df_data: Pandas로 변환할 입력 데이터프레임.

cat_cols: 선택적 열 이름 목록으로, 범주형 데이터 타입으로 변환할 열들이다. 기본값은 None이다.

출력 (Output):

변환된 Pandas 데이터프레임과 범주형 열 이름 목록을 반환한다.

프로세스 (Process):

Pandas 데이터프레임으로 변환:

입력 데이터프레임을 .to_pandas() 메서드를 사용해 Pandas 데이터프레임으로 변환한다.

범주형 열 선택:

cat_cols가 제공되지 않은 경우, 데이터 타입이 "object"인 열을 기본 범주형 열로 선택한다.

범주형 데이터 타입으로 변환:

선택된 범주형 열을 .astype("category") 메서드를 사용해 범주형 데이터 타입으로 변환한다.

변환된 데이터프레임과 범주형 열 목록 반환:

최종적으로 변환된 Pandas 데이터프레임과 범주형 열

reduce_mem_usage(df)

이 함수는 데이터프레임의 모든 열을 순회하며 데이터 유형을 수정하여 메모리 사용량을 줄이는 기능을 수행한다.

입력 (Input):

df: 입력 데이터프레임.

출력 (Output):

메모리 사용량이 줄어든 데이터프레임을 반환한다.

프로세스 (Process):

초기 메모리 사용량 계산

df.memory_usage()를 사용하여 데이터프레임의 초기 메모리 사용량을 계산하고, 이를 start_mem 변수에 저장한다.

각 열을 순회하면서 데이터 유형 수정

열의 데이터 유형이 category인 경우 해당 열을 건너뛴다.

category가 아닌 열에 대해,

열의 최소값(c_min)과 최대값(c_max)을 결정한다.

열의 데이터 유형이 정수형(int)인 경우,

데이터가 int8, int16, int32, int64에 적합한지 확인하고, 적합한 데이터 유형으로 변환한다.

열의 데이터 유형이 부동 소수점형(float)인 경우,

데이터가 float16, float32, float64에 적합한지 확인하고,

적합한 데이터 유형으로 변환한다.

열의 데이터 유형이 object(문자열)인 경우 변환을 건너뛴다.

최종 메모리 사용량 계산:

수정 후 데이터프레임의 최종 메모리 사용량을 계산하고, 이를 end_mem 변수에 저장한다.

메모리 사용량 감소 출력:

초기 메모리 사용량과 최종 메모리 사용량을 출력하고, 감소 비율을 퍼센트로 계산하여 출력한다.

2.3 구현 방안

LightGBM (Light Gradient Boosting Machine) [1]

모델 설명: LightGBM은 마이크로소프트에서 개발한 그래디언트 부스팅 프레임워크이다. 이 모델은 의사결정 트리 기반의 알고리즘으로, 여러 개의 약한 학습자를 결합하여 강력한 예측 모델을 만드는 부스팅 방법을 사용한다. 주요 특징은 다음과 같다:

리프 중심 트리 성장 방식: 기존의 트리 성장 방식과 달리, 리프 중심으로 트리를 성장시켜 더 균형 잡힌 트리를 생성하고 학습 속도를 향상시킨다. 이는 메모리 사용량을 줄이고 학습 시간을 단축하는 데 큰 도움이 된다.

빠른 학습 속도 및 낮은 메모리 사용량: LightGBM은 매우 빠른 학습 속도와 낮은 메모리 사용량을 자랑한다. 이는 대규모 데이터셋을 처리할 때 특히 유용하다.

자동 처리: 결측값 처리 및 범주형 변수 자동 처리 기능을 제공하여 데이터 전처리를 간소화한다. 이를 통해 데이터 준비 과정이 줄어들어 모델 학습이 더욱 신속하게 이루어진다.

병렬 학습 지원: 다중 스레드 및 다중 GPU를 활용하여 학습 속도를 더욱 높일 수 있다. 이는 모델 학습 시간을 크게 단축시킬 수 있는 장점이다.

선정 이유:

대규모 데이터 처리: 신용 평가 데이터셋이 많은 행과 열을 포함하고 있으며, 특히 많은 수의 범주형 변수를 포함하여 해당 모델이 적합하다고 판단하였다.

빠른 학습 속도: LightGBM은 리프 중심 트리 성장 방식을 사용해 학습 속도가 매우 빠르다. 이는 대규모 데이터를 다루는 데 있어 중요한 장점이라 생각하였다.

효율적인 메모리 사용: LightGBM은 메모리 사용량이 적어 대규모 데이터셋을 처리할 때 유리하다고 판단하였다.

CatBoost (Categorical Boosting) [2]

모델 설명: CatBoost는 Yandex에서 개발한 그래디언트 CatBoost는 Yandex에서 개발한 그래디언트 부스팅 프레임워크로, 특히 범주형 데이터 처리에 강점을 가진다. 주요 특징은 다음과 같다:

범주형 변수 처리: CatBoost는 별도의 인코딩 없이도

범주형 데이터를 효율적으로 처리할 수 있는 알고리즘을 제공한다. 이를 통해 범주형 데이터의 전처리 과정이 단순해지고, 모델의 성능이 향상된다.

순열 기반 부스팅: 오버피팅을 방지하기 위해 순열을 이용한 데이터 샘플링 방식을 사용하여 모델의 일반화 성능을 높인다. 이는 모델이 과적합되지 않도록 도와주며, 안정적인 예측을 가능하게 한다.

높은 예측 성능: CatBoost는 다양한 데이터셋에서 높은 예측 성능을 보이며, 특히 범주형 변수가 많은 데이터셋에서 뛰어난 성능을 발휘한다.

빠른 학습 속도: 효율적인 알고리즘으로 빠른 학습 속도를 제공하며, 대규모 데이터셋에서도 성능을 유지한다. 이는 실시간 응용 프로그램에서도 유용하게 사용할 수 있는 장점이다.

사용의 용이성: 최소한의 데이터 전처리로도 높은 성능을 발휘할 수 있으며, 다양한 파라미터 튜닝 옵션을 제공한다. 이를 통해 사용자는 손쉽게 모델의 성능을 최적화할 수 있다.

선정 이유:

범주형 데이터 처리: 신용 평가 데이터셋은 많은 범주형 변수를 포함하고 있다. CatBoost는 이러한 범주형 변수를 효율적으로 처리하며, 별도의 인코딩 작업 없이도 높은 성능을 낼 수 있다고 판단하였다.

고성능: CatBoost는 여러 벤치마크에서 높은 예측 성능을 보인다. 이는 신용 리스크 예측에서도 유리하다 판단하였다.

오버피팅 방지: 순열 기반의 부스팅 방식을 사용해 오버피팅을 방지한다. 이는 신용 데이터와 같이 복잡한 데이터셋에서 중요한 장점이라 생각하였다.

Voting Model

모델 설명: Voting Model은 여러 개의 기본 모델을 결합해 예측을 수행하는 앙상블 기법이다. 각 모델의 예측 결과를 평균 내거나 다수결 투표 방식으로 최종 예측을 결정한다. 이번 구현에서는 모델을 결합해 예측 결과를 평균 내는 방식을 사용하였다.

적용 방법:

LightGBM 모델과 CatBoost 모델을 각각 5개씩 사용하여 총 10개의 모델을 앙상블로 결합한다.

LightGBM의 경우 Train데이터에 대해 GridSearchCV를 통해 로컬에서 학습한 다음 가장 높은 AUC점수를 가진 파라미터를 사용하였고, 이렇게 훈련된 모델을 총 5개 사용하였다.(파라미터 동일)

```
[LGBMClassifier(colsample_bynode=0.8, colsample_bytree=0.8, device='gpu',
                extra_trees=True, learning_rate=0.05, max_depth=10, metric='auc',
                n_estimators=2000, num_leaves=64, objective='binary',
                random_state=42, reg_alpha=0.1, reg_lambda=10,
                sample_weight='balanced', verbose=-1),
 LGBMClassifier(colsample_bynode=0.8, colsample_bytree=0.8, device='gpu',
                extra_trees=True, learning_rate=0.05, max_depth=10, metric='auc',
                n_estimators=2000, num_leaves=64, objective='binary',
                random_state=42, reg_alpha=0.1, reg_lambda=10,
                sample_weight='balanced', verbose=-1),
 LGBMClassifier(colsample_bynode=0.8, colsample_bytree=0.8, device='gpu',
                extra_trees=True, learning_rate=0.05, max_depth=10, metric='auc',
                n_estimators=2000, num_leaves=64, objective='binary',
                random_state=42, reg_alpha=0.1, reg_lambda=10,
                sample_weight='balanced', verbose=-1),
 LGBMClassifier(colsample_bynode=0.8, colsample_bytree=0.8, device='gpu',
                extra_trees=True, learning_rate=0.05, max_depth=10, metric='auc',
                n_estimators=2000, num_leaves=64, objective='binary',
                random_state=42, reg_alpha=0.1, reg_lambda=10,
                sample_weight='balanced', verbose=-1),
 LGBMClassifier(colsample_bynode=0.8, colsample_bytree=0.8, device='gpu',
                extra_trees=True, learning_rate=0.05, max_depth=10, metric='auc',
                n_estimators=2000, num_leaves=64, objective='binary',
                random_state=42, reg_alpha=0.1, reg_lambda=10,
                sample_weight='balanced', verbose=-1),
```

Catboost 또한 LightGBM과 동일.

```
{'nan_mode': 'Min', 'gpu_ram_part': 0.95, 'eval_metric': 'AUC', 'combinations_ctr': ['Borders:CtrBorderCount=15:CtrBorderType=Uniform:TargetBorderCount=1:TargetBorderT
ype=MinEntropy:Prior=0/1:Prior=0.5/1:Prior=1/1', 'FeatureFreq:CtrBorderCount=15:CtrBorderType=Median:Prior=0/1'], 'iterations': 6000, 'fold_permutation_block': 64, 'le
af_estimation_method': 'Newton', 'observations_to_bootstrap': 'TestOnly', 'random_score_type': 'NormalWithModelSizeDecrease', 'counter_calc_method': 'SkipTest', 'grow_
policy': 'SymmetricTree', 'penalties_coefficient': 1, 'boosting_type': 'Plain', 'ctr_history_unit': 'Sample', 'feature_border_type': 'GreedyLogSum', 'bayesian_matrix_r
eg': 0.10000000149011612, 'one_hot_max_size': 2, 'devices': '-1', 'eval_fraction': 0, 'pinned_memory_bytes': '104857600', 'force_unit_auto_pair_weights': False, 'l2_le
af_reg': 3, 'random_strength': 1, 'rsm': 1, 'boost_from_average': False, 'gpu_cat_features_storage': 'GpuRam', 'fold_size_loss_normalization': False, 'max_ctr_complexi
ty': 4, 'model_size_reg': 0.5, 'simple_ctr': ['Borders:CtrBorderCount=15:CtrBorderType=Uniform:TargetBorderCount=1:TargetBorderType=MinEntropy:Prior=0/1:Prior=0.5/1:Pr
ior=1/1', 'FeatureFreq:CtrBorderCount=15:CtrBorderType=MinEntropy:Prior=0/1'], 'pool_metainfo_options': {'tags': {}}, 'use_best_model': True, 'meta_l2_frequency': 0,
'class_names': [0, 1], 'random_seed': 0, 'depth': 6, 'ctr_target_border_count': 1, 'has_time': False, 'border_count': 128, 'min_fold_size': 100, 'data_partition': 'Fea
tureParallel', 'bagging_temperature': 1, 'classes_count': 0, 'auto_class_weights': 'None', 'leaf_estimation_backtracking': 'AnyImprovement', 'best_model_min_trees': 1,
'min_data_in_leaf': 1, 'add_ridge_penalty_to_loss_function': False, 'loss_function': 'Logloss', 'learning_rate': 0.029999999329447743, 'meta_l2_exponent': 1, 'score_fu
nction': 'Cosine', 'task_type': 'GPU', 'leaf_estimation_iterations': 10, 'bootstrap_type': 'Bayesian', 'max_leaves': 64, 'permutation_count': 4}
```

세션 제한 시간 때문에 모델을 joblib파일로 만들어 캐글 노트북에 업로드하고 불러오는 방식을 사용

각 모델의 예측 결과를 평균 내어 최종 예측 값을 도출한다.

범주형 데이터 처리 시, CatBoost 모델을 위해 범주형 변수를 문자열로 변환한다.

[sub.csv 생성]

이미 학습 완료된 LightGBM 모델과 CatBoost모델을 사용하여 test데이터 셋에 대해 예측을 수행하고 최종 예측 확률을 sub.csv에 저장한다.

[submission.csv 생성] => sub.csv의 확률 조정

```
df_train['target']=0
df_test['target']=1
df_train=pd.concat([df_train,df_test])
y = df_train["target"]
df_train= df_train.drop(columns=["target",
"case_id", "WEEK_NUM"])
```

위와 같이 설정하고 LightGBM 모델로 train데이터에 대해 학습시킨다.

이후 test 데이터에 대해 아래와 같은 로직으로 코드를 구현한다.

```
threshold = 0.996
correction = 0.05
df_test =
df_test.drop(columns=["WEEK_NUM",'target'])
df_test = df_test.set_index("case_id")

y_pred =
pd.Series(model.predict_proba(df_test)[:,1],
index=df_test.index)
df_subm =
pd.read_csv("/kaggle/working/sub.csv")
df_subm = df_subm.set_index("case_id")

df_subm.loc[y_pred < threshold, 'score'] =
(df_subm.loc[y_pred < threshold, 'score'] *
threshold - correction).clip(0)
```

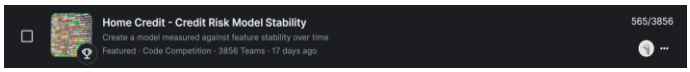
모델이 해당 데이터가 test데이터셋에서 온 것이라고 예측한 확률이 threshold보다 낮으면, 기존 확률*threshold-correction으로 갱신해준다.

이를 통해 데이터 분포의 차이, 즉 훈련 데이터와 테스트 데이터 간의 분포 차이를 고려하여 모델이 예측을 수행할 때, 그 신뢰성을 높인다.

훈련 데이터와 테스트 데이터의 분포 차이가 다를 경우, 데이터의 통계적 특성 또한 다를 수 있기 때문에, 이를

반영하여 예측결과를 조정할 수 있다.

3. 대회 결과



3856팀 중 565등 달성.

4. 참고문헌

[1] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree." Advances in Neural Information Processing Systems (NIPS), 2017.

[2] A. Dorogush, V. Ershov, and A. Gulin, "CatBoost: gradient boosting with categorical features support." arXiv preprint arXiv:1810.11363, 2018.