design

P3+J3^u! Holdings

Table of Contents

1. **purpose:**
   1. The project will collate data about equities from specified sources and data about a specific trading account.
   2. It will analyze that data to determine profitable behavior for the account based on its current positions and the market data.
   3. It will execute technically indicated market orders via an automated broker API.
2. **Rough Outline:**

* 1. **data retrieval:**
     1. Quote data structure in memory {done 6/6}
     2. Quote data attributes (xml) {done 6/6}
     3. Yahoo API stock retrieval class {done 6/6}
     4. Local data storage writer
     5. Local data storage reader
     6. Stock History (quote series) class
     7. Interface for getting quotes over time/maintaining them in memory
     8. “Driver” that continuously downloads quotes from config file during the day
     9. Config file listing files to get {file exists/is formatted; incomplete list 6/6}
  2. **data analysis:**
     1. Indicator wrapper class/interface
        1. Makes the “Driver” component agnostic about what Indicator it uses
        2. Indicators should take inputs and return Actions

class Action implements Runnable

{

// Needs a logging facility in the constructor

public enum ActionType { BUY, SELL, SELL\_SHORT, NO\_ACTION }

public ActionType action;

public float confidence;

public float price;

public run; // execute? w/e a runnable does

}

* + - * 1. Should log each action taken
    1. Simple Moving Average on a stock history for time interval
       1. Average of the bid/ask spread per quote (should I bother? See below)
       2. Most-accurate (ask when currently short/no position, bid when long)
       3. Returns a no-action Action object
    2. SMA Crossover for two time intervals
       1. Returns an appropriate Action object based on data up to that point
    3. SMA Crossover for three time intervals { future }
    4. SMA Crossover profitability analysis
       1. Calls SMA crossover on subsequent (overlapping) periods of data; i.e. with 1000 data points, would call SMA crossover on points 0-100, 1-101, 2-102, etc.
       2. Records the results of performing the returned Action object at each
    5. SMA Crossover comparer
       1. Test with short and long periods of time (50-100 quotes per stock and 1000-5000 quotes per stock)
       2. Iterative crossover comparer (store comparison results, compare multiple
    6. “Driver” to run the analysis
       1. loads data from the local data storage reader in to StockHistories
       2. calls the SMA Crossover comparer on each discrete stock history
       3. outputs/stores the results
    7. Neural Net { future }
       1. Takes multiple different indicators and a stock history as input (i.e. 5 min/30 minutes SMA X and 2 minutes/10 minutes SMA X)
       2. Builds a neural net which balances the output of them for the best profitability over the given time frame
       3. Experiment with feeding it the top two/three/four performing indicators for a given stock, the top/mid/worst, etc.
    8. Neural Net 4 Market Data { future }
       1. Takes multiple indicators and multiple equity histories
          1. Should be able to handle indices as inputs
       2. Builds a neural net for behavior on a single equity history using the indicators for that history + the other input histories
  1. **trade:**
     1. “Driver” will loop through:
        1. download latest quote (every N seconds)
        2. Apply the selected indicator to the quote
        3. Execute the returned action
     2. Trader class will perform network calls to the broker
        1. It will authenticate as necessary
        2. It will use the most reasonable available encryption protocols, with a preference for asymmetric cryptography during connection establishment and a non-AES, trusted symmetric cryptographic system for data transmission
        3. Should be able to send buy, sell, short, stop/limit orders
        4. Should be able to send put/call options { future }

1. schedule:
   1. **data retrieval:**
      1. Quote data structure in memory {done 6/6}
      2. Quote data attributes (xml) {done 6/6}
      3. Yahoo API stock retrieval class {done 6/6}
      4. Config file listing files to get {verified format 6/6}
      5. Local data storage writer {in progress 13/6}
      6. Local data storage reader {13/6}
      7. “Driver” that continuously downloads quotes from config file during the day {13/6}
      8. Stock History (quote series) class {20/6}
      9. Interface for getting quotes over time/maintaining them in memory {20/6}
      10. Config file listing files to get {find more equities 20/6}
   2. **data analysis:**
      1. Indicator wrapper class/interface {27/6}
      2. Simple Moving Average on a stock history for time interval {27/6}
         1. Average of the bid/ask spread per quote (should I bother? See below)
         2. Most-accurate (ask when currently short/no position, bid when long)
         3. Returns a no-action Action object
      3. SMA Crossover for two time intervals {11/7}
         1. Returns an appropriate Action object based on data up to that point
      4. SMA Crossover profitability analysis {18/7}
         1. Calls SMA crossover on subsequent (overlapping) periods of data; i.e. with 1000 data points, would call SMA crossover on points 0-100, 1-101, 2-102, etc.
         2. Records the results of performing the returned Action object at each
      5. SMA Crossover comparer {18/7}
         1. Test with short and long periods of time (50-100 quotes per stock and 1000-5000 quotes per stock)
         2. Iterative crossover comparer (store comparison results, compare multiple
      6. “Driver” to run the analysis {25/7}
         1. loads data from the local data storage reader in to StockHistories
         2. calls the SMA Crossover comparer on each discrete stock history
         3. outputs/stores the results
   3. **trade:**
      1. “Driver” will loop through: { tbd }
         1. download latest quote (every N seconds)
         2. Apply the selected indicator to the quote
         3. Execute the returned action
      2. Trader class will perform network calls to the broker { tbd }
         1. It will authenticate as necessary
         2. It will use the most reasonable available encryption protocols, with a preference for asymmetric cryptography during connection establishment and a non-AES, trusted symmetric cryptographic system for data transmission
         3. Should be able to send buy, sell, short, stop/limit orders