Exploiting Wall Street

An algorithmic approach to investing
Goldman Sachs Fired 99% of Traders
And Replaced Them With Robots
What disadvantages do you have as an investor?
Beating the market: Everybody tries to do beat it, but few succeed.

“Yes, you may be able to beat the market, but with investment fees, taxes, and human emotion working against you, you're more likely to do so through luck than skill. If you can merely match the S&P 500, minus a small fee, you'll be doing better than most investors.”

-investopedia
Algorithmic vs Buy and Hold

S&P 500

- Benchmark
- Algorithm Returns

Year


$ - $100,000 $200,000 $300,000
The Algorithm
elif (signal['action'] == "sell" and positionPrice > 0.0):
    cash += todayCandle['close']*positionSize
    positionSize = 0
    positionPrice = 0.0
    if (lastTradePrice < todayCandle['close']):
        winningTrades += 1
    else:
        losingTrades += 1
    lastTradePrice = todayCandle['close']
elif (signal['action'] == "buy" and positionPrice == 0.0):
    positionPrice = todayCandle['close']
    positionSize = int(cash/positionPrice)
    cash -= positionSize*positionPrice
    if (lastTradePrice > todayCandle['close']):
        winningTrades += 1
    else:
        losingTrades += 1
    lastTradePrice = todayCandle['close']
class Event(object):
    pass

class MarketEvent(Event):
    ''' Handles the event of receiving a new market order update with corresponding bars '''
    def __init__(self):
        self.type = 'MARKET'

class SignalEvent(Event):
    def __init__(self, symbol, datetime, signalType, signalStrength):
        self.type = 'SIGNAL'
        self.symbol = symbol
        self.datetime = datetime
        self.signalType = signalType
        self.signalStrength = signalStrength

class OrderEvent(Event):
    def __init__(self, symbol, orderType, quantity, direction, stop=None, stop_type=None):
        self.type = 'ORDER'
        self.symbol = symbol
        self.orderType = orderType
        self.quantity = quantity
        self.direction = direction
        self.stop = stop
        self.stop_type = stop_type

    def printOrder(self):
        print("Order : Symbol= %s, Type= %s, Quantity= %s, Direction= %s" % (self.symbol, self.orderType, self.quantity, self.direction))

class FillEvent(Event):
    def __init__(self, timeindex, symbol, exchange, quantity, direction, fill_cost, commission=None):
        self.type = 'FILL'
        self.timeindex = timeindex
        self.symbol = symbol
        self.exchange = exchange
        self.quantity = quantity
        self.direction = direction
        self.fill_cost = fill_cost #holding value in dollars
        self.commission = commission #should never be anything if using alpaca
The Road Ahead
Questions?
<p>| Year | Benchmark | Algorithm Returns | Algorithm Returns A Benchmark Returns | AlgoReturns | Algo vs Benchmark | Winning Trades | Losing Trades | Total Trades | Max DrawDown | Accuracy |
|------|-----------|-------------------|---------------------------------------|-------------|------------------|----------------|--------------|-------------|-------------|-----------|----------|
| 1994 | $ 24,421.50 | $ 29,691.09 | $ 28,752.88 | -2.31% | 18.76% | 21.08% | 50 | 42 | 92 | -2.55% | 54.35% |
| 1995 | $ 33,631.91 | $ 31,130.68 | $ 29,904.54 | 34.53% | 24.52% | -10.00% | 49 | 57 | 106 | -0.85% | 46.23% |
| 1996 | $ 30,054.38 | $ 33,398.58 | $ 31,718.86 | 20.22% | 33.59% | 13.38% | 51 | 51 | 102 | -3.04% | 50.00% |
| 1997 | $ 32,613.00 | $ 36,533.01 | $ 34,226.41 | 30.45% | 46.13% | 15.68% | 52 | 52 | 104 | -3.56% | 50.00% |
| 1998 | $ 31,568.00 | $ 32,654.69 | $ 31,123.75 | 26.27% | 30.62% | 4.35% | 52 | 66 | 118 | -5.02% | 44.07% |
| 1999 | $ 29,668.75 | $ 36,521.03 | $ 34,216.82 | 18.68% | 46.08% | 27.41% | 55 | 48 | 103 | -2.85% | 53.40% |
| 2000 | $ 22,039.50 | $ 33,549.30 | $ 31,839.44 | -11.84% | 34.20% | 46.04% | 54 | 50 | 104 | -3.53% | 51.92% |
| 2001 | $ 21,602.70 | $ 29,570.68 | $ 28,656.55 | -13.59% | 18.28% | 31.87% | 51 | 57 | 108 | -6.76% | 47.22% |
| 2002 | $ 19,145.91 | $ 30,834.24 | $ 29,667.39 | -23.42% | 23.34% | 48.75% | 62 | 57 | 119 | -5.04% | 52.10% |
| 2003 | $ 31,269.68 | $ 33,096.47 | $ 31,477.17 | 25.08% | 32.39% | 7.31% | 53 | 42 | 95 | -2.67% | 55.79% |
| 2004 | $ 26,954.01 | $ 30,057.79 | $ 29,046.23 | 7.82% | 20.23% | 12.42% | 61 | 49 | 110 | -1.19% | 55.45% |
| 2005 | $ 25,524.55 | $ 30,002.17 | $ 29,001.73 | 2.10% | 20.01% | 17.91% | 54 | 50 | 104 | -1.84% | 51.92% |
| 2006 | $ 20,102.38 | $ 31,603.53 | $ 30,282.82 | 12.73% | 26.41% | 13.68% | 52 | 34 | 86 | -3.01% | 60.47% |
| 2007 | $ 25,566.75 | $ 28,073.50 | $ 27,458.80 | 2.35% | 12.29% | 9.95% | 53 | 61 | 114 | -3.41% | 46.49% |
| 2008 | $ 15,340.80 | $ 25,457.67 | $ 25,366.13 | -38.64% | 1.83% | 40.47% | 52 | 63 | 115 | -4.48% | 45.22% |
| 2009 | $ 30,757.44 | $ 39,559.73 | $ 36,647.78 | 23.03% | 58.24% | 35.21% | 50 | 56 | 106 | -5.02% | 47.17% |
| 2010 | $ 27,916.50 | $ 30,502.17 | $ 29,401.73 | 11.67% | 22.01% | 10.34% | 45 | 51 | 96 | -3.27% | 46.88% |
| 2011 | $ 24,723.50 | $ 24,176.64 | $ 24,313.31 | -1.11% | -3.29% | -2.19% | 48 | 64 | 112 | -7.79% | 42.86% |
| 2012 | $ 27,769.95 | $ 29,572.61 | $ 28,658.09 | 11.08% | 18.29% | 7.21% | 48 | 59 | 107 | -1.35% | 44.86% |
| 2013 | $ 31,766.68 | $ 29,402.36 | $ 28,521.89 | 27.07% | 17.61% | -9.46% | 43 | 76 | 119 | -2.11% | 36.13% |
| 2014 | $ 27,774.90 | $ 30,402.88 | $ 29,322.31 | 10.99% | 21.61% | 10.62% | 56 | 48 | 104 | -2.30% | 53.85% |
| 2015 | $ 24,668.27 | $ 31,163.54 | $ 29,930.83 | -1.33% | 24.65% | 25.98% | 60 | 48 | 104 | -3.17% | 55.56% |
| 2016 | $ 27,717.72 | $ 27,415.24 | $ 26,932.20 | 10.87% | 9.66% | -1.21% | 44 | 62 | 106 | -3.65% | 41.51% |
| 2017 | $ 29,621.46 | $ 27,580.33 | $ 27,064.27 | 18.49% | 10.32% | -8.16% | 37 | 63 | 100 | -1.05% | 37.00% |
| 2018 | $ 23,040.75 | $ 28,019.26 | $ 27,415.41 | -7.84% | 12.08% | 19.91% | 51 | 45 | 96 | -4.27% | 53.13% |
| Total | $ 48,334.00 | $ 144,969.19 | $ 115,975.35 | 193.34% | 579.88% | 386.54% | 1283 | 1351 | 2634 | - | - |
| Average | $ 26,933.36 | $ 30,798.77 | $ 29,639.01 | 7.73% | 23.20% | 15.46% | 51.32 | 54.04 | 105.36 | -3.35% | 48.94% |
| Std Dev | $ 4,411.27 | $ 3,435.58 | $ 2,748.46 | 17.65% | 13.74% | 16.18% | 5.55 | 9.16 | 8.35 | 1.69% | 6.00% |</p>
<table>
<thead>
<tr>
<th>Year</th>
<th>Benchmark</th>
<th>Algorithm Returns</th>
<th>Returns After Tax</th>
<th>Benchmark Returns</th>
<th>AlgoReturns</th>
<th>Aalgo vs Benchmark</th>
<th>Winning Trades</th>
<th>Losing Trades</th>
<th>Total Trades</th>
<th>Max DrawDown</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1994</td>
<td>$24,422</td>
<td>$29,691</td>
<td>$28,752.88</td>
<td>-2.31%</td>
<td>18.76%</td>
<td>21.08%</td>
<td>50</td>
<td>42</td>
<td>92</td>
<td>-2.55%</td>
<td>54.35%</td>
</tr>
<tr>
<td>1995</td>
<td>$32,956</td>
<td>$37,214</td>
<td>$34,771</td>
<td>34.94%</td>
<td>25.34%</td>
<td>-9.61%</td>
<td>100</td>
<td>98</td>
<td>198</td>
<td>-2.55%</td>
<td>50.51%</td>
</tr>
<tr>
<td>1996</td>
<td>$39,580</td>
<td>$49,714</td>
<td>$44,771</td>
<td>20.10%</td>
<td>33.59%</td>
<td>13.49%</td>
<td>151</td>
<td>149</td>
<td>300</td>
<td>-3.04%</td>
<td>50.33%</td>
</tr>
<tr>
<td>1997</td>
<td>$52,026</td>
<td>$73,365</td>
<td>$63,692</td>
<td>31.44%</td>
<td>47.57%</td>
<td>16.13%</td>
<td>205</td>
<td>199</td>
<td>404</td>
<td>-3.55%</td>
<td>50.74%</td>
</tr>
<tr>
<td>1998</td>
<td>$66,096</td>
<td>$95,847</td>
<td>$81,678</td>
<td>27.04%</td>
<td>30.64%</td>
<td>3.60%</td>
<td>257</td>
<td>265</td>
<td>522</td>
<td>-5.03%</td>
<td>49.23%</td>
</tr>
<tr>
<td>1999</td>
<td>$78,725</td>
<td>$140,085</td>
<td>$117,068</td>
<td>19.11%</td>
<td>46.15%</td>
<td>27.05%</td>
<td>312</td>
<td>313</td>
<td>625</td>
<td>-5.03%</td>
<td>49.92%</td>
</tr>
<tr>
<td>2000</td>
<td>$70,317</td>
<td>$186,166</td>
<td>$153,933</td>
<td>-10.68%</td>
<td>32.89%</td>
<td>43.58%</td>
<td>367</td>
<td>363</td>
<td>730</td>
<td>-5.03%</td>
<td>50.27%</td>
</tr>
<tr>
<td>2001</td>
<td>$61,265</td>
<td>$220,287</td>
<td>$181,214</td>
<td>-12.87%</td>
<td>18.32%</td>
<td>31.19%</td>
<td>419</td>
<td>419</td>
<td>838</td>
<td>-6.76%</td>
<td>50.00%</td>
</tr>
<tr>
<td>2002</td>
<td>$47,291</td>
<td>$275,841</td>
<td>$225,673</td>
<td>-22.81%</td>
<td>25.23%</td>
<td>48.04%</td>
<td>483</td>
<td>474</td>
<td>957</td>
<td>-6.76%</td>
<td>50.47%</td>
</tr>
<tr>
<td>2003</td>
<td>$59,646</td>
<td>$378,925</td>
<td>$308,140</td>
<td>26.12%</td>
<td>37.37%</td>
<td>11.25%</td>
<td>536</td>
<td>515</td>
<td>1051</td>
<td>-6.76%</td>
<td>51.00%</td>
</tr>
<tr>
<td>2004</td>
<td>$64,786</td>
<td>$455,463</td>
<td>$369,371</td>
<td>8.62%</td>
<td>20.20%</td>
<td>11.58%</td>
<td>597</td>
<td>565</td>
<td>1162</td>
<td>-6.76%</td>
<td>51.38%</td>
</tr>
<tr>
<td>2005</td>
<td>$66,737</td>
<td>$546,910</td>
<td>$442,528</td>
<td>3.01%</td>
<td>20.08%</td>
<td>17.07%</td>
<td>652</td>
<td>614</td>
<td>1266</td>
<td>-6.76%</td>
<td>51.50%</td>
</tr>
<tr>
<td>2006</td>
<td>$75,908</td>
<td>$700,103</td>
<td>$565,082</td>
<td>13.74%</td>
<td>28.01%</td>
<td>14.27%</td>
<td>704</td>
<td>648</td>
<td>1352</td>
<td>-6.76%</td>
<td>52.07%</td>
</tr>
<tr>
<td>2007</td>
<td>$78,369</td>
<td>$786,689</td>
<td>$634,351</td>
<td>3.24%</td>
<td>12.37%</td>
<td>9.13%</td>
<td>758</td>
<td>708</td>
<td>1466</td>
<td>-6.76%</td>
<td>51.71%</td>
</tr>
<tr>
<td>2008</td>
<td>$48,369</td>
<td>$800,811</td>
<td>$645,649</td>
<td>-38.28%</td>
<td>1.80%</td>
<td>40.08%</td>
<td>811</td>
<td>770</td>
<td>1581</td>
<td>-6.76%</td>
<td>51.30%</td>
</tr>
<tr>
<td>2009</td>
<td>$59,732</td>
<td>$1,324,473</td>
<td>$1,064,578</td>
<td>23.49%</td>
<td>65.39%</td>
<td>41.90%</td>
<td>862</td>
<td>824</td>
<td>1686</td>
<td>-6.76%</td>
<td>51.13%</td>
</tr>
<tr>
<td>2010</td>
<td>$67,402</td>
<td>$1,630,391</td>
<td>$1,309,313</td>
<td>12.84%</td>
<td>23.10%</td>
<td>10.26%</td>
<td>908</td>
<td>874</td>
<td>1782</td>
<td>-6.76%</td>
<td>50.95%</td>
</tr>
<tr>
<td>2011</td>
<td>$67,268</td>
<td>$1,576,460</td>
<td>$1,266,168</td>
<td>-0.20%</td>
<td>-3.31%</td>
<td>-3.11%</td>
<td>956</td>
<td>938</td>
<td>1894</td>
<td>-7.80%</td>
<td>50.48%</td>
</tr>
<tr>
<td>2012</td>
<td>$76,332</td>
<td>$1,865,715</td>
<td>$1,497,572</td>
<td>13.47%</td>
<td>18.35%</td>
<td>4.87%</td>
<td>1004</td>
<td>997</td>
<td>2001</td>
<td>-7.80%</td>
<td>50.17%</td>
</tr>
<tr>
<td>2013</td>
<td>$98,994</td>
<td>$2,261,516</td>
<td>$1,814,213</td>
<td>29.69%</td>
<td>21.21%</td>
<td>-8.47%</td>
<td>1048</td>
<td>1071</td>
<td>2119</td>
<td>-7.80%</td>
<td>49.46%</td>
</tr>
<tr>
<td>2014</td>
<td>$110,169</td>
<td>$2,742,438</td>
<td>$2,198,950</td>
<td>11.29%</td>
<td>21.27%</td>
<td>9.98%</td>
<td>1105</td>
<td>1119</td>
<td>2224</td>
<td>-7.80%</td>
<td>49.69%</td>
</tr>
<tr>
<td>2015</td>
<td>$109,274</td>
<td>$3,343,912</td>
<td>$2,680,129</td>
<td>-0.81%</td>
<td>21.93%</td>
<td>22.74%</td>
<td>1167</td>
<td>1167</td>
<td>2334</td>
<td>-7.80%</td>
<td>50.00%</td>
</tr>
<tr>
<td>2016</td>
<td>$119,812</td>
<td>$3,669,460</td>
<td>$2,940,568</td>
<td>9.64%</td>
<td>9.74%</td>
<td>0.09%</td>
<td>1212</td>
<td>1228</td>
<td>2440</td>
<td>-7.80%</td>
<td>49.67%</td>
</tr>
<tr>
<td>2017</td>
<td>$143,037</td>
<td>$4,077,434</td>
<td>$3,266,947</td>
<td>19.38%</td>
<td>11.12%</td>
<td>-8.27%</td>
<td>1250</td>
<td>1290</td>
<td>2540</td>
<td>-7.80%</td>
<td>49.21%</td>
</tr>
<tr>
<td>2018</td>
<td>$132,794</td>
<td>$4,584,300</td>
<td>$3,672,440</td>
<td>-7.16%</td>
<td>12.43%</td>
<td>19.59%</td>
<td>1301</td>
<td>1335</td>
<td>2636</td>
<td>-7.80%</td>
<td>49.36%</td>
</tr>
</tbody>
</table>

**TOTAL**

- Benchmark: 212.06%
- Algorithm Returns: 599.55%

**AVERAGE**

- Benchmark: 16.31%
- Algorithm Returns: 46.12%

**STD DEV**

- Benchmark: 17.77%
- Algorithm Returns: 14.71%
THATS THE REINDEXED DATA

<table>
<thead>
<tr>
<th>Date</th>
<th>open</th>
<th>high</th>
<th>low</th>
<th>close</th>
<th>volume</th>
<th>unadjustedClose</th>
<th>unadjustedVolume</th>
<th>changed</th>
<th>changePercent</th>
<th>vwap</th>
<th>label</th>
<th>changeOverTime</th>
<th>Name: 2014-04-07, dtype: object</th>
</tr>
</thead>
<tbody>
<tr>
<td>'2014-04-07'</td>
<td>168.368</td>
<td>168.694</td>
<td>166.611</td>
<td>166.956</td>
<td>1.40803e+08</td>
<td>None</td>
<td>1.40803e+08</td>
<td>-1.8657</td>
<td>-1.105</td>
<td>167.536</td>
<td>Apr 7, 14</td>
<td>0</td>
<td>3775</td>
</tr>
</tbody>
</table>
Okay so this is where it gets tricky. The method `update_timeindex` handles the new holdings tracking. First it obtains the latest prices from the market data handler and creates a new dictionary of symbols to represent the `current_positions`. These are changed when a FillEvent is given. Which is handled in a later function. The method appends this set of `current_positions` to the `all_positions` list. Next the holdings are updated in a similar manner, with the exemption that the market value is recalculated by multiplying the `current_positions` count with the closing price of the last bar.

```py
...  
```

```py
i.e. self.current_positions[s] * bars[s][0][5]  
```

then the new holdings are appended to `all_holders`

```py
@def update_timeindex(self, event):
...
```

```py
# Adds a new record to the positions matrix/table for the current market
# data bar. This reflects the previous bar.
# this means all current market data is "known"
# This is the function that actually uses the MarketEvent from the queue
print('inside Portfolio.update_timeindex')
bars = {}  
for sym in self.symbollist:
    bars[sym] = self.bars.get_latest_bars(symb, N=1)

# Update positions
index = 0
dp = dict((x,y) for x, y in [(s, 0) for s in self.symbollist])
dp['date'] = bars[0][0][1]
for x in self.symbollist:
    dp[x] = self.current_positions[x]
# Append current positions
self.all_positions.append(dp)

# Update holdings:
```

```py
dh = dict((x,y) for x, y in [(s, 0) for s in self.symbollist])
dh['date'] = bars[0][0][1]
dh['cash'] = float(self.current_holding['cash'])
dh['total'] = float(self.current_holdings['cash'])
```

```py
for s in self.symbollist:
    # the real value of portfolio.
    market_value = float(float(self.current_positions[s]) * float(bars[s][0][5]))
    print(market_value)
    dh[5] = market_value
dh['total'] = market_value
```

```py
self.all_holders.append(dh)
```
from DataHandler import HistoricSQLDataHandler
from ExecutionDriver import SimulatedExecutionHandler
import PortfolioDriver
import Strategy
import sys
is_py2 = sys.version[0] == '2'
if is_py2:
    import Queue as queue
else:
    from multiprocessing import Queue
import time
import queue

events = queue.Queue()
symbolList = []
symbolList(0) = 'SPY Data'
bars = HistoricSQLDataHandler(events, 'SELECT * FROM', symbolList)
strategy = Strategy.StirlingBeatsBuffer(bars, events)  # Trend Strat
strategy = Strategy.SimulateStrategy(bars, events)  # Benchmark
port = PortfolioDriver.BasicPortfolio(bars, events, '2014-04-07')
broker = SimulatedExecutionHandler(events)

while True:
    if bars.continueBacktest == True:
        bars.updateBars()
    else:
        break
    while True:
        try:
            event = events.get(False)
        except queue.Empty:
            print('The Queue Is Currently Empty')
            break
        else:
            if event is not None:
                print(event)
                If event.type == 'MARKET':
                    print('MARKET EVENT RECEIVED')
                    print(event)
                    strategy.calculate_signals(event)
                    port.update_timeIndex(event)
                elif event.type == 'SIGNAL':
                    print('SIGNAL EVENT RECEIVED')
                    input('Press Enter To Continue')
                    port.updateSignal(event)
                elif event.type == 'ORDER':
                    print('ORDER EVENT RECEIVED')
                    input('Press Enter To Continue')
                    broker.execute_order(event)
                elif event.type == 'FILL':
                    print('FILL EVENT RECEIVED')
                    input('Press Enter To Continue')
                    port.updateFill(event)
                print('Analysis Complete')