미래에셋 자산운용

AI 혁신본부 AI 혁신팀 사전과제

개요

- 일정기간동안 측정된 주식 데이터셋이 주어집니다.
- Train set과 Test set이 주어집니다.
- Train set에 대해서 학습을 진행하고, Test Set을 통한 prediction을 만드는 것이 목표입니다.

데이터 설명

Input Data

- 데이터셋의 각 column은 날짜정보와 종목정보, 그리고 Feature set으로 이루어져 있습니다. Feature set은 blur 처리되어 있습니다.
- Feature는 각 종목들의 유의미하다고 판단되는 데이터 값으로 이루어져 있습니다.

Target Data

- Train set에 대해서는 정답 데이터 2개가 주어지고, Test set에 대해서는 정답이 주어지지 않습니다.
- 정답 데이터들은 각 샘플 시간 기준으로 다음 단위시간(T) 수익률로 만들어집니다.
  - 정답 데이터1: 단위시간(T) 수익률 (train_target.csv) -> Regression
  - 정답 데이터2: 특정 한 시점에서 종목들의 단위 시간 수익률(정답데이터 1)을 5분위로 나누어 분류한 Target (train_target2.csv) -> Classification

세부사항

- 어떤 정답을 학습시키느냐에 따라 regression 접근, 혹은 5분위 중 어떤 위치에 있을지 예측하는 Classification 접근법이 있습니다.
- 이를 포함해서 수험자 재량에 따라 데이터에 변형을 가하는 등의 접근 방식을 써도 좋습니다.
  - 예를 들어 classification 문제로 예시를 들면 수익률을 5분위 대신 2분위로 나누어서 binary classification으로 변형해도 좋습니다. 다만 변형할 경우 해당 부분에 대한 설명의 기재를 부탁드립니다.
  - 어떠한 접근 방법이든 하나만 선택해도 됩니다.
- 모든 데이터를 학습에 사용할 필요는 없습니다.
  - 예를 들어 특정 Feature는 필요없다고 판단하면 버리셔도 무방하고, 새로운 Feature를 만들어서 사용하셔도 됩니다.
- 실제 모델 결과보다 모델을 만들기까지의 과정이 중요합니다.
- 어떠한 논리로 분석을 진행하였는지 설명을 세부적으로 적어주시길 바랍니다. (코멘트/주석 방식 or 보고서)
  - SCORE보다는 적어주신 주석/코멘트나 보고서 내용이 평가 비중이 높습니다.
- 딥러닝 프레임워크(Tensorflow, Pytorch, Keras, Theano 등)를 사용한 모델링 과정을 하나 이상 넣어주시길 바랍니다.
- Deep learning 모델 외에도 추가적으로 모델을 사용하는 것은 제한이 없습니다.

제출 양식

- Test set에 대한 모델의 예측값을 csv 형태로 저장하여 소스코드와 같이 제출해야합니다.
- 소스코드 형식은 .ipynb(jupyter notebook) 형식이 선호되고 .py 파일도 제출가능합니다.
- 결과물들이 모두 포함된 압축파일(.zip)로 제출해주십시요.

추가 문의 사항:

- AI 혁신팀 고정욱 매니저
- 02-3774-2262(OH : 9:00 ~ 18:00)
- jungwook.ko@miraeasset.com

- 아래 소스코드는 제출 양식과 진행 방식의 전달을 위해 간략하게 진행한 예시입니다.

Data load

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

In [2]:
```
X_train = pd.read_csv('./data/train_data.csv') #훈련 데이터
Y_train = pd.read_csv('./data/train_target.csv') # 훈련 데이터에 대한 정답데이터 for regression
Y2_train = pd.read_csv('./data/train_target2.csv') # 훈련 데이터에 대한 정답데이터 for classification
X_test = pd.read_csv('./data/test_data.csv') # 테스트 데이터
```

In [3]:
```
X_train.tail()
```

Out[3]:

	td	code	F001	F002	F003	F004	F005	F006	F007	F008	...	F037	F038	F039	F040	
83559	T274	A791	5.411604	0.000000	0.560828	-0.101983	-8.063605	0.029134	0.026586	-6.890195	...	-18.278760	-0.000252	-0.009729	4.0	-0.00
83560	T274	A793	82.562142	0.000000	8.971392	-0.008897	22.222222	0.206389	0.033859	19.184370	...	64.794007	0.021994	-0.004664	2.0	0.11
83561	T274	A794	0.186385	0.000893	2.374996	-0.052632	25.110132	0.029708	0.030444	9.984827	...	30.275229	0.093428	0.000383	5.0	-0.02
83562	T274	A795	-5.515355	-0.005224	4.898941	-0.037037	6.101190	0.097354	0.023531	-0.481948	...	-1.109570	0.031893	-0.009076	2.0	-0.00
83563	T274	A796	3.056981	-0.005908	2.252730	-0.035714	-17.596567	0.140226	0.035148	-6.005738	...	-20.661157	-0.003084	-0.005149	2.0	0.05

5 rows × 48 columns

In [4]:
```
Y_train.tail()
```

Out[4]:

	td	code	target
83559	T274	A791	0.026318
83560	T274	A793	-0.034091
83561	T274	A794	0.001761
83562	T274	A795	0.040673
83563	T274	A796	0.005208

In [5]:
```
Y2_train.tail()
```

Out[5]:

	td	code	target
83559	T274	A791	3
83560	T274	A793	0
83561	T274	A794	2
83562	T274	A795	3
83563	T274	A796	2

In [6]:
```
X_test.tail()
```

Out[6]:

	td	code	F001	F002	F003	F004	F005	F006	F007	F008	...	F037	F038	F039	F040	
11613	T310	A790	0.407656	NaN	9.867299	0.000000	-6.334372	NaN	0.036413	-7.040488	...	-25.454545	-0.024005	NaN	7.0	
11614	T310	A793	43.023917	-0.003035	9.985884	-0.008333	-3.834356	0.055146	0.040303	-2.480406	...	-9.913793	0.065610	-0.009397	2.0	-0.01
11615	T310	A794	-4.534972	0.000000	1.751161	0.000000	8.661417	-0.016977	0.027230	-0.698376	...	-1.895735	0.006033	-0.020032	5.0	0.01
11616	T310	A795	-17.974223	0.000000	4.192958	0.003632	11.420983	0.100086	0.025969	-8.893335	...	-23.308958	0.000907	0.001045	2.0	-0.01
11617	T310	A796	-1.651783	-0.029640	2.452572	-0.433829	-4.824561	0.018919	0.040217	-5.178204	...	-21.090909	-0.048585	-0.012903	2.0	-0.04

5 rows × 48 columns

In [7]:
```
X_train = X_train.set_index(['td', 'code'])
Y_train = Y_train.set_index(['td', 'code'])
Y2_train = Y2_train.set_index(['td', 'code'])
X_test = X_test.set_index(['td', 'code'])
```

EDA

In [8]:
```
X_train.shape, Y_train.shape, Y2_train.shape, X_test.shape
```

Out[8]:
```
((83564, 46), (83564, 1), (83564, 1), (11618, 46))
```

In [9]:
```
X_train.describe()
```

Out[9]:

	F001	F002	F003	F004	F005	F006	F007	F008	F009	F010	...
count	80147.000000	72494.000000	82138.000000	83448.000000	83448.000000	69456.000000	82518.000000	82504.000000	71432.000000	70884.000000	...
mean	7.894370	0.000915	3.212757	0.202291	2.075079	0.052969	0.024497	1.753955	0.909854	-0.000027	...
std	22.101300	0.039230	17.267118	2.020238	14.543002	0.069505	0.011561	8.342232	0.601707	0.016406	...
min	-52.092279	-3.832419	0.169382	-76.840000	-59.385189	-0.866186	0.005690	-26.092628	-0.165837	-0.367137	...
25%	-6.103297	-0.001195	0.919447	-0.020000	-5.553225	0.020651	0.017816	-4.073285	0.440529	-0.001739	...
50%	3.184114	0.000000	1.551248	0.000000	0.440529	0.042782	0.022341	0.917061	0.800000	0.000000	...
75%	16.197738	0.000400	3.090994	0.059686	7.606289	0.073355	0.028253	6.597034	1.234568	0.001397	...
max	470.723992	4.133958	983.606913	345.800000	691.925065	1.670955	0.383658	109.968661	10.000000	2.307626	...

8 rows × 46 columns

In [10]:
```
X_test.describe()
```

Out[10]:

	F001	F002	F003	F004	F005	F006	F007	F008	F009	F010	...
count	11086.000000	9532.000000	11562.000000	11617.000000	11617.000000	9177.000000	11390.000000	11507.000000	9470.000000	9356.000000	...
mean	5.547566	-0.001151	3.907227	0.902523	-0.122685	0.050943	0.027775	-1.136299	0.976132	-0.000940	...
std	20.564398	0.020017	8.417691	14.744368	15.264813	0.068831	0.013069	7.346431	0.702352	0.011895	...
min	-29.721986	-0.976340	0.161449	-38.600000	-49.504950	-0.825617	0.007313	-24.110365	0.012950	-0.502640	...
25%	-7.672988	-0.001494	0.852407	-0.023139	-8.493590	0.018199	0.019740	-6.206570	0.409836	-0.002002	...
50%	0.693708	0.000000	1.727881	0.000000	-1.408451	0.040220	0.025025	-2.023613	0.819672	-0.000167	...
75%	12.594850	0.000157	3.967130	0.014085	6.226415	0.070894	0.031825	2.922284	1.408451	0.000461	...
max	196.038447	0.392012	189.233466	396.807830	273.619632	1.537437	0.124297	70.851211	11.111111	0.157010	...

8 rows × 46 columns

In [11]:
```
# imputation with -1
X_train.fillna(-1, inplace = True)
X_test.fillna(-1, inplace = True)
```

Modeling

- classification or regression

In [12]:
```
#use classifier in example
from sklearn.ensemble import ExtraTreesClassifier
```

```
C:\Users\Mirae\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosting.py:29: DeprecationWarning: numpy.core.umath_t
ests is an internal NumPy module and should not be imported. It will be removed in a future NumPy release.
  from numpy.core.umath_tests import innerld
```

In [13]:
```
model = ExtraTreesClassifier(n_estimators=100, max_depth = 20)
```

In [14]:
```
# 만약 regression이라면 Y_train 사용
model.fit(X_train.values, Y2_train.values)
```

```
C:\Users\Mirae\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: DataConversionWarning: A column-vector y was passed w
hen a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
```

Out[14]:
```
ExtraTreesClassifier(bootstrap=False, class_weight=None, criterion='gini',
           max_depth=20, max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

In [15]:
```
print(model.score(X_train.values, Y2_train.values))
```

```
0.8002728447656886
```

Make prediction

In [16]:
```
pred = model.predict(X_test)
```

Make submission

In [17]:
```
submission = pd.DataFrame(pred, columns = ['target'], index = X_test.index)
```

In [18]:
```
submission.head()
```

Out[18]:

		target
td	**code**	
T275	**A005**	0
	A006	1
	A007	2
	A008	4
	A012	0