Sarit Maitra

1.4K Followers About

BRENT CRUDE OIL FUTURES PRICE MOVEMENTS PREDICTION

Classification Algorithm to Predict Price Movements & Performance Testing

ML to generate buy/sell signals

Sarit Maitra 1 day ago · 7 min read ★

Image by author

Prediction and classification are important and of great interest because successful prediction of stock prices lead to attractive 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.

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	B Vol
2020-12-22	50.65	50.86	49.56	50.08	-0.83	50.21	192602.0	211345.0	NaN	NaN	14
2020-12-23	49.70	51.59	49.20	51.20	1.12	50.56	179707.0	178792.0	NaN	NaN	29
2020-12-24	51.11	51.77	50.62	51.29	0.09	51.19	94528.0	135884.0	1800.0	NaN	3
2020-12-28	51.25	52.02	50.53	50.86	NaN	51.28	81059.0	128364.0	NaN	NaN	
2020-12-29	51.02	51.63	50.86	51.09	0.23	51.30	87379.0	105934.0	3828.0	NaN	1

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)
```

```
BC.describe()
```

:

	Open	High	Low	Settle	Volume	Prev. Day Open Interest

min	19.900000	21.290000	15.980000	19.330000	9974.000000	0.000000
25%	53.777500	54.500000	52.692500	53.617500	157878.750000	173074.250000
50%	71.655000	72.725000	70.885000	71.800000	207487.500000	254343.500000
75%	106.855000	107.877500	105.850000	106.887500	266177.000000	366036.000000
max	126.580000	128.400000	125.000000	126.650000	773838.000000	676659.000000

```
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 45.98%
The probability of price changes between 3% and -3% is 85.10%
The probability of price changes between 5% and -5% is 93.91%
The probability of price changes more than 5% is 2.33%
The probability of price changes less than -5% is 3.76%
```

```
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(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());
```

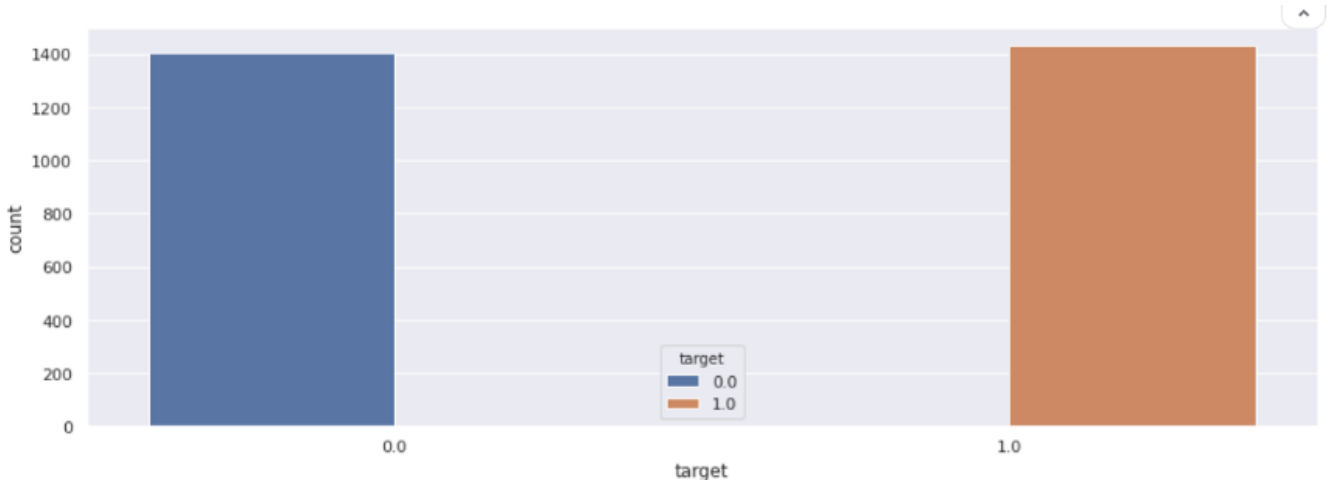
```
Minimum price [2018- till date]
19.33 2020-04-21 00:00:00
Maximum price [2018- till date]
86.29 2018-10-03 00:00:00


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 = pd.DataFrame(index = BC.index)
qq['h_o'] = (BC['High'].pct_change() - BC['Open'].pct_change()) #
distance between Highest and Opening price
qq['l_o'] = (BC['Low'].pct_change() - BC['Open'].pct_change()) #
distance between Lowest and Opening price
qq['gain'] = (BC['Settle'].pct_change() - BC['Open'].pct_change())
qq['price_diff'] = BC['Settle'].diff()
qq['close'] = BC['Settle']

lags = 3
# Create the shifted lag series of prior trading period close values
for i in range(0, lags):
    qq["Lag%s" % str(i+1)] = BC["Settle"].shift(i+1).pct_change()

qq['HL'] = (BC['High'] - BC['Settle']) / BC['Settle']
# creating more features
qq['norm_returns'] = BC['Settle'].pct_change()
qq['vol_increment'] = BC.Volume.diff() / BC.Volume
qq['pdoi'] = BC['Prev. Day Open Interest'].pct_change()
qq = qq.dropna()
```

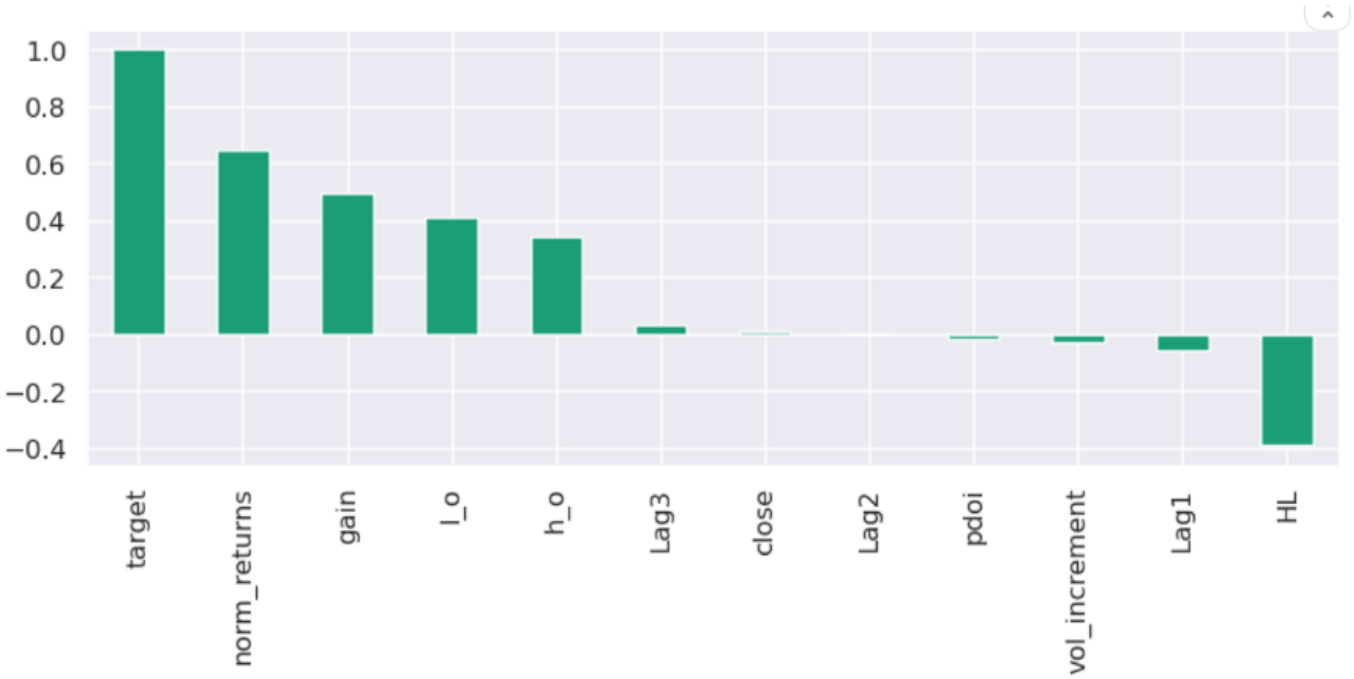
Target creation:

```
sns.countplot(x = 'target', data=qq, hue='target')
plt.show()
```



Checking Correlation:

```
sns.set(style='darkgrid', context='talk', palette='Dark2')
plt.figure(figsize=(14,5))
qq.corr()['target'].sort_values(ascending = False).plot(kind='bar')
plt.show()
```



```
# creating X, y variables
X = qq.drop(columns= ['target', 'Lag1', 'Lag2', 'Lag3', 'pdoi',
             'close','vol_increment'], axis=1)
y = qq.target.astype(np.integer)
```

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: 2832
Training Observations: 2266
Testing Observations: 566
```

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,
```

```
                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: 1.000000 (0.000000)
LR: 0.836291 (0.025296)
KNN: 0.952346 (0.012957)
LDA: 0.974858 (0.016397)
```

```
LSVC: 0.879102 (0.024703)
RSVM: 0.967354 (0.013153)
```

Here, we see that, Linear Discriminant Analysis (LDA) scored 97.48%. We have to remember that, LDA projects data onto a lower-dimensional space which provides maximum class separability. Features that are derived from LDA are linear combinations of the original ones. Moreover, LDA is equivalent to linear regression and thus, can be formulated as a least squares problem.

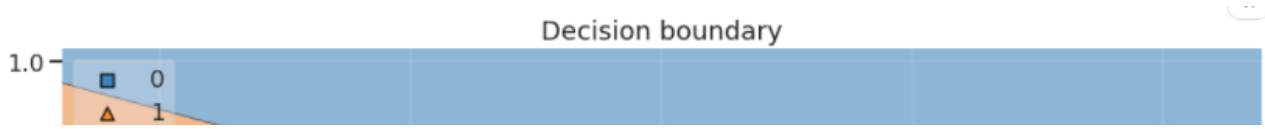
Decision surface:

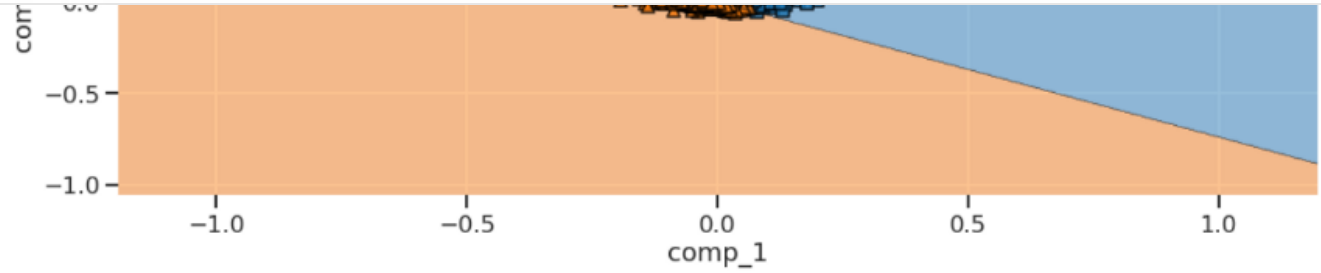
Let us view the decision surface by applying PCA.

```
whiten = False
random_state = 2020

pca_2c = PCA(n_components=2, whiten = whiten, random_state = random_state)
x_pca_2c = pca_2c.fit_transform(xTrain)
print(x_pca_2c.shape)
print(pca_2c.explained_variance_ratio_.sum()); print()
```

```
(2266, 2)
0.8905526756919616
```

```
lda = LinearDiscriminantAnalysis().fit(x_pca_2c,yTrain)
yhat = lda.predict(x_pca_2c)

# plot decision region to visualize
plot_decision_regions(x_pca_2c, yTrain, clf=lda, legend=2)
# axes annotations
plt.xlabel('comp_1'); plt.ylabel('comp_2')
plt.title('Decision boundary')
plt.show()
```






```
print('\n Classification Report:\n', classification_report(yTrain, yhat))
```

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

           0       0.96      0.81      0.88      1139
           1       0.84      0.96      0.90      1127

    accuracy                           0.89      2266
   macro avg       0.90      0.89      0.89      2266
weighted avg       0.90      0.89      0.89      2266
```

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.

```
m = LinearDiscriminantAnalysis().fit(X,y)
qq.loc[:,'Predicted_Signal'] = m.predict(X)
qq = qq[['close', 'norm_returns', 'target', 'Predicted_Signal']]
qq.loc[:,'Strategy'] = qq.loc[:,'Predicted_Signal'].diff()
print(qq)
```

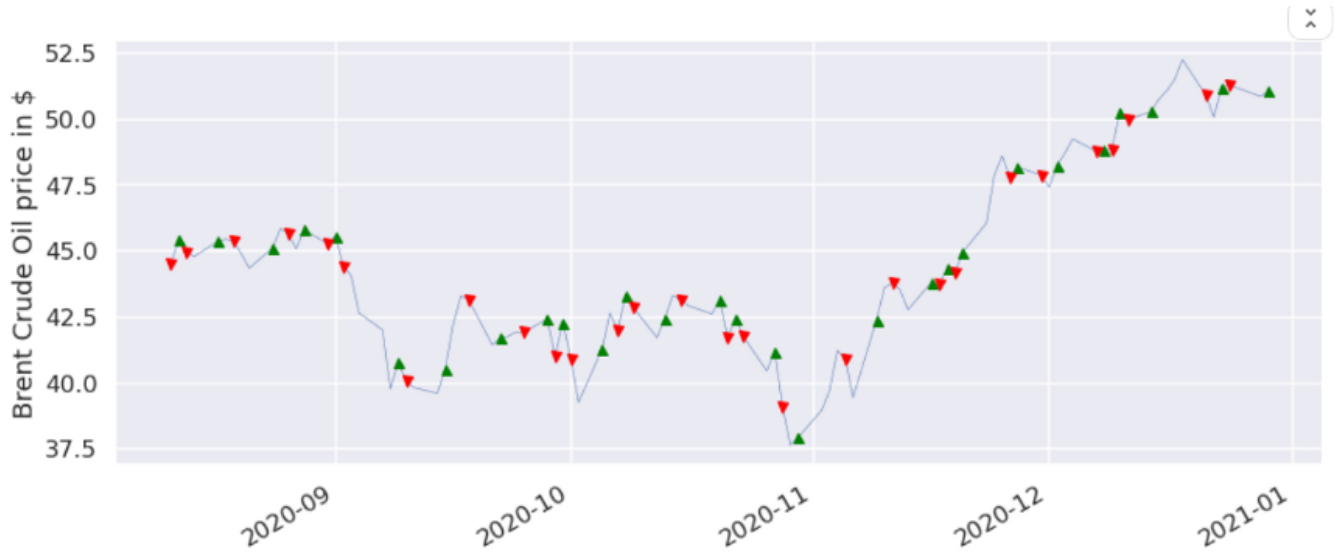
Date	close	norm_returns	target	Predicted_Signal	Strategy
2010-01-08	81.37	-0.001718	0.0	0	NaN
2010-01-11	80.97	-0.004916	0.0	0	0.0

2010-01-13	78.31	-0.012484	0.0	0	0.0
2010-01-14	77.82	-0.006257	0.0	0	0.0

Above, at predicted signal column, we have 0 when we need to buy, and we have 1 when we need to sell. However, it is not advisable to keep buying or selling based on continuous predicted signals. Therefore, to limit the number of transactions, we have used differencing in strategy column.

Signal visualization:

```
# collect last 100 positions for visualization
dd = qq[-100:]
# visualize last 100 positions
fig = plt.figure(figsize = (15,6))
ax1 = fig.add_subplot(111, ylabel='Brent Crude Oil price in $')
dd["close"][-100:].plot(ax=ax1, color='b', lw=.5)
ax1.plot(dd.loc[dd.Strategy ==1.0].index[-100:], dd.close[dd.Strategy
== 1.0][-100:],
         '^', markersize=7, color='green')
ax1.plot(dd.loc[dd.Strategy ==-1.0].index[-100:],
dd.close[dd.Strategy == -1.0][-100:],
         'v', markersize=7, color='red')
plt.show()
```



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. We have 10 years of historical records here.

```
class Portfolio:
    def __init__(self):
        self.lotA = 1
        self.lotB = 1
        self.contract = 1
        self.initial_cash = 10000
        self.buy = (np.where(qq.Strategy == 1,
                                        self.lotA * self.contract
*qq.close, 0))
        self.sell = (np.where(qq.Strategy == (-1),
                                        self.lotB * self.contract
*qq.close, 0))

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

    def cash_delta(self):
        self.cash_delta = Portfolio().buy + Portfolio().sell
        return self.cash_delta

    def end_bal(self):
        self.end_bal = Portfolio().initial_cash +
Portfolio().cash_delta().cumsum()
        return self.end_bal

    def end_pos(self):
        self.end_pos = qq.Strategy.cumsum()
        return self.end_pos

p = Portfolio()
qq['Buy'] = p.buy
qq['Sell'] = p.sell
qq['CashDelta'] = p.cash_delta()
```

```
                    CashDelta ,  EndBalance ,  EndPosition ]]
```



```
qq['pnl'] = qq['EndBalance'] + (qq.EndPosition * qq.close *
Portfolio().contract)
qq.loc[:, ['Date', 'close', 'EndBalance', 'EndPosition', 'pnl']]
```

Concluding remarks:

We can see that our PnL grows from $10,000 to $130,000 in 10 years. However, this is just a simple example of classification based ML and testing to start your algorithmic trading journey. 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