08_making_out_of_sample_predictions

September 29, 2021

1 Long-Short Strategy, Part 5: Generating out-of-sample predictions

In this section, we'll start designing, implementing, and evaluating a trading strategy for US equities driven by daily return forecasts produced by gradient boosting models.

As in the previous examples, we'll lay out a framework and build a specific example that you can adapt to run your own experiments. There are numerous aspects that you can vary, from the asset class and investment universe to more granular aspects like the features, holding period, or trading rules. See, for example, the **Alpha Factor Library** in the Appendix for numerous additional features.

We'll keep the trading strategy simple and only use a single ML signal; a real-life application will likely use multiple signals from different sources, such as complementary ML models trained on different datasets or with different lookahead or lookback periods. It would also use sophisticated risk management, from simple stop-loss to value-at-risk analysis.

Six notebooks cover our workflow sequence:

1. preparing_the_model_data: we engineer a few simple features from the Quandl Wiki data
2. trading_signals_with_lightgbm_and_catboost: we tune hyperparameters for LightGBM and CatBoost to select a model, using 2015/16 as our validation period.
3. evaluate_trading_signals: we compare the cross-validation performance using various metrics to select the best model.
4. model_interpretation: we take a closer look at the drivers behind the best model's predictions.
5. `making_out_of_sample_predictions` (this noteboook): we predict returns for our out-of-sample period 2017.
6. backtesting_with_zipline: evaluate the historical performance of a long-short strategy based on our predictive signals using Zipline.

1.1 Imports & Settings

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

[40]:
```
%matplotlib inline

from time import time
import sys, os
```

```
from pathlib import Path

import pandas as pd
from scipy.stats import spearmanr

import lightgbm as lgb
from catboost import Pool, CatBoostRegressor

import matplotlib.pyplot as plt
import seaborn as sns
```

[3]:
```
sys.path.insert(1, os.path.join(sys.path[0], '..'))
from utils import MultipleTimeSeriesCV
```

[4]:
```
sns.set_style('whitegrid')
```

[5]:
```
YEAR = 252
idx = pd.IndexSlice
```

[6]:
```
scope_params = ['lookahead', 'train_length', 'test_length']
daily_ic_metrics = ['daily_ic_mean', 'daily_ic_mean_n', 'daily_ic_median',␣
↪'daily_ic_median_n']
lgb_train_params = ['learning_rate', 'num_leaves', 'feature_fraction',␣
↪'min_data_in_leaf']
catboost_train_params = ['max_depth', 'min_child_samples']
```

1.2 Generate LightGBM predictions

1.2.1 Model Configuration

[7]:
```
base_params = dict(boosting='gbdt',
                   objective='regression',
                   verbose=-1)

categoricals = ['year', 'month', 'sector', 'weekday']
```

[8]:
```
lookahead = 1
store = Path('data/predictions.h5')
```

1.2.2 Get Data

[9]:
```
data = pd.read_hdf('data.h5', 'model_data').sort_index()
```

[10]:
```
labels = sorted(data.filter(like='_fwd').columns)
features = data.columns.difference(labels).tolist()
label = f'r{lookahead:02}_fwd'
```

[11]:
```
data = data.loc[idx[:, '2010':], features + [label]].dropna()
```

[12]:
```
for feature in categoricals:
    data[feature] = pd.factorize(data[feature], sort=True)[0]
```

[13]:
```
lgb_data = lgb.Dataset(data=data[features],
                       label=data[label],
                       categorical_feature=categoricals,
                       free_raw_data=False)
```

1.2.3 Generate predictions

[14]:
```
lgb_ic = pd.read_hdf('data/model_tuning.h5', 'lgb/ic')
lgb_daily_ic = pd.read_hdf('data/model_tuning.h5', 'lgb/daily_ic')
```

[15]:
```
def get_lgb_params(data, t=5, best=0):
    param_cols = scope_params[1:] + lgb_train_params + ['boost_rounds']
    df = data[data.lookahead==t].sort_values('ic', ascending=False).iloc[best]
    return df.loc[param_cols]
```

[ ]:
```
for position in range(10):
    params = get_lgb_params(lgb_daily_ic,
                            t=lookahead,
                            best=position)

    params = params.to_dict()

    for p in ['min_data_in_leaf', 'num_leaves']:
        params[p] = int(params[p])
    train_length = int(params.pop('train_length'))
    test_length = int(params.pop('test_length'))
    num_boost_round = int(params.pop('boost_rounds'))
    params.update(base_params)

    print(f'\nPosition: {position:02}')

    # 1-year out-of-sample period
    n_splits = int(YEAR / test_length)
    cv = MultipleTimeSeriesCV(n_splits=n_splits,
                              test_period_length=test_length,
                              lookahead=lookahead,
                              train_period_length=train_length)

    predictions = []
    start = time()
    for i, (train_idx, test_idx) in enumerate(cv.split(X=data), 1):
        print(i, end=' ', flush=True)
```

```
        lgb_train = lgb_data.subset(used_indices=train_idx.tolist(),
                                    params=params).construct()

        model = lgb.train(params=params,
                          train_set=lgb_train,
                          num_boost_round=num_boost_round,
                          verbose_eval=False)

        test_set = data.iloc[test_idx, :]
        y_test = test_set.loc[:, label].to_frame('y_test')
        y_pred = model.predict(test_set.loc[:, model.feature_name()])
        predictions.append(y_test.assign(prediction=y_pred))

    if position == 0:
        test_predictions = (pd.concat(predictions)
                            .rename(columns={'prediction': position}))
    else:
        test_predictions[position] = pd.concat(predictions).prediction

by_day = test_predictions.groupby(level='date')
for position in range(10):
    if position == 0:
        ic_by_day = by_day.apply(lambda x: spearmanr(
            x.y_test, x[position])[0]).to_frame()
    else:
        ic_by_day[position] = by_day.apply(
            lambda x: spearmanr(x.y_test, x[position])[0])
print(ic_by_day.describe())
test_predictions.to_hdf(store, f'lgb/test/{lookahead:02}')
```

1.3 Generate CatBoost predictions

[27]:
```
lookaheads = [1, 5, 21]
```

[28]:
```
label_dict = dict(zip(lookaheads, labels))
```

1.3.1 Model Configuration

[17]:
```
lookahead = 1
store = Path('data/predictions.h5')
```

1.3.2 Get Data

[20]:
```
data = pd.read_hdf('data.h5', 'model_data').sort_index()
```

[21]:
```
labels = sorted(data.filter(like='_fwd').columns)
features = data.columns.difference(labels).tolist()
label = f'r{lookahead:02}_fwd'
```

[22]:
```
data = data.loc[idx[:, '2010':], features + [label]].dropna()
```

[23]:
```
for feature in categoricals:
    data[feature] = pd.factorize(data[feature], sort=True)[0]
```

[31]:
```
cat_cols_idx = [data.columns.get_loc(c) for c in categoricals]
```

[32]:
```
catboost_data = Pool(label=data[label],
                     data=data.drop(label, axis=1),
                     cat_features=cat_cols_idx)
```

1.3.3 Generate predictions

[36]:
```
catboost_ic = pd.read_hdf('data/model_tuning.h5', 'catboost/ic')
catboost_ic_avg = pd.read_hdf('data/model_tuning.h5', 'catboost/daily_ic')
```

[37]:
```
def get_cb_params(data, t=5, best=0):
    param_cols = scope_params[1:] + catboost_train_params + ['boost_rounds']
    df = data[data.lookahead==t].sort_values('ic', ascending=False).iloc[best]
    return df.loc[param_cols]
```

[ ]:
```
for position in range(10):
    params = get_cb_params(catboost_ic_avg,
                    t=lookahead,
                    best=position)

    params = params.to_dict()

    for p in ['max_depth', 'min_child_samples']:
        params[p] = int(params[p])
    train_length = int(params.pop('train_length'))
    test_length = int(params.pop('test_length'))
    num_boost_round = int(params.pop('boost_rounds'))
    params['task_type'] = 'GPU'

    print(f'\nPosition: {position:02}')

    # 1-year out-of-sample period
    n_splits = int(YEAR / test_length)
    cv = MultipleTimeSeriesCV(n_splits=n_splits,
                              test_period_length=test_length,
                              lookahead=lookahead,
                              train_period_length=train_length)
```

```
    predictions = []
    start = time()
    for i, (train_idx, test_idx) in enumerate(cv.split(X=data), 1):
        print(i, end=' ', flush=True)
        train_set = catboost_data.slice(train_idx.tolist())

        model = CatBoostRegressor(**params)
        model.fit(X=train_set,
                  verbose_eval=False)

        test_set = data.iloc[test_idx, :]
        y_test = test_set.loc[:, label].to_frame('y_test')
        y_pred = model.predict(test_set.loc[:, model.feature_names_])
        predictions.append(y_test.assign(prediction=y_pred))

    if position == 0:
        test_predictions = (pd.concat(predictions)
                            .rename(columns={'prediction': position}))
    else:
        test_predictions[position] = pd.concat(predictions).prediction

by_day = test_predictions.groupby(level='date')
for position in range(10):
    if position == 0:
        ic_by_day = by_day.apply(lambda x: spearmanr(x.y_test, x[position])[0]).
 ↪to_frame()
    else:
        ic_by_day[position] = by_day.apply(lambda x: spearmanr(x.y_test,␣
 ↪x[position])[0])
print(ic_by_day.describe())
test_predictions.to_hdf(store, f'catboost/test/{lookahead:02}')
```

[ ]: