Sharpe Ratio 극대화를 통한 Lending Club 데이터의 부도 확률 예측

5조 강민채 이선유 정인경

목차

변수 선택

최종적으로 선택된 변수

‘fico_range_low’
‘fico_range_high’
‘loan_amnt’
'installment'
'emp_length'
'home_ownership'
'annual_inc'
‘purpose’
‘dti
’revol_bal’
'revol_util'

1. Lending Club 사이트 및 Lasso 모델 실습 참조

Lending Club의 가입 과정에서 이자율 산출과 Personal Plan 추천을 위해 제공하는 질문 목록과 Lasso 모델 과정에서 사용된 변수들을 참고하여 주관적인 기준으로 변수 선정

2. LightGBM 모델 활용

본격적인 모델 구축 및 학습 전 거의 모든 변수들을 독립변수로 활용해 LightGBM 모델 학습, 그 과정에서 중요도가 높게 나온 변수들을 참고

3. 모델 수정 과정에서의 변수 선택

LightGBM, RandomForest, XGBoost 등의 모델들을 구축해 돌려보며 변수들을 수정하였고, 모든 모델에서 극단적으로 높은 importance를 가지는 변수들(int_rate, last_fico_range, term)에 보수적으로 접근하며 최종 선정

데이터 전처리

결측치 처리

- 데이터의 사이즈(292만 행), 트리 기반 모델의 결측치 민감도, Sharpe Ratio 산식 등을 종합적으로 고려하여 결측치가 포함된 데이터는 삭제

전처리

- loan_status: charged off/default → 1, else → 0
- term: '36 months' → 3, '60 months' → 5로 변환
- emp_length
  - 1~9 years는 문자열 삭제
  - +10 years는 문자열 삭제 후 10으로 변환
  - <1 years는 문자열 삭제 후 0.5로 변환
- purpose, home_ownership: 더미변수로 변환
- revol_util, int_rate: 문자열 삭제

```python
# default(종속변수): charged off/default=1, fully paid=0
df['default'] = df['loan_status'].apply(lambda x: 1 if x in ['Charged Off', 'Default'] else 0)

# default 값의 비율: 부도 케이스는 362981, 부도가 아닌 케이스는 2562512.
df['default'].value_counts()
```

```
default
0    2562512
1     362981
Name: count, dtype: int64
```

```python
# term 전처리:'36 months' → 3, '60 months' → 5로 변환
df['term'] = df['term'].str.extract('(\d+)').astype(float) / 12
```

```python
# emp_length 전처리

def clean_emp_length(val):
    if pd.isnull(val):
        return np.nan
    val = str(val).strip()
    if val == '< 1 year':
        return 0.5
    if '10+' in val:
        return 10
    digits = ''.join(filter(str.isdigit, val))
    return float(digits) if digits else np.nan

# 전처리 적용
df['emp_length'] = df['emp_length'].apply(clean_emp_length)

# 결측치 제거
df = df.dropna(subset=['emp_length'])
```

```python
# revol_util 전처리
df['revol_util'] = df['revol_util'].astype(str).str.strip().str.rstrip('%').replace('n/a', np.nan).astype(float)
df = df.dropna(subset=['revol_util'])
```

```python
# int_rate 전처리
df['int_rate'] = df['int_rate'].astype(str).str.strip().str.rstrip('%').astype(float)
```

```python
# purpose 전처리
df = pd.get_dummies(df, columns=['purpose'], drop_first=True)
```

```python
# home_ownership 전처리
df['home_ownership'] = df['home_ownership'].replace({'NONE': 'OTHER', 'ANY': 'OTHER'})
df = pd.get_dummies(df, columns=['home_ownership'], drop_first=True)
```

모델 선택

왜 트리 모델(LightGBM, RandomForest, XGBoost)을 고려하였는가?

1. Lending Club 데이터의 비선형성
2. 복잡한 구조를 포착하는 트리 모델의 성능
3. 데이터 스케일링 등 전처리에 민감하지 않다는 장점
4. 실제로 트리 모델이 높은 정확도와 안정적인 결과 도출

최종 선택
Random Forest

- 세 모델 중 하이퍼파라미터 튜닝 민감도가 가장 낮다는 장점
- Sharpe Ratio 극대화라는 명확한 목적에 대해, Random Forest 모델이 가장 안정적이고 일관된 성능
- 다만 LightGBM에 비해 학습이 오래 걸린다는 단점

모델 학습

```python
# 1. 10만 행 무작위 샘플링
df_sampled = df.sample(n=100000, random_state=42)

# 2. 독립변수 / 종속변수 분리
X = df_sampled.drop(columns=['default'])
y = df_sampled['default']

# 3. stratified train-test-split (20% test / 20% val / 60% train)
X_temp, X_test, y_temp, y_test = train_test_split(
    X, y, test_size = 0.2,
    random_state=42,
    stratify=y)
X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp, test_size = 0.25,
    random_state=42,
    stratify=y_temp)

# 4. original_data split
original_sample = original_data.loc[df_sampled.index]
original_temp, original_test = train_test_split(original_sample, stratify=y, test_size=0.2, random_state=42)
original_train, original_val = train_test_split(original_temp, stratify=y_temp, test_size=0.25, random_state=42)
```

1. 10만 개 데이터 샘플링

- 292만 개 데이터를 모두 학습시키는 데에 너무 오랜 시간이 소요되는 문제
- 10만 개로 샘플링한 데이터로 기본 모델 구축 후, 292만 개 전체 데이터에 적용하여 일반화

2. Train:Validation:Test=60:20:20

- 데이터 사이즈를 감안할 때, train set에 60%의 데이터를 할당하기에 충분하다고 판단

3. Downsampling

- 부도(default = 1) 데이터와 비부도(default =0) 데이터의 비율이 각각 0.12와 0.88
- 부도 데이터와 비부도 데이터의 비율이 같아지도록 downsampling

모델 학습

```python
# 2. stratified train-test-split (20% test / 20% val / 60% train)
X_temp, X_test, y_temp, y_test = train_test_split(
    X, y, test_size = 0.2,
    random_state=42,
    stratify=y)

X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp, test_size = 0.25,
    random_state=42,
    stratify=y_temp)

# 3. original_data와 동일한 index 기준 split
original_temp = original_data.loc[X_temp.index]
original_test = original_data.loc[X_test.index]
original_train = original_temp.loc[X_train.index]
original_val = original_temp.loc[X_val.index]

# 4. 확인
print("Train class ratio:\n", y_train.value_counts(normalize=True))
print("Validation class ratio:\n", y_val.value_counts(normalize=True))
print("Test class ratio:\n", y_test.value_counts(normalize=True))
```

```
Train class ratio:
 default
0    0.877856
1    0.122144
Name: proportion, dtype: float64
Validation class ratio:
 default
0    0.877855
1    0.122145
Name: proportion, dtype: float64
Test class ratio:
 default
0    0.877857
1    0.122143
Name: proportion, dtype: float64
```

1. 10만 개 데이터 샘플링

- 292만 개 데이터를 모두 학습시키는 데에 너무 오랜 시간이 소요되는 문제
- 10만 개로 샘플링한 데이터로 기본 모델 구축 후, 292만 개 전체 데이터에 적용하여 일반화

2. Train:Validation:Test=60:20:20

- 데이터 사이즈를 감안할 때, train set에 60%의 데이터를 할당하기에 충분하다고 판단

3. Downsampling

- 부도(default = 1) 데이터와 비부도(default =0) 데이터의 비율이 각각 0.12와 0.88
- 부도 데이터와 비부도 데이터의 비율이 같아지도록 downsampling

모델 학습

```python
# Train set downsampling
from sklearn.utils import resample

X_train_balanced = pd.concat([X_train, y_train], axis=1)
default_data = X_train_balanced[X_train_balanced['default']==1]
non_default_data = X_train_balanced[X_train_balanced['default']==0]

non_default_downsampled = resample(
    non_default_data,
    replace=False,
    n_samples = len(default_data),
    random_state=42)

balanced_train_data = pd.concat([default_data, non_default_downsampled])

X_train = balanced_train_data.drop(columns=['default'])
y_train = balanced_train_data['default']
```

1. 10만 개 데이터 샘플링

- 292만 개 데이터를 모두 학습시키는 데에 너무 오랜 시간이 소요되는 문제
- 10만 개로 샘플링한 데이터로 기본 모델 구축 후, 292만 개 전체 데이터에 적용하여 일반화

2. Train:Validation:Test=60:20:20

- 데이터 사이즈를 감안할 때, train set에 60%의 데이터를 할당하기에 충분하다고 판단

3. Downsampling

- 부도(default = 1) 데이터와 비부도(default =0) 데이터의 비율이 각각 0.12와 0.88
- 부도 데이터와 비부도 데이터의 비율이 같아지도록 downsampling

하이퍼파라미터 튜닝

하이퍼파라미터	특성	최적화
n_estimators	생성할 트리의 개수 트리 개수가 많을수록 성능 좋아지지만 학습 시간 증가	[100, 200]
max_depth	트리가 깊으면 과적합 위험 존재 트리 얕으면 덜 복잡하지만 과소적합의 위험 존재	[5, 10]
min_samples_split	노드를 분할하기 위한 최소 샘플 수 너무 작으면 과적합 위험, 너무 크면 트리가 성장하지 못할 위험	[2, 5]
min_samples_leaf	리프 노드(마지막 노드)에 최소한 n개 샘플은 있어야 한다는 제한	[2, 3]
max_features	각 트리 노드를 분할할 때 고려할 feature의 개수 값 작을수록 트리 다양성 증가하지만 성능 저하의 문제 발생	sqrt

모델 학습

```python
# 1. 하이퍼파라미터 정의
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [5, 10],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [2, 3],
    'max_features': ['sqrt'],
    'bootstrap': [True]
}

# threshold를 더 촘촘하게 정의
thresholds = np.linspace(0.05, 1.0, 100)
risk_free_rate = 0.03

# 2. 모든 조합 생성 및 샘플링
all_combinations = list(product(*param_grid.values()))
param_names = list(param_grid.keys())
param_combinations = sample(all_combinations, min(20, len(all_combinations)))
```

train set

```
Best Configuration:
n_estimators: 200
max_depth: 10
min_samples_split: 5
min_samples_leaf: 2
max_features: sqrt
bootstrap: True
threshold: 0.7505050505050505
Best Sharpe Ratio: 1.3394
```

test set

```
Test Set Sharpe Ratio: 1.3387
```

모델 학습

수익률 계산의 문제?

- approved & default=0 → 수익률 r
- not approved → 무위험 수익률 0.03
- approved & default → 수익률 0으로 가정
  - *approved & default의 경우는 손실을 가정하여 마이너스 수익률을 반영할 것인가?*

```python
# Sharpe Ratio 루프 준비
best_sharpe = -np.inf
best_config = {}

# 전체 루프 구조
for combo in param_combinations:
    params = dict(zip(param_names, combo))
    rf = RandomForestClassifier(**params, random_state=42, n_jobs=-1)
    rf.fit(X_train, y_train)
    y_proba_val = rf.predict_proba(X_val)[:, 1]  # 1의 확률만 추출

    for threshold in thresholds:
        predictions = (y_proba_val >= threshold).astype(int)

        approved = (predictions == 0)
        not_approved = (predictions == 1)

        # 여기에서 연환산 수익률 계산
        annualized_return = (
            approved * (y_val == 0) * (
                ((original_val['loan_amnt'] * ((1 + original_val['int_rate'] / 100) ** original_val['term'])) / original_val['loan_amnt'])
                ** (1 / original_val['term']) - 1
            ) +
            not_approved * (
                ((original_val['loan_amnt'] * ((1 + 0.03) ** original_val['term'])) / original_val['loan_amnt'])
                ** (1 / original_val['term']) - 1
            )
        ).dropna()

        mean_ret = annualized_return.mean()
        std_ret = annualized_return.std()
        sharpe = (mean_ret - 0.03) / std_ret if std_ret > 0 else -np.inf

        if sharpe > best_sharpe:
            best_sharpe = sharpe
            best_config = {**params, 'threshold': threshold}
```

모델 학습

수익률 계산의 문제?

- approved & default=0 → 수익률 r
- not approved → 무위험 수익률 0.03
- approved & default → 수익률 0으로 가정
  - *approved & default의 경우는 손실을 가정하여 마이너스 수익률을 반영할 것인가?*

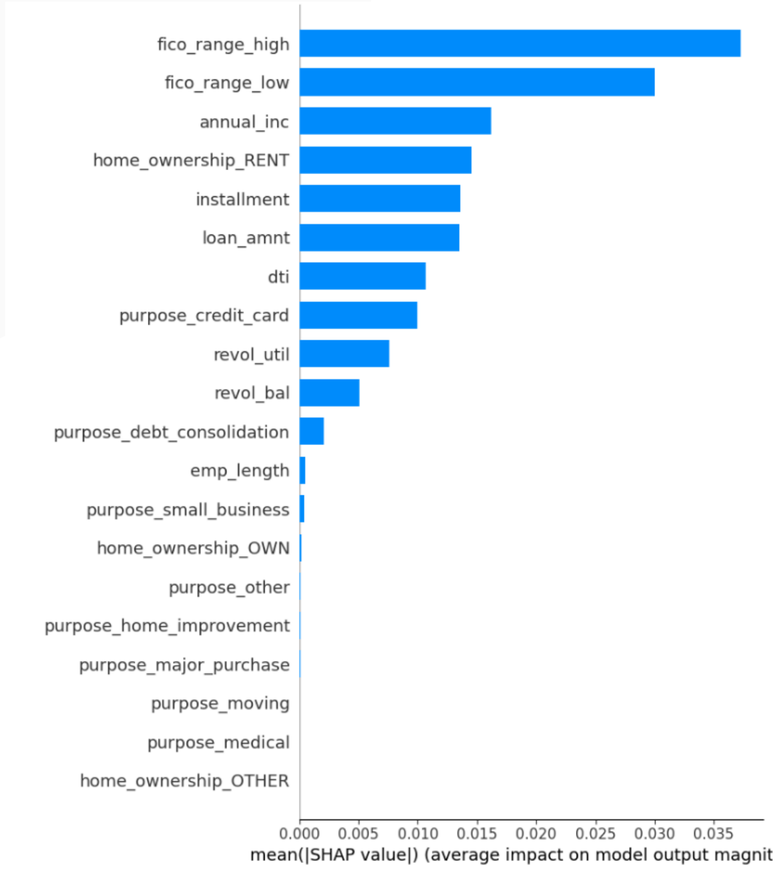
```
# 여기에서 연환산 수익률 계산
annualized_return = (
    approved * (y_val == 0) * (
        ((original_val['loan_amnt'] * ((1 + original_val['int_rate'] / 100) ** original_val['term'])) / original_val['loan_amnt'])
        ** (1 / original_val['term']) - 1
    ) +
    not_approved * (
        ((original_val['loan_amnt'] * ((1 + 0.03) ** original_val['term'])) / original_val['loan_amnt'])
        ** (1 / original_val['term']) - 1
    )
).dropna()

mean_ret = annualized_return.mean()
std_ret = annualized_return.std()
sharpe = (mean_ret - 0.03) / std_ret if std_ret > 0 else -np.inf
```

```
# 여기에서 연환산 수익률 계산
# 1. 승인 + fully paid → 이자 수익률
profit_paid = (y_val == 0) * approved * (
    ((1 + original_val['int_rate'] / 100) ** original_val['term']) ** (1 / original_val['term']) - 1
)

# 2. 승인 + default → 원금 손실 수익률 (-1)
loss_default = (y_val == 1) * approved * (-1)

# 3. 승인 안 된 대출 → 무위험 이자
risk_free = not_approved * (
    ((1 + risk_free_rate) ** original_val['term']) ** (1 / original_val['term']) - 1
)

# 4. 전체 수익률
annualized_return = (profit_paid + loss_default + risk_free).dropna()

mean_ret = annualized_return.mean()
std_ret = annualized_return.std()
sharpe = (mean_ret - 0.03) / std_ret if std_ret > 0 else -np.inf
```

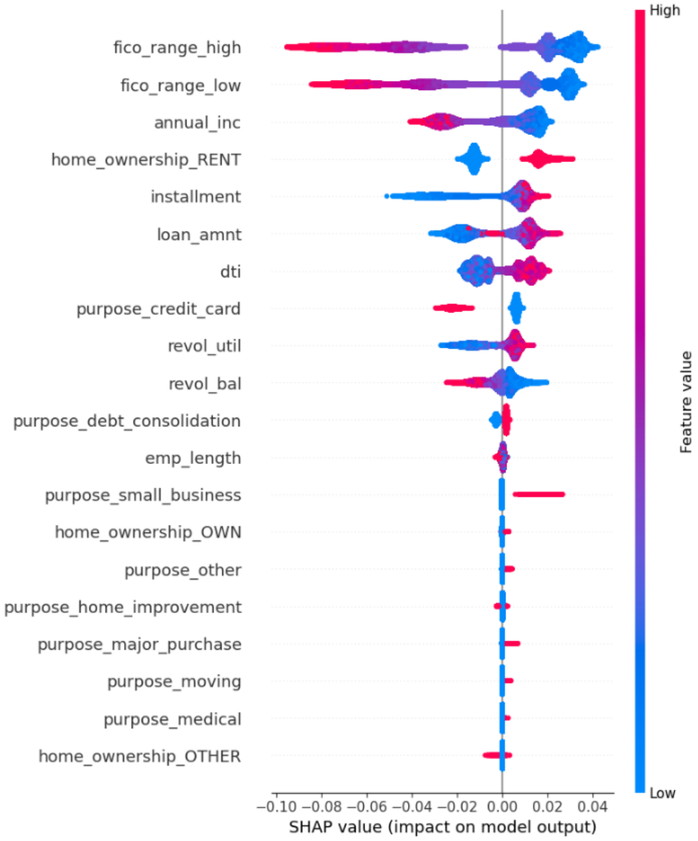
```
Best Configuration:
n_estimators: 200
max_depth: 10
min_samples_split: 2
min_samples_leaf: 2
max_features: sqrt
bootstrap: True
threshold: 0.3
Best Sharpe Ratio: 0.0283
```

결과 및 해석

	mean_abs_shap_value
fico_range_high	0.037283
fico_range_low	0.030028
annual_inc	0.016171
home_ownership_RENT	0.014559
installment	0.013584
loan_amnt	0.013537
dti	0.010705
purpose_credit_card	0.009962
revol_util	0.007629
revol_bal	0.005035
purpose_debt_consolidation	0.002050
emp_length	0.000502
purpose_small_business	0.000431
home_ownership_OWN	0.000202
purpose_other	0.000123
purpose_home_improvement	0.000103
purpose_major_purchase	0.000068
purpose_moving	0.000024
purpose_medical	0.000007
home_ownership_OTHER	0.000004



Sharpely Value



개선할 점 및 질문

변수 선택의 문제

- 독립변수를 선택하는 과정에서 개인의 주관이 얼마나 반영?
- 더 풍부한 독립변수를 선택해야 할 필요성

수익률 계산의 문제

- 대출 승인이 났으나 부도가 났을 경우, 마이너스 손실의 반영?

데이터 사용의 문제

- 하이퍼파라미터 튜닝과 모델 학습에 동일한 train-validation 데이터가 사용되는 경우, 통계적으로 문제가 없는지?