WORLDQUANT UNIVERSITY

FINANCIAL ENGINEERING

GROUP WORK PROJECT

MODELING AND STRATEGY DEVELOPMENT

Group members:

Name	Email	Non-contribution
Quyen Ho Thanh	thquyen11@hotmail.com	
Wei Hao Lew	lewweihao93@hotmail.com	
Lu Liu	liulu0502@gmail.com	
Truong Nguyen	nmtruong93@gmail.com	
Yernur Orakbayev	yernur.orakbayev@nu.edu.kz	

June 2020

Contents

1. Background

The aim of this report is to implement the machine learning algorithmic trading strategy and use the performance metrics to make analysis. The research covers the following terms:

- LSTM theory and application
- Classification's performance metrics: confusion matrix, recall, precision, F1-score, ROC-AUC curves and accuracy

In this report, we will also build a fund factsheet for the new investment strategy by training with data and generating trading signals and list pros and cons for our strategy.

2. Research

2.1. LSTM

During the 90's the research on training RNNs was widely conducted in the data science field. In order to avoid the problem of long-term dependencies, Hochreiter and Schmidhuber (1997) brought up a modified method called the Long Short-term Memory (LSTM), which can sidestep the modeling longer term dependencies by improving the capacity of memorization and introduction of a "gate" into the cell. Gers & Schmidhuber (2000) introduced the popular LSTM variant through the complement of "peephole connections." Cho, et al. (2014) came up with a more beautiful version on the RNN, the Gated Recurrent Unit, or GRU, connecting the forget and input gates into a single "update gate." It also makes some other changes by merging the cell state and hidden state with simpler results compared with standard LSTM models. Yao, et al. (2015) and Koutnik, et al. (2014) proposed different approaches to handle the problem of long-term dependencies, for example Depth Gated RNNs and Clockwork RNNs.

Long Short-Term Memory models are deep recurrent neural network architecture and can be utilized as powerful time-series models. The LSTM model has the following elements:

- Cell state (c_t) - The information flows through a system named the cell state with both short-term and long-term memories stored.
- Hidden state (h_t) – LSTM has two types of hidden states: a "slow" state c_t addressing the vanishing gradient problem, and a "fast" state h_t allowing the system to make complex decisions over short periods of time. The result of hidden state depends on current input, previous hidden state and current cell input.
- Input gate (i_t) – It manages the flow of input activations into the memory cell and controls what new information is added to cell state from current input.

- Forget gate (f_t) - A sigmoid layer, also the "forget gate layer", will decide what information will be thrown away from the cell state, which means the information that is of less important or less required will be deleted via multiplication of a filter.
- Output gate (o_t) – It controls the output flow of cell activations into the rest of the network and conditionally decides what to output from the memory.

Seen in the figure below:

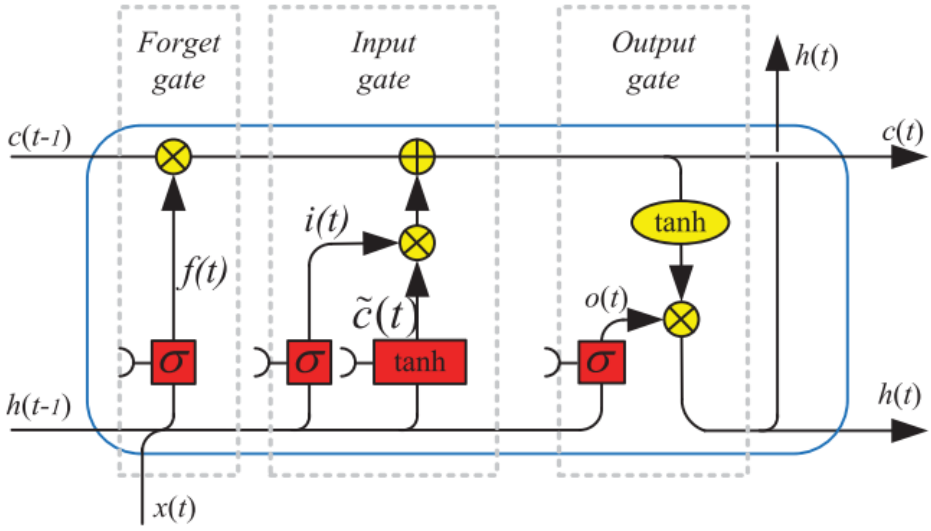


Figure 1. LSTM model

Based on the connections shown in Figure above, the LSTM model can be mathematically expressed as follows:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i$$
$$\tilde{C}_t = tanh(W_c x_t + U_c h_{t-1} + b_c)$$
$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$
$$C_t = i_t * \tilde{C}_t + f_t * C_{t-1}$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + V_o C_t + b_o)$$
$$h_t = o_t * tanh(C_t)$$

Where: x_t is the input data of the memory cell layer at time t; W_i, W_f, W_c, W_o, U_i, U_f, U_c, U_o and V_o are weight parameters; b_i, b_f, b_c and b_o are bias vectors.

Long Short-Term Memory model is the one of the most effective solution in deep learning so far with high computational complexity. LSTM is doing better than traditional recurrent neural network on issues involving long time lags and is trying to solve the vanishing gradient problem.

LSTMs can be utilized to predict stock price behavior and process time series. One disadvantage of LSTM is that training the LSTMs sometimes takes longer time than the RNNs.

2.2. Performance metrics of classification model

- **Confusion matrix**

Confusion matrix measures the achievement of a classification output which is generated by the machine learning model. In general, confusion matrix is a table consists of 4 boxes represent the combination of predicted and actual values.

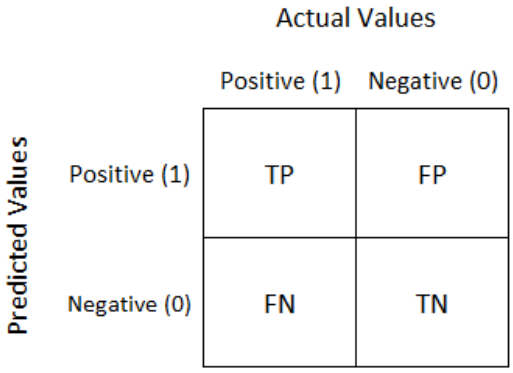


Figure 2. The Matrix

True Positive (TP)

- Predicted output: True
- Actual output: True

False Positive (FP)

- Predicted output: False
- Actual output: True

False Negative (FN)

- Predicted output: False
- Actual output: False

True Negative (TN)

- Predicted output: True
- Actual output: False

- **Recall**

If we only look at all the cases that the actual output is true, recall describes how much we predicted correctly

$$\text{Recall} = TP / (TP+FN)$$

Precision

Consider all the cases that we predict positive, precision illustrate how many of them are actual positive.

$$\text{Precision} = TP / (TP+FP)$$

F1-score

In case there are two models with low precision and high recall or with high precision and low recall, it is quite challenged to compare them effectively. Therefore, F-score help us to measure the Recall and Precision altogether and provide a comparable metric.

$$\text{F-score} = (2*\text{Recall}*\text{Precision}) / (\text{Recall} + \text{Precision})$$

AUC – ROC Curve

Upon the confusion matrix, we can build the ROC (Receiver Operating Characteristics) and AUC (Area Under the Curve) in order to measure the achievement of classification model at different thresholds.

ROC curve is constructed by plotting TPR (True Positive Rate) against the FPR (False Positive Rate) on y-axis and x-axis, respectively.

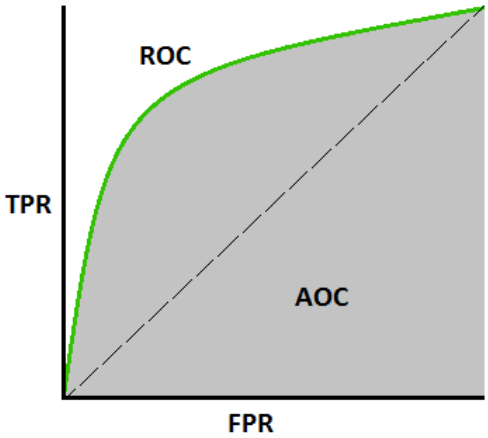


Figure 3. The AUC – ROC Curve

TPR is a synonym of Recall, as a result, it is calculated as:

$$TPR = TP / (TP+FN)$$

FPR describes how much we predict the output wrongly

$$FPR = FP / (TP+FN)$$

From the ROC curve, we can calculate AUC as the area under the curve. The higher value of AUC is the better performance of the classification model. Because the higher value of AUC, the ROC curve is move upward and lean to the TPR y-axis, therefore, at each threshold value in

the curve, there are higher y-axis value and smaller x-axis value, which mean the model provide more True Positive prediction than False Positive prediction.

Classification Accuracy

Classification Accuracy is one of many metrics to measure the accuracy of the classification machine learning model. In short, it measures how many correct predictions the model made over the total number of predictions.

Classification Accuracy = Amount of correct predictions / Total amount of predictions

Classification Accuracy = (TP + TN) / (TP + TN + FP + FN)

The weakness of this approaching is when the sample data has more than 2 classes, the output can be very bias. For example, if the sample has 10% of class A and 90% of class B. The machine learning model can easily get 90% accuracy by only predict the class B, and it will drop dramatically if there is only 40% of class B in the sample

3. Results

Data preparation and processing

Import vital libraries and prepare the financial data

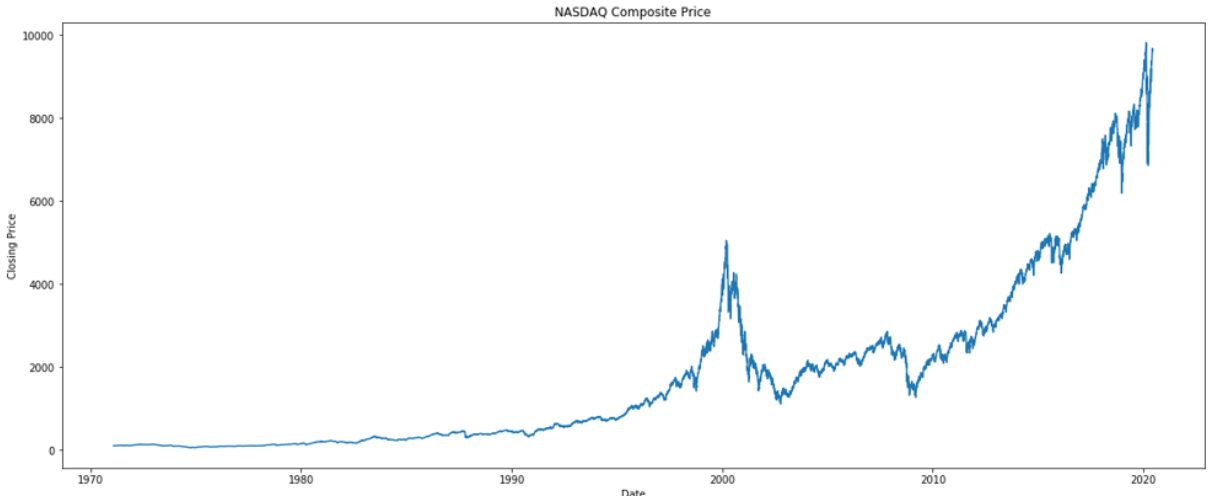
In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pyfolio as pf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM
from sklearn.preprocessing import MinMaxScaler
import yfinance as yf
from sklearn.metrics import confusion_matrix, r2_score, accuracy_score, classification_report, roc_curve, roc_auc_score
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Download the NASDAQ Composite data

In [2]:
```
nasdaq = yf.download("^IXIC", start="1971-02-05", end="2020-06-05", progress=False)
nasdaq.sort_index(inplace=True)
```

Plot daily close price

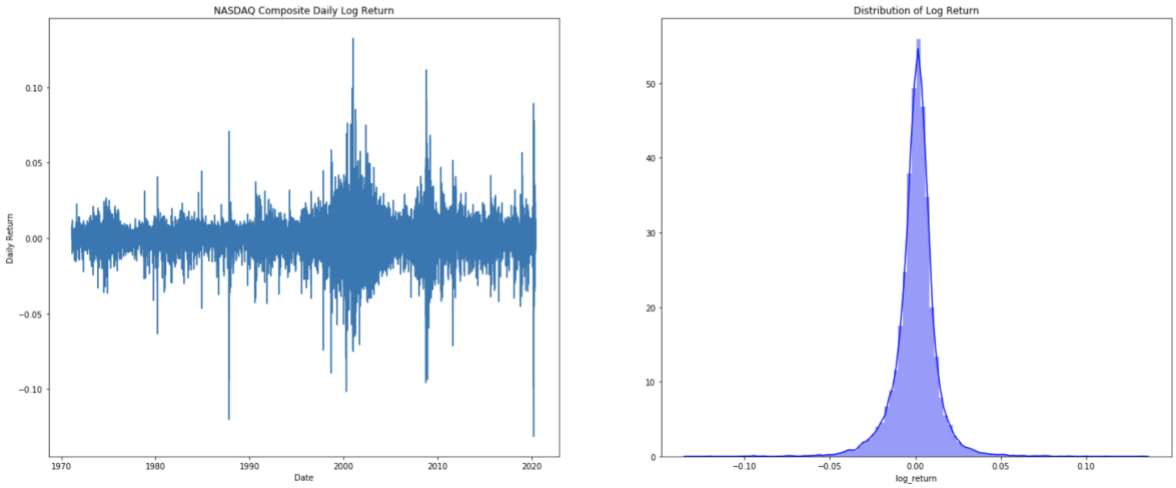
In [3]:
```
plt.figure(figsize=(20, 8))
plt.plot(nasdaq['Close'])
plt.xlabel("Date")
plt.ylabel("Closing Price")
plt.title("NASDAQ Composite Price")
plt.show()
```



After looking at the daily returns chart for Nasdaq we can conclude that the returns are quite volatile at some years and the stock can move +/- 15% on any given day.

- 1980s: Saving and loan crisis
- 1987s: Black Monday
- 2000s: Dot-com bubble, the collapse of internet companies
- 2008: Financial Crisis
- 2020: Covid pandemic

```
[4]: nasdaq['log_return'] = np.log(nasdaq.Close/nasdaq.Close.shift(1))

plt.figure(figsize=(25, 10))
plt.subplot(121)
plt.plot(nasdaq.log_return)
plt.xlabel("Date")
plt.ylabel("Daily Return")
plt.title("NASDAQ Composite Daily Log Return")

plt.subplot(122)
sns.distplot(nasdaq.log_return, bins=100, kde=True, color='blue')
plt.title('Distribution of Log Return')
plt.show()
```



Then prepare the training and testing data

- Select Close Price for modeling
- Scale data as following:

$$X_{scale} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

```
[5]: dataset = nasdaq[['Close']].values
     scaler = MinMaxScaler(feature_range=(0, 1))
     scaled_data = scaler.fit_transform(dataset)
```

Because the prices have serial correlation, then train and label created:

- A single data point contains 60 features corresponding 60 previous data points
- A label (ground truth) is data point at day 61

```
[6]: rolling_windows = 60
     features, labels = [], []
     for i in range(rolling_windows, len(scaled_data)):
         features.append(scaled_data[i-rolling_windows:i, 0])
         labels.append(scaled_data[i, 0])

     features, labels = np.array(features), np.array(labels)
     (features.shape, labels.shape)
```

```
[6]: ((12382, 60), (12382,))
```

Train test split with 80% training size

```
[7]: training_size = 0.8
     x_train, y_train = features[:int(len(features)*training_size), :], labels[:int(len(labels)*training_size)]
     x_test, y_test = features[int(len(features)*training_size):, : ], labels[int(len(labels)*training_size):]

     print("x_train: ", x_train.shape, " - y_train: ", y_train.shape)
     print("x_test: ", x_test.shape, " - y_test: ", y_test.shape)
```

```
x_train:  (9905, 60)  - y_train:  (9905,)
x_test:  (2477, 60)  - y_test:  (2477,)
```

Reshape the data for inputting the model

```
[8]: x_train, y_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1)), np.reshape(y_train, (-1, 1))
     x_test, y_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1)), np.reshape(y_test, (-1, 1))

     print('Train: ', (x_train.shape, y_train.shape))
     print('Test: ', (x_test.shape, y_test.shape))
```

```
Train:  ((9905, 60, 1), (9905, 1))
Test:  ((2477, 60, 1), (2477, 1))
```

Modeling and evaluation

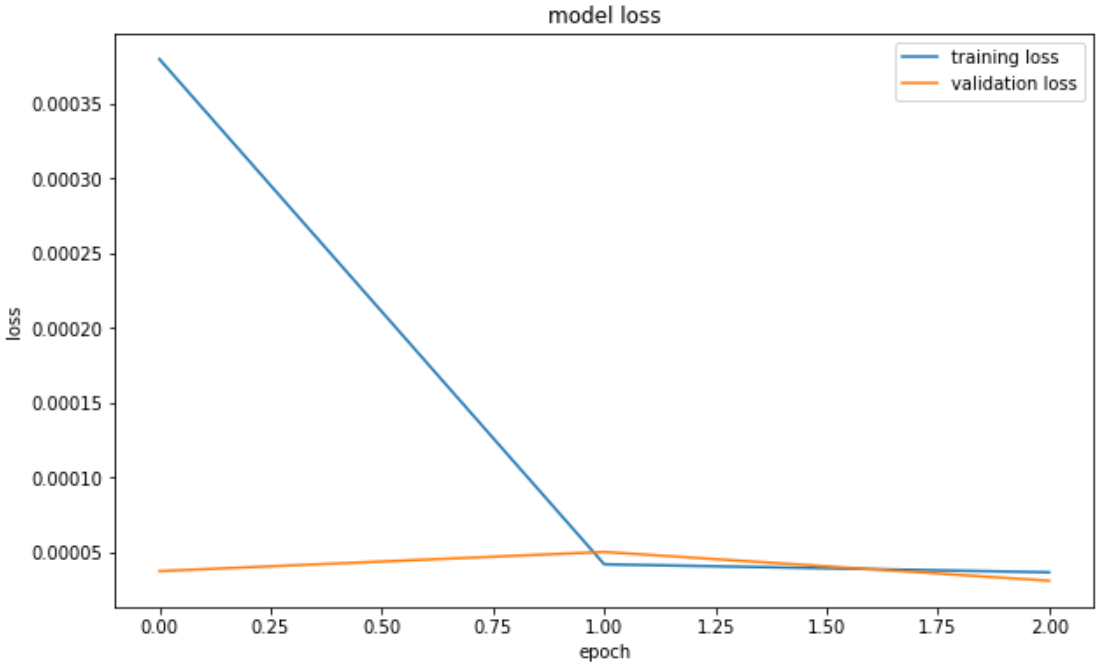
Training the LSTM model. The model has 30,651 parameters in 2 LSTM layers and 1 Dense layer. In this model, we predict the close price.

```
[9]: model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1], x_train.shape[2])))
model.add(LSTM(units=50))
model.add(Dense(1))
model.summary()
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=3, batch_size=32, verbose=1, validation_split=.2)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 60, 50)            10400
_________________________________________________________________
lstm_1 (LSTM)                (None, 50)                20200
_________________________________________________________________
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 30,651
Trainable params: 30,651
```

Plot the training and validation loss. The below graph shows good with no overfitting.

```
In [10]: plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['training loss', 'validation loss'], loc='upper right')
plt.show()
```



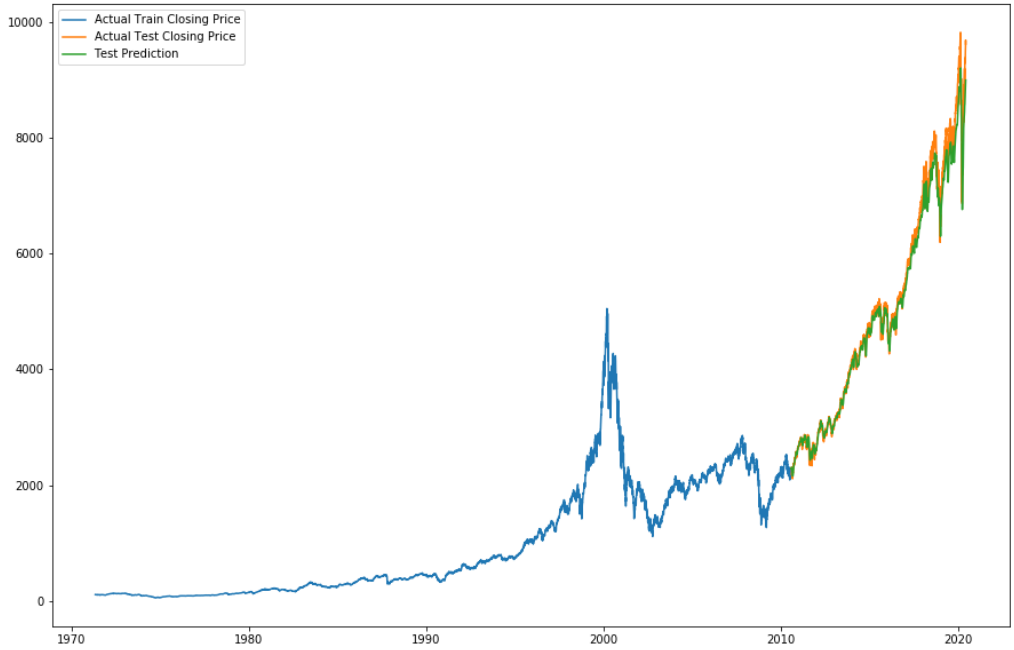
Using root mean square (RMS) and R_2 score to measure the performance of model. The RMS and the R_2 score are 0.0217 and 98.83%, respectively.

```
In [11]: predicted_price = model.predict(x_test)

         rms = np.sqrt(np.mean(np.power((y_test - predicted_price), 2)))
         r2_score = r2_score(y_test, predicted_price) *100
         (rms, r2_score)

Out[11]: (0.021727117823190218, 98.83302270911345)
```

Plot actual testing set and predicted testing set.

```
In [12]: truncated_nasdag = nasdaq[rolling_windows: ]
```

```
In [13]: train = truncated_nasdag[['Close']][: int(len(features)*training_size)]
         test = truncated_nasdag[['Close']][int(len(features)*training_size): ]
         test['Prediction'] = scaler.inverse_transform(predicted_price)
         plt.figure(figsize=(15, 10))
         plt.plot(train['Close'], label='Actual Train Closing Price')
         plt.plot(test['Close'], label='Actual Test Closing Price')
         plt.plot(test['Prediction'], label='Test Prediction ')
         plt.legend(loc='upper left')
         plt.show()
```



- **Trading strategy evaluation and backtesting.**

We will use a simple moving average to create trading strategy. Using the predicted price from the LSTM model, we generate moving average signals based on a long and short window size. If the short moving average is larger than the long moving average, we enter a long position. Otherwise, we exit the position.

- Long mavg: Long moving average
- Short mavg: Short moving average

- If short_mavg > long_mavg → Buy
- If short_mavg < long_mavg → Sell

We first prepare the signals. We generate the moving average columns using the predicted stock prices and shift them upwards by 1 to prevent lookahead bias. In other words, we are using the moving average value of the previous day to make trading decisions for the next day.

Next, we can start with the actual backtesting. We generate a column of positions that we are expected to have at each point in time, based on the moving average signals. Based on the positions, we are able to generate our portfolio value at each point in time, where a long position of 1 will simply translate to a portfolio value of the asset closing price at that point in time. If the position is 0, our portfolio value is simply just the price which we exited the position at. This is equivalent to the most recent closing price during our most recent long position. This can be achieved by filling up all the remaining na values, with the most recent value using the ffill method.

Lastly, for our initial positions at the start of the backtesting where we have yet to enter any positions, we fill them up with the stock value at the starting date. Then, we compute a new column strategyReturns which computes the log returns based on our changes in the portfolio value.

A plot of buy and sell signals are shown below.

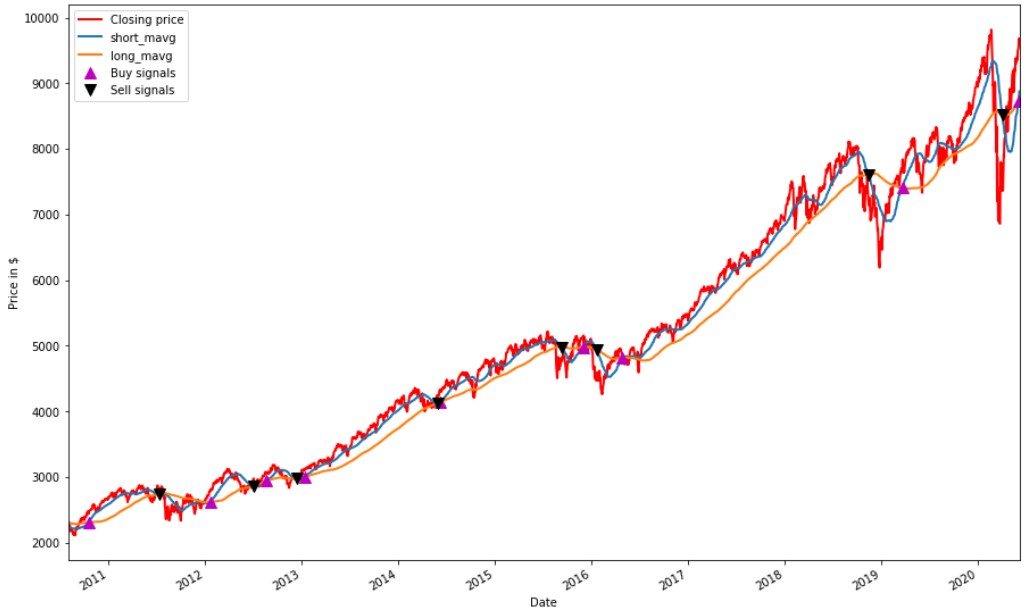
In [14]:
```
# Strategy Parameters
short_window = 40
long_window = 150

startTestDate = truncated_nasdaq[int(len(features)*training_size): ].index[0]
```

In [15]:
```
def create_signals_df(df_close, startTestDate):
    # Initialize the signal DataFrame with signal column
    signals = pd.DataFrame(index=df_close.index)
    signals['Close'] = df_close['Close']

    signals['position'] = 0.0
    signals['portfolio'] = float("Nan")
    # Create short and long simple moving average over the short and long window
    signals['short_mavg'] = df_close['Close'].rolling(window=short_window, min_periods=1, center=False).mean()
    signals['long_mavg'] = df_close['Close'].rolling(window=long_window, min_periods=1, center=False).mean()

    #Shift up by 1 to prevent lookahead bias
    signals['long_mavg'] = signals['long_mavg'].shift(1)
    signals['short_mavg'] = signals['short_mavg'].shift(1)

    #Create signals: if short mavg > long mavg --> buy, otherwise.
    signals['position'][short_window:] = np.where(signals['short_mavg'][short_window:] > signals['long_mavg'][short_window:], 1.0, 0.0)
    # Generate trading orders
    signals['action'] = signals['position'].diff()

    # Generate trading orders
    signals['portfolio'][signals['position'] == 1] = signals['Close']
    signals['portfolio'] = signals['portfolio'].fillna(method='ffill')
    signals['portfolio'] = signals['portfolio'].fillna(signals.loc[startTestDate]['Close'])
    signals['strategyReturns'] = np.log(signals.portfolio/signals.portfolio.shift(1))

    return signals
```

In [16]:
```
def plot_signals(signals_df):
    fig = plt.figure(figsize=(15, 10))
    ax1 = fig.add_subplot(111, ylabel='Price in $')
    signals_df['Close'].plot(ax=ax1, color='r', lw=2., label='Closing price')
    signals_df[['short_mavg', 'long_mavg']].plot(ax=ax1, lw=2.)

    # Plot the buy signals
    ax1.plot(signals_df.loc[signals_df.action == 1.0].index,
             signals_df.short_mavg[signals_df.action == 1.0],
             '^', markersize=10, color='m', label='Buy signals')

    # Plot the sell signals
    ax1.plot(signals_df.loc[signals_df.action == -1.0].index,
             signals_df.short_mavg[signals_df.action == -1.0],
             'v', markersize=10, color='k', label='Sell signals')

    plt.legend(loc='upper left')
    plt.show()
```

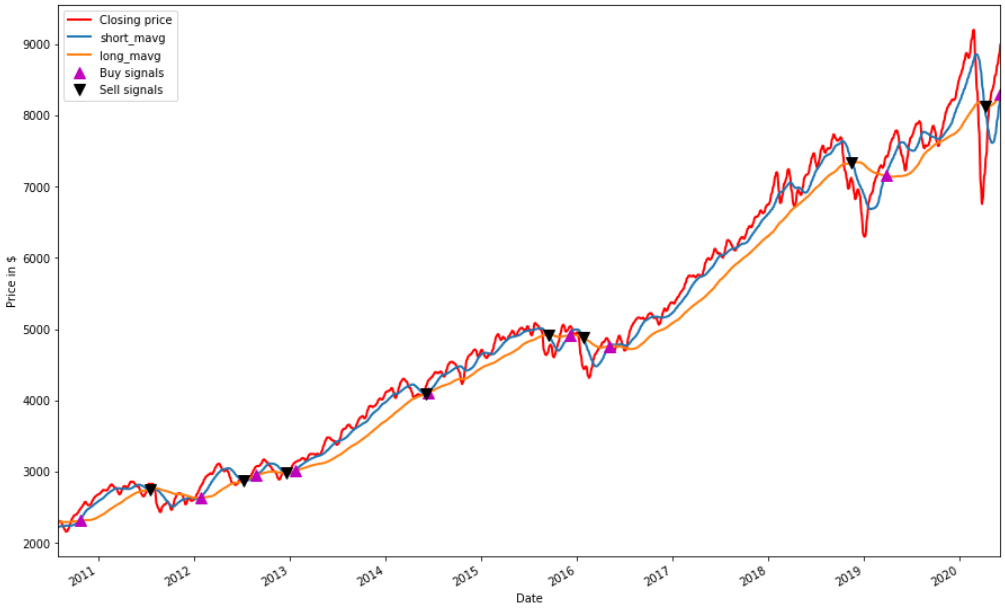
Actual signal is shown below

In [17]:
```
signals = create_signals_df(truncated_nasdag, startTestDate)
plot_signals(signals.query('index >= @startTestDate'))
```



Prediction signal is shown below

In [18]:
```
predicted_data = pd.concat([train, test[['Prediction']].rename(columns={'Prediction': 'Close'})])
signals_predicted = create_signals_df(predicted_data, startTestDate)
plot_signals(signals_predicted.query('index >= @startTestDate'))
```



The classification report show that the accuracy is 99%.

```
In [19]: signals_test = signals.query('index >= @startTestDate')
         signals_predicted_test = signals_predicted.query('index >= @startTestDate')

In [20]: confusion_matrix(signals_test.action, signals_predicted_test.action, labels=[-1, 0, 1])

Out[20]: array([[   0,    8,    0],
                [   8, 2443,    9],
                [   0,    9,    0]], dtype=int64)

In [21]: # Accuracy score of confusion matrix
         accuracy_score(signals_test.action, signals_predicted_test.action)

Out[21]: 0.9862737182075091

In [22]: print(classification_report(signals_test.action, signals_predicted_test.action))

                precision    recall  f1-score   support

          -1.0       0.00      0.00      0.00         8
           0.0       0.99      0.99      0.99      2460
           1.0       0.00      0.00      0.00         9

      accuracy                           0.99      2477
     macro avg       0.33      0.33      0.33      2477
  weighted avg       0.99      0.99      0.99      2477
```

Performance Metrics.

Nevertheless, we see observe that confusion matrix presumes poor prediction for sell and buy signals. Additionally, classification report computes 0 precision, recall and f1-score. Moreover, its accuracy score is very high, that may lead to wrong conclusion about correctness of

confusion matrix. With deeper analysis, we can get more reliable and correct estimates for this performance metrics.

	tvalue_minus1_t	tvalue_minus1_p	tvalue_plus1_t	tvalue_plus1_p	delta_minus1	delta_plus1	new_minus1	new_plus1
0	2011-07-13	2011-07-19	2010-10-20	2010-10-28	-6 days	-8 days	0.0	-2.0
1	2012-07-03	2012-07-12	2012-01-23	2012-01-27	-9 days	-4 days	-3.0	2.0
2	2012-12-13	2012-12-19	2012-08-20	2012-08-24	-6 days	-4 days	0.0	2.0
3	2014-05-30	2014-06-05	2013-01-14	2013-01-22	-6 days	-8 days	0.0	-2.0
4	2015-09-09	2015-09-15	2014-06-09	2014-06-16	-6 days	-7 days	0.0	-1.0
5	2016-01-21	2016-01-27	2015-11-30	2015-12-07	-6 days	-7 days	0.0	-1.0
6	2018-11-13	2018-11-19	2016-04-27	2016-05-03	-6 days	-6 days	0.0	0.0
7	2020-04-01	2020-04-07	2019-03-20	2019-03-26	-6 days	-6 days	0.0	0.0

Now we see that our model has very strong 6-day shift(lag) in prediction of time of signals. After taking it into consideration we can recalculate our Performance Metrics. To be more precise, only exact date matches would be considered. In addition, we should add previously dropped observation of 9th buy signal.

```
conf_matr = confusion_matrix(final_df.testy, final_df.yhat_classes)
print("New confusion matrix \n",conf_matr)
```

```
New confusion matrix
 [[7 1]
 [7 2]]
```

```
# accuracy: (tp + tn) / (p + n)
accuracy = accuracy_score(final_df.testy, final_df.yhat_classes)
print('Accuracy: %f' % accuracy)
# precision tp / (tp + fp)
precision = precision_score(final_df.testy, final_df.yhat_classes)
print('Precision: %f' % precision)
# recall: tp / (tp + fn)
recall = recall_score(final_df.testy, final_df.yhat_classes)
print('Recall: %f' % recall)
# f1: 2 tp / (2 tp + fp + fn)
f1 = f1_score(final_df.testy, final_df.yhat_classes)
print('F1 score: %f' % f1)
```

```
Accuracy: 0.529412
Precision: 0.666667
Recall: 0.222222
F1 score: 0.333333
```

As we only used exact match, i.e. considered worst case scenario, so our performance metrics results are considerably lower. In reality, taking into account additional timespan of one-two

trading days to catch signal would have considerable positive effect on scores, which will be more "real" values for performance metrics.

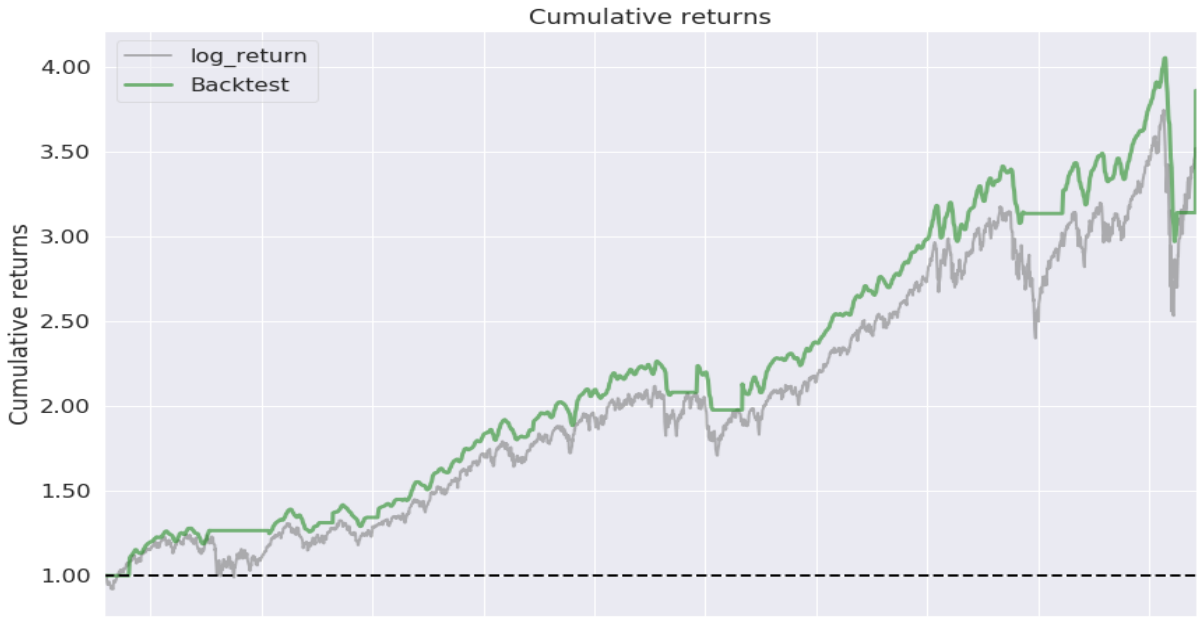
Fund Factsheet

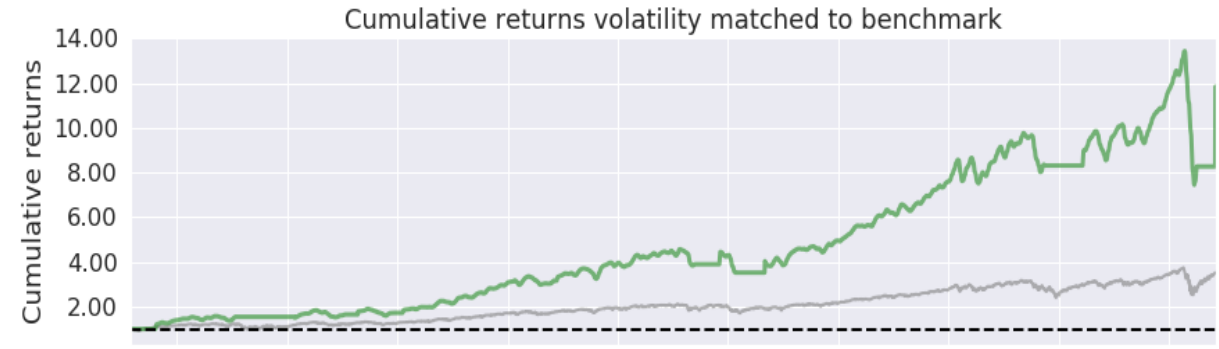
Our strategy involves state of the art machine learning models, trained with huge amounts of clean data and generating trading signals that help us to make timely trading decisions in the market. These trading signals are further smoothened so that we are able to filter out noise and extract meaningful information from the market. Our strategy seeks interest in long-term growth/returns, rather than short-term profits. This helps us to further minimize risk and volatility, helping our investors achieve a more stable return.

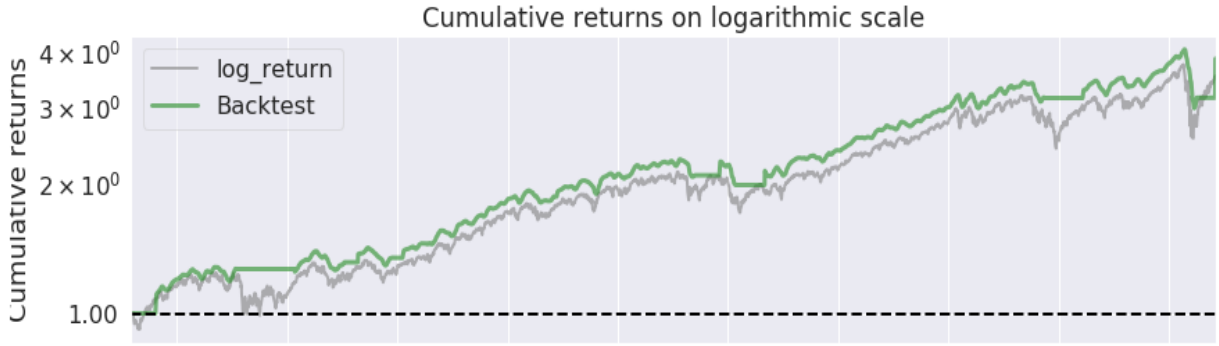
The following is the performance report of our fund for the last 10 years.

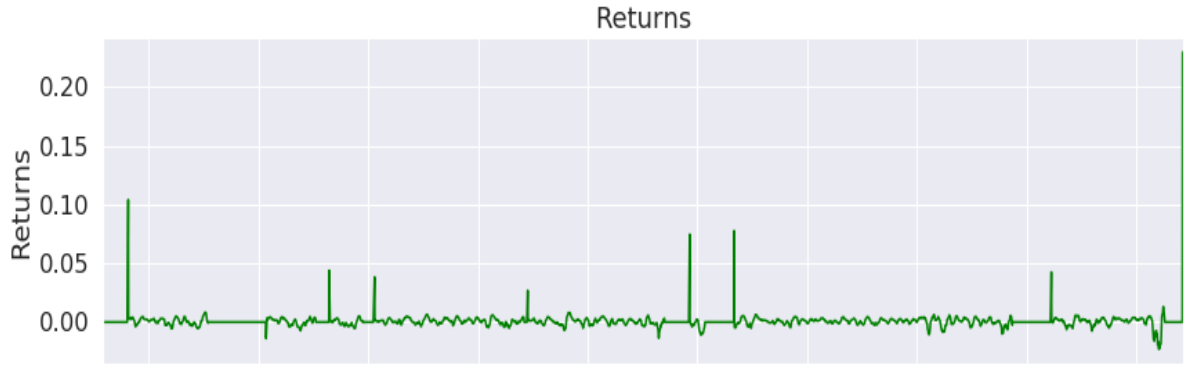
Start date	2010-08-03
End date	2020-06-04
Total months	117
	Backtest
Annual return	14.7%
Cumulative returns	285.8%
Annual volatility	10.2%
Sharpe ratio	1.40
Calmar ratio	0.55
Stability	0.97
Max drawdown	-26.7%
Omega ratio	1.70
Sortino ratio	4.07
Skew	21.61
Kurtosis	701.23
Tail ratio	1.07
Daily value at risk	-1.2%
Alpha	0.16
Beta	-0.01

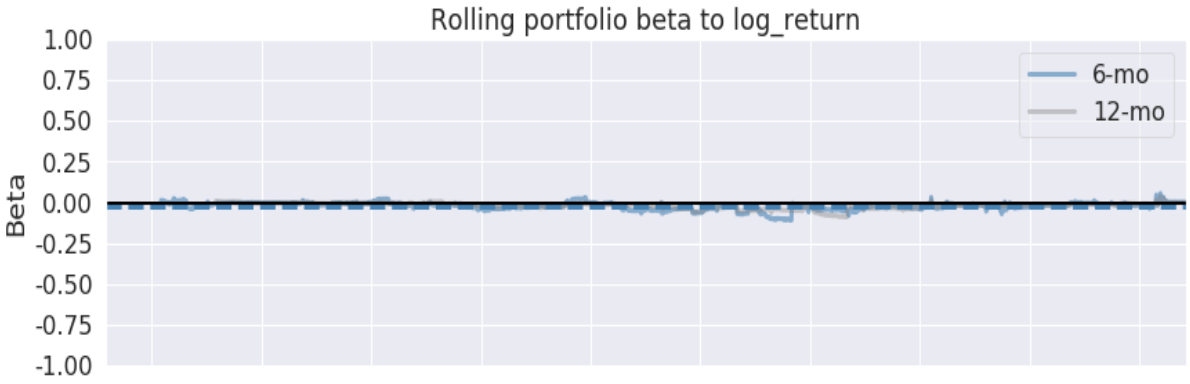
Worst drawdown periods	Net drawdown in %	Peak date	Valley date	Recovery date	Duration
0	26.73	2020-02-24	2020-03-26	NaT	NaN
1	12.71	2015-07-27	2016-01-26	2016-08-12	275
2	9.89	2018-09-07	2018-11-02	2019-04-29	167
3	9.56	2012-04-05	2012-06-08	2012-09-17	118
4	8.92	2012-09-26	2012-11-23	2013-02-14	102

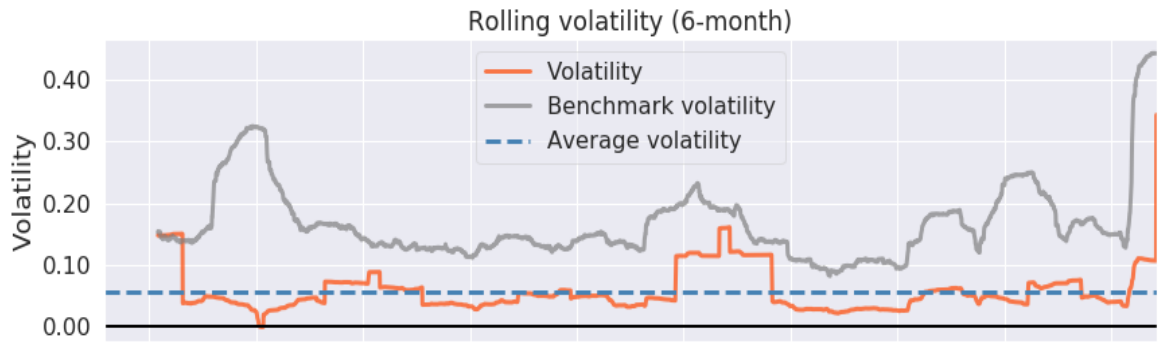

Cumulative returns
log_return
Backtest
Cumulative returns
4.00
3.50
3.00
2.50
2.00
1.50
1.00

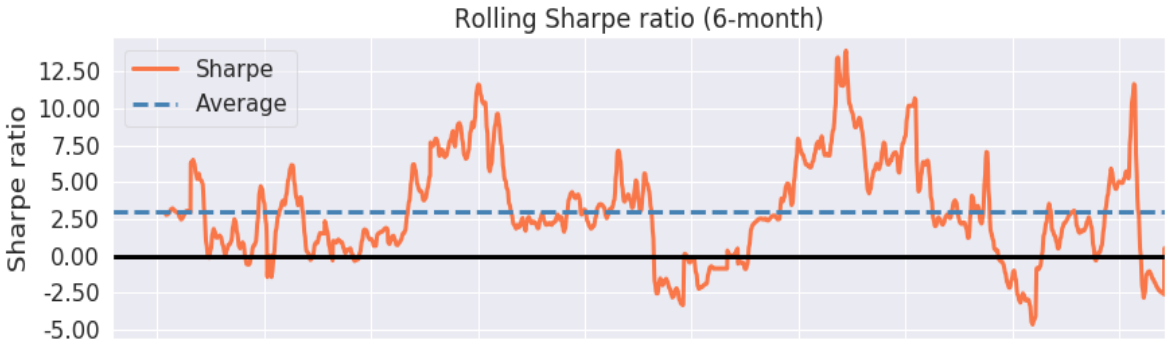

Cumulative returns volatility matched to benchmark
Cumulative returns
14.00
12.00
10.00
8.00
6.00
4.00
2.00

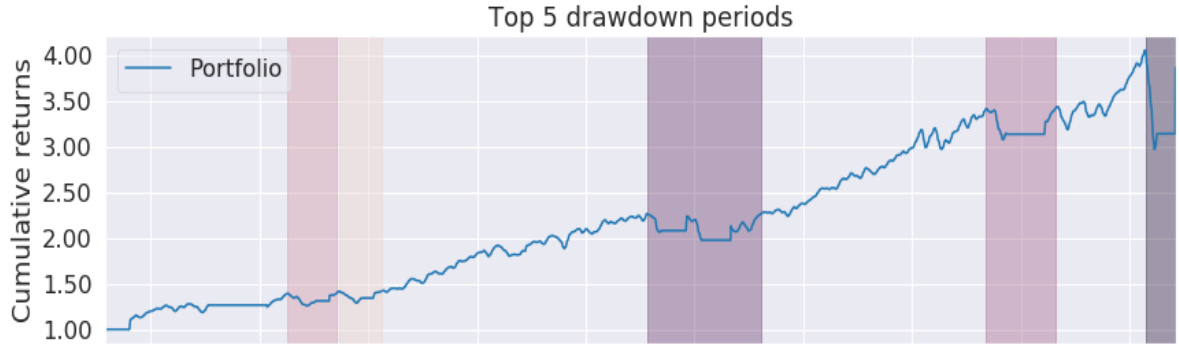

Cumulative returns on logarithmic scale
Cumulative returns
4×10^0
3×10^0
2×10^0
1.00
log_return
Backtest

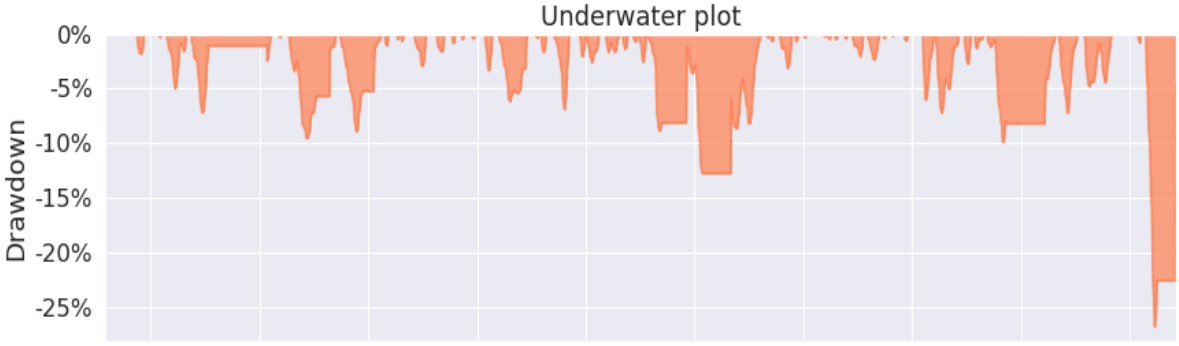

Returns
Returns
0.20
0.15
0.10
0.05
0.00

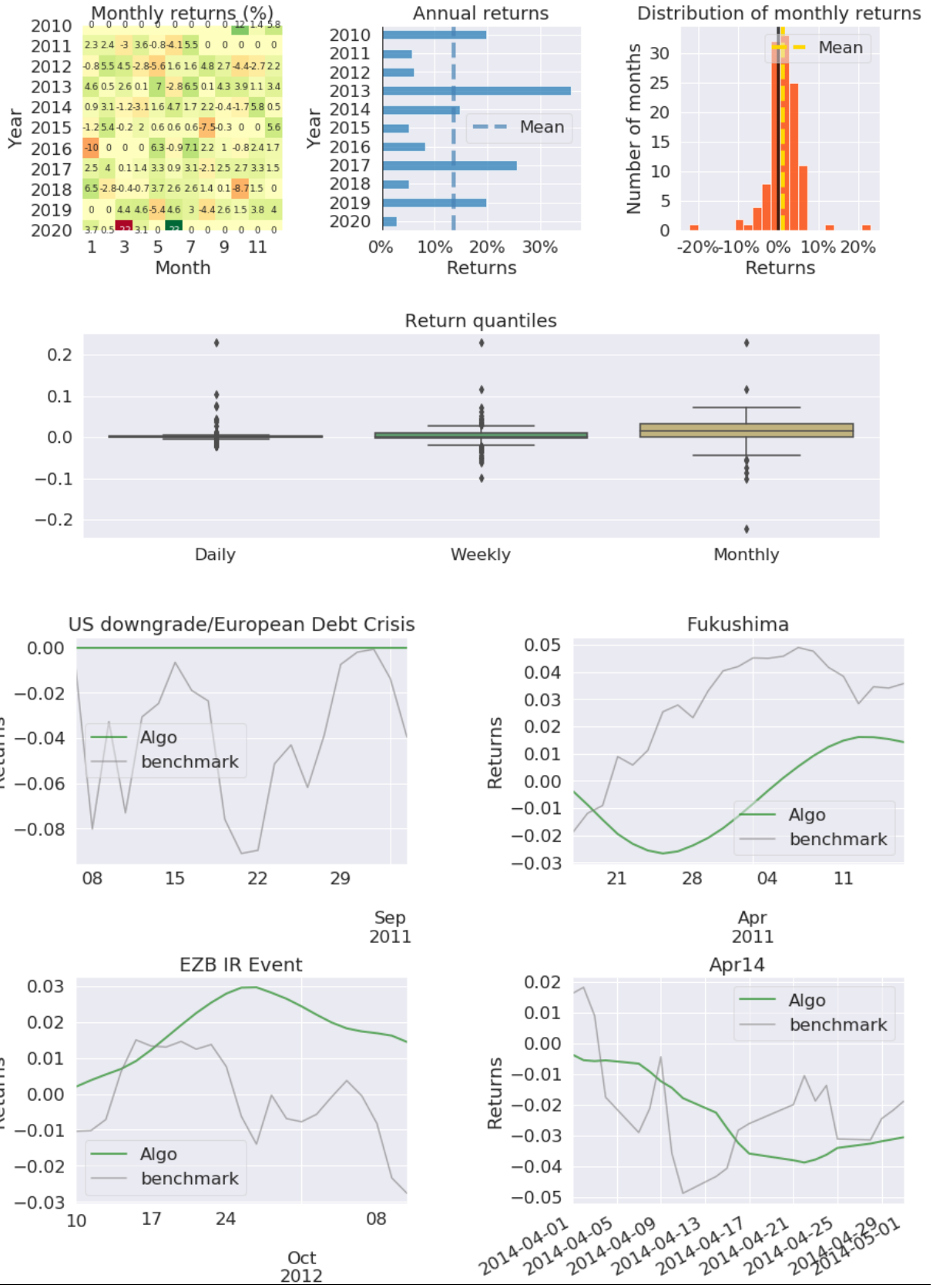

Rolling portfolio beta to log_return
Beta
1.00
0.75
0.50
0.25
0.00
-0.25
-0.50
-0.75
-1.00
6-mo
12-mo

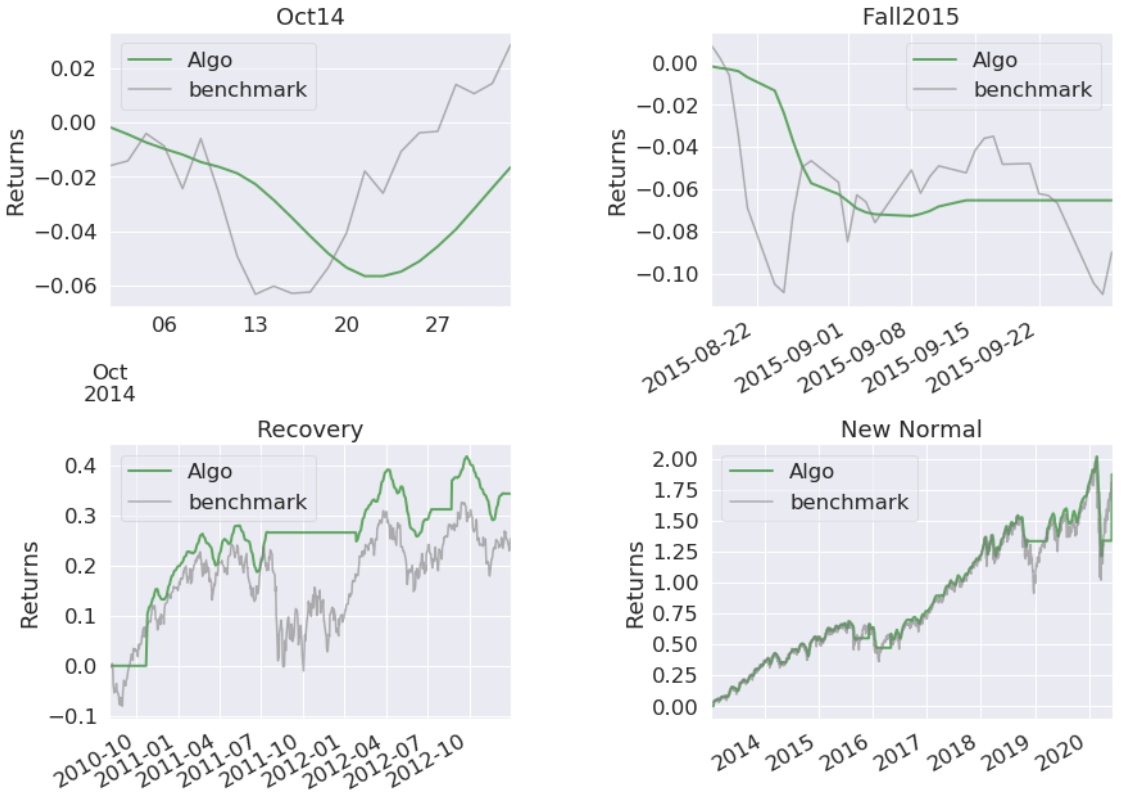

Rolling volatility (6-month)
Volatility
Benchmark volatility
Average volatility
Volatility
0.40
0.30
0.20
0.10
0.00


Rolling Sharpe ratio (6-month)
Sharpe
Average
Sharpe ratio
12.50
10.00
7.50
5.00
2.50
0.00
-2.50
-5.00


Top 5 drawdown periods
Portfolio
Cumulative returns
4.00
3.50
3.00
2.50
2.00
1.50
1.00


Underwater plot
Drawdown
0%
-5%
-10%
-15%
-20%
-25%


Monthly returns (%)
Year
Month
Annual returns
Mean
Returns
Distribution of monthly returns
Number of months
Mean
Returns
Return quantiles
Daily
Weekly
Monthly
US downgrade/European Debt Crisis
Returns
Algo
benchmark
Sep
2011
Fukushima
Returns
Algo
benchmark
Apr
2011
EZB IR Event
Returns
Algo
benchmark
Oct
2012
Apr14
Returns
Algo
benchmark



For the past 10 years, our strategy has beat the NASDAQ benchmark, with an overall alpha of 0.16. From the graph of the cumulative returns, we can see that our strategy has consistently generated higher returns than the index, with an average annual return of 14.7%, with a Sharpe ratio of 1.40, compared to the benchmark Sharpe ratio of 0.47. One of the reasons for our higher Sharpe ratio, other than higher returns, is our ability to manage risks. Looking at the rolling volatility plot, the volatility of our returns is consistently lower than that of the benchmark for the later parts of the year. This can also be seen from our low beta value of -0.01, which shows that the returns of our strategy are not affected by the market's returns. We also see from the annual returns chart that our strategy has positive returns every year for all 10 years.

One downfall of our strategy is our huge max drawdown of 25%. Investors with a low risk appetite would not be able to tolerate. Perhaps, this can be improved by implementing a stop loss signal in our strategy, to exit all positions once the value has fallen by more than a certain threshold. Compared to the benchmark, our strategy does not do as well when it comes to recovering from the financial crisis. Perhaps, the financial crisis represented a major regime change and our machine learning model should be re-trained to adapt to the new regime, whenever there is a structural break.

4. Conclusion

- In this report, we have presented the step by step to building a trading strategy using machine learning from data preparation, processing, modeling, evaluation and backtesting. Our simple strategy outperforms the benchmark with alpha 0.16.
- The data we took is only close price, then transform it into 60 features. The LSTM algorithm shows the power of solving time series problems with high accuracy without many hidden layers. We believe that collecting more data like macro econometrics will result in better results.
- Our solution is using the regression to predict the price, then using the predicted price as an input of classification evaluation (due to unbalance labels, we cannot classify directly). This problem may magnify the error and take more time to take trading orders. Future solutions should be approached in a different direction.
- The performance metrics are really important, depend on the target of problems. In this report, we present almost methods from regression, classification and adjust to adapt the situation.
- Performance indicators are consistent, even with assumption of exact time match predictions of buy and sell signal with tested results. Nevertheless, with more soft "realistic" assumption, they would be even more consistent.
- The Funfact sheet is tracking the trading strategy performance with many criteria that make us easy to manage the portfolio returns and change the strategy if needed.

References

Chung, Junyoung, et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling." arXiv preprint arXiv:1412.3555 (2014)

Classification: ROC Curve and AUC | Machine Learning Crash Course. (n.d.). Retrieved June 11, 2020, from https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

Dauphin, Yann N., et al. "Language Modeling with Gated Convolutional Networks." arXiv preprint arXiv:1612.08083 (2016)

Gers, F. A.; Schmidhuber, J. & Cummins, F. A. (2000), 'Learning to Forget: Continual Prediction with LSTM.', Neural Computation 12 (10), 2451-2471

Gregor, Karol, et al. "DRAW: A recurrent neural network for image generation." arXiv preprint arXiv:1502.04623 (2015)

Kiros, Ryan, et al. Advances in neural information processing systems. 2015

Narkhede, S. (2019, August 29). Understanding Confusion Matrix. Retrieved June 11, 2020, from https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

Narkhede, S. (2019, May 26). Understanding AUC - ROC Curve. Retrieved June 11, 2020, from https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5