Trading-AutoARIMA-Model

September 29, 2021

1 AutoARIMA on Stock Prices

[1]:
```
# Importing Libraries
import pandas as pd
import numpy as np
from pmdarima.arima import AutoARIMA
import plotly.express as px
from statistics import mean
import plotly.graph_objects as go
from tqdm.notebook import tqdm
from sklearn.metrics import mean_squared_error
from datetime import date, timedelta
import yfinance as yf
```

Choosing Stocks that have significantly lost value in the past few years

[2]:
```
# Getting the date five years ago to download the current timeframe
years = (date.today() - timedelta(weeks=260)).strftime("%Y-%m-%d")

# Stocks to analyze
stocks = ['GE', 'GPRO', 'FIT', 'F']

# Getting the data for multiple stocks
df = yf.download(stocks, start=years).dropna()

print("Rows in DataFrame: ", df.shape[0])
```

```
[*********************100%***********************]  4 of 4 completed
Rows in DataFrame:  1255
```

[3]:
```
# Storing the dataframes in a dictionary
stock_df = {}

for col in set(df.columns.get_level_values(0)):

    # Assigning the information (High, Low, etc.) for each stock in the␣
↪dictionary
    stock_df[col] = df[col]
```

2 Preprocessing Data

Scale the data using a logarithmic scale. Also rounding the log result by 2 decimal points in order to reduce any unnecessary noise.

[4]:
```
# Finding the log returns
stock_df['LogReturns'] = stock_df['Adj Close'].apply(np.log).diff().dropna()

# Trying out Moving average
stock_df['MovAvg'] = stock_df['Adj Close'].rolling(10).mean().dropna()

# Logarithmic scaling of the data and rounding the result
stock_df['Log'] = stock_df['MovAvg'].apply(np.log).apply(lambda x: round(x, 2))
```

3 Visualizing the Data

[5]:
```
px.line(stock_df['MovAvg'],
        x=stock_df['MovAvg'].index,
        y=stock_df['MovAvg'].columns,
        labels={'variable': 'Stock',
                'value': 'Price'},
        title='Moving Average')
```

[6]:
```
px.line(stock_df['Log'],
        x=stock_df['Log'].index,
        y=stock_df['Log'].columns,
        labels={'variable': 'Stock',
                'value': 'Log Scale'},
        title='Log of Moving Averages')
```

3.1 Optimum Parameter Search Function

[7]:
```
opt_param = AutoARIMA(start_p=0, start_q=0,
                      start_P=0, start_Q=0,
                      max_p=8, max_q=8,
                      max_P=5, max_Q=5,
                      error_action='ignore',
                      information_criterion='bic',
                      suppress_warnings=True)

for stock in tqdm(stocks):

    opt_param.fit(stock_df['Log'][stock])

    print(f'Summary for {stock}', '--'*20)
    display(opt_param.summary())
```

```
HBox(children=(FloatProgress(value=0.0, max=4.0), HTML(value='')))
```

```
Summary for GE ----------------------------------------
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 1246
Model:               SARIMAX(3, 1, 0)   Log Likelihood                4601.907
Date:                Wed, 16 Sep 2020   AIC                          -9195.814
Time:                        20:24:14   BIC                          -9175.307
Sample:                             0   HQIC                         -9188.103
                               - 1246
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.2809      0.027     10.510      0.000       0.229       0.333
ar.L2          0.4130      0.025     16.294      0.000       0.363       0.463
ar.L3          0.1377      0.025      5.431      0.000       0.088       0.187
sigma2      3.602e-05   1.17e-06     30.733      0.000    3.37e-05    3.83e-05
===================================================================================
Ljung-Box (Q):                       74.54   Jarque-Bera (JB):                 70.
 ↪92
Prob(Q):                              0.00   Prob(JB):                          0.
 ↪00
Heteroskedasticity (H):               2.56   Skew:                             -0.
 ↪03
Prob(H) (two-sided):                  0.00   Kurtosis:                          4.
 ↪17
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients␣
 ↪(complex-step).
"""
```

```
Summary for GPRO ----------------------------------------
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 1246
Model:               SARIMAX(2, 1, 0)   Log Likelihood                4244.156
Date:                Wed, 16 Sep 2020   AIC                          -8482.312
Time:                        20:24:19   BIC                          -8466.932
Sample:                             0   HQIC                         -8476.529
```

```
                             - 1246
Covariance Type:                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.4957      0.025     20.216      0.000       0.448       0.544
ar.L2          0.3498      0.026     13.374      0.000       0.299       0.401
sigma2      6.396e-05   2.05e-06     31.134      0.000    5.99e-05     6.8e-05
===================================================================================
Ljung-Box (Q):                      137.89   Jarque-Bera (JB):                86.
↪04
Prob(Q):                              0.00   Prob(JB):                         0.
↪00
Heteroskedasticity (H):               1.03   Skew:                             0.
↪22
Prob(H) (two-sided):                  0.73   Kurtosis:                         4.
↪21
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients␣
↪(complex-step).
"""

Summary for FIT ----------------------------------------

<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 1246
Model:               SARIMAX(1, 2, 0)   Log Likelihood                4287.748
Date:                Wed, 16 Sep 2020   AIC                          -8571.496
Time:                        20:24:20   BIC                          -8561.244
Sample:                             0   HQIC                         -8567.641
                             - 1246
Covariance Type:                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.4179      0.023    -17.851      0.000      -0.464      -0.372
sigma2      5.937e-05   1.74e-06     34.080      0.000     5.6e-05    6.28e-05
===================================================================================
Ljung-Box (Q):                      153.50   Jarque-Bera (JB):               191.
↪75
Prob(Q):                              0.00   Prob(JB):                         0.
↪00
```

```
Heteroskedasticity (H):               0.57   Skew:                            0.
↪23
Prob(H) (two-sided):                  0.00   Kurtosis:                        4.
↪87
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients␣
↪(complex-step).
"""

Summary for F ----------------------------------------

<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 1246
Model:               SARIMAX(3, 1, 0)   Log Likelihood                4651.277
Date:                Wed, 16 Sep 2020   AIC                          -9294.553
Time:                        20:24:26   BIC                          -9274.046
Sample:                             0   HQIC                         -9286.842
                               - 1246
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.1654      0.027      6.029      0.000       0.112       0.219
ar.L2          0.3871      0.025     15.589      0.000       0.338       0.436
ar.L3          0.2234      0.025      9.043      0.000       0.175       0.272
sigma2      3.328e-05   1.23e-06     27.162      0.000    3.09e-05    3.57e-05
===================================================================================
Ljung-Box (Q):                       83.32   Jarque-Bera (JB):               18.
↪48
Prob(Q):                              0.00   Prob(JB):                        0.
↪00
Heteroskedasticity (H):               1.43   Skew:                            0.
↪03
Prob(H) (two-sided):                  0.00   Kurtosis:                        3.
↪59
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients␣
↪(complex-step).
"""
```

4 Using the ARIMA Model

Using the price history from the past N days to make predictions

[76]:
```
# Days in the past to train on
days_to_train = 180

# Days in the future to predict
days_to_predict = 5

# Establishing a new DF for predictions
stock_df['Predictions'] = pd.DataFrame(index=stock_df['Log'].index,
                                       columns=stock_df['Log'].columns)

# Iterate through each stock
for stock in tqdm(stocks):

    # Current predicted value
    pred_val = 0

    # Training the model in a predetermined date range
    for day in tqdm(range(1000,
                          stock_df['Log'].shape[0]-days_to_predict)):

        # Data to use, containing a specific amount of days
        training = stock_df['Log'][stock].iloc[day-days_to_train:day+1].dropna()

        # Determining if the actual value crossed the predicted value
        cross = ((training[-1] >= pred_val >= training[-2]) or
                 (training[-1] <= pred_val <= training[-2]))

        # Running the model when the latest training value crosses the␣
↪predicted value or every other day
        if cross or day % 2 == 0:

            # Finding the best parameters
            model    = AutoARIMA(start_p=0, start_q=0,
                                 start_P=0, start_Q=0,
                                 max_p=8, max_q=8,
                                 max_P=5, max_Q=5,
                                 error_action='ignore',
                                 information_criterion='bic',
                                 suppress_warnings=True)

            # Getting predictions for the optimum parameters by fitting to the␣
↪training set
            forecast = model.fit_predict(training,
```

```
                                                    n_periods=days_to_predict)

            # Getting the last predicted value from the next N days
            stock_df['Predictions'][stock].iloc[day:day+days_to_predict] = np.
↪exp(forecast[-1])


            # Updating the current predicted value
            pred_val = forecast[-1]
```

```
HBox(children=(FloatProgress(value=0.0, max=4.0), HTML(value='')))

HBox(children=(FloatProgress(value=0.0, max=241.0), HTML(value='')))


HBox(children=(FloatProgress(value=0.0, max=241.0), HTML(value='')))


HBox(children=(FloatProgress(value=0.0, max=241.0), HTML(value='')))


HBox(children=(FloatProgress(value=0.0, max=241.0), HTML(value='')))
```

5 Predictions vs Actual Values

[77]:
```
# Shift ahead by 1 to compare the actual values to the predictions
pred_df = stock_df['Predictions'].shift(1).astype(float).dropna()

pred_df
```

[77]:

Date	F	FIT	GE	GPRO
2019-09-27	8.844738	3.908333	8.977144	4.695571
2019-09-30	8.844738	3.908333	8.977144	4.695571
2019-10-01	8.684210	3.948133	9.207331	4.953032
2019-10-02	8.684210	3.948133	9.023872	4.953032
2019-10-03	8.579128	3.770709	8.851652	5.278704
…	…	…	…	…
2020-09-10	6.889510	6.372468	6.203521	4.195989
2020-09-11	6.889510	6.372468	6.203521	4.195989
2020-09-14	6.889510	6.372468	6.203521	4.195989
2020-09-15	6.889510	6.372468	6.203521	4.195989
2020-09-16	6.889510	6.372468	6.203521	4.195989

[245 rows x 4 columns]

5.1 Plotting the Predictions

Comparing the actual values with the predictions

[78]:
```
for stock in stocks:

    fig = go.Figure()

    # Plotting the actual values
    fig.add_trace(go.Scatter(x=pred_df.index,
                             y=stock_df['MovAvg'][stock].loc[pred_df.index],
                             name='Actual Moving Average',
                             mode='lines'))

    # Plotting the predicted values
    fig.add_trace(go.Scatter(x=pred_df.index,
                             y=pred_df[stock],
                             name='Predicted Moving Average',
                             mode='lines'))

    # Setting the labels
    fig.update_layout(title=f'Predicting the Moving Average for the Next␣
↪{days_to_predict} days for {stock}',
                      xaxis_title='Date',
                      yaxis_title='Prices')

    fig.show()
```

5.2 Evaluation Metric

[79]:
```
for stock in stocks:

    # Finding the root mean squared error
    rmse = mean_squared_error(stock_df['MovAvg'][stock].loc[pred_df.index],
                              pred_df[stock],
                              squared=False)

    print(f"On average, the model is off by {rmse} for {stock}\n")
```

```
On average, the model is off by 0.33234762849920296 for GE

On average, the model is off by 0.15697167679670937 for GPRO

On average, the model is off by 0.27098970427817703 for FIT
```

```
On average, the model is off by 0.19597127039371282 for F
```

6 Trading Signal

Turning the model into a Trading Signal

[80]:
```python
def get_positions(difference, thres=3, short=True):
    """
    Compares the percentage difference between actual values and the respective␣
 ↪predictions.

    Returns the decision or positions to long or short based on the difference.

    Optional: shorting in addition to buying
    """

    if difference > thres/100:

        return 1


    elif short and difference < -thres/100:

        return -1


    else:

        return 0
```

6.0.1 Creating a Trading DF

Note: *On Preventing Lookahead Bias*

For example, if the model is ran after hours and a position is established on the next day's opening, then a shift ahead of 1 is ok. But if a position is established on the next day, near the close, then it needs to be shifted ahead by 2, because the newly established position missed any gains or losses that day. These are due to the fact that gains or losses in the day are determined when a trade is entered.

(This can also determine how long the predicted forecast remains valid.)

[81]:
```python
# Creating a DF for trading the model
trade_df = {}

# Getting the percentage difference between the predictions and the actual␣
 ↪values
```

```
trade_df['PercentDiff'] = (stock_df['Predictions'].dropna() /
                           stock_df['MovAvg'].loc[stock_df['Predictions'].
 ↪dropna().index]) - 1

# Getting positions
trade_df['Positions'] = trade_df['PercentDiff'].applymap(lambda x:␣
 ↪get_positions(x,
                                                                                          ␣
 ↪ thres=1,
                                                                                          ␣
 ↪ short=True) / len(stocks))

# Preventing lookahead bias by shifting the positions
trade_df['Positions'] = trade_df['Positions'].shift(2).dropna()

# Getting Log Returns
trade_df['LogReturns'] = stock_df['LogReturns'].loc[trade_df['Positions'].index]

display(trade_df['PercentDiff'].tail(20))
display(trade_df['Positions'].tail(20))
```

```
                      F          FIT          GE        GPRO
Date
2020-08-18    0.0353646 -7.70806e-06  0.00369288  -0.0186238
2020-08-19   0.00366808  0.000501029   0.0142783  -0.0463313
2020-08-20   0.00495961   0.00238728   0.0152097  -0.0327468
2020-08-21   -0.0340648   0.00432124  0.00436857   -0.038719
2020-08-24   -0.0325388   0.00432124  0.00498465  -0.0379223
2020-08-25   0.00356951    0.0030807  -0.0273696 -0.00462865
2020-08-26   -0.0340272   0.00119314   -0.023769 -0.00297693
2020-08-27   -0.0180615  -0.00113811  0.00267291  0.00159371
2020-08-28   -0.0166348   0.00661095  0.00344825  0.00326617
2020-08-31   0.00327803    0.0064618  -0.0289893  -0.0237918
2020-09-01    0.0041554   0.00740654  -0.0252128  -0.0162515
2020-09-02   0.00298591   0.00666406 -0.00159516   -0.035194
2020-09-03   0.00327803    0.0071366 -0.00237045  -0.0203072
2020-09-04 -0.000216194  -0.00217568 -0.00407183  -0.0581904
2020-09-08 -0.000941098  -0.00123773  0.00355205  -0.0404064
2020-09-09  -0.00137553  7.34863e-05  -0.0242968  -0.0429948
2020-09-10  -0.00267657   0.00274874  -0.0168747  -0.0229388
2020-09-11  -0.00397423   0.00527973  -0.0085471 -0.00202422
2020-09-14  -0.00655944   0.00623215 -0.00120419   0.0198054
2020-09-15  -0.00970098   0.00591447  0.00267025   0.0374556

               F  FIT    GE  GPRO
Date
2020-08-18  0.25  0.0  0.25  0.00
2020-08-19  0.25  0.0  0.00 -0.25
```

```
2020-08-20  0.25  0.0  0.00 -0.25
2020-08-21  0.00  0.0  0.25 -0.25
2020-08-24  0.00  0.0  0.25 -0.25
2020-08-25 -0.25  0.0  0.00 -0.25
2020-08-26 -0.25  0.0  0.00 -0.25
2020-08-27  0.00  0.0 -0.25  0.00
2020-08-28 -0.25  0.0 -0.25  0.00
2020-08-31 -0.25  0.0  0.00  0.00
2020-09-01 -0.25  0.0  0.00  0.00
2020-09-02  0.00  0.0 -0.25 -0.25
2020-09-03  0.00  0.0 -0.25 -0.25
2020-09-04  0.00  0.0  0.00 -0.25
2020-09-08  0.00  0.0  0.00 -0.25
2020-09-09  0.00  0.0  0.00 -0.25
2020-09-10  0.00  0.0  0.00 -0.25
2020-09-11  0.00  0.0 -0.25 -0.25
2020-09-14  0.00  0.0 -0.25 -0.25
2020-09-15  0.00  0.0  0.00  0.00
```

6.1 Plotting the Positions

[87]:
```
# Getting the number of positions
pos = trade_df['Positions'].apply(pd.value_counts)

# Plotting total positions
fig = px.bar(pos,
             x=pos.index,
             y=pos.columns,
             title='Total Positions',
             labels={'variable':'Stocks',
                     'value':'Count of Positions',
                     'index':'Type of Position'})

fig.show()
```

7 Calculating and Plotting the Potential Returns

7.1 Returns on Each Individual Stock

[83]:
```
# Calculating Returns by multiplying the positions by the log returns
returns = trade_df['Positions'] * trade_df['LogReturns']

# Calculating the performance as we take the cumulative sum of the returns and␣
↪transform the values back to normal
performance = returns.cumsum().apply(np.exp)

# Plotting the performance per stock
```

```
px.line(performance,
        x=performance.index,
        y=performance.columns,
        title='Returns Per Stock Using ARIMA Forecast',
        labels={'variable':'Stocks',
                'value':'Returns'})
```

7.2 Returns on the Overall Portfolio

[86]:
```
# Returns for the portfolio
returns = (trade_df['Positions'] * trade_df['LogReturns']).sum(axis=1)

# Returns for SPY
spy = yf.download('SPY', start=returns.index[0]).loc[returns.index]

spy = spy['Adj Close'].apply(np.log).diff().dropna().cumsum().apply(np.exp)

# Calculating the performance as we take the cumulative sum of the returns and␣
↪transform the values back to normal
performance = returns.cumsum().apply(np.exp)

# Plotting the comparison between SPY returns and ARIMA returns
fig = go.Figure()

fig.add_trace(go.Scatter(x=spy.index,
                         y=spy,
                         name='SPY Returns',
                         mode='lines'))

fig.add_trace(go.Scatter(x=performance.index,
                         y=performance.values,
                         name='Portfolio Returns',
                         mode='lines'))

fig.update_layout(title='SPY vs ARIMA Overall Portfolio Returns',
                  xaxis_title='Date',
                  yaxis_title='Returns')

fig.show()
```

```
[*********************100%***********************]  1 of 1 completed
```

[ ]:
```
```

[ ]:
```
```