Sarit Maitra

1.5K Followers About

BRENT CRUDE OIL FUTURES PRICE MOVEMENTS PREDICTION

Classification Algorithm to Predict Price Movements & Strategy Evaluation Technique

Algorithmic trading strategy evaluation based on crude oil data set

Sarit Maitra Dec 30, 2020 · 7 min read ★

Image bu author

Prediction and classification are important and of great interest for the simple fact that successful prediction of stock prices lead to rewarding benefits. However, there is no universal common set of rules but a series of highly complicated and quite difficult tasks are involved for such prediction.

Here, we will show a simple use case to showcase how classification rule can be applied to obtain a trading strategy and conclude with a performance testing of the strategy by

running a simple script.

Let us load the data from Quandl.

```
BC = BC.loc['2010-01-01':,]
BC.sort_index(ascending=True, inplace=True)
BC.tail()
```

```
Brent Crude Futures, Continuous Contract
```

Date	Open	High	Low	Settle	Change	Wave	Volume	Prev. Day Open Interest	EFP Volume	EFS Volume	Block Volume
2021-03-15	69.08	70.03	67.82	68.88	-0.34	68.84	258336.0	428150.0	600.0	NaN	3659.0
2021-03-16	68.81	68.94	67.36	68.39	-0.49	68.07	245273.0	409207.0	1200.0	NaN	2520.0
2021-03-17	68.39	68.89	66.96	68.00	-0.39	67.82	304547.0	402603.0	3000.0	NaN	2209.0
2021-03-18	67.77	68.15	61.45	63.28	-4.72	65.25	509041.0	385091.0	2400.0	NaN	4642.0
2021-03-19	62.99	64.95	62.07	64.53	1.25	63.48	441967.0	323686.0	237.0	NaN	3910.0

Let us investigate the data and do necessary pre-processing for machine learning.

```
print(BC.columns); print(BC.shape)
```

```
Index(['Open', 'High', 'Low', 'Settle', 'Change', 'Wave', 'Volume', 'Prev. Day Open Interest'
'EFP Volume', 'EFS Volume', 'Block Volume'], dtype='object')
(2838, 11)
```

	Open	High	Low	Settle	Volume	Prev. Day Open Interest
count	2895.000000	2895.000000	2895.000000	2895.000000	2895.000000	2895.000000
mean	76.381551	77.316739	75.395893	76.388238	213551.528152	278960.546114
std	26.669618	26.736489	26.593451	26.711405	90689.912591	143996.432409
min	19.900000	21.290000	15.980000	19.330000	9974.000000	0.000000
25%	54.075000	54.720000	53.030000	53.935000	158802.000000	174495.500000
50%	70.820000	71.600000	69.680000	70.540000	208936.000000	256528.000000
75%	106.595000	107.670000	105.590000	106.680000	268737.500000	374654.000000
max	126.580000	128.400000	125.000000	126.650000	773838.000000	693406.000000

```
print('\033[4mProbability of +/-(1%); +/-(3%); +/-%(5) change in
price (Data -> 2018- till date)\033[0m')

print ("The probability of price changes between 1%% and -1%% is
%1.2f%% " %
       (100*daily_ret[(daily_ret > -0.01) & (daily_ret <
0.01)].shape[0] / daily_ret.shape[0]))
print ("The probability of price changes between 3%% and -3%% is
%1.2f%% " %
       (100*daily_ret[(daily_ret > -0.03) & (daily_ret <
0.03)].shape[0] / daily_ret.shape[0]))
print ("The probability of price changes between 5%% and -5%% is
%1.2f%% " %
       (100*daily_ret[(daily_ret > -0.05) & (daily_ret <
0.05)].shape[0] / daily_ret.shape[0]))
print ("The probability of price changes more than 5%% is %1.2f%%" %
       (100*daily_ret[daily_ret > 0.05].shape[0] /
daily_ret.shape[0]))
print ("The probability of price changes less than -5%% is %1.2f%%" %
       (100*daily_ret[daily_ret < -0.05].shape[0] /
daily_ret.shape[0]))
```

```
Probability of +/-(1%); +/-(3%); +/-%(5) change in price (Data -> 2018- till date)
The probability of price changes between 1% and -1% is 46.08%
The probability of price changes between 3% and -3% is 85.40%
The probability of price changes between 5% and -5% is 94.21%
The probability of price changes more than 5% is 2.17%
The probability of price changes less than -5% is 3.62%
```

```
print('\033[4mMinimum price [2018- till date]\033[0m')
print(round(data['Settle'].min(),2), data['Settle'].idxmin());
print('\033[4mMaximum price [2018- till date]\033[0m')
print(round(data['Settle'].max(),2), data['Settle'].idxmax());
print('\n')

print('\033[4mMinimum daily % return [2018- till date]\033[0m')
print(round(daily_ret.min(),2)*100, daily_ret.idxmin());
print('\033[4mMaximum daily % return [2018- till date]\033[0m')
print(round(daily_ret.max()*100, 2), daily_ret.idxmax());
```

```
Minimum price [2018- till date]
19.33 2020-04-21 00:00:00
Maximum price [2018- till date]
86.29 2018-10-03 00:00:00
```

```
Minimum daily % return [2018- till date]
-24.0 2020-04-21 00:00:00
Maximum daily % return [2018- till date]
21.02 2020-04-02 00:00:00
```

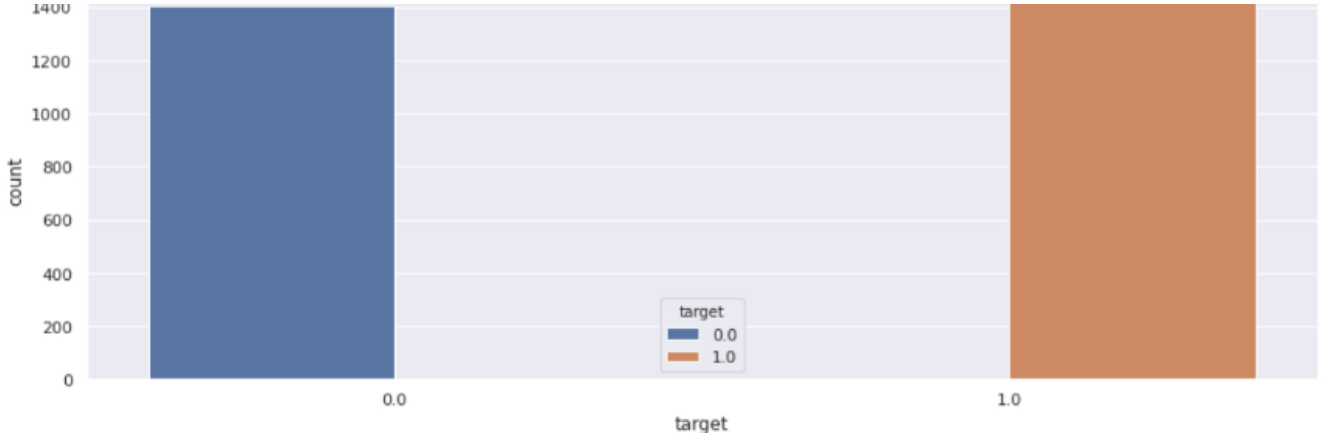
Features creation:

```
qq = BC['2015':].copy()
qq = qq[['Open', 'High', 'Low', 'Settle', 'Volume']]
qq['h_o'] = (BC['High']- BC['Open']) # distance between Highest and
Opening price
qq['l_o'] = (BC['Low'] - BC['Open']) # distance between Lowest and
Opening price
qq['gain'] = (BC['Settle']- BC['Open'])
qq['dailyChange'] = (qq['Settle'] - qq['Open']) / qq['Open']
qq['closeReturn'] = BC['Settle'].pct_change()
qq['priceDirection'] = (qq['Settle'].shift(-1) - qq['Settle'])
qq['volIncrement'] = BC.Volume.diff() / BC.Volume
qq = qq.dropna()
qq.head()
```

Date	Open	High	Low	Settle	Volume	h_o	l_o	gain	dailyChange	closeReturn	volIncrement	target
2015-01-02	58.02	58.54	55.48	56.42	173740.0	0.52	-2.54	-1.60	-0.027577	-0.015873	0.168649	0.0
2015-01-05	56.25	56.30	52.66	53.11	284892.0	0.05	-3.59	-3.14	-0.055822	-0.058667	0.390155	0.0
2015-01-06	53.20	53.60	50.52	51.10	331869.0	0.40	-2.68	-2.10	-0.039474	-0.037846	0.141553	1.0
2015-01-07	51.15	51.84	49.66	51.15	304199.0	0.69	-1.49	0.00	0.000000	0.000978	-0.090960	0.0
2015-01-08	51.06	51.91	49.81	50.96	238964.0	0.85	-1.25	-0.10	-0.001958	-0.003715	-0.272991	0.0

Target creation:

```
# positive value = 1, otherwise, 0
qq.loc[:,"target"] = np.where(qq['priceDirection']> 0, 1.0, 0.0)
qq.drop(['priceDirection'],1, inplace=True)
qq.head()
```



Train/Test split:

```
X = qq.drop(columns= ['target', 'High',
                      'Open', 'Low', 'Settle',
                      'Volume','target'], axis=1)
y = qq.target.astype(np.integer)
print(X)
```

Date	h_o	l_o	gain	dailyChange	closeReturn	volIncrement
2021-03-12	0.31	-0.56	-0.37	-0.005317	-0.005888	-0.160764
2021-03-15	0.95	-1.26	-0.20	-0.002895	-0.004912	0.038311
2021-03-16	0.13	-1.45	-0.42	-0.006104	-0.007114	-0.053259
2021-03-17	0.50	-1.43	-0.39	-0.005703	-0.005703	0.194630
2021-03-18	0.38	-6.32	-4.49	-0.066254	-0.069412	0.401724

Let us introduce gaps between the training and test set to mitigate the temporal dependence. Here the split function splits the data into Kfolds. The test sets are untouched, while the training sets get the gaps removed.

```
# training and test sets
gkcv = GapKFold(n_splits=5, gap_before=2, gap_after=1)

for tr_index, te_index in gkcv.split(X, y):
    xTrain, xTest = X.values[tr_index], X.values[te_index];
    yTrain, yTest = y.values[tr_index], y.values[te_index];
```

```
print('Observations: %d' % (len(xTrain) + len(xTest)))
print('Training Observations: %d' % (len(xTrain)))
print('Testing Observations: %d' % (len(xTest)))
```

```
Observations: 1601
Training Observations: 1281
Testing Observations: 320
```

Testing different algorithms:

```
xgb = XGBClassifier()
logreg2 = LogisticRegression(solver='lbfgs')
knn = KNeighborsClassifier(5)
lda = LinearDiscriminantAnalysis()
qda = QuadraticDiscriminantAnalysis()
lsvc = LinearSVC()
rsvm = SVC(C=1000000.0, cache_size=200, class_weight=None,
                      coef0=0.0, degree=3, gamma=0.0001, kernel =
'rbf', max_iter=-1, probability=False, random_state=None,
shrinking=True, tol=0.001, verbose=False)
rfc = RandomForestClassifier(
              n_estimators=1000, criterion='gini',
              max_depth=None, min_samples_split=2,
              min_samples_leaf=1, max_features='auto',
              bootstrap=True, oob_score=False, n_jobs=1,
              random_state=None, verbose=0)

# prepare configuration for cross validation test harness
seed = 42
# prepare models
models = []
models.append(('XGB', XGBClassifier()))
models.append(('LR', LogisticRegression(solver='lbfgs')))
models.append(('KNN', KNeighborsClassifier()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('RF', RandomForestClassifier(
              n_estimators=1000, criterion='gini',
              max_depth=None, min_samples_split=2,
              min_samples_leaf=1, max_features='auto',
              bootstrap=True, oob_score=False, n_jobs=1,
              random_state=None, verbose=0)))
models.append(('QDA', QuadraticDiscriminantAnalysis()))
models.append(('LSVC', LinearSVC()))
models.append(('RSVM', SVC(C=1000000.0, cache_size=200,
class_weight=None,
                      coef0=0.0, degree=3, gamma=0.0001,
```

```
kernel='rbf',
                        max_iter=-1, probability=False,
random_state=None,
                        shrinking=True, tol=0.001, verbose=False)))

# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
    cv_results = model_selection.cross_val_score(model, xTrain,
yTrain, cv=gkcv, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
XGB: 0.506639 (0.023163)
LR: 0.535542 (0.022796)
KNN: 0.516795 (0.038544)
LDA: 0.543358 (0.025270)
RF: 0.499638 (0.027869)
QDA: 0.517607 (0.031733)
```

```
LSVC: 0.535545 (0.025269)
RSVM: 0.543367 (0.032586)
```

Ensemble Model:

```
LR = LogisticRegression(solver='lbfgs',penalty='l2',random_state = 0)
LDA = LinearDiscriminantAnalysis()

ensembleModel = VotingClassifier(
    estimators=[
                ('LDA', LDA),
                ('LR', LR)
                ],
    voting='hard'
    ).fit(X, y)

print(cross_val_score(ensembleModel, X, y, cv=gkcv).mean()
```

```
0.5321300623052959
```

Here, ensemble model (combining Logic Regression & Linear Discriminant Analysis) happened to give the best score (53.20%) with 5 fold cross validation.

Prediction:

```
yhat = ensembleModel.predict(X)
print('\n Classification Report:\n', classification_report(y, yhat))
```

```
 Classification Report:
              precision    recall  f1-score   support

           0       0.53      0.43      0.48       765
           1       0.56      0.66      0.60       838

    accuracy                           0.55      1603
   macro avg       0.55      0.54      0.54      1603
weighted avg       0.55      0.55      0.54      1603
```

After fitting the model with complete dataset, we can predict out-of-sample for future forecasting. Here, we have exercised in-sample prediction to further showcase how predicted signals can be used in trading as buy/sell signals.

```
yhat = pd.DataFrame(yhat)
yhat.index = y.index
qq['PredictedSignal'] = yhat

"""Now that we have the PredictedSignal column, let us collect the
record from 2021"""

prices = qq[['Settle', 'closeReturn', 'target', 'PredictedSignal']]
['2021-01-01':].copy()
prices.tail()
```

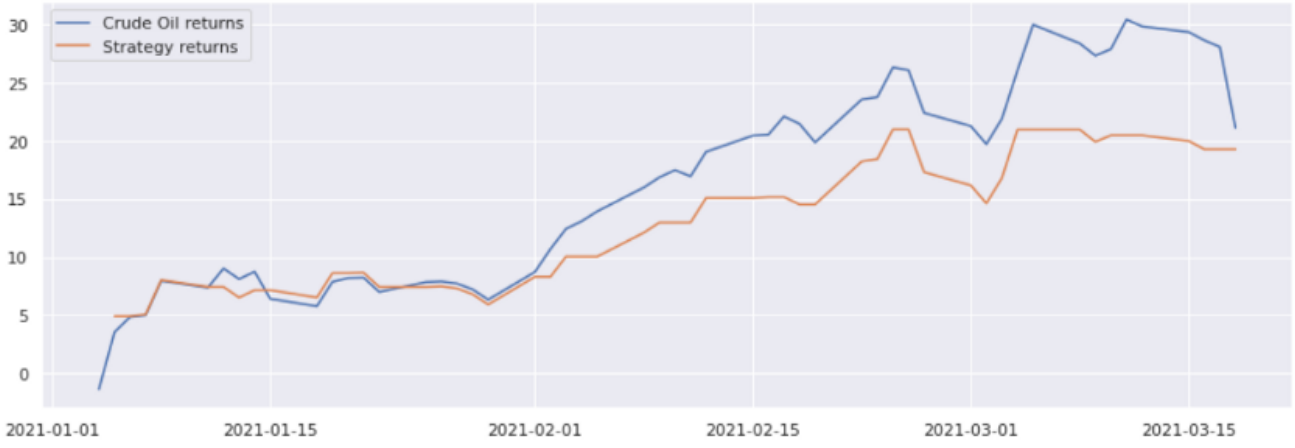
	Settle	closeReturn	target	PredictedSignal

Date				
2021-03-12	69.22	-0.005888	0.0	1
2021-03-15	68.88	-0.004912	0.0	1
2021-03-16	68.39	-0.007114	0.0	0
2021-03-17	68.00	-0.005703	0.0	0
2021-03-18	63.28	-0.069412	1.0	1

Above, at predicted signal column, we have ‘0’ when we need to buy, and we have ‘1’ when we need to sell. Let us find out cumulative returns and cumulative strategy return based on PredictedSignal to find out an indication of the strength of our strategy.

```
prices['cumReturns'] = 100*prices['closeReturn'].cumsum()
prices['cumStrategyReturns']= 100*(prices['closeReturn'] *
prices['PredictedSignal'].shift(1)).cumsum()

plt.figure(figsize=(15,5))
plt.plot(prices['cumReturns'], label = 'Crude Oil returns')
plt.plot(prices['cumStrategyReturns'], label = 'Strategy returns')
plt.legend(loc='best')
plt.show()
```




```
print('Number of trades (buy) = ', (prices['PredictedSignal']==1).sum())
print('Number of trades (sell) = ', (prices['PredictedSignal']==0).sum())
```

```
Number of trades (buy) =  34
Number of trades (sell) =  20
```

Here, we see that, our ML model predicted 54 buy/sell signals in 2021; however, strategic cumulative return falling short of actual return based on historical values since 1 Jan 2021.

Now, it is not advisable to keep buying or selling based on continuous predicted signals. Therefore, to limit the number of transactions, we will use differencing to the strategy column.

```
"""
we will limit the number of orders by restricting ourselves to the
number of positions on the market.
we will apply diff() to the column signal:
"""
prices['Strategy'] = prices['PredictedSignal'].diff(2)
```

Signal visualization:

```
print(prices.Strategy.value_counts())

# Buy/Sell signals plot
buys = prices.loc[prices["Strategy"] == 1];
sells = prices.loc[prices["Strategy"] == -1];

# Plot
fig = plt.figure(figsize=(20, 5));
plt.plot(prices.index, prices['Settle'], lw=2., label='Price');

# Plot buy and sell signals
# up arrow when we buy one share
plt.plot(sells.index, prices.loc[sells.index]['Settle'], '^',
markersize=10, color='g', lw=2., label='Buy');
# down arrow when we sell one share
plt.plot(buys.index, prices.loc[buys.index]['Settle'], 'v',
markersize = 10, color='r', lw=2., label='Sell');
plt.ylabel('Price (USD)'); plt.xlabel('Date');
plt.title('Last 24 Buy and Sell signals'); plt.legend(loc='best');
plt.show()
```

```
 0.0   22
-1.0   15
 1.0   15
```

```
Name: Strategy, dtype: int64
```

Here, we see that few signal are generated; we will validate the profitability of our new strategy based on PredictedSignal column.

Performance testing:

Well, the final objective is to monetize the buy/sell signals that are generated by the models by implementing some strategy. The trading strategy requires that at the end of each trading day, we decide to buy stocks if the signal is up for the next day and there is money in the account, or, sell stocks at hand if the signal is down for the next day. Otherwise, there is no action.

We will do the performance evaluation with a hypothetical amount of $10,000 and check if our P&L amount grows.

```
prices = prices.reset_index()
class Portfolio:
    def __init__(self):
        self.lotA = 1
        self.lotB = 1
        self.contract = 1
        self.initialCash = 10000
        self.buy = (np.where(prices.Strategy == 1, self.lotA *
self.contract *prices.Settle, 0))
        self.sell = (np.where(prices.Strategy == (-1), self.lotB *
self.contract *prices.Settle, 0))

    def long_amt(self):
        self.buy = (np.where(prices.Strategy == 1,Portfolio().lotA *
Portfolio().contract *prices.Settle, 0))
        return self.buy

    def cashDelta(self):
```

```
        self.cashDelta = Portfolio().buy + Portfolio().sell
        return self.cashDelta

    def endBalance(self):
        self.endBalance = Portfolio().initialCash +
Portfolio().cashDelta().cumsum()
        return self.endBalance

    def endPosition(self):
        self.endPosition = prices.Strategy.cumsum()
        return self.endPosition

p = Portfolio()
prices['Buy'] = p.buy
prices['Sell'] = p.sell
prices['cashDelta'] = p.cashDelta()
prices['endBalance'] = p.endBalance()
prices['endPosition'] = p.endPosition()
prices.loc[:, ['Date', 'Settle', 'Strategy','Buy', 'Sell',
                'cashDelta', 'endBalance', 'endPosition']]
```

	Date	Settle	Strategy	Buy	Sell	cashDelta	endBalance	endPosition
49	2021-03-12	69.22	1.0	69.22	0.00	69.22	11594.69	0.0
50	2021-03-15	68.88	1.0	68.88	0.00	68.88	11663.57	1.0
51	2021-03-16	68.39	-1.0	0.00	68.39	68.39	11731.96	0.0
52	2021-03-17	68.00	-1.0	0.00	68.00	68.00	11799.96	-1.0
53	2021-03-18	63.28	1.0	63.28	0.00	63.28	11863.24	0.0

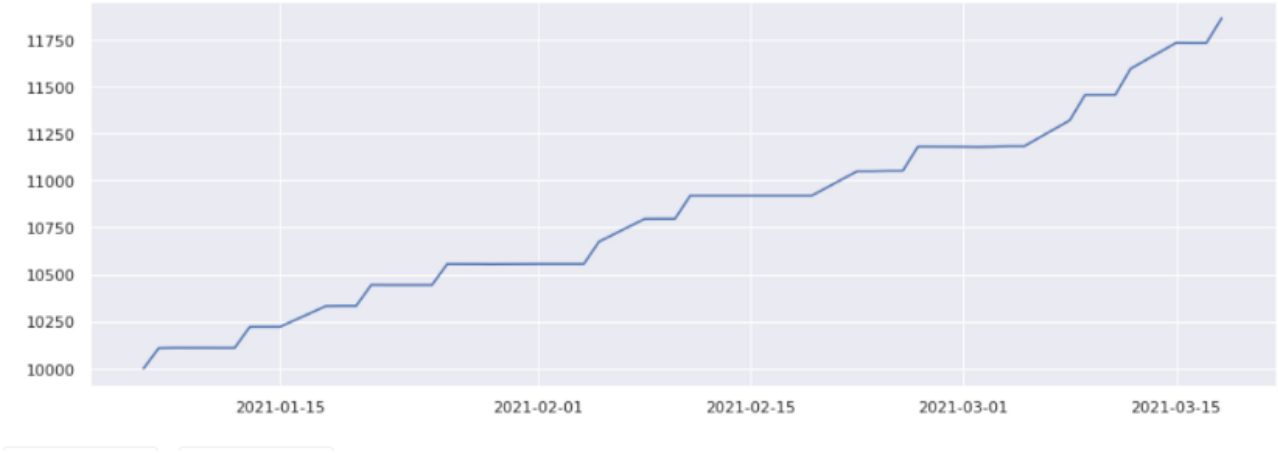
Profit & Loss metrics:

```
prices['pnl'] = prices['endBalance'] + (prices.endPosition *
prices.Settle * Portfolio().contract)
prices.loc[:, ['Date', 'Settle', 'endBalance', 'endPosition', 'pnl']]

plot = prices.set_index('Date')
plt.figure(figsize=(15,5))
plt.plot(plot.pnl)
```

	Date	Settle	endBalance	endPosition	pnl
44	2021-03-05	69.36	11252.18	-1.0	11182.82

45	2021-03-08	68.24	11320.42	0.0	11320.42
46	2021-03-09	67.52	11387.94	1.0	11455.46
47	2021-03-10	67.90	11455.84	0.0	11455.84
48	2021-03-11	69.63	11525.47	-1.0	11455.84
49	2021-03-12	69.22	11594.69	0.0	11594.69
50	2021-03-15	68.88	11663.57	1.0	11732.45
51	2021-03-16	68.39	11731.96	0.0	11731.96
52	2021-03-17	68.00	11799.96	-1.0	11731.96
53	2021-03-18	63.28	11863.24	0.0	11863.24



Concluding remarks:

So, we could see that, our portfolio has grown from $10,000 to $11,863 in a quarter. However, this is a simplified approach of classification based ML and testing. We have to remember the involvement of risk elements and managing risks in trading strategies are critical. Risks could be money losses or could lead to legal troubles. Unless managed well, the entire exercise would collapse to become nightmare.

I can be reached here.

Disclaimer: The programs described here are experimental and should be used with caution for any commercial purpose. All such use at your own risk.

Classification Performance Testing Trade Signal Algorithmic Trading