Operations Research Techniques in the Formulation of an Investment Strategy

Ivan Busharov

Tyler Diller

Haris Krijestorac

21-292, Carnegie Mellon University, Fall 2008

Contents

1. **Abstract** **pg. 3**
2. **Introduction** **pg. 3**
3. **Prediction** **pg. 3**
    - **3.1. Moving Averages** **pg. 4**
        - **3.1.1. Simple Moving Average**
        - **3.1.2. Weighted Moving Average**
    - **3.2. Modified Regression** **pg. 5**
4. **Portfolio Optimization** **pg. 6**
5. **The Testing Framework** **pg. 10**
6. **Analysis** **pg. 13**
7. **Conclusion** **pg. 14**
8. **Appendix** **pg. 15**

Abstract

In this paper we will discuss the steps towards setting up the framework for a securities investment algorithm. The paper will go over various prediction algorithms, as well a portfolio allocation strategy based on a genetic algorithm. Reasonable evidence is given that the algorithms presented are effective on their own. However, they also leave room for more algorithms and heuristics to be integrated into the overall strategy.

Introduction

The idea for this project is a marriage of two ideas: a stock movement prediction algorithm, and a portfolio allocation algorithm. The portfolio allocation algorithm takes the predicted movements of a set of stocks and accordingly assigns a percentage of the portfolio to allocate to each stock. In Operations Research terminology, we are solving the constrained problem of maximizing return on investment, while minimizing the variance of the portfolio. We want an algorithm that is expected to make money, and do so with relatively high certainty. We will go through the prediction algorithms and portfolio allocation algorithms separately, before consolidating them and showing our results.

Some may argue that our attempt at an investment strategy is futile, since many are currently in existence. However, the key is that these algorithms rely on others not knowing how they work! As a result, these algorithms are kept in secret and it is essentially up to individuals or coalitions of people to come up with an investment strategy. This paper will present a basic investment strategy. As would be the case with any such strategy, there is virtually infinite room for refinement. Our goal was to bring our algorithm to a point at which it was useful as is, but still leave room for refinements. This paper can provide virtually anyone interested in creating an investment strategy with a starting off point.

Prediction

The following is an overview of our various prediction algorithms. Although we do suggest the modified regression approach, we will present the evolution of our prediction algorithms used. Each algorithm works on the same concept: take closing values of the previous n periods, and predict the closing value of the (n+1)st period. The parameter of n can easily be adjusted in our code (see appendix). If we define a ‘period’ as more than one day (e.g., one month), then we take the average of all closing values over that period. Therefore number of input values and span of applicability for the prediction output can both be adjusted in accordance with the purposes of an allocation algorithm.

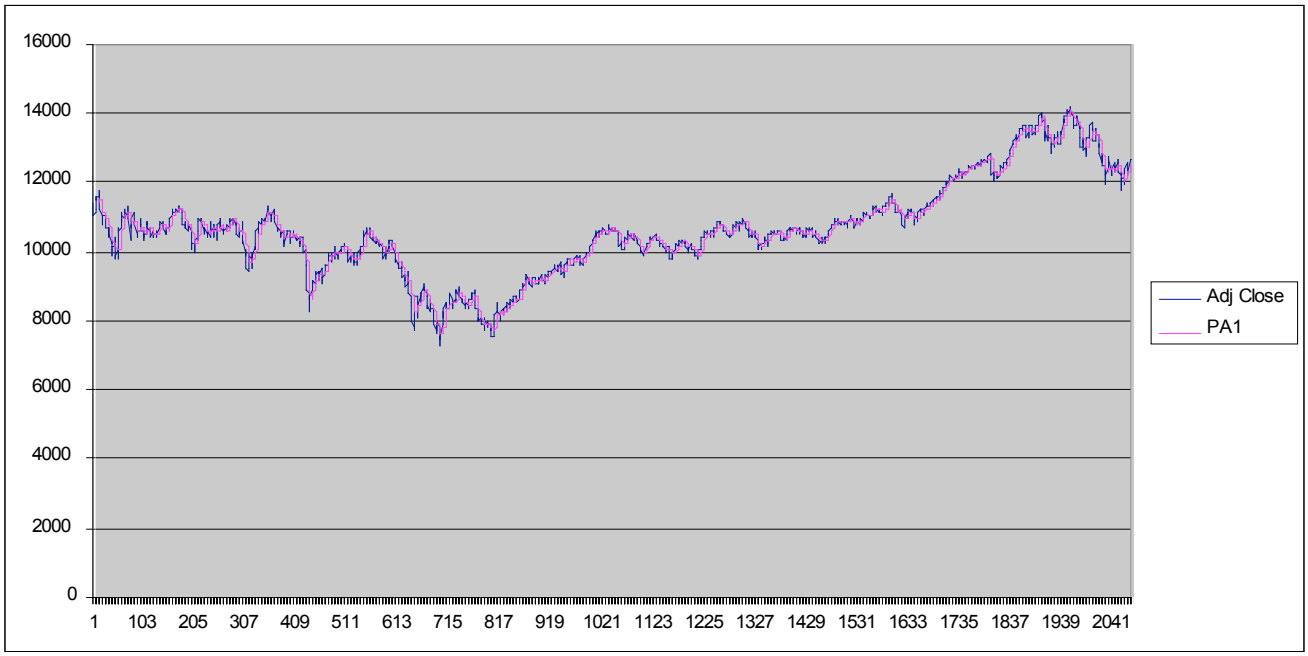
Moving Average:

Moving averages a very basic forecasting technique. We will go through two types of moving averages, with the second one more accurate than the first.

Moving Average 1: Simple Moving Average

In this approach, we take the previous n closing values (or averages of closing values), average them, and use this value as our prediction for the (n+1)st time period. The advantage of this algorithm is that it is very easy to implement, and can yield reasonable results for small periods and small values of n. However, the main disadvantage is that the input data lags. For example, with n=20, we would not want the value from 20 days ago to have as much of an effect on our prediction as the value from 1 day ago. But with a simple average, this is indeed the case.

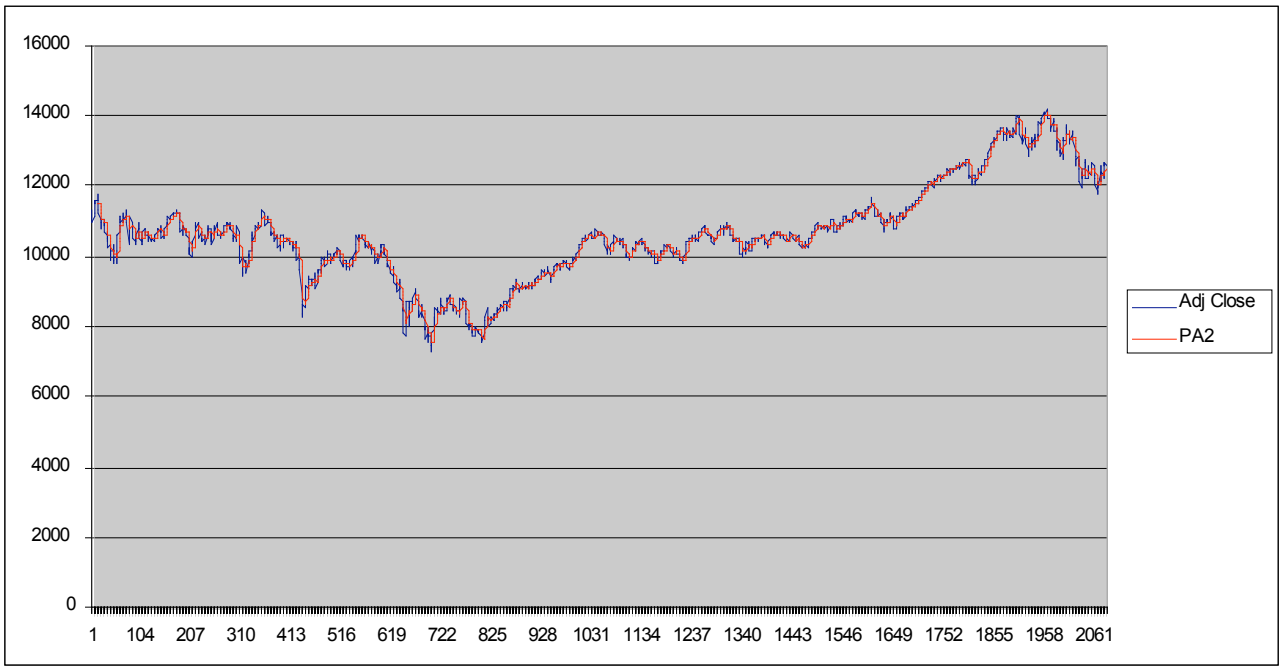
The following is a spreadsheet depicting how the simple moving average algorithm predicted values of the Dow Jones Industrial Average since the beginning of 2000. (n=10, period = 1 day). It is evident from the graph that the predictor suffers from lag, since the predicted



Moving Average 2: Weighted Moving Average

This approach takes a step towards rectifying the lag effect of the simple moving average by applying appropriate weights to each value and taking a weighted average. For example, with n=20, the value from 20 days ago will have a weight of 1, the value from 19 days ago will have a weight of 2, and so forth until the value from yesterday has a weight of 20. Note that this weighting system can also be modified many ways, and weights do not even have to be linear. We simply chose this heuristic, but as all heuristics go, it can be subject to modification.

The following is a spreadsheet depicting how the weighted moving average algorithm predicted values of the Dow Jones Industrial Average since the beginning of 2000. (n=10, period = 1 day)



Modified Regression:

Our third approach combines typical linear data modeling techniques such as extrapolative forecasting with our own logic-based heuristics. This method takes an input of the closing values of previous two time periods, as well as the high and low values of the previous day.

In order to eventually understand this algorithm, one should first picture a simple extrapolation. The closing values of the two previous days make a line. Extrapolating the line to tomorrow yields a prediction for tomorrow's value. However, there was a significant amount of randomness in the randomness of the closing value of tomorrow. Throughout the day, the value of the stock fluctuated between its high and low values. As a result, the value that the stock winds up closing can be viewed as a random number between that high and low.

Our modified regression approach takes this randomness into account by adjusting the simple extrapolation by pulling the line of extrapolation closer to the midpoint between the high and the low values. Mathematically, it can be described as follows (all variables italicized):

- Take the closing values of the previous two days and find their difference [their difference = *net*]
- Introduce variable *pchange*, which will be between 0 and 1, and defined:
    - If yesterday's closing value (*pt*) > yesterday's midpoint (*mdpt*)

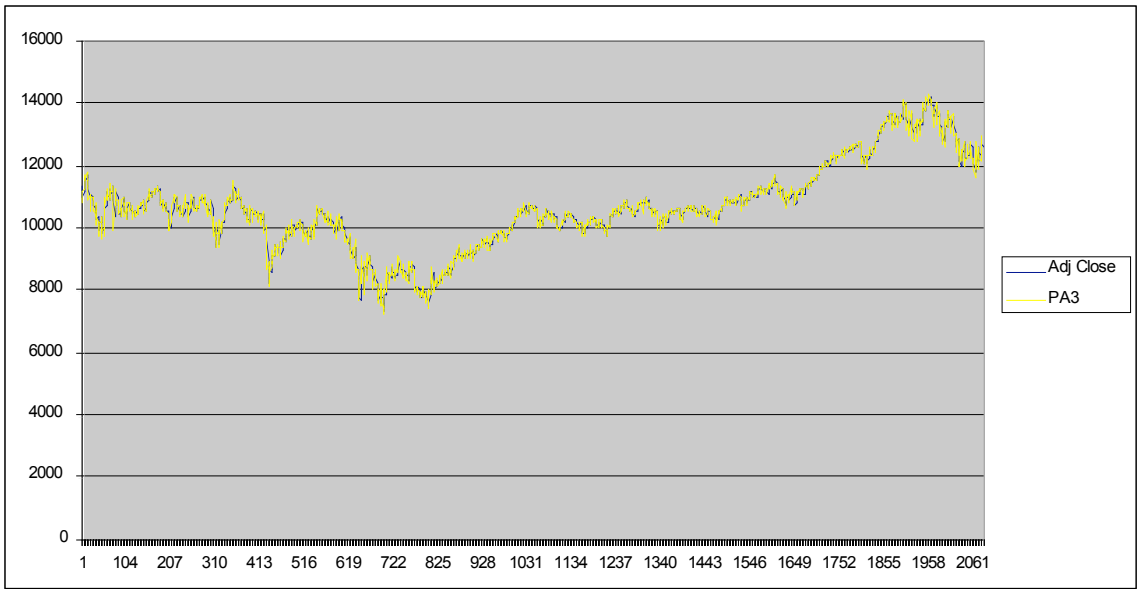
        → *pchange* = (*pt*/*mdpt*)
    - If yesterday's closing value < yesterday's midpoints (*mdpt*)

→ *pchange* = [1 – (*pt*/*mdpt*)]

- Predicted close value = *(pchange * net) + pt*

To summarize, extrapolate the line of yesterday's closing value with today's closing value. Also extrapolate the line of yesterday's closing value with the midpoint value of today (the average of today's high and today's low values). By introducing *pchange*, the algorithm takes into account the randomness of yesterday's closing value, and algorithmically produces an extrapolation line somewhere between the two aforementioned lines.

The following is a spreadsheet depicting how the weighted moving average algorithm predicted values of the Dow Jones Industrial Average since the beginning of 2000. (n=10, period = 1 day). As you can see, the line representing the real closing values and the line representing our predictions are virtually overlapping.



Portfolio Optimization

So now we have the results for our predictor algorithm, and the question becomes: how to allocate a sum of cash we have to effectively make the best investment choice. The data we will use will include both the predictor algorithm and the fundamental analysis, to demonstrate the differences in the portfolio allocations based on the expected value inputs. Another reason would be to use expected values that are obtained by an industry-standard method, and by a new method that is not widespread in its use.

The main defining property of stocks is the fact that they are risky securities, meaning that the returns are in no way guaranteed and the stock price is very likely to drop in value, giving a negative return to the shareholder. Given a certain degree of confidence in the expected return of a security, the variance of the realized return vs. the expected return would serve as an adequate measure of risk. Using these measures, one can implement an algorithm to pick the stocks that have the highest returns for given risk, and vice versa. The algorithms to accomplish such a task are described as a part of the Modern Portfolio Theory, which deals with optimizing a collection of defined assets.

A Nobel-Prize winning paper by Harry Markowitz created a notion of an optimized portfolio, a collection of stocks that carries the least amount of risk. This risk was defined as the previous variances of stock returns, and the covariances of pairs of securities. The constraints to the problem include a set expected portfolio return, and the condition that stock proportions add up to 1. The main optimization problem is presented as such:

Minimize: $W = \sum\sum w_i w_j \sigma_{ij}$

Constraints: $\sum w_i = 1$

$0 < w_i < 1$

Given: w_i – weight of asset in portfolio, a proportion of total money invested in a security

σ_{ij} – covariance between assets i and j

The main problem is augmented by the condition that various investors have a varying taste for risk; hence a λ-value will denote the investor's risk preference. The λ value will modify the objective function value by multiplying the sum of expected returns, and the inverse of λ will multiply sum of variances:

Minimize: $W = \lambda [\sum\sum w_i w_j \sigma_{ij}] + (1 - \lambda) [\sum w_i \mu_i]$

Constraints: $\sum w_i = 1$ for $i = 1 \ldots N$

$0 < w_i < 1$ for $i = 1 \ldots N$

$0 < \mu_i < 1$ for $i = 1 \ldots N$

Given: w_i – proportion of asset in portfolio

μ_i – expected return of an asset

λ – measure of risk averseness

The given optimization problem is solved by quadratic programming, involving systems of equations. This method is an effective solution, giving a result in polynomial time, and giving the absolute optimal portfolio. The obvious problem occurs when we increase the size of our portfolio to look at a broad spectrum of assets, thus increasing our time complexity by a higher order than $O(h^3)$ In addition we have to resort to mixed-integer programming if the optimization problem is faced with a constraint limiting the number of present assets in the collection:

Thus the problem we want to consider is such:

Minimize: $W = \lambda [\sum\sum w_i w_j \sigma_{ij}] + (1 - \lambda) [\sum w_i \mu_i]$

Constraints: $\sum w_i = 1$ for $i = 1 \ldots N$

$0 < w_i < 1$ for $i = 1 \ldots N$

$0 < \mu_i < 1$ for $i = 1 \ldots N$

$\sum I(w_i) = K$ I = indicator function = 1 if asset is present in portfolio

Given: w_i – proportion of asset in portfolio

μ_i – expected return of an asset

λ – measure of risk averseness

We will consider a genetic algorithm to solve this problem, because unlike the QP method, it has a O(1) complexity. Let us look at the algorithm:

Genetic Algorithm:

For all λ values that we want to consider:

1. Create a solution set consisting of randomly created solutions
2. Iterate the following until desired accuracy
    a. Get a pair of solutions by binary tournament
    b. Cross their properties to produce a new solution
    c. Evaluate the objective function of child
    d. Compare the child to the worst solution in the solution set
    e. If the child is better, replace the worst solution with child
3. Plot out the efficient frontier

Binary tournament:

1. Randomly input 2 sets containing 2 solutions each
2. In each set:
    a. Pick the best solution by evaluating the objective function
3. Return a pair of solutions

Crossing genes:

Rules:

1. If an asset is present:
    a. in both parents: it is present in the child, weight is randomly assigned from either of the parents
    b. in one parent: 50% chance of being present in child with same weight
    c. in neither: the asset is not present in child
2. Mutate genes:
    a. 50% chance of increasing weight by 10%
    b. 50% chance of decreasing weight by 10%

The testing framework:

To see how effectively the algorithm performs, we established an 11-asset portfolio that would be used for creating an efficient frontier. The number of assets used is typical for a portfolio optimization scenario, and would reflect a real-life problem faced by wealth managers on a day-to-day basis.

The test portfolio consists of stocks ranging in their capitalization, and volatility, thus creating a perfect test case. The stocks in our portfolio are:

DJI, GOOG, S&P500, ESLR, FSLR, MSFT, XOM, TM, SFEG, LUV, SBUX

We used historical daily closing price data to calculate the covariance matrix. Among the assets, the least volatile stock was Exxon Mobil Corporation, one of the biggest oil companies, which avoided huge losses even as the price of oil jumped and dropped dramatically over the course of the past two years

For the expected returns of stocks, we used two conjectures to predict the price. The first method used our predictor algorithms from the first part of the paper and created a vector of predicted returns. The predictor is based on technical analysis, and thus uses previous historic data to predict the future movements. This method of analyzing stocks has a number of followers,

and has been proven to work to a certain degree of profitability. We will use this data to create an optimized portfolio based on this theory.

However, there exists a different philosophy regarding stock prices, and that has to do with the underlying value of the stock at any given time. This theory is called fundamental analysis, and emphasizes that stocks will generally follow the movement in the underlying value of the company, thus an investor would need to look at the company earnings, cashflows, and capital acquisitions to make an informed decision dealing with the movement of the stock in the future. The projections for each company were based on the availability of cash, and the recurring cashflows, projected revenue, and projected expenses. The emphasis was put on cash availability, due to the recent developments in the credit/debt markets it becomes apparent that cash is king and companies that are financially secure will return the highest profit.

Overall the predictor values gave a grim outlook on the basis of the previous losses, especially in the year 2008. There is no denying that the current economic situation is terrible, and may get worse, therefore this prediction can be attributed to a bearish outlook on the market in the future. The fundamental analysis evaluation gave a more promising outlook on the stock market, given that the current prices are depressed a significant amount. Since fundamental analysis figures out the intrinsic value by taking the (Current Valuation) / (Shares Outstanding) some of the stocks had been below their value, and the expected turn of events is the realization of that value.

We will use the two sets of data to create a two optimal portfolios with $K = 7$, meaning we will restrict our portfolios to have exactly 7 assets. The results will be shown in the form of the efficient frontier.

The covariance matrix and the two expected predictions are as follows:

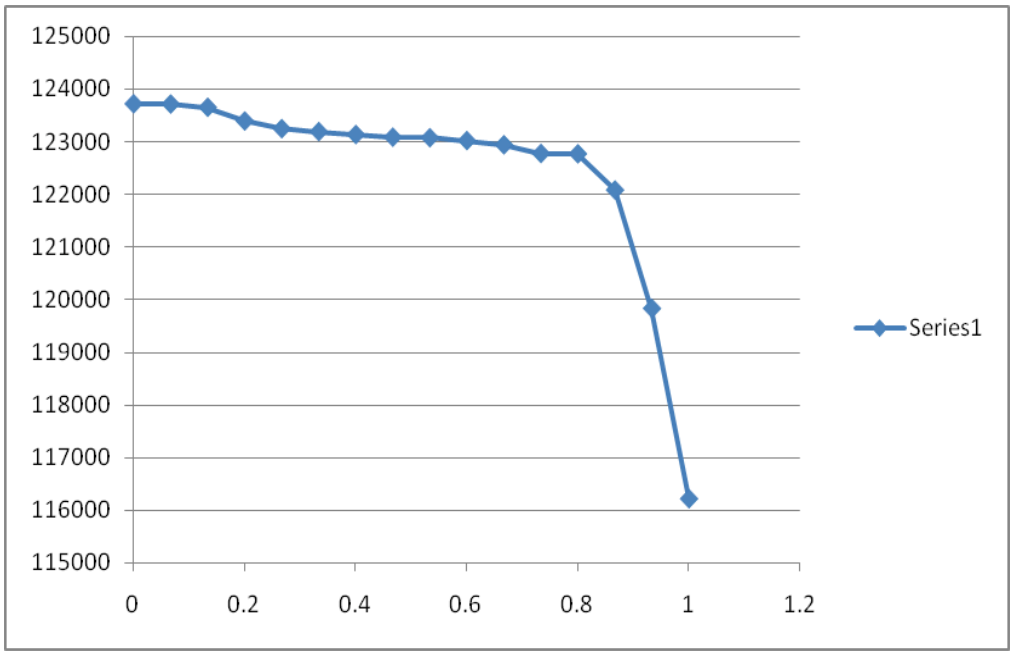
	DJI	*GOOG*	*S&P500*	*ESLR*	*FSLR*	*MSFT*	*XOM*	*TM*	*SFEG*	*LUV*	*SBUX*
DJI	0.0118	0	0	0	0	0	0	0	0	0	0
GOOG	0.0154	0.0379	0	0	0	0	0	0	0	0	0
S&P500	0.0127	0.0162	0.0141	0	0	0	0	0	0	0	0
ESLR	0.0264	0.0578	0.029	0.1316	0	0	0	0	0	0	0
FSLR	-0.0415	0.1754	-0.0667	0.2762	8.3522	0	0	0	0	0	0
MSFT	0.0096	0.0182	0.0103	0.033	0.02	0.0116	0	0	0	0	0
XOM	0.0064	0.0132	0.0060	0.0188	0.1228	0.0060	0.0084	0	0	0	0
TM	0.012	0.0104	0.014	0.0238	0.2069	0.0087	0.0021	0.0185	0	0	0
SFEG	-0.0029	0.0157	0.0006	0.0084	0.1883	0.0045	0.0103	0.0086	0.0357	0	0
LUV	0.0061	0.0043	0.0074	0.0062	0.0842	0.0029	0.0005	0.0085	0.0043	0.0108	0
SBUX	0.0154	0.0108	0.0183	0.0246	0.4009	0.0107	0.0004	0.0249	0.012	0.0128	0.038

Using Proprietary Predictor				Using Economic Indicators and Fundamental			
Company	Predicted	E[return]		Company	Predicted	E[return]	
DJI	6261.18	419.1	0.7437	DJI	8623.48	419.1	1.0243
GOOG	210.79	275.1	0.7662	GOOG	290.43	275.1	1.0557
S&P500	624.23	848.81	0.7354	S&P500	857.29	848.81	1.01
ESLR	1.23	2.47	0.5182	ESLR	1.2	2.47	0.4858
FSLR	107.49	116.12	0.9257	FSLR	123.56	116.12	1.0641
MSFT	16.29	19.15	0.8507	MSFT	22.32	19.15	1.1655
XOM	72.09	77.6	0.9289	XOM	86.21	77.6	1.1108
TM	35.75	61.95	0.5771	TM	75.44	61.95	1.2178
SFEG	0.49	0.59	0.8305	SFEG	0.73	0.59	1.2373
LUV	8.04	8.33	0.9652	LUV	10.23	8.33	1.2281

The algorithm came up with two sets of solutions for varying degrees of risk appetite. The results are displayed in the form of four graphs: two graphs are showing the efficient frontier for the problem, and the two following graphs show the composition of the portfolio for varying values of lambda.

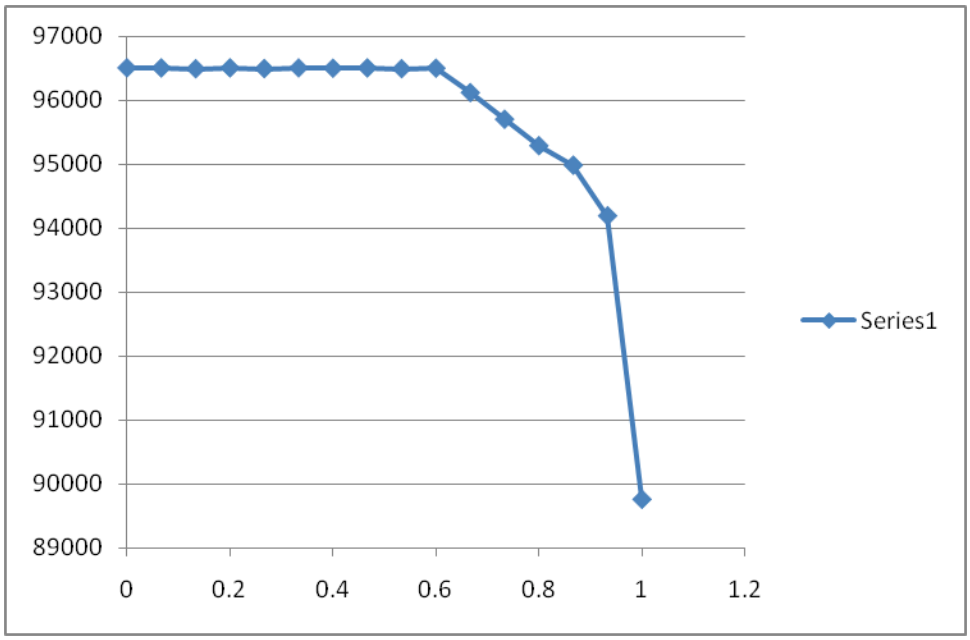
Efficient frontier – Fundamental Analysis

LAMBDA	RETURN
0	123718.5
0.066667	123713.6
0.133333	123651.5
0.2	123391.2
0.266667	123246.6
0.333333	123190.4
0.4	123133.7
0.466667	123083.9
0.533333	123072.8
0.6	123019
0.666667	122940.1
0.733333	122773.9
0.8	122767.6
0.866667	122079.5
0.933333	119827.7
1	116207.8



Efficient frontier – OR Predictor Analysis

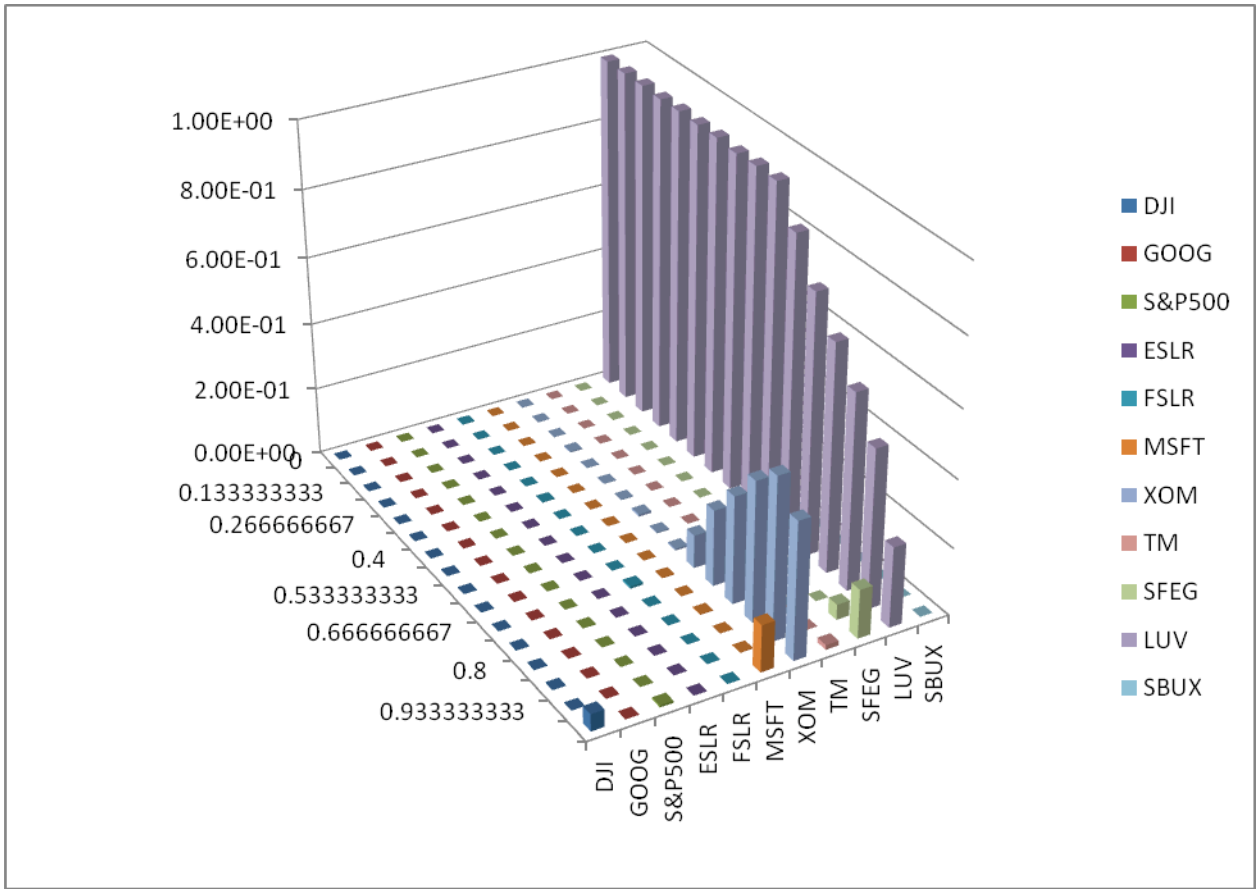
LAMBDA	RETURN
0	96505.71858
0.0666667	96505.25097
0.1333333	96492.32008
0.2	96505.56866
0.2666667	96494.3681
0.3333333	96505.26967
0.4	96500.67211
0.4666667	96505.34093
0.5333333	96492.68344
0.6	96500.37555
0.6666667	96119.01049
0.7333333	95702.15838
0.8	95288.62184
0.8666667	94983.36136
0.9333333	94192.78974
1	89759.42402



Analysis:

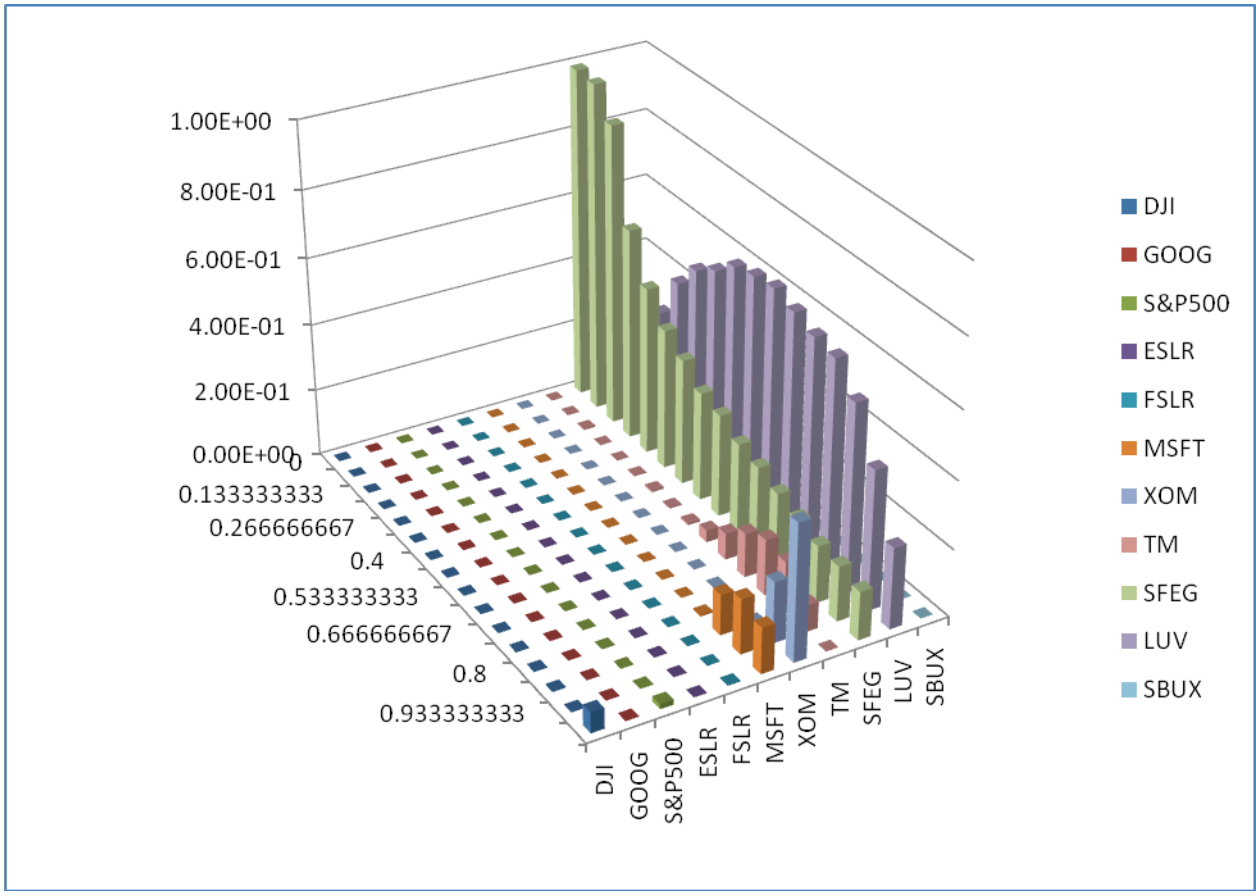
As we can see the efficient frontier is downward-sloping function, because high values of lambda convey a high degree of risk averseness. Therefore, the smaller the lambda value is, the higher the expected return of the portfolio. It is also worth reiterating that the efficient frontier outputs un-dominated portfolios, meaning, for a given measure of risk, the resulting solution gives the portfolio with the highest possible expected value. In the OR predictor model, we can see that for any measure of risk, we are still expecting to lose at least 4% off our portfolio, which means that it makes no sense to keep money in the stock market, given that we can invest money in virtually riskless securities. In the Fundamental Analysis model, we can see the risk-averse portfolio expects an 11% return, which is the long-run average return on stock indices, which are considered very safe investments. Therefore we can see that both models produce un-dominated portfolios, and the algorithm works as expected.

Collections of Assets – OR Predictor Model



This graph plots the weights of each asset in the portfolio, given the value of lambda on the left. We can see that as we decrease the risk-averseness of our portfolio, the stock with the highest return –LUV, becomes the dominant investment.

Collections of Assets – Fundamental Analysis Model



This graph plots the various asset weights as a function of lambda. Note, as risk averseness of the investor increases, the asset collection becomes more and more diversified, leading to a decrease in portfolio variance. As we decrease the value of lambda, the portfolio starts containing more of SFEG, which has the highest expected return out of all the stocks in our model.

Conclusion

From these results we can conclude that the algorithm performs up to specification, decreasing the portfolio variance for a given value of risk. The efficient frontiers that are constructed using different predictor methods, and the subsequent genetic algorithm optimization create a curve, which an investor could use to pick the best collection of assets, given that he knows his own appetite for risk, or a desired expected return. The results give us effective measures for proportions of assets to be included in our real-world example of a portfolio of stocks, showing that as risk decreases, diversification becomes more important, at the cost of expected portfolio return. The algorithm evaluation demonstrated that a rather substantial problem of 11 assets could be solved in millisecond time, using a fixed number of iterations, compared with a polynomial time QP algorithm that performs poorly as the number of assets increases. Overall the objective for creating new/uncommon optimization methods was very successful, and provides real-world functionality to real investment problems. Please see attached Appendix for Python and Java code for the predictor and optimizer methods respectively.

Appendix

Prediction algorithms:

```
class Predictor


 def initialize(file="test2.csv")
  @stock_file= file
  @file_lines = Array.new
  read_file
 end


 def read_file
  if @file_lines.empty?
   @file = File.open(@stock_file)
   @file_lines = @file.readlines()
  end
 end


 #Prediction Algorithm 1: Simple Moving Average
 #Description: Take the previous n close values and average them to obtain your prediction for n+1 case
 # By default, n is set to 4, but the method can be called on any n as shown in the test code at the very bottom
 def simpleMovingAvg(n=4)
  #n = n-1
  i=0
  previous = Array.new(n)


  while(i<=n)      #need a sample size of at least n before making a prediction - insert nil values until you have n values
   previous[i] = nil
   i= i+1
  end


  itr = 0
  @file_lines.each do |line|
```

```
    puts "Iteration" + itr.to_s
    puts line
    itr+=1
    dateV, openV, highV, lowV, closeV, volumeV, adjCloseV, allOthersV = line.split(",")
    puts "Date: " + dateV + ", Close: " + closeV

    if(allValsNumeric?(previous))
      puts " Close Before that: " + previous[previous.length-1].to_s
      puts "Prediction: " + average(previous).to_s
    end

    previous= update(previous, Float(closeV))
    puts previous
  end
end

#Prediction Algorithm 2: Weighted Moving Average
#Description: Takes linearly weighted average of previous n close values and average them to obtain prediction
# By default, n is set to 4, but the method can be called on any n as shown in the test code at the very bottom
def weightedMovingAvg(n=4)
  i=0
  previous = Array.new(n)

  while(i<=n)       #need a sample size of at least n before making a prediction - insert nil values until you have n values
    previous[i] = nil
    i= i+1
  end

  itr = 0
  @file_lines.each do |line|        #reads each line of the file
    puts "Iteration" + itr.to_s
    puts line
    itr+=1
    dateV, openV, highV, lowV, closeV, volumeV, adjCloseV, allOthersV = line.split(",")
```

```
    puts "Date: " + dateV + ", Close: " + closeV

    if(allValsNumeric?(previous))
      puts "Prediction: " + weightedAverage(previous).to_s    #displays the prediction according to the moving average
    end

    previous= update(previous, Float(closeV))
    puts previous
  end
end

#Prediction Algorithm 3: Modified Regression
#See description in paper
def modifiedRegression
  i=0
  previous = Array.new(n)

  while(i<=2)       #need initial sample size of 2
    previous[i] = nil
    i= i+1
  end

  itr = 0
  @file_lines.each do |line|
    puts "Iteration" + itr.to_s
    puts line
    itr+=1
    dateV, openV, highV, lowV, closeV, volumeV, adjCloseV, allOthersV = line.split(",")
    middle = (highV + lowV) / 2
    puts "Date: " + dateV + ", Close: " + closeV + "High: " + highV + "Low: " + lowV + "Midpoint: " + middle

    #Here is where we calculate the prediction
    net = closeV - previous[previous.length-1]
    point = closeV - lowV
```

```
    if point > middle
      point = point - middle
      pchange = (point / middle) * 1
          if net < 0
                  net = -1 * net
          end
    end
    if point < middle
      pchange = (1 - (point / middle)) * 1
          if net < 0
                  net = -1 * net
          end
    end
    if point = middle
      pchange = 0
    end

   predV = (pchange*net) + closeV

   if(allValsNumeric?(previous))
     puts " Close Before that: " + previous[previous.length-1].to_s
     puts "Prediction: " + predV
   end

   previous= update(previous, Float(closeV))
   puts previous
  end
 end

#This 'helper' method updates the array storing the past n close values in a moving average
#It kicks the most outdated value out of the array, shifts all values down one index, and fills in the
# most recent value, val
def update(previous, val)
```

```
  i=1
  temp = Array.new(previous.length)
  while(i<temp.length)
    temp[i-1]=previous[i]
    i = i+1
  end
  temp[temp.length-1]=val
  return temp
end

#This 'helper' method calculates the average of the values of an array
#This method should never be called unless allValsNumeric? returns true
def average(previous)
  total = 0
  previous.each do |value|
    total = Float(total + value)
  end
  return total/previous.length
end

#This 'helper' method calculates the WEIGHTED average of the values of an array
#For example, in an array of 4, the most 'recent' value will have a weight of 4, the second most recent weight 3, etc.
#This method should never be called unless allValsNumeric? returns true
def weightedAverage(previous)
  total = 0
  weight = previous.length
  previous.each do |value|
    total = Float(total + weight*value)
    weight = weight - 1
  end
  return total/(previous.length * 2)
end

#This 'helper' checks if all the values in an array are numeric
```

```
  def allValsNumeric?(previous)
    previous.each do |value|
      if !value.kind_of?(Numeric)
        return false
      end
    end
    return true
  end
end

#Test Code
test1 = Predictor.new
test1.simpleMovingAvg(2)
```

Genetic Algorithm:

```
package portfoliooptimization;
import java.util.*;
import java.io.*;
/**
 *
 * @author Ivan
 */
public class Optimizer {
    Random rand = new Random();
    private double[] expectedValues;
    private double[][] covarMatrix;
    private int numAssets;
    private double minProportion;
    private double maxProportion;
    /**
     * @param args the command line arguments
     */
    public Optimizer(double[] expectedValues, double[][] covMatrix,
                    int numAssets, double minProportion, double maxProportion){
        this.expectedValues = expectedValues;
        this.covarMatrix = covMatrix;
        this.numAssets = numAssets;
        this.minProportion = minProportion;
        this.maxProportion = maxProportion;
    }

    public Solution[] GAheuristic(){
        Solution[] map = new Solution[16];
        int LAMBDA = 15;
        for(int l = 0; l <= LAMBDA; l++){
            double lambda = ((double)l)/((double)LAMBDA);
            Solution[] d = new Solution[25];
            for(int i = 0; i < 25; i++)
                d[i] = new Solution(expectedValues, covarMatrix, numAssets, lambda);

            for(int iter = 0; iter < 3000; iter++){
                Arrays.sort(d);
                //System.out.println("Just before bin tourn");
```

```
            Solution child = binaryTournament(d, lambda);
            //System.out.println("Just after");
            if(d[24].compareTo(child) > 0)
                d[24] = child;
        }
        Arrays.sort(d);

        //map.put(lambda, d[0]);
        map[l]=d[0];
    }
    return map;
}

public Solution binaryTournament(Solution[] array, double lambda){
    //System.out.println("starting binary tournament");
    HashSet<Integer> set = new HashSet<Integer>();
    while(set.size() < 4)
        set.add(rand.nextInt(array.length));
    int[] parents = new int[set.size()];
    int counter = 0;
    for(int i : set)
        parents[counter++] = i;

    Solution parent1;
    Solution parent2;

    if(array[parents[0]].compareTo(array[parents[1]]) > 0 )
        parent1 = array[parents[0]];
    else
        parent1 = array[parents[1]];

    if(array[parents[2]].compareTo(array[parents[3]]) > 0 )
        parent2 = array[parents[2]];
    else
        parent2 = array[parents[3]];

  return crossGenes(parent1, parent2, lambda);

}

public Solution crossGenes(Solution s1, Solution s2, double lambda){
    //System.out.println("starting crossGenes");
    double[] parent1 = s1.getWeights();
    double[] parent2 = s2.getWeights();
    double[] childArray = new double[parent1.length];

    for(int i = 0; i < parent1.length; i++){
        if(parent1[i] > 0.0001 && parent2[i] > 0.0001){        /// asset present in both parents
            if(rand.nextBoolean())
                childArray[i] = parent1[i];
            else
                childArray[i] = parent2[i];

        }else if(parent1[i] > 0.0001 || parent2[i] > 0.0001){  /// present in just one
            if(rand.nextBoolean()){
                if(rand.nextBoolean())
                    childArray[i] = parent1[i];
                else
                    childArray[i] = parent2[i];
            }else{
                childArray[i] = 0.0;
            }
        }else{              /// not present
             childArray[i] = 0.0;
        }
    }
    //System.out.println("starting mutate");
    mutate(childArray);
    reWeight(childArray);
    //System.out.println("returning new solution");
```

```
    return new Solution(childArray, this.expectedValues, this.covarMatrix, this.numAssets, lambda);
}

public void reWeight(double[] weights){
    double sum = 0.0;
    for(double c_i : weights)
        sum += c_i;

    double multiplier = 1.0/sum;
    for(int i = 0; i < weights.length ; i++)
        weights[i] = weights[i]* multiplier;

}

public void mutate(double[] array){
    int counter = 0;
    for(int i = 0; i < array.length; i++){
        if(array[i] > 0.0001){
            if(rand.nextBoolean())
                array[i] = array[i]*1.1;
            else
                array[i] = array[i]*0.9;
            counter++;
        }
    }

    int newAssets = this.numAssets - counter;

    if(newAssets > 0){
        for(int j = 0; j < newAssets; j++){
            int newIndex = 0;
            while(array[newIndex] > 0.0001){            /// random asset
                newIndex = rand.nextInt(array.length);
                //System.out.println("while loop");
                //System.out.println(Arrays.toString(array));
            }
            array[newIndex] = rand.nextDouble();
        }
    }else if(newAssets < 0){            /// decrease the assets until desired number
        while(newAssets < 0){
            int index = rand.nextInt(array.length);
            if(array[index] > 0.0001){
                array[index] = 0.0;
                newAssets++;
            }

        }

    }

}

public double[] geneticOptimizer(){
    return expectedValues;
}
```