DESIGN DOCUMENT

Objective:

To get the trading position for the market.

Description:

The trading algorithm is based in the following formulation: -

$$\min\left\{\frac{1}{2}\left(\|\omega\|^2 + A\theta^2\right) + \alpha\sum_{t=1}^{T} a_t - \beta\sum_{t=1}^{T} b_t + \gamma\sum_{t=1}^{T}(c_t + d_t)\right\} \quad (1)$$

Subject to:-

$$r_t\left[\omega^T\emptyset(x^{t-1}) + \theta\right)] + (a_t - b_t) = 0 \qquad \text{such that } a_t, b_t \geq 0$$

$$\delta\omega^T\left[\emptyset(x^t) - \emptyset(x^{t-1})\right] + (c_t - d_t) = 0 \qquad \text{such that } c_t, d_t \geq 0$$

The formulation named as "OMEGA RATIO FORMULATION" aims at minimizing the losses (contained in a_t), transaction costs (contained in c_t and d_t) and maximizing the profits (contained in b_t).

The Ø's represents the samples. Each sample is BarsBack dimensional i.e. each sample has past BarsBack features which can simply be the closing prices or some complex Indicator or a combination of both.

The T is the TradingWindowSize (T>=1).

α, β, γ are the weighing factors for the cumulative losses, profits and transaction costs respectively. The condition α>β ensures that minimizing losses is more important than maximizing profits. So we carefully implement this condition while selecting the parameters.

r_t is the return at time t given by r_t=price$_t$-price$_{t-1}$

δ is the transaction cost incurred for the scrip being traded.

The above formulation is our primal formulation. After we apply KKT conditions and convert the above into dual formulation, it becomes:-

$$Min\frac{1}{2}\sum_{t=1}^{T}\sum_{u=1}^{T}\Big\{\lambda_t r_t \lambda_u r_u \phi(x^{t-1})^T\phi(x^{u-1}) + \lambda_t r_t g_u \delta[\phi(x^{t-1})^T\phi(x^{u}) - \phi(x^{t-1})^T\phi(x^{u-1})] + g_t g_u \delta^2[\phi(x^{t})^T\phi(x^{u}) - \phi(x^{t})^T\phi(x^{u-1}) - \phi(x^{u})^T\phi(x^{t-1}) + \phi(x^{t-1})^T\phi(x^{u-1})] + \frac{1}{A}\lambda_t r_t \lambda_u r_u + \lambda_u r_u g_t \delta[\phi(x^{u-1})^T\phi(x^{t}) - \phi(x^{u-1})^T\phi(x^{t-1})]\Big\} \qquad (2)$$

Constrained to the following conditions: -

$\beta \leq \lambda_t \leq \alpha$

$-\gamma \leq g_t \leq \gamma$

Here, $\Phi(x^t)^T\Phi(x^u) = K(x^t, x^u)\ or\ Kernel(x^t, x^u)$. The kernel function can be one of the many commonly used kernels such as rbf kernel, polynomial kernel etc.

Minimizing equation (2) gives set of λ's and g's which gives us ω and θ.

$\omega=\sum_{t=1}^{T}\lambda_t r_t \Phi(x^{t-1}) + \sum_{t=1}^{T} g_t \delta(\Phi(x^t) - \Phi(x^{t-1}))$

$\theta=\frac{1}{A}\sum_{t=1}^{T}\lambda_t r_t$

Ideally, the current position in the market is given by sgn($\omega^T\emptyset(x^t) + \theta$) .But to inculcate the confidence of trader into our strategy and to rule out the possibility of bad trades, we compute the current position as: -

If TempPosition[t]> threshold and TempPosition[t-1]>threshold and HighPrice[t]>HighPrice[t-1] and shortmovingaverage[t]>longmovingaverage[t], Then Current Position=1. {i.e If we are confident enough, enter the trade}

ElseIf TempPosition[t]> threshold and Position[t-1]==1 and shortmovingaverage[t]>longmovingaverage[t], Then Current Position=1 {Continue your position as long as trend is positive and Trader is confident}

ElseIf TempPosition[t]< -threshold and Temposition[t-1]<-threshold and HighPrice[t]<HighPrice[t-1] and shortmovingaverage[t]<longmovingaverage[t], Then Current Position=-1 . {i.e If we are confident enough, enter the trade}

ElseIf TempPosition[t]< -threshold and position[t-1]==-1 and shortmovingaverage[t]<longmovingaverage[t], Then Current Position=-1 {Continue your position as long as trend is negative and Trader is confident}

Else, Current Position=0

Where , $(\omega^T \emptyset(x^t) + \theta/|| \omega ||)$ can be termed as TempPosition. Division by $|| \omega ||$ helps to normalize the output of trader.

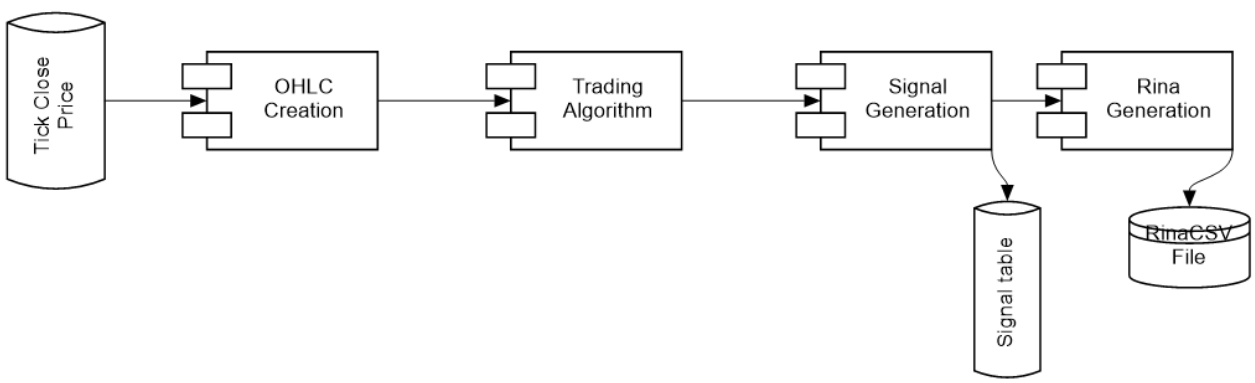
Thus alpha, beta, gamma, dell, A, threshold, kernelfunc, TradingWindowSize, BarsBack, BarTimeInterval are the parameters of algorithm.

So the algorithm requires past T closing prices to compute the current position. The output of the of the trading algorithm is ternary with 1, -1, 0 as the 3 possible states representing the long, short, and neutral position in the market respectively.

The preprocessing and post processing can be done can be done in an outside module. Preprocessing involves reading the data prices for each tick and preparing the OHLC bars for the time interval specified by BarTimeInterval. Sequence of states can be converted into Trading Signals post processing. We stand Long as long as the output is +1, Short as long as the output is -1 and Neutral whenever the output is 0. Static Trailing and Stop Loss can be applied after we take positions in the market.

After we get the position for current time instant or current bar, we repeat the procedure for next bar by shifting the entire T sized Trading Window to the right exactly by 1 bar.

<u>Component Relationship Diagram:</u>



<u>Design Considerations:</u>

<u>Assumptions</u>:

- The input stream is available for every tick in a continuous manner.
- The signal file is generated for every entry and exit of the trade.
- The RinaCSV file is generated at the end of the trading session.
- The ternary output of Trader is stored for reference purpose.

Constraints:

- The input to algorithm is rounded off to 2 places of decimal.
- The rounding off can be done using methods OHLCCreation before calling the algorithm.

Optimizer Used:

ISMO algorithm is being used to optimize the dual objective.

The update rule for λ_k and g_k is given by:-

$$\lambda_k^{new} = \lambda_k^{old} - \left(\frac{f_{old}(x_{k-1})}{r_k\left(K_{k-1,k-1} + \frac{1}{A}\right)} \right)$$

$$g_k^{new} = g_k^{old} - \left(\frac{f_{old}(x_k) - f_{old}(x_{k-1})}{\delta_k\left(K_{k,k} + K_{k-1,k-1} - 2K_{k,k-1}\right)} \right)$$

Where,

$$f_{old(x_k)} = \sum_{u=1}^{T} \lambda_u r_u \left(K_{k,u-1} + \frac{1}{A}\right) + \sum_{u=1}^{T} g_u \delta\left(K_{k,u} - K_{k,u-1}\right)$$

And, $K_{k,k} = Kernel(x_k, x_k)$

DataStructure Used:

A class with the following UML diagram:-

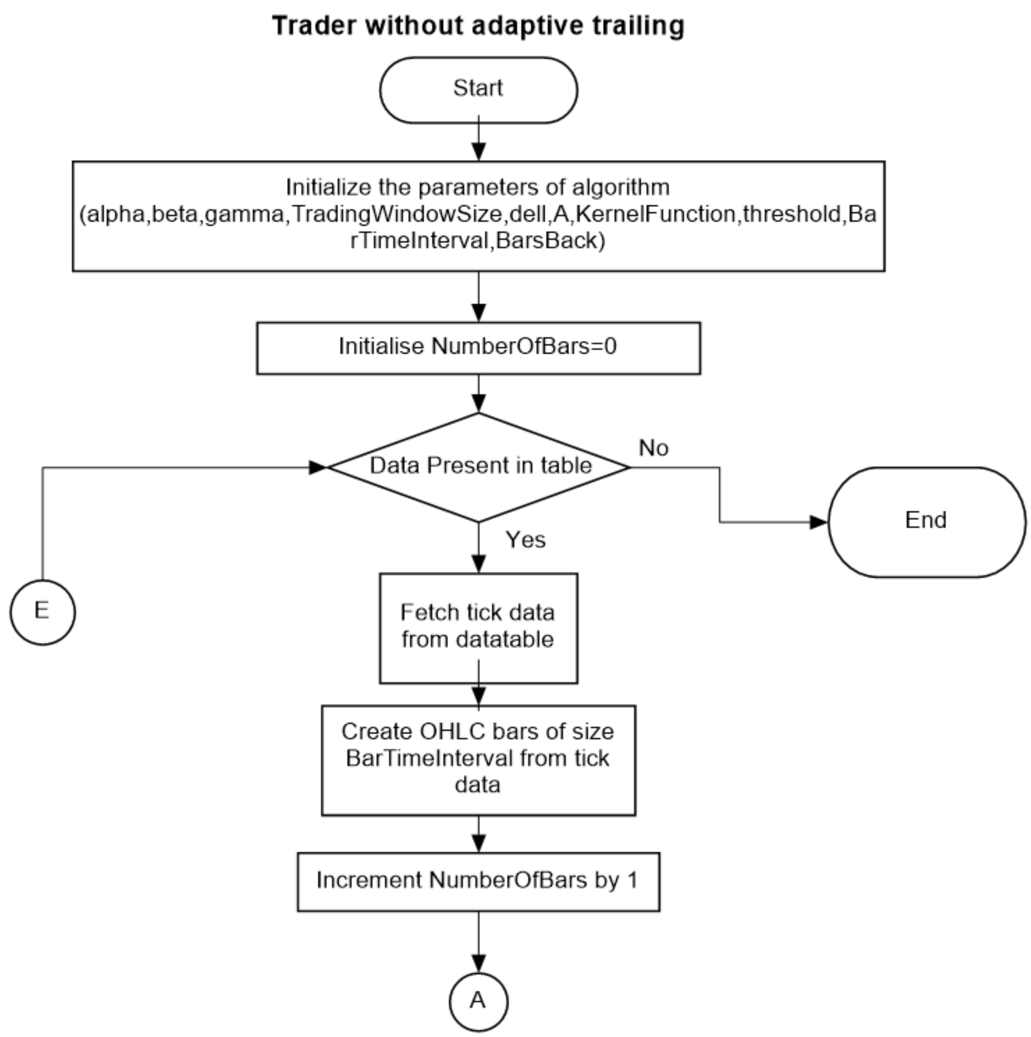
Trader
Private: m_str_dbName m_str_DataTableName m_str_SignalTableName m_str_RinaFileName m_str_UserName m_str_Password m_f_alpha m_f_beta m_f_gamma m_f_dell m_i_TradingWindowSize m_f_threshold m_i_BarsBack m_f_A m_i_BarTimeInterval m_i_TotalTicks m_i_t m_str_SessionCloseTime m_f_OHLCMatrix m_i_Position m_f_TempPosition m_f_FeatureMatrix m_f_Return m_KernelFunc() m_login()
Public: m_TblCreation() m_OHLCCreation() m_TradingAlgorithm() m_RinaGeneration()

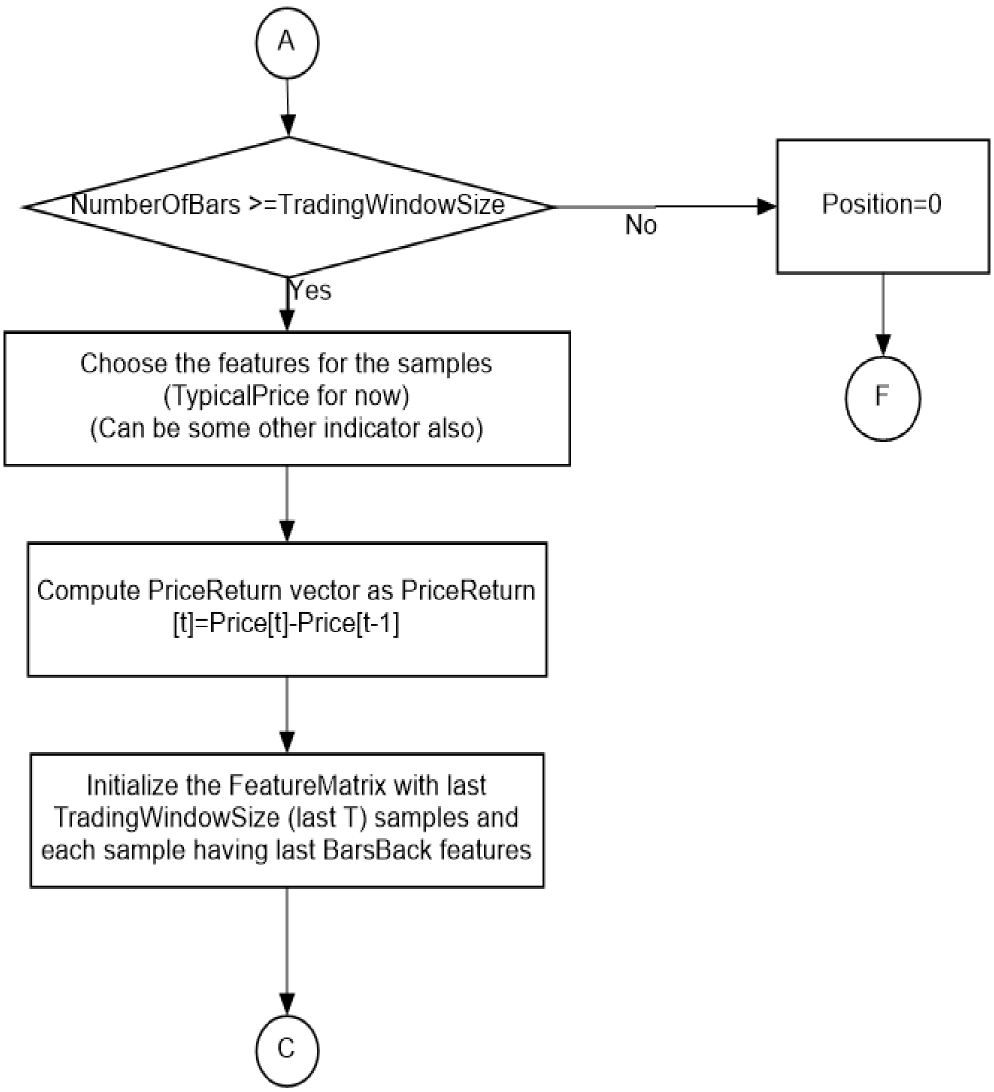
Milestones

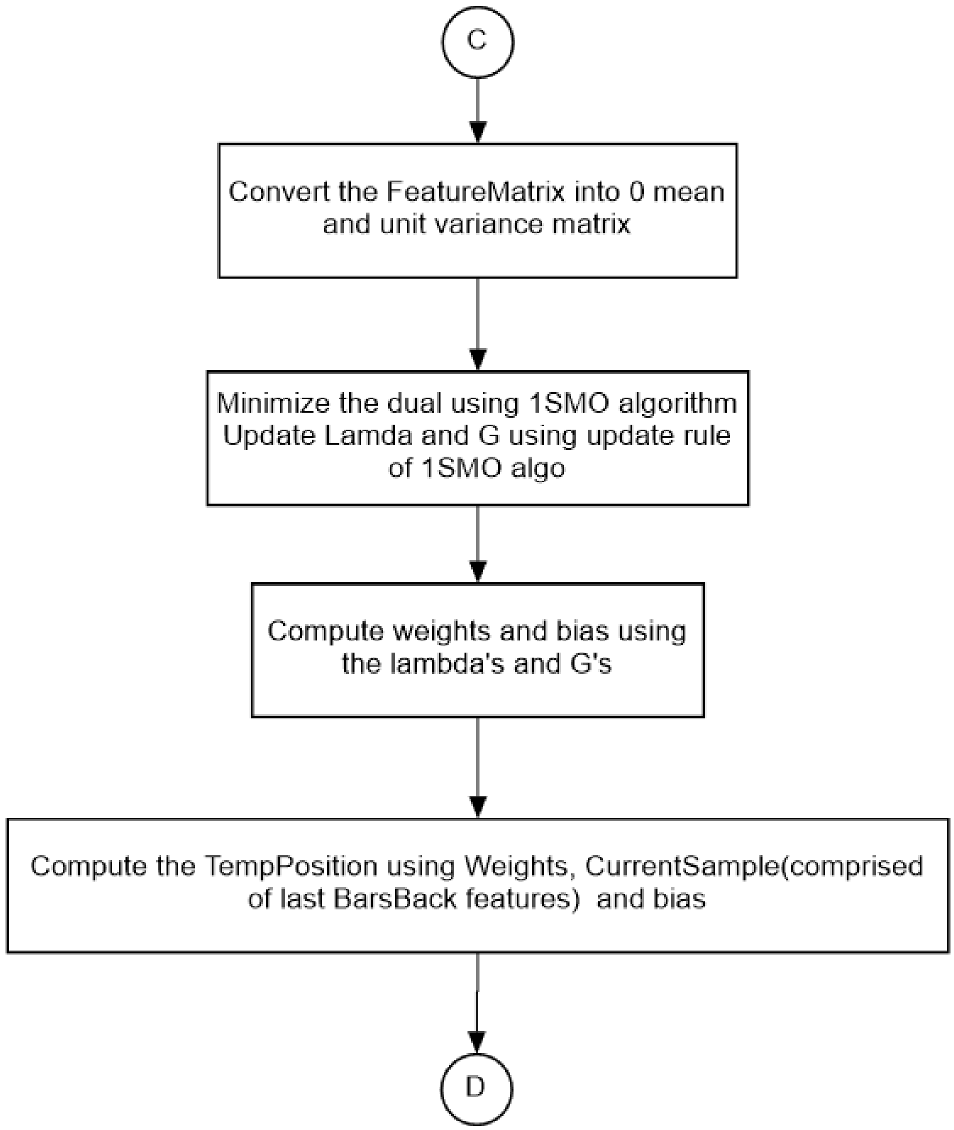
- Estimated date for completion: June 19, 2015

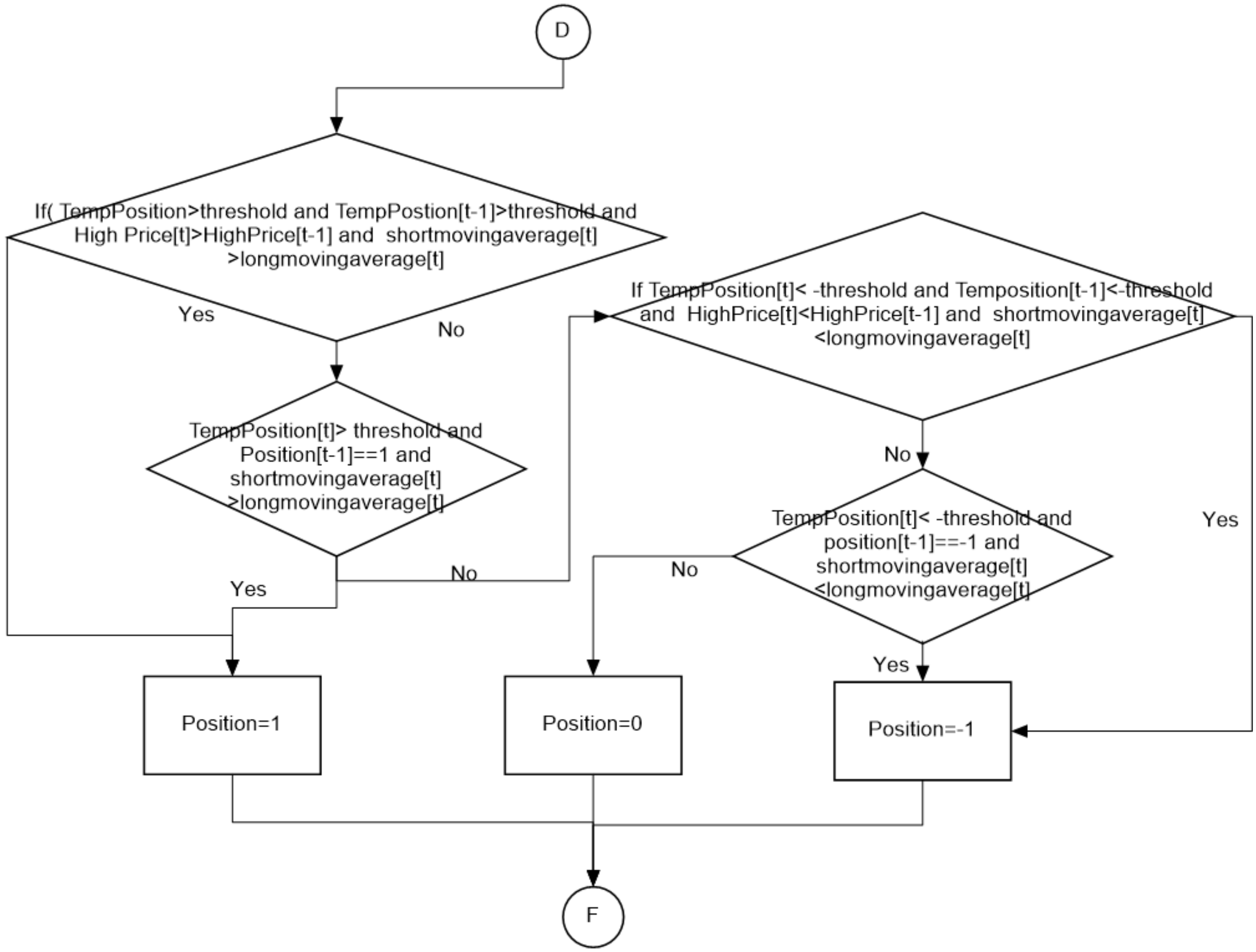
- Code and Use Cases --- Estimated date for completion: June 25, 2015

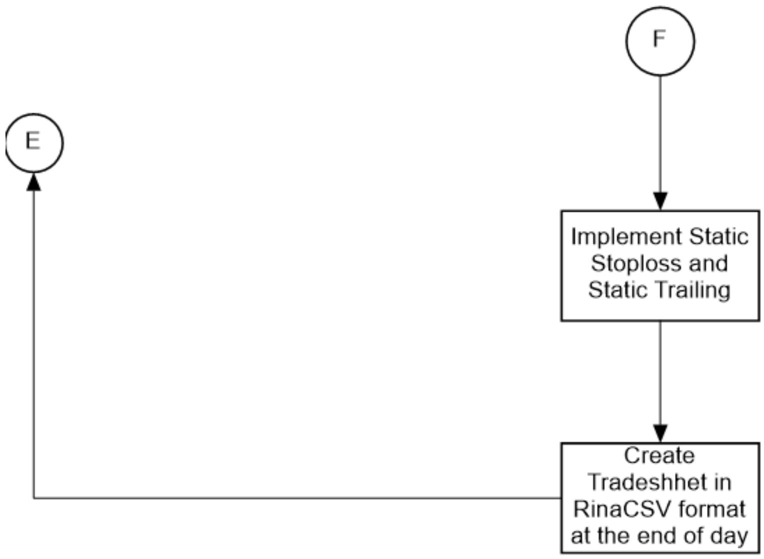
Flowchart for TradingAlgorithm:




A
NumberOfBars >=TradingWindowSize
No
Position=0
F
Yes
Choose the features for the samples
(TypicalPrice for now)
(Can be some other indicator also)
Compute PriceReturn vector as PriceReturn
[t]=Price[t]-Price[t-1]
Initialize the FeatureMatrix with last
TradingWindowSize (last T) samples and
each sample having last BarsBack features
C


C
Convert the FeatureMatrix into 0 mean and unit variance matrix
Minimize the dual using 1SMO algorithm Update Lamda and G using update rule of 1SMO algo
Compute weights and bias using the lambda's and G's
Compute the TempPosition using Weights, CurrentSample(comprised of last BarsBack features) and bias
D


D
If(TempPosition>threshold and TempPostion[t-1]>threshold and High Price[t]>HighPrice[t-1] and shortmovingaverage[t] >longmovingaverage[t]
Yes
No
TempPosition[t]> threshold and Position[t-1]==1 and shortmovingaverage[t] >longmovingaverage[t]
Yes
No
If TempPosition[t]< -threshold and Temposition[t-1]<-threshold and HighPrice[t]<HighPrice[t-1] and shortmovingaverage[t] <longmovingaverage[t]
No
Yes
TempPosition[t]< -threshold and position[t-1]==-1 and shortmovingaverage[t] <longmovingaverage[t]
No
Yes
Position=1
Position=0
Position=-1
F



Pseudocode for the Trading Algorithm:-

Aim: To compute the position for time instant ‘t’ i.e. Position[t]

Input: Open, High, Low, Close (All Prices are tick data)
Output: Position for the market.

1. Initialize m_f_A
2. Initialize m_f_Alpha
3. Initialize m_f_Beta
4. Initialize m_f_Gamma
5. Initialize m_f_Dell
6. Initialize m_i_TradingWindowSize
7. Initialize m_i_BarsBack
8. Initialize m_i_BarTimeInterval
9. Initialize m_i_TotalTicks=0
10. Initialize m_f_Threshold
11. Initialize l_i_TickNumber=0

12. Initialize l_str_TickDate=NULL
13. Initialize l_str_TickTime=NULL
14. Initialize l_f_TickOpen=0.0
15. Initialize l_f_TickHigh=0.0
16. Initialize l_f_TickLow=0.0
17. Initialize l_f_TickClose=0.0
18. Initialize m_i_t=0
19. For each tick do:
    19.1. Read l_srt_TickDate, l_str_TickTime, l_f_TickOpen, l_f_TickHigh, l_f_TickLow, l_f_TickClose from m_str_DataTableName.
    19.2. If No Data Present in table then exit
    19.3. Increment l_i_TickNumber by 1
    19.4. Increment m_i_TotalTicks by 1
    19.5. If l_i_TickNumber==m_i_BarTimeInterval or l_str_TickTime==m_str_SessionCloseTime
        19.5.1. Create OHLC bar of size m_i_BarTimeInterval
        19.5.2. Append l_f_BarOpen[t], l_f_BarHigh[t],l_f_ BarLow[t],l_f_ BarClose[t] to m_f_OHLCMatrix.
        19.5.3. Increment m_i_t by 1.
        19.5.4. Set l_i_TickNumber=0
        19.5.5. If m_i_t==1
            19.5.5.1. Set m_f_ReturnVector[t]=0
        19.5.6. Else If m_i_t>1
            19.5.6.1. Set m_f_ReturnVector[t]=BarClose[t]-BarClose[t-1]
        19.5.7. l_f_FeatureVector =MovingAverage(TypicalPrice)
        19.5.8. l_f_ShortMA=MovingAverage(BarClose)
        19.5.9. l_f_LongMA=MovingAverage(BarClose)
        19.5.10. If m_i_t<m_i_BarsBack
            19.5.10.1. m_f_FeatureMatrix[1:m_i_BarsBack][t]=Zeros(BarsBack,1)
        19.5.11. Else if t>=BarsBack
            19.5.11.1. m_f_FeatureMatrix[1:BarsBack][t]=l_f_FeatureVector[t-BarsBack:t]
        19.5.12. If m_i_t<m_i_TradingWindowSize
            19.5.12.1. M_i_Position[t]=0
            19.5.12.2. Go to step 19.5.14

19.5.12.3. End If
19.5.13. Else IF m_i_t>=m_i_TradingWindowSize
19.5.13.1. l_f_Phi=m_f_FeatureMatrix[t-TradingWindowSize:t][:]
19.5.13.2. Convert l_f_Phi to 0 mean and unit variance matrix
19.5.13.3. Initialize l_f_Lambda=zeros(1,TradingWindowSize)
19.5.13.4. Initialize l_f_G=zeros(1,TradingWindowsize)
19.5.13.5. Initialize l_i_iterate=0
19.5.13.6. While (l_i_iterate<l_i_MaxIterations) OR ($|\lambda^{new} - \lambda^{old}|$<epsilon and $|g^{new} - g^{old}|$<epsilon)
19.5.13.6.1. For k= 1 to m_i_TradingWindowSize
19.5.13.6.1.1. Compute λ_k^{new} using 1SMO update rule.
19.5.13.6.2. For k= 1 to m_i_TradingWindowSize
19.5.13.6.2.1. Compute g_k^{new} using 1SMO update rule.
19.5.13.6.3. l_i_iterate=l_i_iterate+1
19.5.13.7. Compute l_f_W.
19.5.13.8. Compute l_f_theta
19.5.13.9. Compute m_f_TempPosition[t]=(W^T(l_f_Phi[t][:])+Theta)/(|W|)
19.5.13.10. Compute m_i_Position[t]
19.5.13.10.1. if (m_f_TempPosition[t]>m_f_Threshold) and (m_f_TempPosition[t-1]>m_f_Threshold) and (l_f_BarHigh[t]>l_f_BarHigh[t-1]) and (l_f_ShortMA[t]>l_f_LongMA[t])
19.5.13.10.1.1. m_i_Position[t]=1
19.5.13.10.2. Else If (m_f_TempPosition[t]>m_f_Threshold) and m_i_Position[t-1]==1 and (l_f_ShortMA[t]>l_f_LongMA[t]):
19.5.13.10.2.1. m_i_Position[t]=1
19.5.13.10.3. Else IF (m_f_TempPosition[t]<-m_f_Threshold) and (l_i_TempPosition[t-1]<-m_f_Threshold) and (l_f_BarHigh[t]<HighPrice[t-1]) and (l_f_ShortMA[t]<l_f_LongMA[t]):
19.5.13.10.3.1. m_i_Position[t]=-1
19.5.13.10.4. Else If (m_f_TempPosition[t]<-m_f_Threshold) and m_i_Position[t-1]==-1 and (l_f_ShortMA[t]<l_f_LongMA[t]):
19.5.13.10.4.1. m_i_Position[t]=-1
19.5.13.10.5. Else:
19.5.13.10.5.1. FindPos.Pos[t]=0

19.5.14. If Trade Entry or exit write into data table
19.5.15. Trail your trade
19.5.16. Check for stop loss
19.5.17. If end of session write RinaCSV
19.5.18. Pause if session ends.
19.5.19. Go to step 19