MIME 522: Final Project

Algorithmic Trading Strategies

Zachary Levinson

260466524

Introduction

The goal of this project was to create a program that would inform the user of when to buy stocks based on moving average techniques and through the rebalancing of stocks using a specific strategy called the Amazon+ technique. For design of the project historical data was collected from Yahoo Finance to perform the algorithms at hand due to inaccessibility to an API. Assuming that the historical data was the present and performing actions as newer data was added is how the program simulated real-time stock data. Ideally these strategies would provide a higher rate of return than passive trading techniques that most financial advisors try and sell. Map and vector data structures were used to organize and manipulate data to perform these operations. A challenging part of this project was creating a data structure that was flexible to take multiple inputs so that data could be added easily. A map with a vector data type was used to overcome this problem. In completing this task it was found over a period of 1 year that these strategies beat the risk free rate of 6% in the majority of scenarios. Barrick Gold, Amazon, and an ETF for the 20-year treasury yield were used.

Background

Algorithmic trading is the idea of automated trading using a program such as the one created in this project. The idea is to follow a set of instructions to place an order based on price, quantity, timing, or a combination of these attributes. Removing the influence of human interference and emotion is extremely beneficial for traders as it prevents traders from becoming reliant on a particular stock. In addition, algorithm trading allows traders to change positions faster than what is humanly possible.

The first trading strategy implemented was the moving average strategy which based on tracking a short term and long term moving average and examining the trends for cross overs. In this case, the 50-day and 200-day moving averages were examined. The simple moving average is a technical indicator and it smooths out the day-to-day fluctuations or volatility in the market to improve the understanding of the stocks trend. Although, this is not a perfect assumption it is an effective way to forecast future stock movement. Inadvertently, it does not consider the previous problems of human emotion and world events. The 50-day moving average for example is calculated by taking the closing prices for the previous 50 day of any stock and averaging the prices (=sum of last 50 days closing prices/50). The moving average is updated after each closing price is made available. These moving averages could be calculated based on monthly, weekly, or open prices as well.

The cross over strategy provides a signal to the trader by checking if the moving 50-day moving average has breached the 200-day moving average or vice-versa. Using multiple moving averages allows the trader or program in this case to see the long-term vs short-term trend.

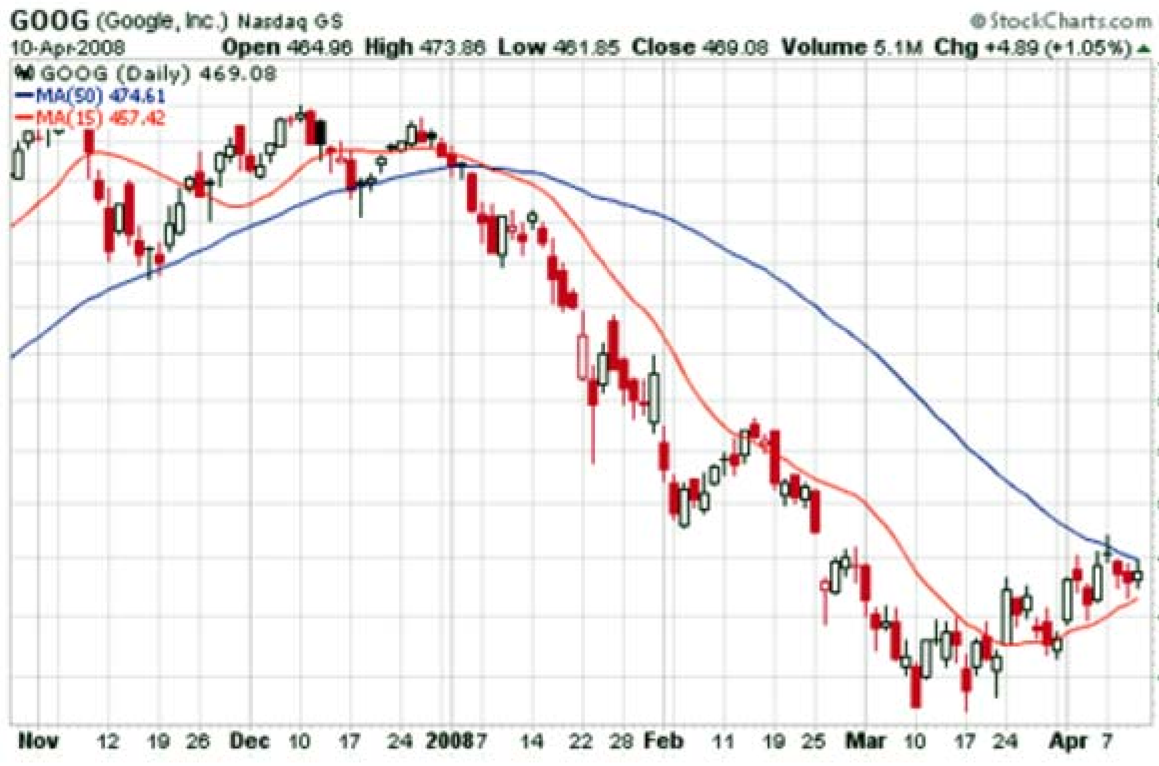


Figure 1: Moving Average Example

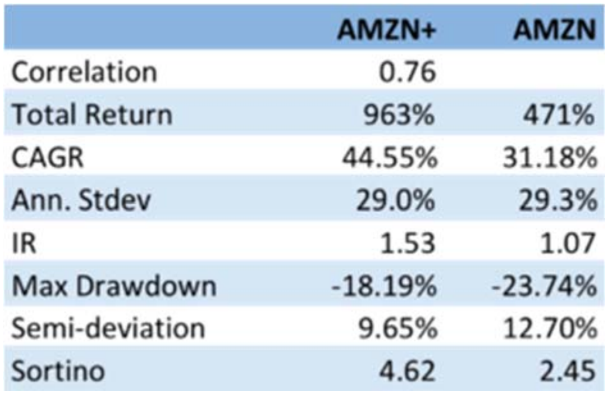
http://www.investopedia.com/articles/technical/052201.asp?rp=i

In Figure 1, the short-term moving average (red) falls below the long-term moving average (blue) indicating a falling trend in the coming periods. This should indicate traders to short sell. The opposite case is when the short-term moving average breaches the long-term moving average indicating the program to buy the stock.

The Amazon+ strategy is a different strategy which only works when targeting a specific correlation between two stocks. A quantitative research blogger Jonathan Kinlay found an interesting relationship between Amazon (AMZN) and an ETF containing 20-year treasury yields called (TMF). The correlation between these two stocks is 0.76, and the performance of the

AMZN+ strategy is about 20% better over a 5-year period (Kinlay 2016). However, the more crucial improvement is the reduction in volatility by owning the combination of these two stocks.

The AMZN+ strategy is a portfolio that contains 66% AMZN, and 34% of the Direxion Daily 20 Yr Trsy Bull 3X ETF (TMF). The idea was to rebalance the portfolio at the end of every month. In this case, a small adjustment was made where if the % of AMZN in the portfolio exceeded 76% or went below 56% the portfolio would automatically rebalance. Improving the performance of the strategy and automating it on a more consistent basis.



	AMZN+	AMZN
Correlation	0.76	
Total Return	963%	471%
CAGR	44.55%	31.18%
Ann. Stdev	29.0%	29.3%
IR	1.53	1.07
Max Drawdown	-18.19%	-23.74%
Semi-deviation	9.65%	12.70%
Sortino	4.62	2.45

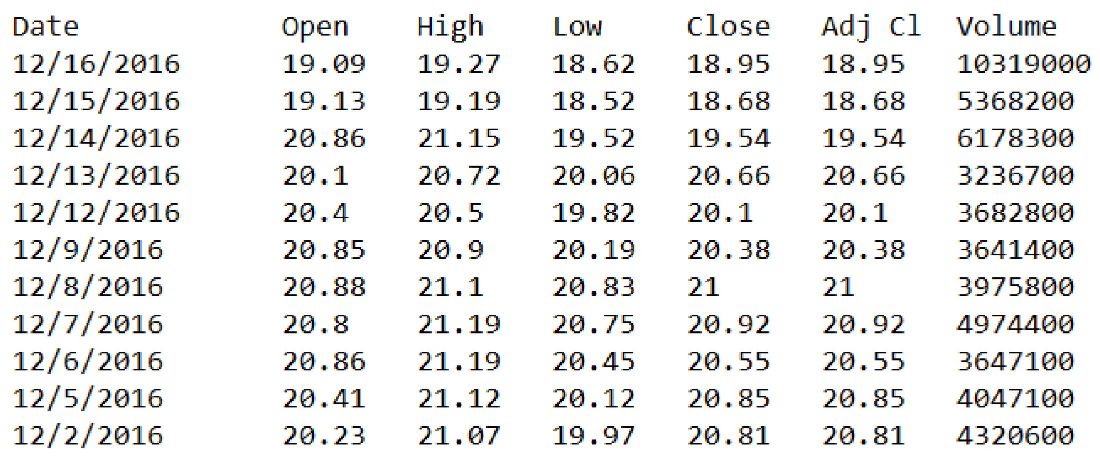
Figure 2: Statistics behind Amazon+

http://jonathankinlay.com/2016/07/the-amazon-killer

The basic principal is to remove human emotion from the equation and buy or sell stocks based on performance indicators rather than human decision principals. In addition, automation of trading is ideal so a trader or anyone for that matter can trade without having to keep up with the stock market itself.

Design

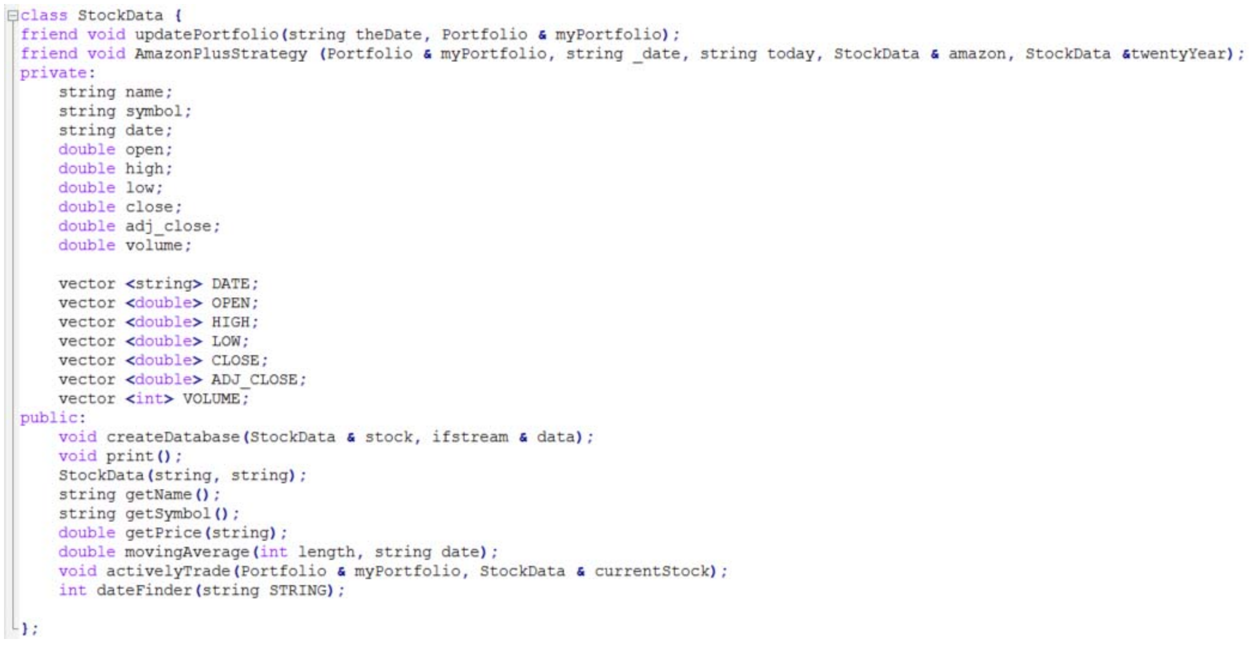
The building blocks of this program are based on the way data is manipulated into a data frame to store important stock information and perform operations on this data. The stock data is input as a text file however it was built in a way that is dynamic using a vector container. The reason a dynamic container was used is when new data is available it can be read into the program and added to end of the vector using the push_back modifier. Class StockData creates an object that contains vectors pertaining to each column in Figure 3. Stock data is stored in these vectors and an instance created for each stock the user is interested in trading.



Date	Open	High	Low	Close	Adj Cl	Volume
12/16/2016	19.09	19.27	18.62	18.95	18.95	10319000
12/15/2016	19.13	19.19	18.52	18.68	18.68	5368200
12/14/2016	20.86	21.15	19.52	19.54	19.54	6178300
12/13/2016	20.1	20.72	20.06	20.66	20.66	3236700
12/12/2016	20.4	20.5	19.82	20.1	20.1	3682800
12/9/2016	20.85	20.9	20.19	20.38	20.38	3641400
12/8/2016	20.88	21.1	20.83	21	21	3975800
12/7/2016	20.8	21.19	20.75	20.92	20.92	4974400
12/6/2016	20.86	21.19	20.45	20.55	20.55	3647100
12/5/2016	20.41	21.12	20.12	20.85	20.85	4047100
12/2/2016	20.23	21.07	19.97	20.81	20.81	4320600

Figure 3: Stock Data Input

A secondary class is used to store the stocks name, symbol, etc. Every time a stock is purchased for the first time its data is then stored in this vector. When buying, or selling a stock the program will check to see if the stock is in the portfolio; if it is not it will be added to the portfolio. If the stock is being sold completely it will delete the stock from the portfolio. This allows the program to know which stocks it already has and which ones it does not.



```
class StockData {
friend void updatePortfolio(string theDate, Portfolio & myPortfolio);
friend void AmazonPlusStrategy (Portfolio & myPortfolio, string _date, string today, StockData & amazon, StockData &twentyYear);
private:
    string name;
    string symbol;
    string date;
    double open;
    double high;
    double low;
    double close;
    double adj_close;
    double volume;

    vector <string> DATE;
    vector <double> OPEN;
    vector <double> HIGH;
    vector <double> LOW;
    vector <double> CLOSE;
    vector <double> ADJ_CLOSE;
    vector <int> VOLUME;
public:
    void createDatabase(StockData & stock, ifstream & data);
    void print();
    StockData(string, string);
    string getName();
    string getSymbol();
    double getPrice(string);
    double movingAverage(int length, string date);
    void activelyTrade(Portfolio & myPortfolio, StockData & currentStock);
    int dateFinder(string STRING);

};
```

Figure 4: StockData Class and Functions

Finally, the third class is used to organize the portfolio where the default constructor builds a portfolio with a $100,000 cash balance. The class has two main double precision variables which are accountCash and investments. These two components are the basis of any portfolio. For each stock in the portfolio the quantity purchased and price paid for the stock is tracked. Using a few calculations and manipulations the average cost of the stock, mkt value, book value, and percent gain and loss are easily calculated. A map data structure is used to store each stock that is purchased and these components are stored in their described position in the vector. An array would have been sufficient however for flexibility a vector was used in case there is a desire to track another indicator in the Portfolio class.

Maps are a container that store elements formed by a combination of a key value and a mapped value. The key in this case is the stock ticker and the mapped value is the ticker This map holds the current holdings in each portfolio so each ticker for the stock and the value is a vector containing these important features.

The last feature is a global variable called trackedStocks, which stores each StockData instance to determine whether that instance or stock exists.

Functions

Class StockData

createDatabase - After an instance is created the data frame is built using the function takes in data from each line and stores them in the appropriate vector using the push_back method. Upon completion, it closes the input file.

print – prints stock data to ensure the database creation is performed properly

StockData – constructor that sets the stock name and symbol

getName – get the name of a stock

getSymbol – get the stock ticker e.g. Barrick Gold Corp = ABX

getPrice – finds the current date and finds the corresponding current adjusted closing price

movingAverage – calculates moving averages of various lengths. In this case a 50 and 200 day moving average were calculated by summing all closing prices and averaging them.

activelyTrade – a function that outputs all that should be implanted using the moving average strategy discussed earlier. Outputs the files to a file called stock_trades.txt.

Class Portfolio

printPortfolio – outputs current portfolio holdings and performance indicators

printAlgorithmResults – is a function created to see what trades are being made and when to ensure the program works properly

Each of the calculate functions do exactly as they say.

buyStock – is a function which will add stocks to the portfolio and purchase them if there is enough cash in the account. Adds stocks that are purchased to the map :: currentHoldings variable using a key that is the stocks name and a vector that stores the important information that pertains to the buyer.

sellStock – is the opposite of buy stock it will a sell several shares or completely sell the stock and remove it from the portfolio.

Other Functions

openInputFile – opens the input file of a given name

amazonPlusInitiate – initiates the amazon plus strategy by using the 66% of the portfolio cash to invest in AMZN and 34% of TMF at current price of initation and 32%.

amazonPlusStrategy – implements trades every time a 10% increase or decrease is made of a stock. It rebalances the portfolio to the original 34 and 66 percent when this target is breached.

Results

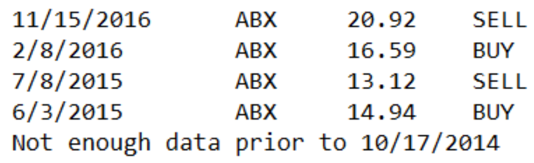
The strategies provide a positive net return of 7-25% when implemented over a duration of one year for the amazon+ strategy and 15-20% over two years using the moving average strategy. However, these results are not guaranteed to be reproducible. Barrick Gold and the Amazon+ strategy provided a rate of return that was significantly better than the 6% risk free rate based off National Bank

Conclusion

Both algorithms results made the algorithm profits and could be optimized more effectively using an API and more precise pricing patterns. This experiment helped build skills using maps and vectors to store and manipulate data. The value of programming these algorithms is extremely high and for further work an optimized algorithmic trading model would be ideal. Instead of using only the closing prices that were used in this program it would be far superior to use real time pricing.

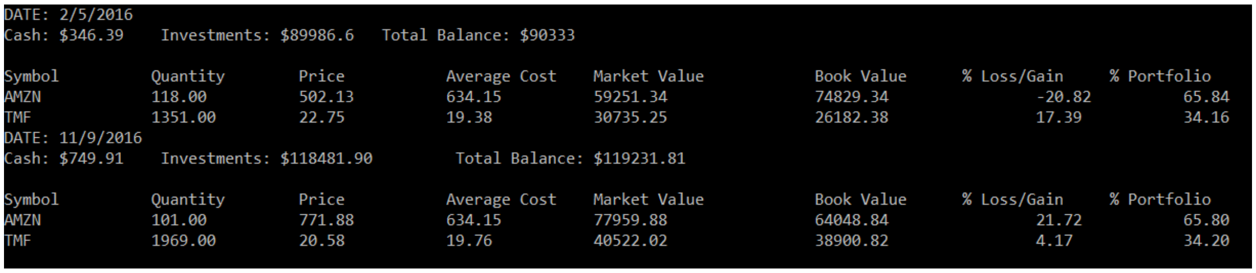
Appendix

Test Results – Moving Average



```
11/15/2016      ABX     20.92    SELL
2/8/2016        ABX     16.59    BUY
7/8/2015        ABX     13.12    SELL
6/3/2015        ABX     14.94    BUY
Not enough data prior to 10/17/2014
```

Test Results – Amazon Plus Strategy



```
DATE: 2/5/2016
Cash: $346.39    Investments: $89986.6   Total Balance: $90333

Symbol         Quantity       Price          Average Cost   Market Value           Book Value     % Loss/Gain    % Portfolio
AMZN           118.00         502.13         634.15         59251.34               74829.34             -20.82         65.84
TMF            1351.00        22.75          19.38          30735.25               26182.38             17.39          34.16
DATE: 11/9/2016
Cash: $749.91    Investments: $118481.90        Total Balance: $119231.81

Symbol         Quantity       Price          Average Cost   Market Value           Book Value     % Loss/Gain    % Portfolio
AMZN           101.00         771.88         634.15         77959.88               64048.84             21.72          65.80
TMF            1969.00        20.58          19.76          40522.02               38900.82             4.17           34.20
```

References

CPP Reference, Std:Map. (2017, January 03). Retrieved January 12, 2017, from CPP Reference, http://en.cppreference.com/w/cpp/container/map

Kinlay, J. The Amazon Killer. (2016, July 24). Retrieved January 11, 2017, from http://jonathankinlay.com/2016/07/the-amazon-killer/

Seth, S. (2016, January 26). Basics of Algorithmic Trading: Concepts and Examples. Retrieved January 11, 2017, from http://www.investopedia.com/articles/active-trading/101014/basics-algorithmic-trading-concepts-and-examples.asp

Yahoo Finance, Stock Data, Retrieved December 16, 2017, from Yahoo Finance https://ca.finance.yahoo.com/