08_predicting_price_movements_with_logistic_regression

September 29, 2021

1 Predicting stock price moves with Logistic Regression

1.1 Imports & Settings

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

[2]:
```
from pathlib import Path
import sys, os
from time import time

import pandas as pd
import numpy as np

from scipy.stats import spearmanr

from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

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

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

[4]:
```
sns.set_style('darkgrid')
idx = pd.IndexSlice
```

[5]:
```
YEAR = 252
```

1.2 Load Data

[6]:
```
with pd.HDFStore('data.h5') as store:
    data = (store['model_data']
            .dropna()
```

```
            .drop(['open', 'close', 'low', 'high'], axis=1))
data = data.drop([c for c in data.columns if 'year' in c or 'lag' in c], axis=1)
```

1.2.1 Select Investment Universe

[7]:
```
data = data[data.dollar_vol_rank<100]
```

1.2.2 Create Model Data

[8]:
```
y = data.filter(like='target')
X = data.drop(y.columns, axis=1)
X = X.drop(['dollar_vol', 'dollar_vol_rank', 'volume', 'consumer_durables'],␣
 ↪axis=1)
```

1.3 Logistic Regression

1.3.1 Define cross-validation parameters

[9]:
```
train_period_length = 63
test_period_length = 10
lookahead =1
n_splits = int(3 * YEAR/test_period_length)

cv = MultipleTimeSeriesCV(n_splits=n_splits,
                          test_period_length=test_period_length,
                          lookahead=lookahead,
                          train_period_length=train_period_length)
```

[10]:
```
target = f'target_{lookahead}d'
```

[11]:
```
y.loc[:, 'label'] = (y[target] > 0).astype(int)
y.label.value_counts()
```

[11]:
```
1    56486
0    53189
Name: label, dtype: int64
```

[12]:
```
Cs = np.logspace(-5, 5, 11)
```

[13]:
```
cols = ['C', 'date', 'auc', 'ic', 'pval']
```

1.3.2 Run cross-validation

[14]:
```
%%time
log_coeffs, log_scores, log_predictions = {}, [], []
for C in Cs:
    print(C)
```

```
    model = LogisticRegression(C=C,
                               fit_intercept=True,
                               random_state=42,
                               n_jobs=-1)

    pipe = Pipeline([
        ('scaler', StandardScaler()),
        ('model', model)])
    ics = aucs = 0
    start = time()
    coeffs = []
    for i, (train_idx, test_idx) in enumerate(cv.split(X), 1):
        X_train, y_train, = X.iloc[train_idx], y.label.iloc[train_idx]
        pipe.fit(X=X_train, y=y_train)
        X_test, y_test = X.iloc[test_idx], y.label.iloc[test_idx]
        actuals = y[target].iloc[test_idx]
        if len(y_test) < 10 or len(np.unique(y_test)) < 2:
            continue
        y_score = pipe.predict_proba(X_test)[:, 1]

        auc = roc_auc_score(y_score=y_score, y_true=y_test)
        actuals = y[target].iloc[test_idx]
        ic, pval = spearmanr(y_score, actuals)

        log_predictions.append(y_test.to_frame('labels').assign(
            predicted=y_score, C=C, actuals=actuals))
        date = y_test.index.get_level_values('date').min()
        log_scores.append([C, date, auc, ic * 100, pval])
        coeffs.append(pipe.named_steps['model'].coef_)
        ics += ic
        aucs += auc
        if i % 10 == 0:
            print(f'\t{time()-start:5.1f} | {i:03} | {ics/i:>7.2%} | {aucs/i:>7.
↪2%}')

    log_coeffs[C] = np.mean(coeffs, axis=0).squeeze()
```

```
1e-05
          3.9 | 010 |  -0.31% |  50.42%
          5.3 | 020 |   1.89% |  51.83%
          7.0 | 030 |   2.84% |  52.01%
          8.5 | 040 |   3.29% |  51.98%
         10.2 | 050 |   3.97% |  52.44%
         11.6 | 060 |   3.96% |  52.27%
         13.1 | 070 |   4.73% |  52.59%
0.0001
          1.8 | 010 |  -0.06% |  50.62%
```

```
          3.8 | 020 |   2.23% |  52.01%
          5.2 | 030 |   3.20% |  52.26%
          7.0 | 040 |   3.34% |  52.08%
          9.0 | 050 |   4.02% |  52.53%
         10.6 | 060 |   4.02% |  52.33%
         12.1 | 070 |   4.83% |  52.67%
0.001
          1.8 | 010 |   0.42% |  50.96%
          3.7 | 020 |   2.53% |  52.14%
          5.4 | 030 |   3.58% |  52.48%
          6.8 | 040 |   3.17% |  52.07%
          8.5 | 050 |   3.83% |  52.49%
         10.2 | 060 |   4.03% |  52.33%
         12.0 | 070 |   4.88% |  52.70%
0.01
          1.7 | 010 |   0.68% |  51.13%
          3.2 | 020 |   2.39% |  51.97%
          4.8 | 030 |   3.64% |  52.41%
          6.6 | 040 |   3.12% |  51.94%
          8.3 | 050 |   3.92% |  52.46%
         10.3 | 060 |   4.16% |  52.30%
         12.4 | 070 |   4.91% |  52.64%
0.1
          2.1 | 010 |   0.65% |  51.11%
          4.0 | 020 |   2.17% |  51.80%
          5.8 | 030 |   3.49% |  52.28%
          7.7 | 040 |   2.94% |  51.80%
          9.5 | 050 |   3.74% |  52.31%
         11.4 | 060 |   3.93% |  52.15%
         13.1 | 070 |   4.59% |  52.46%
1.0
          2.0 | 010 |   0.60% |  51.07%
          4.1 | 020 |   2.13% |  51.77%
          6.1 | 030 |   3.45% |  52.26%
          7.9 | 040 |   2.90% |  51.77%
          9.8 | 050 |   3.68% |  52.28%
         11.6 | 060 |   3.86% |  52.11%
         13.3 | 070 |   4.50% |  52.41%
10.0
          1.7 | 010 |   0.59% |  51.07%
          3.5 | 020 |   2.12% |  51.76%
          5.3 | 030 |   3.45% |  52.25%
          6.8 | 040 |   2.89% |  51.77%
          8.6 | 050 |   3.67% |  52.27%
         10.4 | 060 |   3.85% |  52.11%
         12.2 | 070 |   4.49% |  52.41%
100.0
          1.8 | 010 |   0.59% |  51.07%
```

```
          3.4 | 020 |   2.11% |  51.76%
          5.0 | 030 |   3.45% |  52.25%
          6.8 | 040 |   2.89% |  51.77%
          8.7 | 050 |   3.67% |  52.27%
         10.4 | 060 |   3.85% |  52.11%
         11.9 | 070 |   4.49% |  52.41%
1000.0
          1.4 | 010 |   0.59% |  51.07%
          2.8 | 020 |   2.11% |  51.76%
          4.3 | 030 |   3.44% |  52.25%
          5.7 | 040 |   2.89% |  51.77%
          7.1 | 050 |   3.67% |  52.27%
          8.5 | 060 |   3.85% |  52.11%
          9.9 | 070 |   4.49% |  52.41%
10000.0
          1.4 | 010 |   0.59% |  51.07%
          2.8 | 020 |   2.11% |  51.76%
          4.3 | 030 |   3.45% |  52.25%
          5.8 | 040 |   2.89% |  51.77%
          7.2 | 050 |   3.67% |  52.27%
          8.7 | 060 |   3.85% |  52.11%
         10.1 | 070 |   4.49% |  52.41%
100000.0
          1.4 | 010 |   0.59% |  51.07%
          2.8 | 020 |   2.11% |  51.76%
          4.2 | 030 |   3.45% |  52.25%
          5.6 | 040 |   2.89% |  51.77%
          7.0 | 050 |   3.67% |  52.27%
          8.5 | 060 |   3.85% |  52.11%
          9.8 | 070 |   4.49% |  52.41%
CPU times: user 1min 34s, sys: 1.21 s, total: 1min 35s
Wall time: 2min 18s
```

1.3.3 Evaluate Results

[15]:
```
log_scores = pd.DataFrame(log_scores, columns=cols)
log_scores.to_hdf('data.h5', 'logistic/scores')

log_coeffs = pd.DataFrame(log_coeffs, index=X.columns).T
log_coeffs.to_hdf('data.h5', 'logistic/coeffs')

log_predictions = pd.concat(log_predictions)
log_predictions.to_hdf('data.h5', 'logistic/predictions')
```

[16]:
```
log_scores = pd.read_hdf('data.h5', 'logistic/scores')
```

[17]:
```
log_scores.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 825 entries, 0 to 824
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   C       825 non-null    float64
 1   date    825 non-null    datetime64[ns]
 2   auc     825 non-null    float64
 3   ic      825 non-null    float64
 4   pval    825 non-null    float64
dtypes: datetime64[ns](1), float64(4)
memory usage: 38.7 KB
```

[18]:
```
log_scores.groupby('C').auc.describe()
```

[18]:

C	count	mean	std	min	25%	50%	75%	max
0.00001	75.0	0.524316	0.036131	0.432491	0.499719	0.519363	0.546030	0.625816
0.00010	75.0	0.525045	0.035658	0.442738	0.501438	0.520094	0.546003	0.635535
0.00100	75.0	0.525448	0.036371	0.438676	0.500371	0.520211	0.545218	0.641011
0.01000	75.0	0.525076	0.036905	0.435870	0.500481	0.522237	0.550842	0.641632
0.10000	75.0	0.523552	0.036496	0.427256	0.499836	0.523206	0.549261	0.621749
1.00000	75.0	0.523159	0.036445	0.424981	0.498530	0.521782	0.549033	0.616444
10.00000	75.0	0.523109	0.036423	0.424850	0.498587	0.521569	0.548962	0.615767
100.00000	75.0	0.523104	0.036422	0.424826	0.498593	0.521573	0.548949	0.615716
1000.00000	75.0	0.523102	0.036423	0.424812	0.498597	0.521582	0.548951	0.615728
10000.00000	75.0	0.523102	0.036423	0.424812	0.498595	0.521582	0.548953	0.615728
100000.00000	75.0	0.523102	0.036423	0.424812	0.498595	0.521582	0.548953	0.615728

1.3.4 Plot Validation Scores

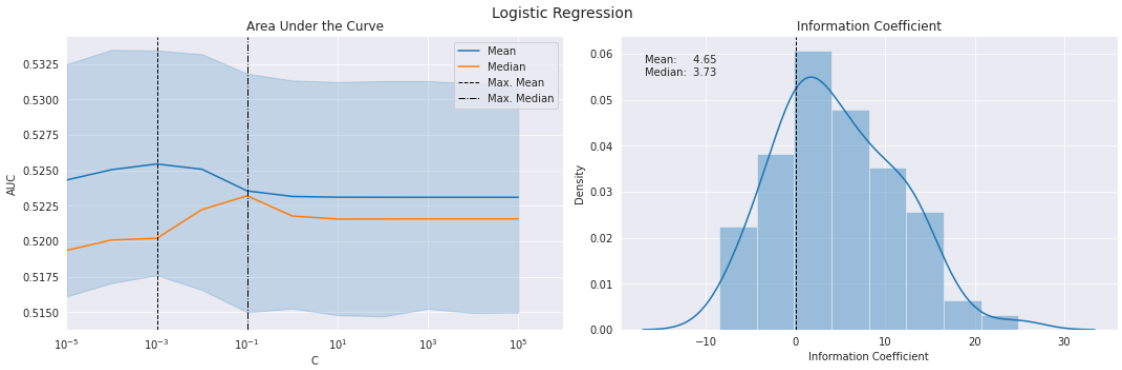
[19]:
```
def plot_ic_distribution(df, ax=None):
    if ax is not None:
        sns.distplot(df.ic, ax=ax)
    else:
        ax = sns.distplot(df.ic)
    mean, median = df.ic.mean(), df.ic.median()
    ax.axvline(0, lw=1, ls='--', c='k')
    ax.text(x=.05, y=.9, s=f'Mean: {mean:8.2f}\nMedian: {median:5.2f}',
            horizontalalignment='left',
            verticalalignment='center',
            transform=ax.transAxes)
    ax.set_xlabel('Information Coefficient')
    sns.despine()
    plt.tight_layout()
```

[20]:
```
fig, axes= plt.subplots(ncols=2, figsize=(15, 5))

sns.lineplot(x='C', y='auc', data=log_scores, estimator=np.mean, label='Mean',␣
 ↪ax=axes[0])
by_alpha = log_scores.groupby('C').auc.agg(['mean', 'median'])
best_auc = by_alpha['mean'].idxmax()
by_alpha['median'].plot(logx=True, ax=axes[0], label='Median', xlim=(10e-6,␣
 ↪10e5))
axes[0].axvline(best_auc, ls='--', c='k', lw=1, label='Max. Mean')
axes[0].axvline(by_alpha['median'].idxmax(), ls='-.', c='k', lw=1, label='Max.␣
 ↪Median')
axes[0].legend()
axes[0].set_ylabel('AUC')
axes[0].set_xscale('log')
axes[0].set_title('Area Under the Curve')

plot_ic_distribution(log_scores[log_scores.C==best_auc], ax=axes[1])
axes[1].set_title('Information Coefficient')

fig.suptitle('Logistic Regression', fontsize=14)
sns.despine()
fig.tight_layout()
fig.subplots_adjust(top=.9);
```


Logistic Regression
Area Under the Curve
Mean
Median
Max. Mean
Max. Median
AUC
C
Information Coefficient
Mean: 4.65
Median: 3.73
Density
Information Coefficient