Stock Prediction Technique : Classification Use Case and Return Computation

Predictive model using a machine learning algorithm

Start it up

[Sarit Maitra](#)

.

Published in

The Startup

.

5 min read

.

Aug 29, 2020

https://sarit-maitra.medium.com/membership

Predictive modeling using machine learning comes with a trick to generalize new cases and not merely memorizing past cases. In order to achieve that, the ML algorithm must look through multiple rows of data, and different features which have significant correlations with target variable. In designing predictive modeling the key is to find a way to identify price trends without the uncertainty and bias of the our mental model.

A successful approach could be classification where we we can formulate our objective is to predict next period’s return. We can predict compared to current period, If next period’s return > 0, then 1, if next periods return <= 0, then -1.

We picked up crypto currency data. The can predict motivation here is that, despite its high volatility, the price of BTC has been an area where significant efforts for price forecast are going on. Here we loading Bitcoin daily data into pandas data frame.

```python
btc = yf.Ticker("BTC-USD")
# get historical market data
hist = btc.history(period="max")
df = hist[['Open', 'High', 'Low', 'Close', 'Volume']]
print(df.tail()); print(); print(df.shape)
```

```
                Open       High        Low      Close       Volume
Date
2020-08-25  11773.59  11778.30  11189.85  11366.13  26301509932
2020-08-26  11366.89  11530.05  11296.99  11488.36  22466660958
2020-08-27  11485.61  11570.79  11185.94  11323.40  23240415076
2020-08-28  11325.30  11545.62  11316.42  11542.50  19807127588
2020-08-29  11545.08  11577.64  11497.06  11563.66  18840709120

(2173, 5)
```

Visualization

Daily candlestick shows the open, high, low, and close price for the day. This real body represents the price range between the open and close of that day's trading. When the real body is filled in red or green, it means the close was lower than the open. Here, we have taken last 30 days data to have a clear visualization.

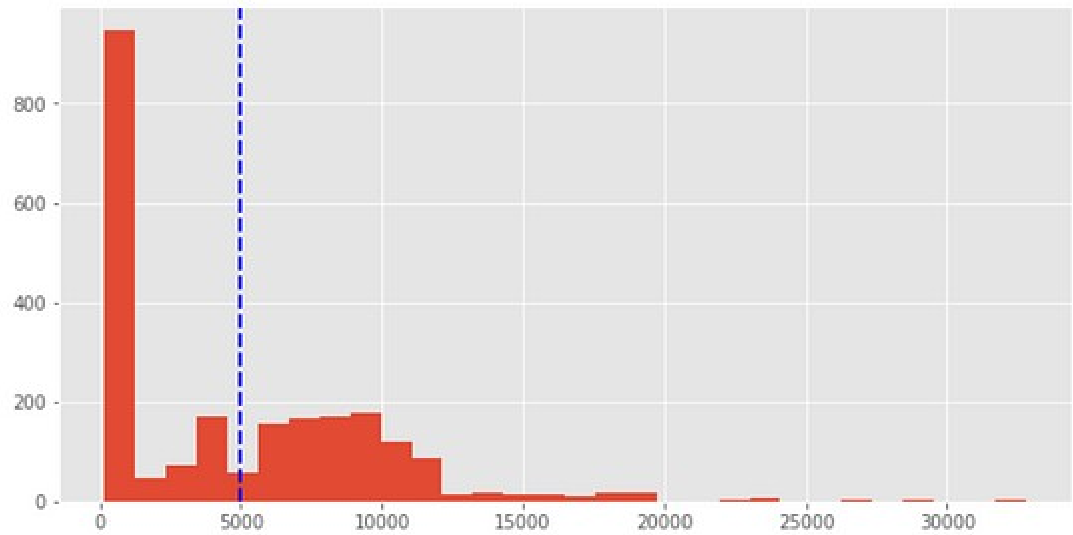
```python
fig = go.Figure(data=[go.Candlestick(x=df.index[-30:],
                                     open=df['Open'][-30:],
                                     high=df['High'][-30:],
                                     low=df['Low'][-30:],
                                     close=df['Close'][-30:])])
fig.update_xaxes(showline=True, linewidth=2, linecolor='black', mirror=True)
fig.update_yaxes(showline=True, linewidth=2, linecolor='black', mirror=True)
fig.update_layout(title='Last 30 days BTC price', yaxis_title='BTC (US$)')
fig.show()
```

Statistics

```
df.describe()
```

	Open	High	Low	Close	Volume
count	2173.000000	2173.000000	2173.000000	2173.000000	2.173000e+03
mean	4337.489268	4448.061431	4220.204993	4342.490152	7.744282e+09
std	4107.245418	4227.794096	3968.729128	4109.325032	1.152054e+10
min	176.900000	211.730000	171.510000	178.100000	5.914570e+06
25%	431.660000	436.020000	424.430000	431.960000	5.936640e+07
50%	3584.500000	3647.330000	3487.170000	3585.120000	1.844620e+09
75%	7836.830000	8076.890000	7615.990000	7871.690000	1.233650e+10
max	19475.800000	20089.000000	18974.100000	19497.400000	7.415677e+10

```
df['Close'].hist(bins=30, figsize=(10,5)).axvline(df['Close'].mean(),
                                                  color='b', linestyle='dashed', linewidth=2)
plt.show()
```



Observations

BTC closing price was not over $4342 for almost half of the time (we can see that from mean value of close price). Blue dashed line represents the median/mean line.

Feature creation:

```python
# Create the shifted lag series of prior trading period close values
lags = 2
for i in range(0, lags):
    df["Lag%s" % str(i+1)] = df["Close"].shift(i+1).pct_change()

df['Open-Close'] = (df.Open - df.Close).pct_change()
df['High-Low'] = (df.High - df.Low).pct_change()
df['volume_gap'] = df.Volume.pct_change()
df.head()
```

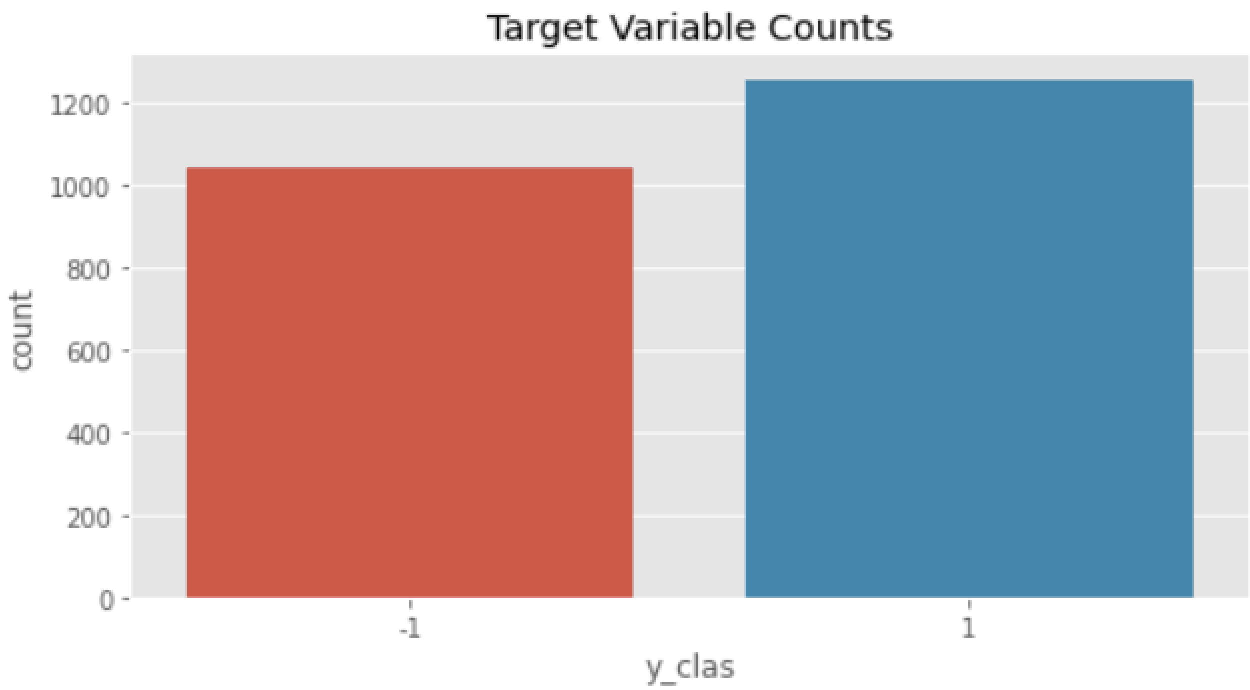
	Open	High	Low	Close	Volume	Lag1	Lag2	Open-Close	High-Low	volume_gap
Date										
2014-09-17	465.864014	468.174011	452.421997	457.334015	21056800	NaN	NaN	NaN	NaN	NaN
2014-09-18	456.859985	456.859985	413.104004	424.440002	34483200	NaN	NaN	2.800702	1.777802	0.637628
2014-09-19	424.102997	427.834991	384.532013	394.795990	37919700	-0.071926	NaN	-0.096020	-0.010353	0.099657
2014-09-20	394.673004	423.295990	389.882996	408.903992	36863600	-0.069843	-0.071926	-1.485583	-0.228390	-0.027851
2014-09-21	408.084991	412.425995	393.181000	398.821014	26580100	0.035735	-0.069843	-1.650972	-0.424027	-0.278961

Target variable:

```python
#Shift -1 for next day's return
df['forward_ret'] = df['Close'].shift(-1) / df['Open'].shift(-1)-1
#If tomorrow's return > 0, then 1; #If tomorrow's return <= 0, then -1
df['y_clas']= -1
df.at[df['forward_ret']>0.0,'y_clas']=1
# Remove it make ensure no look ahead bias
del df['forward_ret']
```

Visualize target distribution:

```
# plot target variable
plt.figure(figsize=(8,4))
sns.countplot('y_clas', data=df)
plt.title('Target Variable Counts')
plt.show()
```



Train/Test split:

```python
# collect necessary features
data =  df[['Close', 'Lag1', 'Lag2', 'Open-Close', 'High-Low', 'volume_gap', 'y_clas']]
data.dropna(inplace=True)

# create X, y set
X = data.drop(['y_clas', 'Close'],1)
y_clas = data.y_clas

SP = 0.80 # split percentage
split = int(SP * len(data))
print('Split:', split)

# Train data set
xTrain = X[:split]; yTrain = y_clas[:split]
# Test data set
xTest = X[split:]; yTest = y_clas[split:]

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

```
Split: 1840
Observations: 2300
Training Observations: 1840
Testing Observations: 460
```

Model fitting and testing:

```python
# 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')))
models.append(('QDA', QuadraticDiscriminantAnalysis()))
models.append(('LSVC', LinearSVC()))
models.append(('RSVM', SVC(C=1000000.0, gamma=0.0001, kernel='rbf')))# evaluate
each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
    kf = model_selection.KFold(n_splits=5, random_state=seed)
    cv_results = model_selection.cross_val_score(model, xTrain, yTrain, cv=kf,
scoring = scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg); print()# iterate over the models
for i in models:
    i[1].fit(xTrain, yTrain)
    pred = i[1].predict(xTest)
```

```
    print("%s:\n%0.3f" % (i[0], i[1].score(xTest, yTest)))
    print("%s\n" % confusion_matrix(pred, yTest))
```

```
XGB: 0.515761 (0.014110)

LR: 0.543478 (0.039640)

KNN: 0.483696 (0.022800)

LDA: 0.536413 (0.032861)

RF: 0.505978 (0.024870)

QDA: 0.523913 (0.057347)

LSVC: 0.547283 (0.039551)

RSVM: 0.547283 (0.042294)
```

Accuracy of over 50% in test sample suggests that the classification model is effective for our use case.

```
XGB:
0.533
[[ 39  45]
 [170 206]]

LR:
0.539
[[  1   4]
 [208 247]]

KNN:
0.507
[[ 84 102]
 [125 149]]

LDA:
0.537
[[  1   5]
 [208 246]]

RF:
0.515
[[ 66  80]
 [143 171]]

QDA:
0.539
[[  5   8]
 [204 243]]

LSVC:
0.537
[[  8  12]
 [201 239]]

RSVM:
0.522
[[ 13  24]
 [196 227]]
```

Here, we see that though RandomForest is not the highest in-terms of accuracy score, but has generated reasonably good amount of signals.

Computing Strategy Returns

With the predicted values of the price movement, we can compute the returns of the strategy.

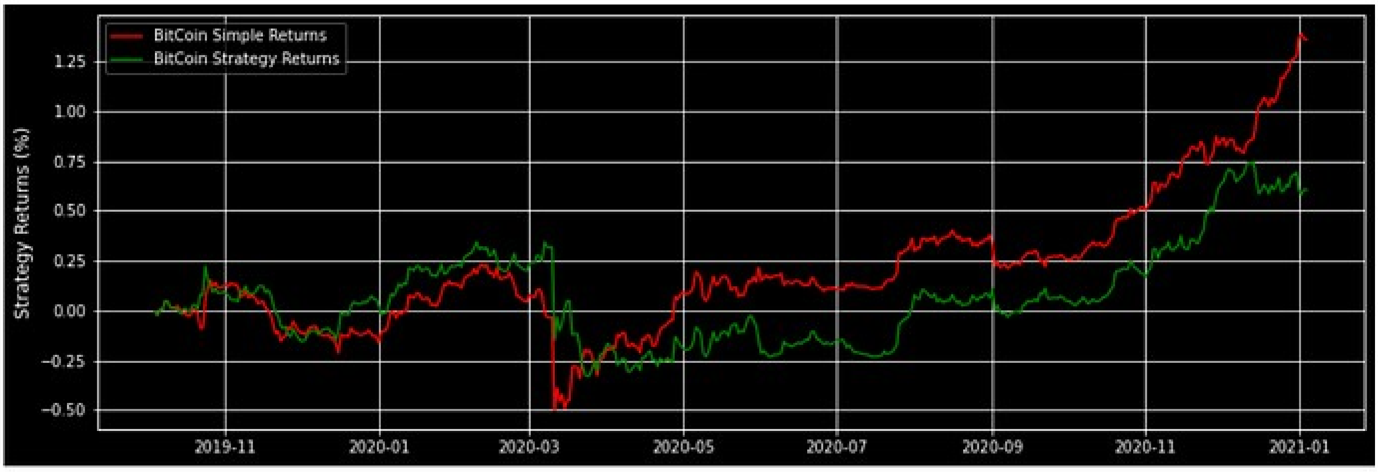
```
DfTrade = data[['Close']].copy()
#Dftrade = DfTrade[DfTrade.index > '2020'].copy()
DfTrade['trade_signal'] = rf.predict(X)
# log returns

"""
log returns of today (log of the close price of today) / close price of yesterday.
log-returns are added to show performance across time
"""
DfTrade['simple_ret'] = np.log(DfTrade['Close']/DfTrade['Close'].shift(1))

"""
the simple_ret values are shifted upwards by one element so that tomorrow's returns are stored against the prices of today
"""
DfTrade['simple_ret'] = DfTrade['simple_ret'].shift(-1)

# Strategy Returns
DfTrade['startegy_ret'] = DfTrade['simple_ret']* DfTrade['trade_signal']
# cumulative returns
DfTrade['cum_ret'] = DfTrade[split:]['simple_ret'].cumsum()
# Cumulative Strategy Returns
DfTrade['startegy_ret'] = DfTrade['simple_ret']* DfTrade['trade_signal']
DfTrade['cum_strategy_ret'] = DfTrade[split:]['startegy_ret'].cumsum()

# visualize the performance
plt.style.use('dark_background')
plt.figure(figsize=(15,5))
plt.plot(DfTrade.cum_ret, color='r',label = 'BitCoin Simple Returns')
plt.plot(DfTrade.cum_strategy_ret, color='g', label = 'BitCoin Strategy Returns')
plt.ylabel("Strategy Returns (%)")
plt.legend()
plt.show()
```



We can observe from above plot that, RandomForest classifier model with 2 lags and 3 hand engineered features yielded positive returns. However, it could not outperform the BitCoin simple return which has been highly volatile and price has gone upwards from around $4000 in last April to around $30,000 currently.

```
print('Market returns:', round(DfTrade['cum_ret'].sum(),2))
print('Trading Strategy returns:', round(DfTrade['cum_strategy_ret'].sum(),2))
```

```
Market returns: 93.55
Trading Strategy returns: 30.37
```

However, we have to remember that, RandomForest could be prone to overfitting and we have to be careful about selecting the right hyperparameters to get robust model. Moreover, RandomForest can capture non-linear relationships, so we can select more features for experimentation.

Key takeaways:

We have shown a simplified version of how to predict and compute if the strategy taken is effective and can beat market standard. However, there are enough rooms for improvements. Efficient Market Hypothesis (EMH) suggest that stock price also depends on new information significantly; therefore, information about people's opinion can be collected from social media and can be added as a predictor, Moreover, the same model can be tested with hourly or minute frequency to check the effectiveness.

I can be reached *here*.

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

Predictive Modeling
Machine Learning
Linear Regression
Technical Analysis

Written by Sarit Maitra

·Writer for

The Startup

Analytics & Data Science Practice Lead