KB 상시감사 인공지능 모델링

프로젝트 개요

인공지능을 활용한 상시감사 지원시스템 구축

- 프로젝트 기간: 2019.11.04 ~ 2020.05.15
- 투입 기간: 2020.02.18 ~ 2020.05.15
- 현업 부서: 영업감사부
- 개발 부서/팀: IT기획부/감사시스템

배경

- 사기대출 관련 영업점 보고가 지속적으로 발생하는 추세
- 사기대출 이상거래로 인한 손실규모는 전체 상각/매각 규모의 약 5%로 추산 (출처: 코리아크레딧뷰. '15 ~ '17년 기준)
- 여신 이상거래를 탐지하는 감사부내 상시감사시스템은 **사후적발 중심, 인력운영 중심**

목표

- 인공지능 기술을 활용한 사기(이상)대출 사전 탐지 및 예방
- 대내외 환경변화에 따른 신종 사기(허위)대출 대응으로 은행의 손실 예방 및 평판리스크 관리
- 점검대상 여신의 위험도 평가 및 자동재학습 기능으로 **감사 효율성 증대 및 통합 점검 관리체계 구축**
- 상시 모니터링의 고도화를 통한 리스크 관리체계 구축

프로젝트 추진 내역

분석 모델링 개요

구분	가계 사기 모형	가계 부실 모형	기업 사기 모형
정의	사기 - 사기로 지정된 건	부실 - 계좌 생성 후 1년 이내 상각된 건	사기 - 사기로 지정된 건
분석대상	신규여신, 창구 접수 등	신규여신, 창구 접수 등	기업신용평가 완료 건 중 사기 건 등
분석기간	3년 (2017 ~ 2019)	2년 (2017 ~ 2018)	사기 4년 (2012 ~ 2015) 정상 3년 (2017 ~ 2019)
분석대상 건수	전체 360216 정상 360001 사기 215 (**0.06%**)	전체 716746 정상 711535 사기 5211 (**0.7%**)	전체 3506 정상 3466 사기 40 (**1.1%**)
테이블 수	12	6	19
컬럼 수	61	60	34
모델	**XGBoost**	**DNN**	**Isolation Forest**
성능지표 AUC	0.999	0.927	0.999
성능지표 AUCPR	0.831	0.422	0.969

분석 배경 및 목적

불균형 데이터 처리

지도학습을 통한 분류 문제에서 불균형 데이터 학습 시 정상 건수 패턴만 집중적으로 학습되어 과적합의 위험이 있기 때문에 불균형 데이터에 대한 처리가 필요함

- 문제: 일반적으로 지도학습 시 기계학습의 성능을 저하시킴
- 평가 기준 변경: 불균형 데이터에 적합한 평가 기준으로 변경
  - 기존: Accuracy
  - 변경: AUCPR, Confusion Matrix, Precision, Recall, F1-Score
- 샘플링: 데이터셋을 변형시켜 전체 클래스의 분포를 균일하게 만드는 방법
  - SMOTE
  - Over-Sampling
  - Under-Sampling
- 모델링: 모델의 파라미터 조정을 통해 데이터 가중치 조절
  - Bootstrap시 소수 관측치들에 더 큰 가중치를 부여하여 샘플 추출 시 많이 선택되게 함
  - XGBoost, RandomForest, DNN
- 아웃라이어 감지, 이상탐지: 비지도 학습으로 새로운 사기 유형 탐지 가능

분석 프로세스

1. 데이터 준비: 데이터 적재 → 품질 검증 → 전처리 → 데이터 결합
2. 변수 선택: EDA(시각화) → 통계적 유의성 검증 → 다중공선성처리 → Random 시뮬레이션 → Bayesian Opt → 입력 데이터셋
3. 모델 개발 및 비교: 다양한 모델 생성
4. 모델 검증 및 선정: 모델 검증 → 최종 모델 선정
5. 통합 테스트: 1차, 2차, 3차 테스트
6. 이행: 운영

로컬 테스트 환경

- **Python3**: Anaconda를 이용해 Python 가상 환경을 설정하고 ML 개발 라이브러리를 설치한다.
- **HDFS**: Docker를 사용해 간단하게 HDFS를 구성한다.

Anaconda

Individual Edition: [Download (https://www.anaconda.com/products/individual)](https://www.anaconda.com/products/individual)

시작

Conda를 이용하면 Python 버전을 자유롭게 선택할 수 있지만,
국민은행 클러스터에서는 리눅스 RPM 패키지로 Python 3.6 버전을 설치했다.

Python 3.6 환경 생성

```
source .bash_profile;
(base) conda create -n kb python=3.6;
```

Python 환경 확인

```
(base) conda activate kb;
(kb) python --version;

# Python 3.6.10 :: Anaconda, Inc.
```

패키지

- nb_conda: Jupyter 브라우저에서 Conda 환경을 사용할 수 있다.
- pyspark: Pyspark 라이브러리 이용
- scikit-optimize
- xgboost
- tensorflow
- shap

설치

```
(kb) conda install -c conda-forge \
     nb_conda nb_conda_kernels pyspark scikit-optimize xgboost tensorflow shap;
```

Jupyter 실행

```
(kb) jupyter notebook
```

디렉터리 구성

- `data` : 원천 데이터 저장
- `lib` : Spark, HDFS Python 라이브러리 파일
- `result` : 데이터 분석 모델 및 결과 데이터 저장
- `result.sample` : 샘플 모델, 결과 데이터
- `images`
- `Analysis.ipynb` : 분석 소스 코드
- `Analysis.html` : 모델 학습 및 예측 과정 정리
- 환경설정`.md` : Anaconda, HDFS 로컬 환경 설정
- `README.md` : 프로젝트 개요

대체 데이터

- 실제 인공지능 모델 개발 시 영업감사부의 개인/기업 데이터 사용
- 이 문서에서는 실제 데이터 대신 Kaggle의 신용카드 사기탐지 데이터 활용
  - 데이터 적재 워크플로우(Sqoop) 생략 가능
  - 데이터 전처리 워크플로우(Hive) 생략 가능
- Kaggle: Credit Card Fraud Detection (https://www.kaggle.com/mlg-ulb/creditcardfraud)
- 저장 경로: `data/creditcard.csv`

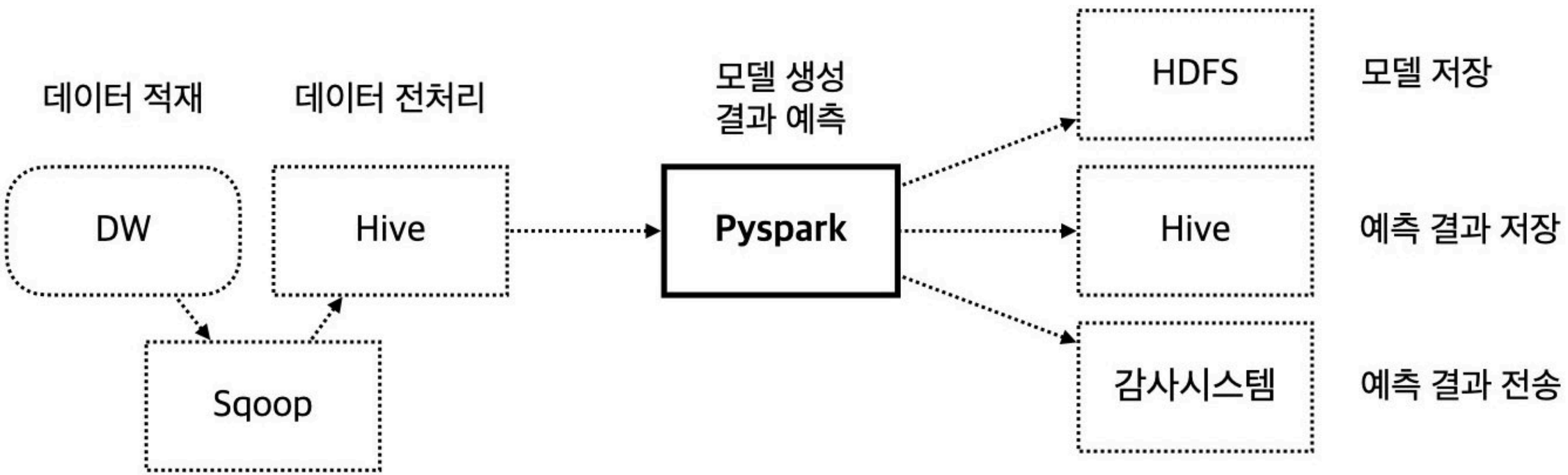
Kaggle에서 `creditcard.csv` 를 다운받아 `data` 디렉터리에 저장한다.

차이점:

- Feature 자료형
  - Kaggle 데이터는 모두 정규화된 수치들이다.
  - KB 데이터는 카테고리형 데이터가 많다.
- Feature 수
  - Kaggle 데이터는 이름 없는 컬럼 28개와 시간, 금액 정보가 담겨있다.
  - 전처리한 KB 데이터는 다음과 같다.
    - 가계 사기, 기업 사기 데이터: 30여개 컬럼
    - 가계 부실: 60여개 컬럼

문제점: KB 데이터에 맞춰 개발한 모델 결과보다 성능 지표가 떨어진다.

KB 상시감사 프로세스



Kaggle 데이터 분석 프로세스

전처리

PyHive 라이브러리는 SSL 인증이 지원되지 않는다.
따라서 KB 데이터는 Hive SQL을 사용해서 전처리했다.

Kaggle 데이터는 간단하게 두 가지 컬럼만 전처리한다.

데이터 불러오기

Kaggle 데이터를 불러와 살펴본다.

In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
df = pd.read_csv('data/creditcard.csv', sep=',')
```

In [3]:
```
df.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

5 rows × 31 columns

- Time: 이전 거래로부터 경과된 시간
- V1 ~ V28: 익명화된 데이터
- Amount: 금액
- Class
  - 0: 정상
  - 1: 사기

데이터 스케일링

Amount와 Time 데이터 활용하기 위해 조정한다.

In [4]:
```
from sklearn.preprocessing import RobustScaler

rob_scaler = RobustScaler()
```

- Robust Scaler: 평균,분산 대신 중앙값, 분위수 사용하여 데이터 조정. 이상치의 영향이 적다.

Amount, Time 데이터 스케일링:

In [5]:
```
df['scaled_amount'] = rob_scaler.fit_transform(df['Amount'].values.reshape(-1,1))
df['scaled_time'] = rob_scaler.fit_transform(df['Time'].values.reshape(-1,1))
```

Amount, Time 컬럼 삭제:

In [6]:
```
df.drop(['Amount', 'Time'], axis=1, inplace=True)
```

컬럼 순서 변경:

In [7]:
```
scaled_amount = df['scaled_amount']
scaled_time = df['scaled_time']

df.drop(['scaled_amount', 'scaled_time'], axis=1, inplace=True)

df.insert(0, 'scaled_amount', scaled_amount)
df.insert(1, 'scaled_time', scaled_time)
```

변경된 데이터 확인:

In [8]:
```
df.head()
```

Out[8]:

	scaled_amount	scaled_time	V1	V2	V3	V4	V5	V6	V7	V8	...	V20	V21	V22	V23	V24	V25	V26	V27	V
0	1.783274	-0.994983	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	...	0.251412	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.0210
1	-0.269825	-0.994983	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	...	-0.069083	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.0147
2	4.983721	-0.994972	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	...	0.524980	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.0597
3	1.418291	-0.994972	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	...	-0.208038	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.0614
4	0.670579	-0.994960	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	...	0.408542	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.2151

5 rows × 31 columns

불균형 데이터

In [9]:
```
print('정상', round(df['Class'].value_counts()[0]/len(df) * 100, 5), '%')
print('사기', round(df['Class'].value_counts()[1]/len(df) * 100, 5), '%')
```

```
정상 99.82725 %
사기 0.17275 %
```

전체 데이터 중 사기로 판별된 데이터는 1%가 되지 않아 모델이 과적합될 수 있다.
불균형 데이터를 처리해야 한다.

무작위 언더 샘플링

실제 KB 데이터를 사용할 때는 언더/오버 샘플링을 사용하지 않았다.
샘플링을 했을 때 유의미한 변화가 없었기 때문이다.

Kaggle 데이터를 사용할 때는 언더 샘플링을 사용한다.
훈련 데이터셋 크기가 줄어들어 빠르게 테스트할 수 있다.

정상과 사기 데이터로 구분:

In [10]:
```
Normal = df[df['Class'] == 0]
Fraud = df[df['Class'] == 1]
```

무작위로 정상 데이터 선택:

In [11]:
```
temp = Normal.sample(frac=1)
NormalSample = temp.loc[temp['Class'] == 0][:Fraud.shape[0]]
```

데이터를 병합한다.

In [12]:
```
UnderSample = pd.concat([Fraud, NormalSample]).sample(frac=1)
```

언더 샘플링한 데이터 확인:

In [13]:
```
UnderSample.shape
```

Out[13]:
```
(984, 31)
```

훈련, 테스트 데이터 생성

데이터를 7:3 비율로 나눠 훈련, 테스트 데이터를 생성한다.

가계 사기/부실 훈련, 테스트 데이터 생성:

In [14]:
```
from sklearn.model_selection import train_test_split
```

In [15]:
```
X = UnderSample.drop('Class', axis=1)
y = UnderSample['Class']
```

In [16]:
```
global X_train, y_train, X_test, y_test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

기업 사기 훈련, 테스트 데이터 생성:

In [17]:
```
global X_train_c, X_test_c, y_test_c

train_c, test_c = train_test_split(UnderSample, test_size=0.3, random_state=1)
X_train_c = train_c.drop('Class', axis=1)
X_test_c = test_c.drop('Class', axis=1)
y_test_c = test_c['Class'].values
```

모델 생성

전처리한 데이터를 공통으로 사용하여 3가지 모델을 생성한다.
실제로는 3가지 모두 다른 데이터셋을 사용한다.

1. 가계 사기 모형: XGBoost
2. 가계 부실 모형: DNN
3. 기업 사기 모형: Isolation Forest

가계 사기 모형

XGBoost로 모델을 생성한다.

In [18]:
```
from xgboost import XGBClassifier
from sklearn.metrics import auc, precision_recall_curve, confusion_matrix

from skopt import gp_minimize
from skopt.space import Space, Real, Integer, Categorical
from skopt.utils import use_named_args
```

In [19]:
```
class HouseFraud:
    dimensions = [
        Real(low=0.01, high=1.0, name='learning_rate'),
        Real(low=1e-9, high=1000, name='reg_lambda'),
        Real(low=1e-9, high=11, name='reg_alpha'),
        Real(low=1e-9, high=0.5, name='gamma'),
        Integer(low=2, high=10, name='max_depth'),
        Integer(low=50, high=100, name='n_estimators')
    ]

    @use_named_args(dimensions)
    def xgb_score(**params):
```

```
        params['scale_pos_weight'] = (y_train.shape[0] - sum(y_train))/sum(y_train)
        params['eval_metric'] = 'aucpr'
        params['n_jobs'] = 5

        _xgb = XGBClassifier()
        _xgb.set_params(**params)
        _xgb.fit(X_train, y_train)
        y_proba = _xgb.predict_proba(X_test)

        precision, recall, _ = precision_recall_curve(y_test, y_proba[:, 1])

        auc_pr = auc(recall, precision)

        params.pop('scale_pos_weight')
        params.pop('eval_metric')
        params.pop('n_jobs')
        params['aucpr'] = auc_pr

        return 1 - auc_pr

    def opt(self):
        optvar_name = [x.name for x in HouseFraud.dimensions]
        optvar_name.append('aucpr')
        res_gp = gp_minimize(HouseFraud.xgb_score, HouseFraud.dimensions, n_jobs = 5, n_calls = 10)
        return res_gp, optvar_name

    def train_and_test_model(self, res_gp, optvar_name):
        params={}
        optvar_name.remove('aucpr')
        for i in range(len(optvar_name)):
            params[optvar_name[i]] = res_gp['x'][i]

        params['scale_pos_weight'] = (y_train.shape[0] - sum(y_train)) / sum(y_train)
        params['eval_metric'] = 'aucpr'
        params['n_jobs'] = 5

        _xgb = XGBClassifier()
        _xgb.set_params(**params)

        _xgb.fit(X_train, y_train)

        y_proba = _xgb.predict_proba(X_test)

        precision, recall, _ = precision_recall_curve(y_test, y_proba[:, 1])
        _xgb.performance_indicator = {}

        cm = confusion_matrix(y_test, y_proba[:, 1]>0.5)
        aucpr = auc(recall, precision)

        _xgb.performance_indicator['confusion_matrix'] = cm
        _xgb.performance_indicator['aucpr'] = aucpr
        _xgb.train_col = list(X_train.columns)

        return y_proba, _xgb
```

In [20]:
```
house_fraud = HouseFraud()
```

gp_minimize 함수로 최적 파라미터 탐색:

In [21]:
```
res_gp, optvar_name = house_fraud.opt()
```

최적화한 XGBoost 모델로 학습 모델과 테스트 결과 생성:

In [22]:
```
y_proba_house_fraud, house_fraud_model = house_fraud.train_and_test_model(res_gp, optvar_name)
```

모델 성능 지표 확인:

In [23]:
```
house_fraud_cm = house_fraud_model.performance_indicator['confusion_matrix']
house_fraud_aucpr = house_fraud_model.performance_indicator['aucpr']
```

In [24]:
```
pd.DataFrame({"예측 거짓": [house_fraud_cm[0][0], house_fraud_cm[0][1]], \
              "예측 참": [house_fraud_cm[1][0], house_fraud_cm[1][1]]})\
.rename(index = {0:'실제 거짓', 1:'실제 참'})
```

Out[24]:

	예측 거짓	예측 참
실제 거짓	143	12
실제 참	3	138

In [25]:
```
house_fraud_aucpr
```

Out[25]: 0.9875409155972033

- AUCPR(Area Under Precision Recall Curve)이 1에 가까이 나올 수록 성능이 좋다.
- Confusion Matrix: 실제 값과 예측 값이 같아야 성능이 좋다.
  - KB 상시감사 모델링 목표는 2종 오류를 0으로 맞추는 것이다.

	예측 거짓	예측 참
실제 거짓	True Negatives	False Positives (1종 오류)
실제 참	False Negatives (2종 오류)	True Positives

가계 부실 모형

Deep Neural Network, DNN으로 모델을 생성한다.

In [ ]:
```
from pyspark.conf import SparkConf
from pyspark.sql import SparkSession
from pyspark.context import SparkContext
from pyspark.sql.types import *
from pyspark.sql.functions import pandas_udf, PandasUDFType, udf
import pyspark.sql.functions as f
from pyspark.ml.feature import MinMaxScaler as pysparkMinMaxScaler
from pyspark.ml.feature import VectorAssembler
from pyspark.ml import Pipeline

from sklearn.metrics import auc, precision_recall_curve, confusion_matrix, f1_score

import skopt
from skopt import gp_minimize
from skopt.space import Space, Real, Integer, Categorical
from skopt.utils import use_named_args
from sklearn.preprocessing import *
from sklearn.metrics import *

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import initializers
from tensorflow.keras.models import Sequential
```

In [27]:
```
class HouseRisk:
    def dnn_create_model(self):
        model = keras.Sequential()
        model.add(keras.layers.Dense(95, input_dim=X_train.shape[1], activation='relu'))
        model.add(keras.layers.Dropout(0.25))

        model.add(keras.layers.Dense(142, activation='tanh'))
        model.add(keras.layers.Dropout(0.25))

        model.add(keras.layers.Dense(70, activation=tf.nn.leaky_relu))
        model.add(keras.layers.Dropout(0))

        model.add(keras.layers.Dense(1, activation='sigmoid'))

        optimizer = tf.keras.optimizers.Adam(0.01)
        model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics = ['accuracy'])
        return model

    def train_and_test_model(self, columns_to_scale):
        aucpr= 0
        history = []
        _iter = 0

        while((aucpr<0.4) & (_iter<10)):
            _iter+=1
            _dnn = self.dnn_create_model()
            _dnn.fit(x=X_train.fillna(0).values, y=y_train.values, epochs=10, batch_size=64, validation_split= 0.3, verbose=0, shuffle=False, use_multiprocessi
ng=True)

            y_proba = _dnn.predict(X_test.fillna(0).values)
            precision, recall, _ = precision_recall_curve(y_test, y_proba)
            proba_scaler = MinMaxScaler().fit(_dnn.predict(X_train.values))
            aucpr = auc(recall, precision)

            temp = {}
            temp['dnn'] = _dnn
            temp['aucpr'] = aucpr
            temp['scaler'] = proba_scaler
            history.append(temp)

        max_aucpr = sorted(history, key=lambda x:x['aucpr'], reverse=True)[0]['aucpr']
        max_scaler = sorted(history, key=lambda x:x['aucpr'], reverse=True)[0]['scaler']
        max_dnn = sorted(history, key=lambda x:x['aucpr'], reverse=True)[0]['dnn']

        _dnn_etc = {'aucpr': max_aucpr, 'train_col': list(X_train.columns), 'proba_scaler': max_scaler, 'scale_col':list(columns_to_scale)}

        return max_dnn, _dnn_etc
```

In [28]:
```
house_risk = HouseRisk()
```

DNN 모델 생성:

In [29]:
```
house_risk_model, house_risk_model_info = house_risk.train_and_test_model(UnderSample)
```

```
WARNING:tensorflow:From /opt/anaconda3/envs/kb/lib/python3.6/site-packages/tensorflow/python/ops/resource_variable_ops.py:435: colocate_with (from tensorflow
.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /opt/anaconda3/envs/kb/lib/python3.6/site-packages/tensorflow/python/keras/layers/core.py:143: calling dropout (from tensorflow.pytho
n.ops.nn_ops) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
WARNING:tensorflow:From /opt/anaconda3/envs/kb/lib/python3.6/site-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_
ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
```

모델 성능 지표 확인:

In [30]:
```
house_risk_model_info['aucpr']
```

Out[30]:
```
0.9799720080943911
```

- AUCPR(Area Under Precision Recall Curve)이 1에 가까이 나올 수록 성능이 좋다.

기업 사기 모형

Isolation Forest로 모델을 생성한다.

In [31]:
```
import sklearn.ensemble as ensemble

from skopt import gp_minimize
from skopt.space import Space, Real, Integer, Categorical
from skopt.utils import use_named_args
```

In [32]:
```
company_fraud_score = None
company_fraud_model = None
```

In [33]:
```
class CompanyFraud:
    BAYESIAN_RUNS = 100
    DEFAULT_OFFSET = -0.423
    DEFAULT_PARAMS = {
        'n_jobs': 4, # CPU cores
        'random_state': 42,
    }

    ORIGINAL_PARAMS = {
        'max_features': 0.7934935258073553,
        'max_samples': 1.0,
```

```
        'n_estimators': 100
    }

    dimensions = [
        Real(low=0.1, high=1.0, name='max_features'),
        Real(low=0.1, high=1.0, name='max_samples'),
        Integer(low=100, high=1000, name='n_estimators')
    ]


    def train_model(data, params):
        forest = ensemble.IsolationForest()
        forest.set_params(**params)
        forest.fit(data)
        return forest

    def test_model(model, X_test, y):
        from sklearn.metrics import (
            roc_auc_score, average_precision_score,
            f1_score, precision_score, recall_score,
            accuracy_score, confusion_matrix
        )

        # 절대 이상치값, 사기여부 추론
        y_proba = np.abs(model.score_samples(X_test))
        y_pred = model.predict(X_test)
        y_pred = np.where(y_pred == -1, 1, 0)

        from sklearn.metrics import precision_recall_curve, auc
        precision, recall, _ = precision_recall_curve(y, y_proba, pos_label=1)
        AUCPR = auc(recall, precision)

        # 여러 지표로 모델 평가
        roc_auc = float(roc_auc_score(y, y_proba))
        pr_auc = auc(recall, precision)
        f1 = list(f1_score(y, y_pred, average=None))
        precision = list(precision_score(y, y_pred, average=None))
        recall = list(recall_score(y, y_pred, average=None))
        accuracy = accuracy_score(y, y_pred)
        cm = [int(x) for x in confusion_matrix(y, y_pred, ).ravel() ]

        return {
            'roc_auc': roc_auc,
            'pr_auc': pr_auc,
            'f1_score': f1,
            'precision': precision,
            'recall': recall,
            'accuracy': accuracy,
            'confusion_matrix': cm
        }

    @use_named_args(dimensions)
    def optimize_model(**params):
        global company_fraud_score, company_fraud_model

        params.update(CompanyFraud.DEFAULT_PARAMS)
        forest = CompanyFraud.train_model(X_train_c, params)
        forest.offset_ = CompanyFraud.DEFAULT_OFFSET
        score = CompanyFraud.test_model(forest, X_test_c, y_test_c)
```

```
        if company_fraud_score is None or score['pr_auc'] > company_fraud_score['pr_auc']:
            company_fraud_score = score
            company_fraud_model = forest

        return 1 - score['pr_auc']

    def opt(self):
        return gp_minimize(CompanyFraud.optimize_model, CompanyFraud.dimensions, n_calls=CompanyFraud.BAYESIAN_RUNS)

    def set_offset(self, model, X_test, y, start=-0.60, end=-0.40, step=0.01):
        highest_score = None
        best_offset = CompanyFraud.DEFAULT_OFFSET

        for offset in np.arange(start, end, step):
            model.offset_ = offset
            score = CompanyFraud.test_model(model, X_test, y)
            print("%.3f" % offset, score['confusion_matrix'])

            # 목표: 2종 오류 0
            # if score['confusion_matrix'][2] == 0:
            #     print('2종 오류 0:', score)
            #     if highest_score is None or score['f1_score'][1] > highest_score['f1_score'][1]:
            #         highest_score = score
            #         best_offset = offset

            if (highest_score is None) or (score['f1_score'][1] > highest_score['f1_score'][1]):
                highest_score = score
                best_offset = offset

        model.offset_ = best_offset
        return model, highest_score
```

In [34]:
```
company_fraud = CompanyFraud()
```

In [ ]:
```
res_gp = company_fraud.opt();
```

In [36]:
```
company_fraud_score
```

Out[36]:
```
{'roc_auc': 0.8850684931506849,
 'pr_auc': 0.8982580075214015,
 'f1_score': [0.7918088737201364, 0.7959866220735786],
 'precision': [0.7891156462585034, 0.7986577181208053],
 'recall': [0.7945205479452054, 0.7933333333333333],
 'accuracy': 0.793918918918919,
 'confusion_matrix': [116, 30, 31, 119]}
```

In [37]:
```
company_fraud_model
```

Out[37]:
```
IsolationForest(max_features=0.1, max_samples=0.11239154014351381,
                n_estimators=1000, n_jobs=4, random_state=42)
```

In [38]:
```
company_fraud_model, company_fraud_score = company_fraud.set_offset(company_fraud_model, np.array(X_test_c), y_test_c)
```

```
-0.600 [146, 0, 145, 5]
-0.590 [146, 0, 139, 11]
-0.580 [146, 0, 130, 20]
-0.570 [146, 0, 122, 28]
-0.560 [146, 0, 117, 33]
-0.550 [146, 0, 114, 36]
-0.540 [146, 0, 113, 37]
-0.530 [146, 0, 112, 38]
-0.520 [146, 0, 110, 40]
-0.510 [146, 0, 108, 42]
-0.500 [146, 0, 106, 44]
-0.490 [146, 0, 98, 52]
-0.480 [143, 3, 86, 64]
-0.470 [140, 6, 74, 76]
-0.460 [136, 10, 64, 86]
-0.450 [136, 10, 52, 98]
-0.440 [133, 13, 41, 109]
-0.430 [121, 25, 37, 113]
-0.420 [114, 32, 27, 123]
-0.410 [93, 53, 15, 135]
```

In [39]:
```
company_fraud_result = {
    'offset': abs(company_fraud_model.offset_) * 100,
    'maxfeatures': company_fraud_model.max_features,
    'maxsamples': company_fraud_model.max_samples,
    'netimators': company_fraud_model.n_estimators,
    'prauc': company_fraud_score['pr_auc'],
    'rocauc': company_fraud_score['roc_auc'],
    'f1score': company_fraud_score['f1_score'][1],
    'precision': company_fraud_score['precision'][1],
    'recall': company_fraud_score['recall'][1],
    'accuracy': company_fraud_score['accuracy'],
    'truenegative': company_fraud_score['confusion_matrix'][0],
    'falsepositive': company_fraud_score['confusion_matrix'][1],
    'falsenegative': company_fraud_score['confusion_matrix'][2],
    'truepositive': company_fraud_score['confusion_matrix'][3],
}
```

In [40]:
```
company_fraud_result
```

Out[40]:
```
{'offset': 41.99999999999998,
 'maxfeatures': 0.1,
 'maxsamples': 0.11239154014351381,
 'netimators': 1000,
 'prauc': 0.8982580075214015,
 'rocauc': 0.8850684931506849,
 'f1score': 0.8065573770491803,
 'precision': 0.7935483870967742,
 'recall': 0.82,
 'accuracy': 0.8006756756756757,
 'truenegative': 114,
 'falsepositive': 32,
 'falsenegative': 27,
 'truepositive': 123}
```

In [41]:
```
result = pd.DataFrame({"거짓": [company_fraud_result['truenegative'], company_fraud_result['falsepositive']], \
                       "참": [company_fraud_result['falsenegative'], company_fraud_result['truepositive']]})\
    .rename(index = {0:'거짓', 1:'참'})

result
```

Out[41]:

	거짓	참
거짓	114	27
참	32	123

In [42]:
```
company_fraud_result['prauc']
```

Out[42]: 0.8982580075214015

- AUCPR(Area Under Precision Recall Curve)이 1에 가까이 나올 수록 성능이 좋다.
- Confusion Matrix: 실제 값과 예측 값이 같아야 성능이 좋다.
  - KB 상시감사 모델링 목표는 2종 오류를 0으로 맞추는 것이다.

	예측 거짓	예측 참
실제 거짓	True Negatives	False Positives (1종 오류)
실제 참	False Negatives (2종 오류)	True Positives

예측

위에서 생성한 3가지 모델로 결과를 예측한다.

1. 가계 사기 모형: XGBoost
2. 가계 부실 모형: DNN
3. 기업 사기 모형: Isolation Forest

SHAP 함수:

In [43]:
```
import shap

def h_shap(model, X):
    explainer = shap.TreeExplainer(model)
    shap_values = explainer.shap_values(X)
    df_shap_values = pd.DataFrame(explainer.shap_values(X), columns = X.columns.values)

    df_importances = pd.DataFrame(columns=['사기 근거 1', '사기 근거 값1', '사기 근거 비중1',
                                           '사기 근거 2', '사기 근거 값2', '사기 근거 비중2',
```

```
                                          '사기 근거 3', '사기 근거 값3', '사기 근거 비중3',
                                          '사기 근거 4', '사기 근거 값4', '사기 근거 비중4',
                                          '사기 근거 5', '사기 근거 값5', '사기 근거 비중5',
                                          '정상 근거 1', '정상 근거 값1', '정상 근거 비중1',
                                          '정상 근거 2', '정상 근거 값2', '정상 근거 비중2',
                                          '정상 근거 3', '정상 근거 값3', '정상 근거 비중3',
                                          '정상 근거 4', '정상 근거 값4', '정상 근거 비중4',
                                          '정상 근거 5', '정상 근거 값5', '정상 근거 비중5'])

    for i in range(len(X)):
        importance = sorted(zip(X.columns, X.values[i], df_shap_values.iloc[i, :]), key=lambda x: x[2], reverse=True)
        frd_imp = importance[:5]
        importance = sorted(zip(X.columns, X.values[i], -1 * df_shap_values.iloc[i, :]), key=lambda x: x[2], reverse=True)
        norm_imp = importance[:5]

        frd_imp.extend(norm_imp)
        df_importances.loc[i] = list(sum(frd_imp, ()))

    df_importances.insert(0, 'index', X.index.values)

    return explainer, shap_values, df_importances

def c_shap(model, X):
    explainer = shap.TreeExplainer(model)
    shap_values = explainer.shap_values(X)
    df_shap_values = pd.DataFrame(explainer.shap_values(X), columns = X.columns.values)

    df_importances = pd.DataFrame(columns=['사기 근거 1', '사기 근거 값1', '사기 근거 비중1',
                                          '사기 근거 2', '사기 근거 값2', '사기 근거 비중2',
                                          '사기 근거 3', '사기 근거 값3', '사기 근거 비중3',
                                          '사기 근거 4', '사기 근거 값4', '사기 근거 비중4',
                                          '사기 근거 5', '사기 근거 값5', '사기 근거 비중5',
                                          '정상 근거 1', '정상 근거 값1', '정상 근거 비중1',
                                          '정상 근거 2', '정상 근거 값2', '정상 근거 비중2',
                                          '정상 근거 3', '정상 근거 값3', '정상 근거 비중3',
                                          '정상 근거 4', '정상 근거 값4', '정상 근거 비중4',
                                          '정상 근거 5', '정상 근거 값5', '정상 근거 비중5'])

    for i in range(len(X)):
        importance = sorted(zip(X.columns, X.values[i], -1 * df_shap_values.iloc[i, :]), key=lambda x: x[2], reverse=True)
        frd_imp = importance[:5]
        importance = sorted(zip(X.columns, X.values[i], df_shap_values.iloc[i, :]), key=lambda x: x[2], reverse=True)
        norm_imp = importance[:5]

        frd_imp.extend(norm_imp)
        df_importances.loc[i] = list(sum(frd_imp, ()))

    df_importances.insert(0, 'index', X.index.values)

    return explainer, shap_values, df_importances
```

SHAP 그래프 함수:

In [44]:
```
import random

shap.initjs()
```

```python
def HouseAdditiveForceVisualizer(data, explainer, shap_values, X):
    row_id = data.index.values[0]
    data_id = data['index'].values[0]
    data_class = data['사기여부'].values[0]
    data_fraud = "%.1f" % data['사기확률'].values[0]
    data_risk = data['부실확률'].values[0]

    print(f"{data_id}번 실제 값: 정상" if data_class == 0 else f"{data_id}번 실제 값: 사기")
    print(f"예측한 사기 확률: {data_fraud}%")
    print(f"예측한 부실 확률: {data_risk}부실")
    plt = shap.force_plot(explainer.expected_value, shap_values[row_id,:], X.iloc[row_id,:], show=False)
    return plt

def CompanyAdditiveForceVisualizer(data, explainer, shap_values, X):
    row_id = data.index.values[0]
    data_id = data['index'].values[0]
    data_class = data['사기여부'].values[0]
    data_fraud = "%.1f" % data['사기확률'].values[0]

    print(f"{data_id}번 실제 값: 정상" if data_class == 0 else f"{data_id}번 실제 값: 사기")
    print(f"예측한 사기 확률: {data_fraud}%")
    plt = shap.force_plot(explainer.expected_value, shap_values[row_id,:], X.iloc[row_id,:], show=False)
    return plt

def HouseVisualizer(importances, explainer, shap_values, X, Class_value = 1):
    data = importances.loc[importances['사기여부'] == Class_value]
    random_index = random.randint(0, len(data))
    data = data.iloc[[random_index]]
    return HouseAdditiveForceVisualizer(data, explainer, shap_values, X), data

def CompanyVisualizer(importances, explainer, shap_values, X, Class_value = 1):
    data = importances.loc[importances['사기여부'] == Class_value]
    random_index = random.randint(0, len(data))
    data = data.iloc[[random_index]]
    return CompanyAdditiveForceVisualizer(data, explainer, shap_values, X), data

def OneHouseVisualizer(importances, explainer, shap_values, X, row_id):
    return HouseAdditiveForceVisualizer(importances.iloc[[row_id]], explainer, shap_values, X)

def OneCompanyVisualizer(importances, explainer, shap_values, X, row_id):
    return CompanyAdditiveForceVisualizer(importances.iloc[[row_id]], explainer, shap_values, X)

def FactorTable(data):
    return pd.DataFrame({
        '사기 근거 항목': [data['사기 근거 1'].values[0], data['사기 근거 2'].values[0], data['사기 근거 3'].values[0], data['사기 근거 4'].values[0], data['사기 근거 5'].va
lues[0]],
        '사기 근거 값': [data['사기 근거 값1'].values[0], data['사기 근거 값2'].values[0], data['사기 근거 값3'].values[0], data['사기 근거 값4'].values[0], data['사기 근거 값
5'].values[0]],
        '사기 근거 비중': [data['사기 근거 비중1'].values[0], data['사기 근거 비중2'].values[0], data['사기 근거 비중3'].values[0], data['사기 근거 비중4'].values[0], data['사기
근거 비중5'].values[0]],
        '정상 근거 항목': [data['정상 근거 1'].values[0], data['정상 근거 2'].values[0], data['정상 근거 3'].values[0], data['정상 근거 4'].values[0], data['정상 근거 5'].va
lues[0]],
        '정상 근거 값': [data['정상 근거 값1'].values[0], data['정상 근거 값2'].values[0], data['정상 근거 값3'].values[0], data['정상 근거 값4'].values[0], data['정상 근거 값
5'].values[0]],
        '정상 근거 비중': [data['정상 근거 비중1'].values[0], data['정상 근거 비중2'].values[0], data['정상 근거 비중3'].values[0], data['정상 근거 비중4'].values[0], data['정상
근거 비중5'].values[0]],
    }).rename(index = {0:'1', 1:'2', 2:'3', 3:'4', 4:'5'})
```

가계 사기/부실 예측

- 사기 예측: XGBoost 모델
- 부실 예측: DNN 모델

사기 예측으로 사전점검 화면에
대출 건 별 중요하게 영향을 미친 변수항목 및 사기확률을 나타낸다.
감사역은 정리된 정상/사기 근거항목을 보고 의사결정을 빠르게 할 수 있다.

사기 예측 결과와 함께 SHAP 라이브러리를 활용하여 근거 항목을 산출한다.

부실 예측 확률은 정상/부실이 아닌 저부실, 중부실, 고부실로 분류하여 활용한다.
실제 부실율도 저부실에서 고부실로 갈수록 높아진다.
저부실일수록 신용등급이 좋고 고부실일수록 신용등급이 나쁘다.

가계 사기 및 부실 모델 예측:

In [45]:
```
y_pred_house_fraud = house_fraud_model.predict_proba(X_test)
y_pred_house_risk = house_risk_model.predict_proba(X_test)
```

가계 부실 확률 분류:

- 저: ~ 20%
- 중: 20% ~ 80%
- 고: 80% ~

In [46]:
```
house_risk_result = np.where(y_pred_house_risk < 0.2, '저', \
                    np.where((y_pred_house_risk >= 0.2) & (y_pred_house_risk <= 0.8), '중', \
                             np.where(y_pred_house_risk >= 0.8, '고', '-')))
```

거래별 모델 예측 근거:

In [47]:
```
house_explainer, house_shap_values, house_importances = h_shap(house_fraud_model, X_test)
house_importances.insert(1, '사기여부', y_test.values)
house_importances.insert(2, '사기확률', (y_pred_house_fraud[:,1] * 100).round(1))
house_importances.insert(3, '부실확률', pd.DataFrame(house_risk_result))
house_importances.head()
```

```
Setting feature_perturbation = "tree_path_dependent" because no background data was given.
```

Out[47]:

	index	사기여부	사기확률	부실확률	사기 근거 1	사기 근거 값 1	사기 근거 비중1	사기 근거 2	사기 근거 값2	사기 근거 비중2	...	정상 근거 비중2	정상 근거 3	정상 근거 값3	정상 근거 비중3	정상 근거 4	정상 근거 값4	정상 근거 비중4	정상 근거 5	정상 근거 값5	정상 근거 비중5
0	234632	1	98.099998	고	V14	-8.485795	1.345451	V4	5.342759	1.053857	...	0.056336	V8	0.123062	0.039585	V17	-1.372629	0.021900	V20	0.313332	0.019152
1	42856	1	98.800003	고	V14	-10.468677	1.290441	V4	7.690772	0.978565	...	0.007625	V28	-0.818970	0.004637	V13	1.184985	0.001701	V22	-1.271509	0.000247
2	165728	0	4.400000	저	V8	-4.136271	0.193060	V20	-0.898750	0.164918	...	1.029965	V10	-0.083725	0.579724	V12	0.536875	0.237924	V11	-1.443451	0.229618
3	167305	1	96.900002	고	V14	-8.490813	1.233970	V4	6.081321	1.152444	...	0.008585	V24	0.163718	0.007625	V13	-1.105710	0.001701	V22	-1.608272	0.000247
4	142696	0	28.000000	저	V4	2.459656	0.970674	V16	-0.642843	0.186302	...	0.406562	V10	-0.094109	0.339166	V12	1.278722	0.275123	V11	-0.268510	0.253928

5 rows × 34 columns

가계 사기/부실 예측 근거 확인

감사역은 사전점검 화면에서 다음과 같은 표와 그래프를 확인한다.

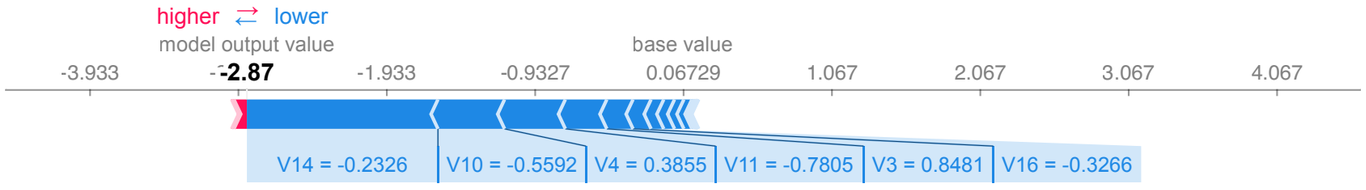
정상 거래 예측 결과

다음은 정상으로 예측한 거래 중 한 가지다.

In [48]:
```
house_fraud_normal_graph, house_fraud_normal_data = HouseVisualizer(house_importances, house_explainer, house_shap_values, X_test, 0)
house_fraud_normal_graph
```

```
48121번 실제 값: 정상
예측한 사기 확률: 5.4%
예측한 부실 확률: 저부실
```

Out[48]:



사기/정상 근거 비중 상위 5개:

In [49]:
```
FactorTable(house_fraud_normal_data)
```

Out[49]:

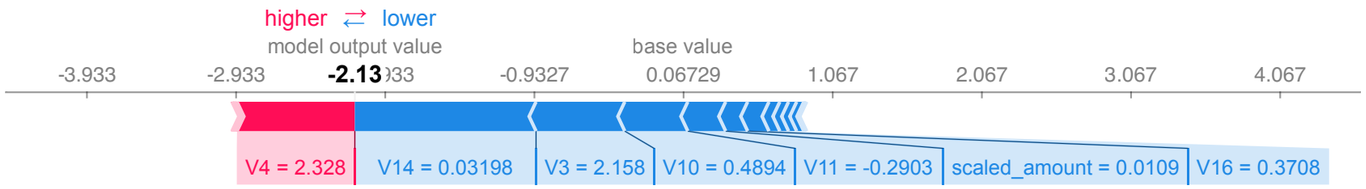
	사기 근거 항목	사기 근거 값	사기 근거 비중	정상 근거 항목	정상 근거 값	정상 근거 비중
1	scaled_amount	2.793684	0.087311	V14	-0.232595	1.287732
2	V5	-1.332437	0.015295	V10	-0.559201	0.448869
3	V28	0.045511	0.004934	V4	0.385452	0.409657
4	scaled_time	-0.483876	0.000000	V11	-0.780459	0.279566
5	V2	-1.173665	0.000000	V3	0.848142	0.176579

정상으로 예측한 다른 거래:

In [50]:
```
house_fraud_normal_graph, house_fraud_normal_data = HouseVisualizer(house_importances, house_explainer, house_shap_values, X_test, 0)
house_fraud_normal_graph
```

```
50472번 실제 값: 정상
예측한 사기 확률: 10.6%
예측한 부실 확률: 저부실
```

Out[50]:



In [51]:
```
FactorTable(house_fraud_normal_data)
```

Out[51]:

	사기 근거 항목	사기 근거 값	사기 근거 비중	정상 근거 항목	정상 근거 값	정상 근거 비중
1	V4	2.328482	0.789859	V14	0.031982	1.214830
2	V5	-0.087609	0.015295	V3	2.158198	0.594864
3	V26	-0.136649	0.010396	V10	0.489432	0.422632
4	V28	0.172899	0.007855	V11	-0.290285	0.253928
5	V1	-0.575168	0.007334	scaled_amount	0.010899	0.147318

두 거래에서 모두 V14 값이 정상 근거 비중이 높게 나타난다.
두 거래에서 사기 근거들은 V14 외 정상 근거들에 의해 비중이 상쇄된다.

감사역은 30여가지가 되는 모든 거래 관련 항목을 보지 않고 비중이 높은 항목들로 빠르게 판단을 할 수 있다.

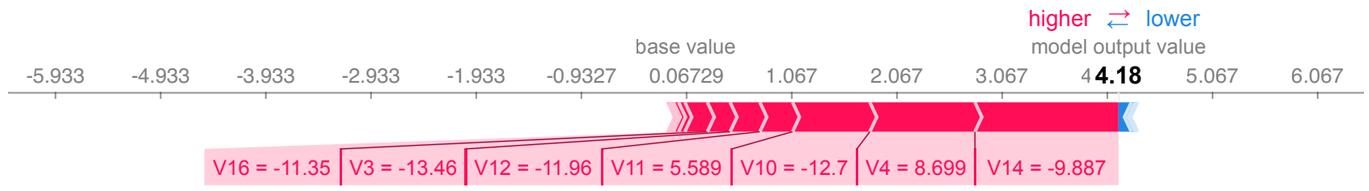
사기 거래 예측 결과

다음은 사기로 예측한 거래 중 한 가지다.

In [52]:
```
house_fraud_fraud_graph, house_fraud_fraud_data = HouseVisualizer(house_importances, house_explainer, house_shap_values, X_test, 1)
house_fraud_fraud_graph
```

```
42936번 실제 값: 사기
예측한 사기 확률: 98.5%
예측한 부실 확률: 고부실
```

Out[52]:



사기/정상 근거 비중 상위 5개:

In [53]:
```
FactorTable(house_fraud_fraud_data)
```

Out[53]:

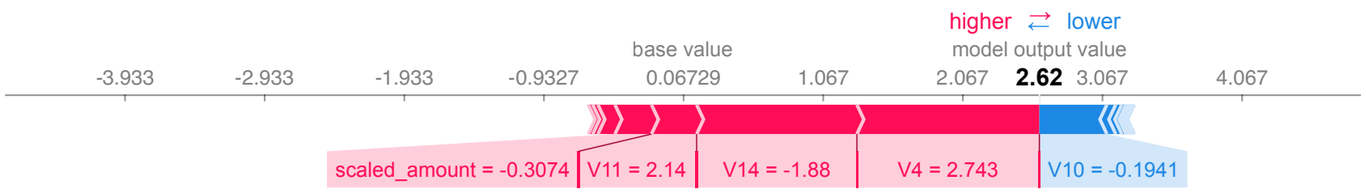
	사기 근거 항목	사기 근거 값	사기 근거 비중	정상 근거 항목	정상 근거 값	정상 근거 비중
1	V14	-9.887214	1.365578	scaled_amount	0.526514	0.121289
2	V4	8.698610	0.998754	V8	8.218191	0.035623
3	V10	-12.695947	0.734681	V26	-0.006168	0.008585
4	V11	5.589362	0.310680	V24	0.673209	0.007625
5	V12	-11.960866	0.283093	V28	-0.747361	0.004637

사기로 예측한 다른 거래:

In [54]:
```
house_fraud_fraud_graph, house_fraud_fraud_data = HouseVisualizer(house_importances, house_explainer, house_shap_values, X_test, 1)
house_fraud_fraud_graph
```

```
93788번 실제 값: 사기
예측한 사기 확률: 93.2%
예측한 부실 확률: 고부실
```

Out[54]:



사기/정상 근거 비중 상위 5개:

In [55]:
```
FactorTable(house_fraud_fraud_data)
```

Out[55]:

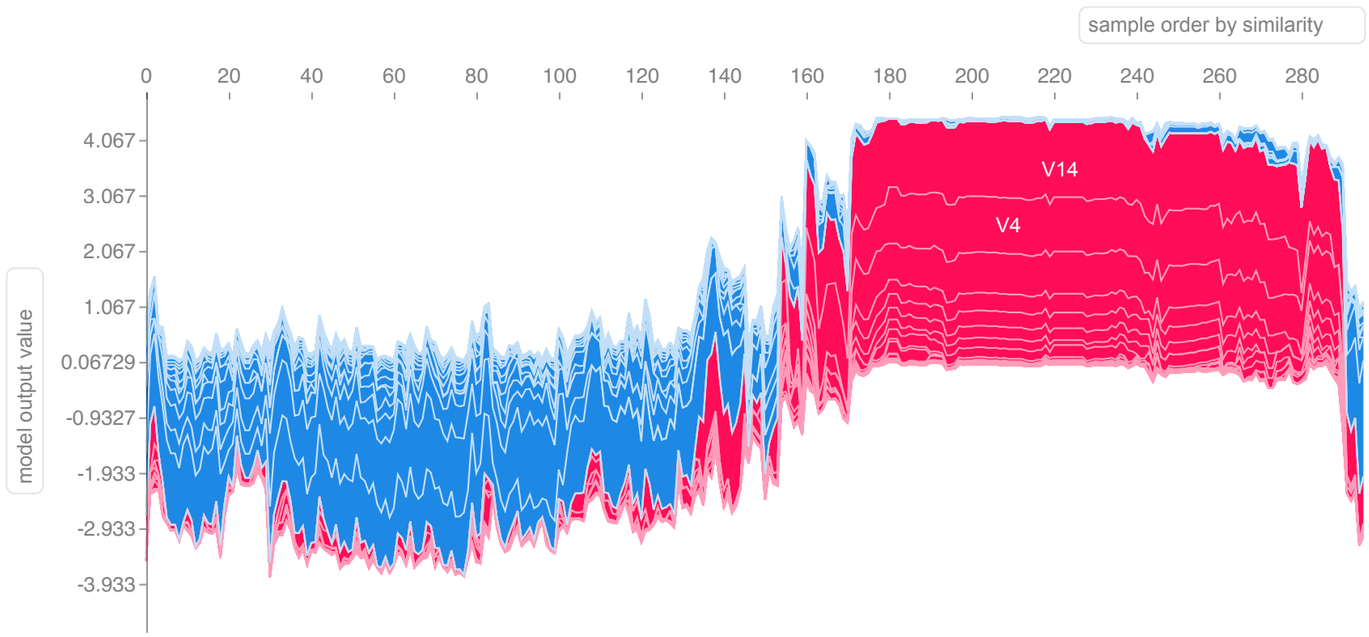
	사기 근거 항목	사기 근거 값	사기 근거 비중	정상 근거 항목	정상 근거 값	정상 근거 비중
1	V4	2.743318	1.306967	V10	-0.194120	0.483165
2	V14	-1.880275	1.147732	V16	1.367433	0.063244
3	V11	2.140057	0.323175	V8	0.176170	0.043509
4	scaled_amount	-0.307413	0.261695	V20	-0.172659	0.029967
5	V17	1.522662	0.121812	V12	-0.276309	0.019536

두 거래에서 모두 V14, V4 값이 사기 근거로 비중이 높게 나타난다.
두 거래에서 정상 근거들은 비중이 낮게 나타난다.

전체 거래 예측 근거 그래프:

In [56]:
```
shap.force_plot(house_explainer.expected_value, house_shap_values, X_test)
```

Out[56]:



기업 사기 예측

- 사기 예측: Isolation Forest 모델

사기 예측으로 사전점검 화면에
대출 건 별 중요하게 영향을 미친 변수항목 및 사기확률을 나타낸다.
감사역은 정리된 정상/사기 근거항목을 보고 의사결정을 빠르게 할 수 있다.

사기 예측 결과와 함께 SHAP 라이브러리를 활용하여 근거 항목을 산출한다.

기업 사기 예측 근거 확인

감사역은 사전점검 화면에서 다음과 같은 표와 그래프를 확인한다.

In [57]:
```
y_pred_fraud_company = np.abs(company_fraud_model.score_samples(X_test_c)) * 100
```

In [58]:
```
company_explainer, company_shap_values, company_importances = c_shap(company_fraud_model, X_test_c)
company_importances.insert(1, '사기여부', y_test_c)
company_importances.insert(2, '사기확률', y_pred_fraud_company.round(1))
company_importances.head()
```

```
Setting feature_perturbation = "tree_path_dependent" because no background data was given.
The sklearn.ensemble.iforest module is  deprecated in version 0.22 and will be removed in version 0.24. The corresponding classes / functions should instead
be imported from sklearn.ensemble. Anything that cannot be imported from sklearn.ensemble is now part of the private API.
```

Out[58]:

	index	사기여부	사기확률	사기 근거 1	사기 근거 값 1	사기 근거 비중1	사기 근거 2	사기 근거 값2	사기 근거 비중2	사기 근거 3	...	정상 근거 비중2	정상 근거 3	정상 근거 값 3	정상 근거 비중3	정상 근거 4	정상 근거 값4	정상 근거 비중4	정상 근거 5	정상 근거 값 5	정상 근거 비중5
0	234632	1	42.3	V1	1.261324	0.831859	scaled_time	0.744381	0.086073	V2	...	0.000000	V3	-5.435019	0.0	V4	5.342759	0.0	V5	1.447043	0.0
1	42856	1	57.3	V1	-11.682215	2.640094	scaled_amount	2.110948	1.085433	V2	...	0.000000	V3	-13.297109	0.0	V4	7.690772	0.0	V5	-10.889891	0.0
2	165728	0	44.0	V1	-0.901641	0.348754	V2	1.747927	-0.000000	V3	...	0.332681	V2	1.747927	0.0	V3	-0.532344	0.0	V4	-0.552070	0.0
3	167305	1	50.4	V1	-6.677212	2.330991	scaled_amount	1.172221	0.520987	V2	...	0.000000	V3	-7.193275	0.0	V4	6.081321	0.0	V5	-1.636071	0.0
4	142696	0	44.7	V1	-2.486331	1.674967	scaled_amount	0.866345	0.224476	V2	...	0.000000	V3	2.740996	0.0	V4	2.459656	0.0	V5	1.698475	0.0

5 rows × 33 columns

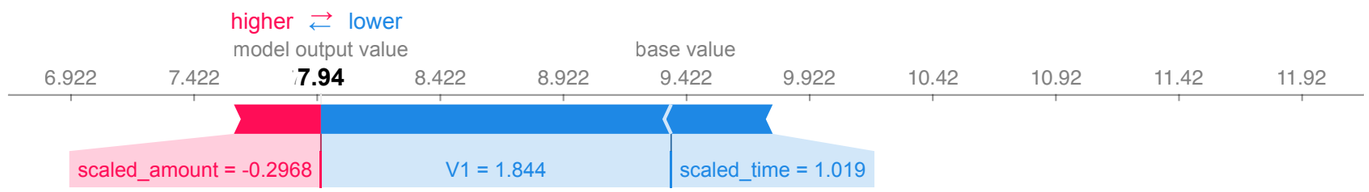
정상 거래 예측 결과

다음은 정상으로 예측한 거래 중 한 가지다.

In [59]:
```
company_fraud_normal_graph, company_fraud_normal_data = CompanyVisualizer(company_importances, company_explainer, company_shap_values, X_test_c, 0)
company_fraud_normal_graph
```

```
283153번 실제 값: 정상
예측한 사기 확률: 41.4%
```

Out[59]:



사기/정상 근거 비중 상위 5개:

In [60]:
```
FactorTable(company_fraud_normal_data)
```

Out[60]:

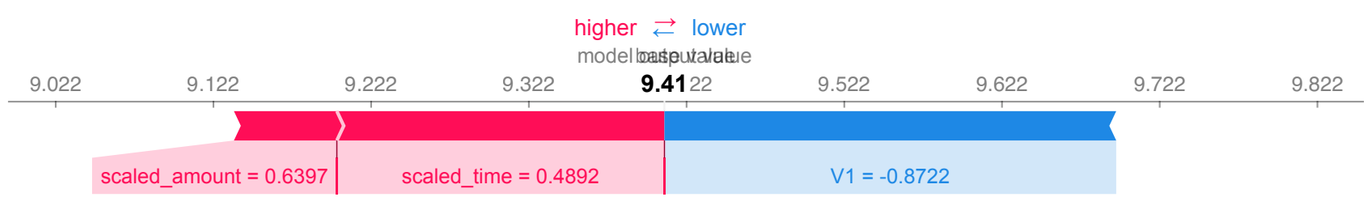
	사기 근거 항목	사기 근거 값	사기 근거 비중	정상 근거 항목	정상 근거 값	정상 근거 비중
1	V1	1.843506	1.422400	scaled_amount	-0.296793	0.353401
2	scaled_time	1.018727	0.412463	V2	0.219116	0.000000
3	V2	0.219116	-0.000000	V3	0.068437	0.000000
4	V3	0.068437	-0.000000	V4	3.571300	0.000000
5	V4	3.571300	-0.000000	V5	-0.036187	0.000000

정상으로 예측한 다른 거래:

In [61]:
```
company_fraud_normal_graph, company_fraud_normal_data = CompanyVisualizer(company_importances, company_explainer, company_shap_values, X_test_c, 0)
company_fraud_normal_graph
```

```
184666번 실제 값: 정상
예측한 사기 확률: 40.2%
```

Out[61]:



사기/정상 근거 비중 상위 5개:

In [62]:
```
FactorTable(company_fraud_normal_data)
```

Out[62]:

	사기 근거 항목	사기 근거 값	사기 근거 비중	정상 근거 항목	정상 근거 값	정상 근거 비중
1	V1	-0.872218	0.286458	scaled_time	0.489197	0.207710
2	V2	1.449383	-0.000000	scaled_amount	0.639698	0.065261
3	V3	0.642204	-0.000000	V2	1.449383	0.000000
4	V4	0.507697	-0.000000	V3	0.642204	0.000000
5	V5	0.835651	-0.000000	V4	0.507697	0.000000

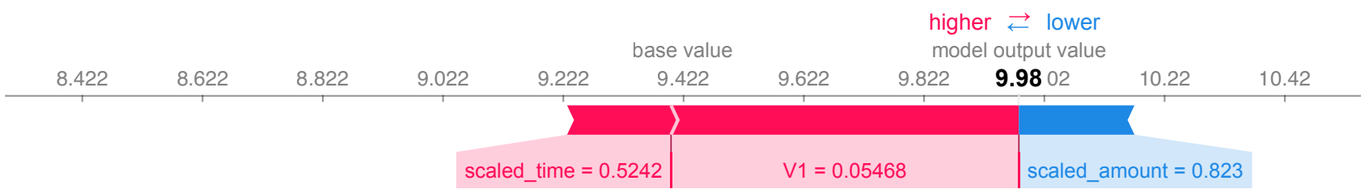
사기 거래 예측 결과

다음은 사기로 예측한 거래 중 한 가지다.

In [63]:
```
company_fraud_fraud_graph, company_fraud_fraud_data = CompanyVisualizer(company_importances, company_explainer, company_shap_values, X_test_c, 1)
company_fraud_fraud_graph
```

```
191544번 실제 값: 사기
예측한 사기 확률: 42.5%
```

Out[63]:



사기/정상 근거 비중 상위 5개:

In [64]:
```
FactorTable(company_fraud_fraud_data)
```

Out[64]:

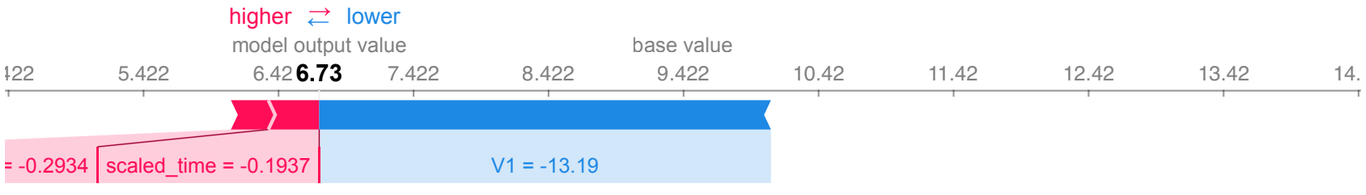
	사기 근거 항목	사기 근거 값	사기 근거 비중	정상 근거 항목	정상 근거 값	정상 근거 비중
1	scaled_amount	0.823028	0.192287	V1	0.054682	0.578553
2	V2	1.856500	-0.000000	scaled_time	0.524160	0.172835
3	V3	-4.075451	-0.000000	V2	1.856500	0.000000
4	V4	4.100098	-0.000000	V3	-4.075451	0.000000
5	V5	-0.800931	-0.000000	V4	4.100098	0.000000

사기로 예측한 다른 거래:

In [65]:
```
company_fraud_fraud_graph, company_fraud_fraud_data = CompanyVisualizer(company_importances, company_explainer, company_shap_values, X_test_c, 1)
company_fraud_fraud_graph
```

```
102441번 실제 값: 사기
예측한 사기 확률: 58.2%
```

Out[65]:



사기/정상 근거 비중 상위 5개:

In [66]:
```
FactorTable(company_fraud_fraud_data)
```

Out[66]:

	사기 근거 항목	사기 근거 값	사기 근거 비중	정상 근거 항목	정상 근거 값	정상 근거 비중
1	V1	-13.192671	3.346366	scaled_time	-0.193670	0.376814
2	V2	12.785971	-0.000000	scaled_amount	-0.293440	0.276938
3	V3	-9.906650	-0.000000	V2	12.785971	0.000000
4	V4	3.320337	-0.000000	V3	-9.906650	0.000000
5	V5	-4.801176	-0.000000	V4	3.320337	0.000000

비지도 학습인 Isolation Forest으로 모델링을 한 기업 사기 예측 모델은
정상과 사기 거래 구분력이 떨어진다.

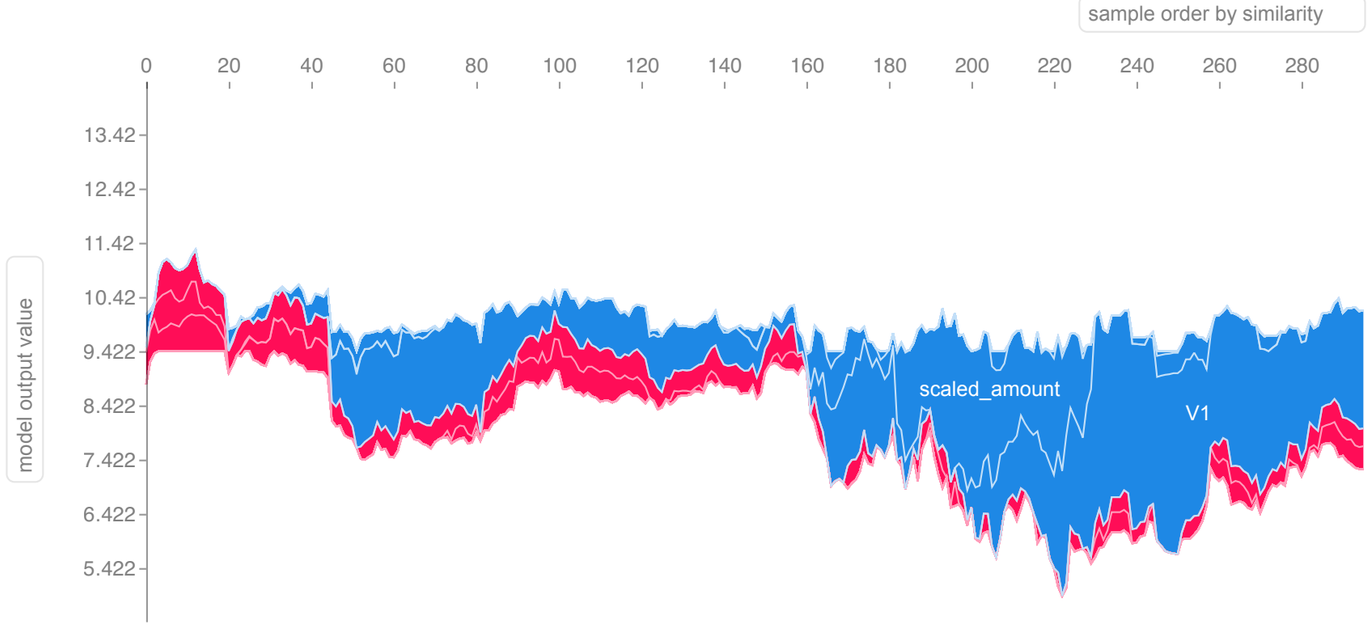
실제 KB 상시감사 데이터에 맞춘 모델링을 사용했기 때문에
Kaggle 데이터와 알맞지 않다.

하지만 정상으로 예측한 거래에서는 정상 근거 비중이 높은 것을 볼 수 있다.

전체 거래 예측 근거 그래프:

In [67]:
```
shap.force_plot(company_explainer.expected_value, company_shap_values, X_test_c)
```

Out[67]:



데이터 저장

데이터 종류:

- 모델
- 모델 성능 지표
- 테스트 결과
- 예측 결과

가계 사기

모델

In [68]:
```
house_fraud_model
```

Out[68]:
```
XGBClassifier(base_score=0.5, booster=None, colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eval_metric='aucpr',
              gamma=0.0011090219240890609, gpu_id=-1, importance_type='gain',
              interaction_constraints=None, learning_rate=0.8769006814172203,
              max_delta_step=0, max_depth=2, min_child_weight=1, missing=nan,
              monotone_constraints=None, n_estimators=94, n_jobs=5,
              num_parallel_tree=1, random_state=0, reg_alpha=4.479414500004676,
              reg_lambda=281.4547321092634, scale_pos_weight=1.0116959064327486,
              subsample=1, tree_method=None, validate_parameters=False,
              verbosity=None)
```

모델 성능 지표

In [69]:
```
house_fraud_model.performance_indicator
```

Out[69]:
```
{'confusion_matrix': array([[143,   3],
        [ 12, 138]]),
 'aucpr': 0.9875409155972033}
```

테스트 결과

In [70]:
```
pd.DataFrame(y_proba_house_fraud).set_index(X_test.index).head()
```

Out[70]:

	0	1
234632	0.019427	0.980573
42856	0.012273	0.987727
165728	0.956377	0.043623
167305	0.031369	0.968631
142696	0.719834	0.280166

예측 결과

In [71]:
```
pd.DataFrame(y_pred_house_fraud).set_index(X_test.index).head()
```

Out[71]:

	0	1
234632	0.019427	0.980573
42856	0.012273	0.987727
165728	0.956377	0.043623
167305	0.031369	0.968631
142696	0.719834	0.280166

가계 부실

모델

In [72]:
```
house_risk_model
```

Out[72]: <tensorflow.python.keras.engine.sequential.Sequential at 0x15a97a780>

모델 성능 지표

In [73]:
```
house_risk_model_info['aucpr']
```

Out[73]: 0.9799720080943911

예측 결과

In [74]:
```
pd.DataFrame(house_risk_result).set_index(X_test.index).head()
```

Out[74]:

	0
234632	고
42856	고
165728	저
167305	고
142696	저

예측 근거

In [75]:
```
house_importances.head()
```

Out[75]:

	index	사기여부	사기확률	부실확률	사기 근거 1	사기 근거 값 1	사기 근거 비중1	사기 근거 2	사기 근거 값2	사기 근거 비중2	...	정상 근거 비중2	정상 근거 3	정상 근거 값3	정상 근거 비중3	정상 근거 4	정상 근거 값4	정상 근거 비중4	정상 근거 5	정상 근거 값5	정상 근거 비중5
0	234632	1	98.099998	고	V14	-8.485795	1.345451	V4	5.342759	1.053857	...	0.056336	V8	0.123062	0.039585	V17	-1.372629	0.021900	V20	0.313332	0.019152
1	42856	1	98.800003	고	V14	-10.468677	1.290441	V4	7.690772	0.978565	...	0.007625	V28	-0.818970	0.004637	V13	1.184985	0.001701	V22	-1.271509	0.000247
2	165728	0	4.400000	저	V8	-4.136271	0.193060	V20	-0.898750	0.164918	...	1.029965	V10	-0.083725	0.579724	V12	0.536875	0.237924	V11	-1.443451	0.229618
3	167305	1	96.900002	고	V14	-8.490813	1.233970	V4	6.081321	1.152444	...	0.008585	V24	0.163718	0.007625	V13	-1.105710	0.001701	V22	-1.608272	0.000247
4	142696	0	28.000000	저	V4	2.459656	0.970674	V16	-0.642843	0.186302	...	0.406562	V10	-0.094109	0.339166	V12	1.278722	0.275123	V11	-0.268510	0.253928

5 rows × 34 columns

기업 사기

모델

In [76]:
```
company_fraud_model
```

Out[76]:
```
IsolationForest(max_features=0.1, max_samples=0.11239154014351381,
                n_estimators=1000, n_jobs=4, random_state=42)
```

모델 성능 지표

In [77]:
```
company_fraud_result
```

Out[77]:
```
{'offset': 41.99999999999998,
 'maxfeatures': 0.1,
 'maxsamples': 0.11239154014351381,
 'netimators': 1000,
 'prauc': 0.8982580075214015,
 'rocauc': 0.8850684931506849,
 'f1score': 0.8065573770491803,
 'precision': 0.7935483870967742,
 'recall': 0.82,
 'accuracy': 0.8006756756756757,
 'truenegative': 114,
 'falsepositive': 32,
 'falsenegative': 27,
 'truepositive': 123}
```

예측 결과

In [78]:
```
pd.DataFrame(y_pred_fraud_company).set_index(X_test_c.index).head()
```

Out[78]:

	0
234632	42.271467
42856	57.317881
165728	44.048021
167305	50.356199
142696	44.703457

예측 근거

In [79]:
```
company_importances.head()
```

Out[79]:

	index	사기여부	사기확률	사기 근거 1	사기 근거 값 1	사기 근거 비중1	사기 근거 2	사기 근거 값2	사기 근거 비중2	사기 근거 3	...	정상 근거 비중2	정상 근거 3	정상 근거 값 3	정상 근거 비중3	정상 근거 4	정상 근거 값4	정상 근거 비중4	정상 근거 5	정상 근거 값 5	정상 근거 비중5
0	234632	1	42.3	V1	1.261324	0.831859	scaled_time	0.744381	0.086073	V2	...	0.000000	V3	-5.435019	0.0	V4	5.342759	0.0	V5	1.447043	0.0
1	42856	1	57.3	V1	-11.682215	2.640094	scaled_amount	2.110948	1.085433	V2	...	0.000000	V3	-13.297109	0.0	V4	7.690772	0.0	V5	-10.889891	0.0
2	165728	0	44.0	V1	-0.901641	0.348754	V2	1.747927	-0.000000	V3	...	0.332681	V2	1.747927	0.0	V3	-0.532344	0.0	V4	-0.552070	0.0
3	167305	1	50.4	V1	-6.677212	2.330991	scaled_amount	1.172221	0.520987	V2	...	0.000000	V3	-7.193275	0.0	V4	6.081321	0.0	V5	-1.636071	0.0
4	142696	0	44.7	V1	-2.486331	1.674967	scaled_amount	0.866345	0.224476	V2	...	0.000000	V3	2.740996	0.0	V4	2.459656	0.0	V5	1.698475	0.0

5 rows × 33 columns

로컬 파일로 저장

In [80]:
```
from joblib import dump
```

가계 사기:

In [81]:
```
dump(house_fraud_model, 'result/house_fraud_model')
pd.DataFrame(y_proba_house_fraud).set_index(X_test.index).to_csv('result/house_fraud_y_test.csv', mode='w')
pd.DataFrame(y_pred_house_fraud).set_index(X_test.index).to_csv('result/house_fraud_y_pred.csv', mode='w')
```

가계 부실:

In [82]:
```
house_risk_model.save('result/house_risk_model')
dump(house_risk_model_info, 'result/house_risk_model_info')
pd.DataFrame(house_risk_result).set_index(X_test.index).to_csv('result/house_risk_y_pred.csv', mode='w')
```

가계 예측 근거:

In [83]:
```
house_importances.to_csv('result/house_importances.csv', mode='w')
```

기업 사기:

In [84]:
```
dump(company_fraud_model, 'result/company_fraud_model')
dump(company_fraud_result, 'result/company_fraud_result')
pd.DataFrame(y_pred_fraud_company).set_index(X_test_c.index).to_csv('result/company_fraud_y_pred.csv', mode='w')
```

기업 예측 근거:

In [85]:
```
company_importances.to_csv('result/company_importances.csv', mode='w')
```

(옵션) HDFS 저장

로컬에 저장한 결과 파일들을 HDFS로 저장한다.

Docker 설치

Docker Desktop: Download (https://www.docker.com/products/docker-desktop)

로컬 HDFS 실행

```
docker run -it --rm \
--name hdfs \
-p 22022:22 -p 8020:8020 \
-p 50010:50010 -p 50020:50020 \
-p 50070:50070 -p 50075:50075 \
mdouchement/hdfs;
```

Work directory 생성

```
hdfs dfs -mkdir -p /user/root;
hdfs dfs -ls -R /;
```

```
drwxr-xr-x   - root supergroup          0 2020-06-05 05:14 /user
drwxr-xr-x   - root supergroup          0 2020-06-05 05:14 /user/root
```

Web UI

localhost:50070 (http://localhost:50070)

Spark 세션 생성 및 HDFS 연결

In [86]:
```
from lib.spark import start_spark
from lib.config import setConfig
from lib.hdfs import HDFS
```

In [87]:
```
config = setConfig('localhost', '9083')
spark = start_spark(app_name='MyApp', config=config)
```

In [88]:
```
hdfs = HDFS(spark, 'localhost', 'localhost')
```

HDFS 파일 업로드

In [89]:
```
hdfs.upload('result', '/user/root')
```

In [90]:
```
hdfs.ls('/user/root', True)
```

```
drwxr-xr-x root supergroup          0 hdfs://localhost/user/root/result
-rw-r--r-- root supergroup    4131584 hdfs://localhost/user/root/result/company_fraud_model
-rw-r--r-- root supergroup        558 hdfs://localhost/user/root/result/company_fraud_result
-rw-r--r-- root supergroup       7317 hdfs://localhost/user/root/result/company_fraud_y_pred.csv
-rw-r--r-- root supergroup     102297 hdfs://localhost/user/root/result/company_importances.csv
-rw-r--r-- root supergroup      42858 hdfs://localhost/user/root/result/house_fraud_model
-rw-r--r-- root supergroup       8419 hdfs://localhost/user/root/result/house_fraud_y_pred.csv
-rw-r--r-- root supergroup       8419 hdfs://localhost/user/root/result/house_fraud_y_test.csv
-rw-r--r-- root supergroup     132419 hdfs://localhost/user/root/result/house_importances.csv
-rw-r--r-- root supergroup     356508 hdfs://localhost/user/root/result/house_risk_model
-rw-r--r-- root supergroup       1178 hdfs://localhost/user/root/result/house_risk_model_info
-rw-r--r-- root supergroup       3121 hdfs://localhost/user/root/result/house_risk_y_pred.csv
```

Spark, HDFS 연결 종료

In [91]:
```
hdfs.close()
spark.stop()
```

HDFS 종료

터미널을 종료하면 자동으로 도커 컨테이너는 삭제된다.

```
exit
```