Necessary ingredients for a successful trading strategy

Julian Manzano

SC AG
Switzerland
julian.manzano@gmail.com
Necessary ingredients for a successful trading strategy

Disclaimer:

- The information provided in this presentation has been given in good faith and every effort has been made to ensure its accuracy. However, the author disclaims any responsibility for the accuracy and completeness of the information or the conclusions that can be drawn from it. Nothing in this presentation constitutes or should be construed to constitute: (i) an offer, advice, invitation or solicitation from the author, his employer or its affiliates to buy or sell any investments or securities, futures, options or other financial instruments; (ii) an invitation or inducement to engage in investment activity or a financial promotion of any kind; or (iii) investment advice or recommendation.

- The views and opinions expressed in this presentation are solely those of the author and do not necessarily represent those of his employer, its management and/or any of its staff.
Presentation outline

When we research and develop trading strategies we are faced with the following questions:

- How do we test strategies?
- How badly are we over-fitting?
- How can we be more robust?
- Is this enough?
Over-fitting and Market risk

Trade Frequency

Scalable.
Easier to simulate

Non-Scalable.
Difficult to simulate

Hedge Funds
Day Traders
Market Makers
Strategy life cycle:

1) think, learn & write code
2) use data & simulate
3) select parameters & products
4) iterate 1) to 3) until targets achieved
5) go into production
6) compare simulation with production
7) understand & fix production and simulation issues
8) go to 1)

Profits! (actually, not that easy)
Let's explore some model selection toy examples. We will asses the over-fitting severity and some techniques to minimize it.
time_steps = 200
all_cases = 100000
step_cost = 0.5/math.sqrt(time_steps)

selection_time = time_steps/2-1
selection_cut = 0

portfolio_PLs = []; selectedPLs = []; selected_cases = 0; selected_cumPL = 0.

for k in range(all_cases):
    drift = - step_cost
    randx = 2*np.random.rand(time_steps)-1
    step_PL = randx + drift
    cum_PL = np.cumsum(step_PL)

    cumPL_s = cum_PL[selection_time]
    stdev_s = math.sqrt(selection_time*np.var(step_PL[:selection_time+1]))

    if cumPL_s > stdev_s*selection_cut: # selection here
        selected_cases += 1
        selected_cumPL += cumPL_s
        portfolio_PLs.append(cum_PL)

weightsEq = np.empty((selected_cases,1)); weightsEq.fill(1./selected_cumPL)
portfolio_PL_Eq = np.array(portfolio_PLs).T.dot(weightsEq)      # normalization
time_steps = 200
all_cases = 100000
step_cost = 0.5/math.sqrt(time_steps)

selection_time = time_steps/2-1
step_bias = 2*step_cost
goodCandidates = 0.05
selection_cut = 0

portfolio_PLs = []; selectedPLs = []; selected_cases = 0, selected_cumPL = 0.

for k in range(all_cases):
    if np.random.rand() < goodCandidates:  # drift = +step_bias - step_cost
        drift = +step_bias - step_cost
    else:  # drift = -step_bias - step_cost
        drift = -step_bias - step_cost
    randx = 2*np.random.rand(time_steps)-1
    step_PL = randx + drift
    cum_PL = np.cumsum(step_PL)

    cumPL_s = cum_PL[selection_time]
    stdev_s = math.sqrt(selection_time*np.var(step_PL[:selection_time+1]))

    if cumPL_s > stdev_s*selection_cut:  # selection here
        selected_cases += 1
        selected_cumPL += cumPL_s
        portfolio_PLs.append(cum_PL)

weightsEq = np.empty((selected_cases,1)); weightsEq.fill(1./selected_cumPL)
portfolio_PL_Eq = np.array(portfolio_PLs).T.dot(weightsEq)  # normalization
Can we do better out-of-sample? Let's try a regularized Markowitz portfolio.

\[
\min_w w^T \Sigma w + \alpha w^T \text{diag}(\Sigma) w + \beta p_{\text{target}} w^T \sqrt{\text{diag}(\Sigma)}
\]

s.t. \quad w \geq 0

\[
p^T w = p_{\text{target}}
\]

where:

\[\Sigma = \text{covariance matrix of step PLs}\]

\[p = \text{average of step PLs}\]

time_steps = 200
all_cases = 100000
step_cost = 0.5/math.sqrt(time_steps)

selection_time = time_steps/2-1
step_bias = 2*step_cost
goodCandidates = 0.05
selection_cut = 0

portfolio_PLs = []; selectedPLs = []; selected_cases = 0; selected_cumPL = 0.

for k in range(all_cases):
    if np.random.rand() < goodCandidates:  drift = +step_bias - step_cost
    else:                                       drift = -step_bias - step_cost
    randx = 2*np.random.rand(time_steps)-1
    step_PL = randx + drift
    cum_PL = np.cumsum(step_PL)

cumPL_s = cum_PL[selection_time]
stdev_s = math.sqrt(selection_time*np.var(step_PL[:selection_time+1]))

if cumPL_s > stdev_s*selection_cut:  # selection here
    selected_cases += 1
    selected_cumPL += cumPL_s
    portfolio_PLs.append(cum_PL)
    selectedPLs.append(step_PL[:selection_time+1])

C = np.cov(selectedPLs)
avePL = np.average(selectedPLs,axis=1)
Ridge_Regularization = 1; L1_Regularization = 1

weightsMR = portfolioSolver(C,avePL,targetAvePL,[],[],Ridge_Regularization,0.)
weightsML = portfolioSolver(C,avePL,targetAvePL,[],[],0.,L1_Regularization)
weightsMw = portfolioSolver(C,avePL,targetAvePL,[],[],0.,0)
weightsEq = np.empty((selected_cases,1)); weightsEq.fill(1./selected_cumPL)
Let's add correlation to our portfolio

time_steps = 200
all_cases   = 100000
step_cost   = 0.5/math.sqrt(time_steps)

selection_time   = time_steps/2-1
step_bias            = 2*step_cost
goodCandidates = 0.05
selection_cut      = 0

corr = 0.3
negCorr = 0.5

ci  = math.sqrt(corr)
co = math.sqrt(1.-corr)
common_factor = 2*np.random.rand(time_steps)-1

for k in range(all_cases):
    if np.random.rand() < goodCandidates:
        drift = +step_bias - step_cost
    else:
        drift = -step_bias - step_cost

    sign       = 1 if np.random.rand() > negCorr else -1
    randx    = 2*np.random.rand(time_steps)-1
    step_PL = co*randx + (sign*ci)*common_factor  + drift
What if we pre-select more aggressively?

time_steps = 200
all_cases = 500000
step_cost = 0.5/math.sqrt(time_steps)

selection_time = time_steps/2-1
step_bias = 2*step_cost
goodCandidates = 0.05
selection_cut = 2

corr = 0.3
negCorr = 0.5

ci = math.sqrt(corr)
co = math.sqrt(1.-corr)
common_factor = 2*np.random.rand(time_steps)-1

for k in range(all_cases):
    if np.random.rand() < goodCandidates:
        drift = +step_bias - step_cost
    else:
        drift = -step_bias - step_cost

    sign = 1 if np.random.rand() > negCorr else -1
    randx = 2*np.random.rand(time_steps)-1
    step_PL = co*randx + (sign*ci)*common_factor + drift
10 million cases. Selection cut = 3 sigmas
A real world example:

Portfolio with 76 strategies selected over \(~22300\) candidates. Weights given by a Ridge Regularized Markowitz portfolio with $\alpha = 2$. 
Out-Ave-PL/In-Ave-PL = 42%
Out-Shape/In-Sharpe = 62%
In-PL-Stdev/Out-PL-Stdev = 148%
In-MaxDrawDown/Out-MaxDrawDown = 93%
How do we test strategies?

- We simulate with different assets and parameters and we apply selection rules in asset/parameter space using a data subset.

- The strategy is given by the trading heuristic together with the sampling and selection rules.
Is this enough?

No, we have to consider aspects outside the “math”:

- Production results can differ from simulated ones.
- Human input still necessary (Syrian conflict, Italian crisis, US debt ceiling, etc.)
- Technology glitches, data problems, simulation assumptions.
- Market changes, new regulations, new exchange rules.
Conclusions:

- We used Monte Carlo simulations to analyze the out-of-sample performance of different selection rules applied to portfolios of strategies.
- We showed that preselection based on individual performance is important (no good pie out of bad apples).
- We showed evidence that Ridge regularization tends to outperform L1 regularization when correlation is significant.

Thank you!