University of Modena and Reggio Emilia

DEPARTEMENT OF PHYSIC, COMPUTER AND MATHEMATICAL SCIENCES

COMPUTER SCIENCES MASTER DEGREE

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Development of interpretable machine learning algorithms for multi-objective optimization of financial portfolios based on market anomalies.

Candidate:
Umberto Paroli

Supervisor:
Prof.essa Federica Mandreoli

Co-Supervisor:
Dott.essa Veronica Guidetti

Anno Accademico 2025–2026

Acknowledgment

I would like to thank Professor F.Mandreoli and Phd. V.Guidetti for their constant help, despite their many commitments, and for guiding me through the final phase of my master's degree.
I dedicate this thesis to my family for all the support and allowing to focus myself completely to my academic career.
To my friends who accompanied me over countless hours of study and travel on public transport.
And finally, to the numerous students who shared their help (and notes) over the years.
Without you, I couldn't have achieved any of this.

Contents

1 Introduction

Factor investing is a financial framework that aims to build a portfolio formation strategy based on numerical anomalies that describe assets and the market [15]. Over the years, numerous studies have developed frameworks to combine multiple anomalies, some involving machine learning and deep learning.

But modern portfolio theory assumes that investors are risk-averse, meaning that they also account for minimizing the risk of financial loss and not just maximizing returns [17]. For such investors, a deep learning model, despite being powerful, is a black box for which one can't easily tell why, from a given input, a specific output is obtained; and therefore not suited for a scenario where financial risk is involved.

Because of the risks in financial investing, there is an interest in exploring explainable machine learning approaches where the logic behind the decisions of a model is clear and where a human expert can directly understand the relationships between input and output.

In this work, we apply symbolic regression, an evolutionary algorithm, to factor investing to produce a ranking score for market assets based on a universe of anomalies. Symbolic Regression is a genetic programming technique, meaning it evolves a population of mathematical equations that we use to compute a ranking score that can be directly understood. The ranking function obtained is explainable by design and is easier to evaluate its benefits and risks.

We also implemented a Bayesian Optimization process to optimize the numerical variables of the equations obtained with Symbolic Regression to achieve better returns and stability.

We applied stationary bootstrap to the fitness function of the algorithm to control the overfitting of individuals. In addition, we used a Superior Predictive Ability Test (SPA test) to organize the final population of the evolutionary process.

In sections 2 and 3, we give an overview of Symbolic Regression and Portfolio Theory.. Section 4 presents the model we proposed, its challenges and the solutions we applied to produce a formation strategy with Symbolic Regression. In section 5, we delve into more details about the Gaussian Process applied to a multi-objective optimization problem (MOO problem) given by multiple fitness functions. We then proceed, in sections 6 and 7, to report the details of the implementation of the model and the data for training and testing. Finally, in Section 8, we report the results achieved.

This thesis showed how the combination of Symbolic Regression and Bayesian Optimization allows to generalize over market anomalies to produce multiple explainable functions. Results also suggest that it's possible to reduce the complexity of the equation while preserving the quality of the solutions, leaving space to further research.

2 Symbolic Regression

2.1 Overview

Many modern machine learning architectures, such as neural networks, have high generalization capabilities and can learn to extract new features and approximate complex non-linear functions. Although being powerful tools for many tasks, modern architectures, especially in recent years, have become exponentially more complex, involving millions or billions of parameters [8] [1]. As a consequence, explaining why a model produces a specific output (explainability), rather than how it was computed (interpretability), has become a difficult task [8].

Many attempts were made to gain insight into how a model produces an output value: for example, by interpreting the kernel weights of a convolutional neural network or by equipping a deep learning model with explainability modules to obtain a simple generalization of its sections. But for many scenarios where a certain degree of risk is involved, black-box models, whose results can only partially be explained, may not meet the safety and/or security requirements [8].

For this reason, there is an interest in exploring architectures that have explainability as an explicit design point, such as Symbolic Regression. Symbolic regression (SR) is a set of machine learning techniques that belongs to genetic programming, meaning it evolves a population of programs. More specifically, SR models mathematical expressions with the goal of learning the underlying relationships or laws from a dataset of observations. An SR problem is usually defined as a standard risk minimization in which the goal is to minimize a loss function over a dataset [8].

The advantage of using an expression to predict a numerical value or a class label is that one can directly understand how each feature of a sample contributes in which way to produce a specific output [8]. The relationship between input and output is perfectly explainable. For example, if we consider a formula such as $2x_1/x_2$, we can see how the result is directly correlated to the value of the feature, x_1, while it is inversely correlated to the value of the feature x_2.

An SR problem requires defining a library of tokens that can be used to "assemble" an expression. They can be basic operands (+, -, $exp()$, ...) or more complex constructs added to the library as a form of prior knowledge injection [8] [12]. The size of the library exponentially increases the costs of training: the more symbols are included, the larger the search space in which to find an optimal solution for the problem.

2.2 Tree-Based Symbolic Regression

There are different ways to implement a Symbolic Regression algorithm, and many details depend on how one represents the members of the population. Some algorithms use the entries of the library as variables of a regression problem in which a linear combination of each token is optimized. Another possible approach is to build a feed-forward neural network in which each neuron implements an item from the library instead of a linear transformation. The population is composed of many different networks and evolves through a genetic algorithm with a fitness function that accounts for both a loss score over the dataset and a complexity term to maintain simple individuals [8].

For this project we chose for tree-based symbolic regression: an expression is modeled as a binary tree where the leaf nodes are the attributes of the dataset and the internal

nodes the operands of the library [8].

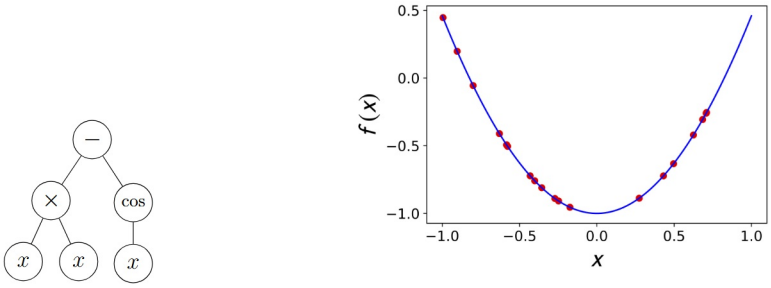


Figure 1: A simple expression represented as a tree and the corresponding plot [8].

To compute an expression, one can recursively combine the value of two child nodes using the operand in the parent node, starting from the leaves up to the root of the tree, obtaining the output of the equation. This result is used as a fitness to evaluate each equation and evolve a population of solutions over multiple generations [8].

The fitness score, which measures the reproductive success of an individual, is a concept taken from biology that we use to evaluate the effectiveness of a candidate solution in the population to solve a problem. It allows us to select the most promising elements and to combine them to improve the "goodness" of the population until we obtain a set of solutions which satisfy us enough for the task we aim to optimize.

A genetic algorithm imitates the natural selection process in which individuals who adapt better to a specific environment have more chances to reproduce and pass on their genes to their offspring. Over the course of multiple generations, the population has adapted to the environment and survives better to its challenges. In this scenario, the environment is given by the Loss function of the problem.

The genetic algorithm starts with an initial population of individuals, which can be randomly generated or hand-crafted to inject a form of domain knowledge or bias into the problem [12]. The algorithm then proceeds iteratively to evaluate the members of the population and compute a fitness score for each of them. The best performing elements are then selected while the others are discarded. Then, new individuals are generated by combining and/or mutating the survivors of the current iteration to replace the discarded expressions. The new population is then passed on to the next iteration to repeat the process. The algorithm iterates until it converges to one or multiple solutions, which describe the dataset well enough, or until a timeout is reached [1].

The tree structure of the individual allows to implement mutation and crossover (combining different individuals) directly over the tree representation. For instance, a mutation can add, remove, or change the content of a node, while two expressions can be combined by selecting a sub-tree from each parent and then switching them.

When computing the fitness score, individuals are usually divided into tournaments. By choosing the tournament size, one selects the number of participants competing against each other and influences the convergence speed of the genetic algorithm.

For the same population, a smaller tournament size means a higher number of “competitions”. Therefore, having fewer individuals competing against each other, suboptimal candidates with a lower fitness score have a higher chance of passing to the next iteration.

On the other hand, a larger tournament size implies greater competition. As a consequence, only the candidates with the highest fitness score among all of the population will survive the current iteration, and the optimization process will converge faster [8].

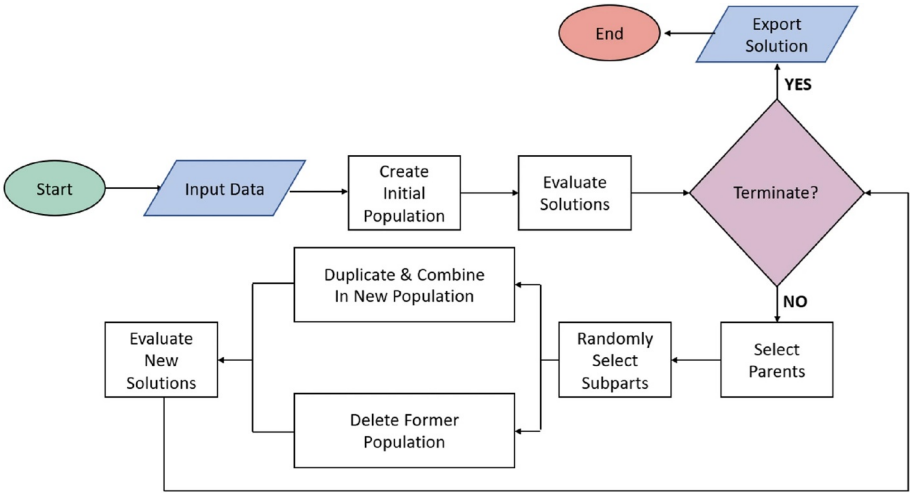


Figure 2: Genetic programming based symbolic regression flowchart[1].

2.3 Overfitting and Complexity

Symbolic Regression, like the majority machine learning algorithms, is prone to overfitting that it presents in the form of overly complex equations. If the genetic algorithm only accounts for a reconstruction error while computing the fitness of the individuals, their tree structure is left free to grow in depth without control. This led to solutions that perform well on the dataset but poorly with new samples, having lost generalization capability.

In general, a solution of too low complexity might indicate poor performance, while a solution of too high complexity could be prone to overfitting [1].

To mitigate the problem, one can include a complexity penalty in the fitness score. This forces the algorithm to find a balance between the reconstruction error and the generalization capability of the solution [1].

When handling tree-based symbolic regression, one can consider a measure correlated with the number of terms present in the tree. Previous works defined this expression as parsimony: the inverse of the number of terms in the tree-expression [12]. In this project, we used a probabilistic approach explained in section 6.

A good practice is to combine a complexity penalty with other regularization techniques, such as early stopping, or by setting hyperparameters of the optimization process to privilege the exploration of the solution space over convergence. A lower tournament size slows the convergence process and allows for the evaluation of many more different solutions.

3 Portfolio Theory

3.1 Portfolio Optimization

Modern portfolio theory is a mathematical framework for assembling a portfolio from a universe of assets to maximize the expected return for a given level of risk. The key idea of this framework is that owning a diversified set of assets reduces financial risk rather than holding only one asset type [17].

Maximizing the returns and lowering the variability of a portfolio are two objectives which can be strictly related. If one wants to raise the returns of a portfolio, it also has to accept an increase in its variability. On the other hand, modern theory assumes that investors are risk-averse: given two portfolios with the same returns but different variability, the one with a lower variance will be more attractive [17]. As a consequence, a wide range of research and real-world applications attempt to build a strategy which uses a set of rules that balance financial risk and rewards.

Portfolio optimization is a field that has been addressed over the years by countless studies, from different perspectives, accounting for different factors, and focusing on different tradeoffs and objectives.

3.2 Factor Investing and Relevance

Factor investing is an investment approach that exploits measurable characteristics of the market or assets to explain differences in risk and returns [15]. Since the development of the Capital Asset Pricing Model (CAPM), one of the earliest single-factor models, many different factors have been proposed and used to build portfolio formation strategies.

Over the years, many recent studies, given an ever-expanding universe of factors and market anomalies, have tried to combine multiple single-factor approaches to create a strategy that combines the strengths and mitigates the weaknesses of their individual components [14][9].

It was shown how, using a Multi-Criteria Decision Model (MCDM) to combine different single-criterion models, one can obtain better performance than the single-factor approaches. Many experiments have also shown how different priorities in portfolio optimization (higher returns instead of more stable performances and vice versa) lead to different compositions of single-factor criteria obtained from the MCDM [9].

Among the wide range of combination methods available, the study of reference adopted only four of the simplest and most popular [9]. They also choose not to apply any significant refinement process to the results for two reasons. First, because such processes tend to cause a loss of generality by being designed for specific scenarios, obtaining worse performances in other cases. Second, because they tend to add too much complexity to the resulting model, making it difficult to explain the obtained result (loss in explainability), which many investors prefer to avoid [9]. In other words, a simpler and more explainable model is usually preferred when dealing with financial risks compared to a more complex one, even if it can lead to better performance.

Another problem MCDM can face is that the universe of market factors changes constantly. Over the years, multiple studies have discovered new factors and market anomalies and have proposed many portfolio formation strategies that account for them. It was also shown that such anomalies often aren't continuously relevant: a small subset of anomalies may be enough to summarize the whole population at a given moment, and

the composition of such a subset changes over time [9].

As a consequence, the best-performing combination of single-criterion models becomes "obsolete" when enough changes in the market diminish the relevance of the factors on which they are based.

From this perspective, we can extract an additional goal for this project: the construction of a model that can achieve explainable solutions to an optimization problem and reflect the relevance changes in the universe of factors and anomalies.

4 Symbolic Regression applied to Portfolio Theory

4.1 Symbolic Regression and Portfolio Optimization

The main goal of this project is to apply a Symbolic Regression Model to a selection of market anomalies to produce a "super-factor"" that can be used to rank anomalies to compose a portfolio optimization strategy.

This model could be used to produce a clear and explainable attribute [8][1] for each asset considered for the portfolio-formation task over a given time period. A mathematical expression produced by an SR model can be directly interpreted to understand which factors are selected by the fitting process and considered relevant for the objective modeled by an appropriate fitness function [8][1]. A human expert can evaluate the results, understand how anomalies are aggregated, the corresponding tradeoffs, and decide on many possible solutions which better suit its vision of the market.

Since the relevance of anomalies changes over time [14], a solution produced by an SR model would eventually become obsolete as the market changes. On a real world application, the proposed model would repeat the optimization process over time, updating the dataset as new data are obtained from observations and new factors get discovered.

4.2 Overfitting and Stationary Bootstrap

As anticipated, Symbolic Regression, like the majority of machine learning (ML) models, is prone to overfitting. In the context of time series and forecasting problems, such as portfolio optimization, overfitting corresponds to learning to perfectly exploit the fluctuations and outliers of the training set to boost returns[2]. As a consequence, the model loses generalization capability and performs poorly on out-of-sample data.

Overfitting isn't just a problem which can occur with excessive or unregulated training of an ML model. Architectural choices, data selection, goal definition, training strategy, and overall evaluation, if done only to obtain higher numbers (Data Snooping Bias and Winner's Curse), lead to results which perform poorly in real-world scenarios despite excellent performance during development [2].

A method to mitigate these problems, among many possible strategies, is bootstrapping. Bootstrap is a practice used in portfolio optimization models to better reconstruct the distribution behind the dataset and to compare and evaluate alternative models [7]. The standard stationary bootstrap divides the global period of observations into multiple blocks of observations. The model is then applied to the block list to estimate one or more features of the data distribution [11][7].

The current project applies stationary bootstrap, already implemented by an eastern library, proposed by Politis and Romano [11].

The list of time blocks used in the stationary bootstrap is obtained as follows: given a list of ordered observations

$$B_{i,b} = \{x_i, ...x_{i+b-1}\}$$

One wants to generate N different time blocks. The beginning of a block is obtained by sampling a list of ids $L = \{L_1, \ldots, L_N\}$ following the geometric distribution, such that $L_i = m$ with probability $(1-p)^{m-1}$ for $m = 1, 2, \ldots$ Independent of L, a list of ids $I = \{I_1, \ldots, I_N\}$ is obtained by sampling from a uniform discrete distribution on $\{1, \ldots, N\}$.

The list of time blocks $\{B_{I_1,L_1}, ..., B_{I_N,L_N}\}$ is obtained so that

$$B_{I_k,L_k} = \{x_{I_k}, x_{I_k+1}, ...x_{I_k+L_{k-1}}\}$$

The block generation process is stopped when N blocks are achieved. Some control mechanism can be applied to avoid time blocks being too small or of excessive length.

In addition to stationary bootstrap, we proposed an alternative solution to reduce overfitting. The dataset is divided into a set of observation blocks of equal length, with eventually one block of a smaller size if the total number of observations isn't perfectly divisible by the block length. The split is performed before the beginning of the training process, and the fitness function is computed every time over a different block from the list. This alternative approach was designed prioritizing simplicity to decrease the computational costs of generating block lists, and provides a comparison for the performance obtained with stationary bootstrap.

As an additional note, to reduce overfitting of a genetic algorithm, a trivial addition is to lower the effort in exploring the solution space by reducing the population size and the maximum number of generations for the fitting process.

4.3 Long-Short Portfolio with Equal Weights

To construct anomaly-sorted portfolios, we rebalance on predetermined reallocation dates and sort stocks into deciles based on the cross-sectional distribution of each characteristic.[1] The first (tenth) decile contains stocks with the lowest (highest) characteristic values. We then form a long-short (high-low) portfolio by going long the decile with the *best* characteristic values (expected to earn higher returns) and short the decile with the *worst* values (expected to earn lower returns). For fair comparison between aggregated-signal portfolios, we exclude stocks with missing anomaly values at each reallocation date so that all strategies are evaluated on a consistent set of tradable stocks.

4.4 Computational Costs

Tree-based Symbolic Regression suffers from an exponential increase in resource consumption depending on the library size and the complexity of the population elements. At each iteration, to explore the search space, new individuals have to be generated and evaluated to compute their fitness score [1].

By combining this knowledge with portfolio theory [17], to compute the fitness score of an individual, it is necessary to:

1. Apply the mathematical expression to the dataset to produce the new super-factor for each asset.

2. Convert the super factor to a ranking score (via normalization or a more complex transformation).

3. Compute a set of weights derived from the super score for a portfolio formation strategy.

[1]Following standard practice, we define the kth portfolio at time t as $P_{k,t} = \{i \mid B_{k-1,t} \leq F_{i,t} \leq B_{k,t}\}$, where $F_{i,t}$ is the sorting variable and $B_{k,t}$ are the decile breakpoints. To prevent empty portfolios when breakpoints coincide, we avoid strict inequalities; consequently, some stocks may appear in multiple portfolios, which does not affect the analysis.

4. Compute the returns (or an alternative metric) of the strategy.

5. Compute a fitness score derived from the average returns (or an alternative metric) of the portfolio.

The process has to be repeated for every element of the population at each iteration of a genetic algorithm. Summing this with additional operations, such as numerical constant optimizations and multi-objective optimization of multiple fitness scores leads to a rapid explosion in computational resources consumption.

Computing an optimized set of weights for a given portfolio composition is a problem studied through different frameworks and strategies, which can lead to complex solutions whose costs in this scenario would be exponentially amplified. To control the costs of computing the fitness scores, it was chosen to implement an equal-weight long-short portfolio.

The approach, while not optimal compared to more refined methods, allows for a diversified, fully invested portfolio that provides a simple scenario to understand and less expensive to compute [16].

These implementation choices leave space for future improvement of the proposed model by adding different investment strategies, comparing the tradeoff, and eventually re-inventing the best approaches.

4.5 Individual Evaluation

The project follows the structure of an Evolutionary Algorithm: a starting population is initialized as the input of the evolutionary process, as shown in Figure 3. At each iteration, individuals with the best fitness are selected, combined, and/or mutated to generate new solutions.

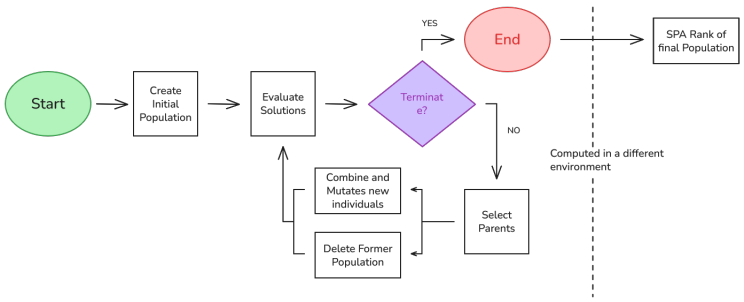


Figure 3: Scheme of our evolutionary algorithm which involves an additional ranking step.

Each new element is also evaluated, and the best individuals are chosen to form the population for the next iteration, while the worst are discarded. The process repeats until the population converges to a satisfying solution or enough time has passed.

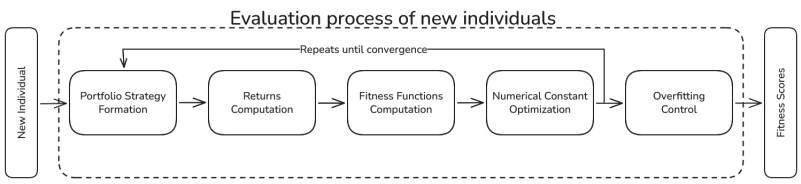


Figure 4: Flowchart of the evaluation of a new individuals

To evaluate a new individual, the algorithm performs the following steps:

1. Formation of a portfolio strategy.
2. Returns computation.
3. Calculation of fitness functions.
4. Optimization of numerical constants.
5. Overfitting control.

4.5.1 Formation of a Portfolio Strategy

To gather the portfolio strategy, the algorithm applies the mathematical equation computed by Symbolic Regression to the entire dataset, combining the anomalies to produce a score. Then, for each date, with a monthly frequency, the assets are sorted; the top decile and the worst decile are selected, with a long and a short position, respectively.

4.5.2 Returns Computation

To obtain the returns associated with a strategy, the program has to compute the weights for the selected assets. For the long position, the assets acquired, 1 is divided by the total number of assets falling in the first decile. For the short position, the assets sold, the method is the same, but the weights are multiplied by -1, reflecting the operation. The remaining assets, not selected for the strategy, have a weight of 0.

Then, for each time step in the dataset, the risk-free returns of each asset are multiplied by its corresponding weight at that date.

4.5.3 Calculation of Fitness Functions

From the returns, we derived the two fitness functions: the average returns and the Sharpe ratio. The average returns are given, trivially, by the average of the returns over a given period of time.

The Sharpe ratio is given by the annualized (multiplied by the square root of 12) mean of the risk-free returns divided by its standard deviation.

$$Sp = \frac{mean(RiskFreeReturns)}{\sigma(RiskFreeReturns)} * \sqrt{12}$$

4.5.4 Optimization of Numerical Constants

The optimization of the variables in the equations generated by the symbolic regression is performed by approximating the optimal solution of a multi-objective optimization problem given by the fitness functions. The algorithm exploits the hypervolumes method to approximate the optimal Pareto frontier.

To evaluate a point in the solution space, it is necessary to reassemble the portfolio and compute the associated returns. To deal with the costs and non-derivability of the process, we applied a Gaussian process to reduce the number of evaluations required to solve the problems. The hypervolumes method and the Gaussian process are later described in section 5.

4.5.5 Overfitting Control

After optimizing the numerical constants in the equations, to control the overfitting, the final scores are obtained by applying the fitness function with one of the two methods to control the overfitting.

For the random-subperiods version of the problem, one interval is randomly selected, and the fitness functions are computed by accounting only for the dates falling in it.

For the stationary bootstrap method, proposed in [11], the fitness functions are applied to each block, and the scores are aggregated. We applied the mean for simplicity, but more sophisticated approaches are possible. Regardless of the method, the final score for each fitness function is returned and saved for the individual.

4.6 Final population model selection

The output of the algorithm is a population of mathematical equations, ideally all with a high fitness score, from which we can extract different portfolio formation strategies. As they are presented, the only way to compare them is through their fitness scores, which can be used to rank the final population.

If multiple fitness functions are used, we can select just one of those or produce a combined score by following some sort of criterion (For example, weighting more the variability over pure returns). But in general, it is desirable to apply a more refined method to evaluate a portfolio strategy.

As shown in Figure 3, to help evaluate these elements, a tournament was implemented where each solution competes against all the others. The process creates a ranking score that divides the population into sorted class, helping to organize them depending on their performance over the test set.

The tournament is based on a Test of Superior Predictive Ability. The SPA test is a test to evaluate the forecasting performance of multiple models. For a given benchmark, the method proposes the null hypothesis that none of the input models is better than the benchmark. If the null hypothesis is discarded for one or more models, we have recognized their forecasting ability. The SPA was designed to evaluate reconstruction errors; therefore, it promotes lower values of a loss function [6].

The procedure was studied to mitigate data snooping bias and be less sensitive to outliers and irrelevant alternatives. The method exploits bootstrap to compare models over multiple time boxes, avoiding premature overfitting [6].

The SPA test was adapted to the current project to produce a ranking for the individuals of the final population. For each equation, the average returns (or the Sharpe Ratios) are computed over the test set, and the corresponding time series substitutes the loss function in the method, forming a portfolio strategy.

The SPA test is applied iteratively, comparing one portfolio strategy as the benchmark with all the others. At each iteration, one strategy is evaluated against the others, and those that don't negate the null hypothesis are discarded and inserted into a list to create a rank. The procedure will be covered in more detail in the implementation section.

As a final note, since the SPA test was designed with loss functions in mind [6], it is necessary to multiply the metrics that one wants to maximize by -1, such as average returns or Sharpe Ratio.

For the sake of reproducibility and simplicity, the SPA tournament is performed in a second moment after the evolutionary algorithm is completed, reconstructing the performance directly from the final populations and with a fixed seed. The process, being

much less demanding of Symbolic Regression in terms of computational resources, can be executed in a different environment.

5 Gaussian Process

5.1 Symbolic Regression and Learning to Rank

A ranking problem is defined over a vector space for the items (objects to rank) and a vector space for the queries, and the goal is to find a rank function which produces a score for each query-item couple. In this scenario, the items are the financial assets at a given time, while the query is given by the fitness functions whose models the performance of a portfolio strategy. Each ranking score computed by the Symbolic Regression has to sort the assets in a way that results optimal for the long-short equal weights strategy.

A Learning to Rank approach solves the problem using machine learning to predict the score using both anomaly values and, possibly, one or more numerical constants [10]. The proposed project exploits Symbolic Regression to rank a set of financial assets (items), where the query is defined by fitness functions which report the "goodness" of the selected items in terms of average returns and the Sharpe Ratio.

The Symbolic Regression algorithm produces a ranking score by synthesizing mathematical expressions from data. The ranking obtained for the assets is used to compute a solution for a multi-objective optimization (MOO) problem where each objective function corresponds to a fitness score for each individual.

Financial anomalies, entries for the dataset, are fixed at each date and can't change; but each expression may also contain one or more numerical constants (-1.9, 0.0287, 0.14, etc. . .). The initial value of such constants is sampled from a given distribution chosen as a hyperparameter of the process.

Despite the misleading name, a numerical constant can undergo an optimization process to compute a better solution for the MOO problem. But there is a catch: optimizing the objective functions requires sampling multiple times as the parameters of the problem are updated, and in this scenario, fitness scores are expensive to compute.

For each new individual in the population, one must apply the corresponding mathematical expression to the dataset, perform the ranking, select the assets for the portfolio strategy, compute the corresponding weights set, and finally compute the fitness functions. The process has to be repeated at each step of the optimization process.

Furthermore, in finance, most anomaly values are released on a monthly or annual basis; therefore, one does not have many data points to support a precise optimization for an MOO problem.

It is necessary to find a strategy to efficiently perform the MOO process, which can deal with the scarcity of data points.

5.2 Bayesian Optimization

Bayesian optimization is a powerful strategy which helps optimize expensive functions by reducing the number of samples required by the process. Given a sample from a dataset and the corresponding objective function f that returns an outcome, we can sequentially collect information composed by a sample and its function to construct a set D, which can be used to define the posterior probability $P(f|D)$ [4].

Given our prior beliefs about the problem, we can rewrite it as the product between the likelihood $P(D|f)$ and the prior probability $p(f)$ [4]:

$$P(f|D) = P(D|f) * P(f) \tag{1}$$

or

$$posterior = likelihood * prior \tag{2}$$

Bayesian optimization involves modeling a real function, expensive to compute, with a surrogate function which can be evaluated much more efficiently. The surrogate function will approximate the likelihood and indirectly the nature of the problem we will optimize. An acquisition function is used to guide the sampling process to select a candidate point and optimize the surrogate function. An iterative process, involving sampling and optimizing, is repeated until the maximum of the surrogate function is found (or a close enough solution) [4].

Such a problem is much more efficient to solve because it reduces the number of evaluations of the expensive real function while reaching the real solution.

If the surrogate function is treated as a joint probability distribution over the variables, assuming a multivariate Gaussian distribution, the method is called a Gaussian Process and allows the exploitation of the probability information from the model for the acquisition function [4].

The method allows to select the number of real points to sample during the optimization, allowing to set a tradeoff between costs and accuracy of the solution found [4].

5.3 Gaussian Process and Multi-Objective Optimization

In multi-objective optimization (MOO), the Pareto Front is a set of all the Pareto Efficient solutions, which means that all the solutions in that set are equally good overall, or more formally, they aren't worse than any other solution in the set [18].

Since in this project the Gaussian Process is used for an MOO problem, the optimization process can produce multiple alternative solutions. It is necessary to apply a method that evaluates and ranks solutions to select the best individual from a Pareto Frontier.

The Hypervolume Method is a strategy that, for each approximation set in the objective space, introduces a reference point to compute a hypervolume for each solution. Qualitatively, the wider the region covered by the hypervolume, the closer the approximation of the real Pareto Front and therefore a real optimal solution [3]. The largest volume is the one selected. The method was chosen for its simplicity and low costs.

Summing up: the final algorithm for a given individual in the population will compute a ranking score that can be optimized for the MOO problem defined by two fitness functions (average returns and Sharpe ratio) through a Gaussian Process, which accounts for the proposed approximation of the real Pareto Front of the problem. It was chosen to perform the Gaussian Process with 50 points, a compromise value to further reduce the costs of the optimization process.

6 Implementation

6.1 Starting Library

The starting point for the current project was a Python library developed for Tree-based Symbolic Regression for multiple Machine Learning scenarios (Classification, Regression, Distribution approximation, etc.). The project was developed with a modular structure, where each component of the algorithm is managed as an object and can be extended by implementing new classes. The fitting process automatically performs the training and validation process, managing early stopping and convergence checks.

A logging system allows for setting the verbosity level of the program and saves the final results as multiple files as a text file and a CSV reporting the final population members and a screenshot of the instantiated problem.

The project was already extended to manage multi-objective optimization and Bayesian optimization to implement a Gaussian Process described in the previous section for previous studies.

6.2 Fitness Functions

The library manages a fitness function as a class, which defines how to apply mathematical expressions for a member of the population and over which data. The main method also manages the optimization process, and it's designed to elaborate only once for new individuals to save resources.

For the current project, two different fitness functions were proposed: one computing the average returns with a monthly reallocation rate, and the second the Sharpe ratio of the same strategy.

Each fitness function has two different implementations: the first version provides a lightweight method to reduce overfitting, inspired by bootstrapping, but simpler and faster to compute. The training set is divided into equal length time blocks, where the individual duration is a hyperparameter. When the fitness function is applied, a random block is sampled from the list, returning the score over that subperiod.

The second version has the same objective, but through a more refined (and expensive) bootstrap process implemented by the library Auto Regressive Conditional Heteroskedasticity (ARCH), an open-source toolbox containing many algorithms and statistical analyses for time series, all tested and well documented [13]. The ARCH library was included in the project dependencies.

As an additional note, the original library was designed thinking of the fitness score mainly as a reconstruction error to minimize, while average returns and Sharpe ratio are values one wants to maximize. The problem is solved trivially by multiplying the output of the fitness functions by -1 during the evaluation process.

The optimization process is parallel by design: at each iteration, new individuals can be evaluated in parallel. To avoid conflict in data access, each individual allocates a copy of the segment of the dataset needed to compute the fitness functions, which is then destroyed after the procedure is complete. The original library is designed to compute the scores and optimize numerical constants only for new individuals, avoiding the process when it isn't necessary.

6.3 Final Population Ranking

To evaluate the final population, the SPA test, implemented by the ARCH library [13], was adapted to rank the strategy associated with each individual. As explained in the evaluation section, the SPA test proposes the null hypothesis that a model isn't better than a benchmark. If the hypothesis is negated, then the proposed model is a better approximation of the ground benchmark.

In the current project, the SPA test is used to iteratively compare each solution to create a ranking. Individuals are placed in an array where the elements in a given position aren't better than the ones in the next entry.

Formally:

Algorithm 1 SPA Ranking

$S \leftarrow \{s_1, s_2, ..., s_n\}$
$L \leftarrow []$
while $S \neq \{\}$ **do**
  $s \leftarrow S[0]$ ▷ Select one solution as benchmark
  $candidates \leftarrow S[0:]$ ▷ Select all solutions except s
  $bettermodels \leftarrow SPAtest(s, c)$ ▷ Returns a set
  if $bettermodels \neq \{\}$ **then**
    push(L,$candidates - bettermodels$) ▷ s is included because not strictly better of itself
    $S \leftarrow bettermodels$
  end if
  if $bettermodels = \{\}$ **then**
    push(L, $S - s$)
    push(L, s)
    $S \leftarrow \{\}$
  end if
end while

At the end of the execution, L will have the better strategies, according to the SPA test, in the first positions; and the worst strategies will fall in the last elements of the list. The algorithm terminates because at each iteration at least one element is removed from S (either the models not better than the benchmark, or just the benchmark itself), and eventually the set will be empty. The actual implementation of the algorithm slightly diverges from this structure to meet the type requirements of the SPA test, which takes only vector and matrices as input and return indices.

6.4 Execution Environment

Symbolic Regression has a high demand for computational resources that is increased by the Gaussian process to optimize the fitness. To sustain the costs, the execution of the project was divided into two parts: the fitting process was split into two scripts, one for each fitness type (random-subperiods and bootstrap), and launched on a cluster of machines sharing a large amount of resources.

The nature of SR allows for high-level parallelism during the exploration of the solution spaces by evaluating new individuals at the same time. To avoid data access conflicts when computing the fitness scores, each individual copies the portion of the dataset needed for

itself and destroys it at the end of the operation. This choice raises the memory consumed by the algorithm, but avoids problems of consistency and dependency.

The output of such a process, composed of the final population and the associated fitness score, is transferred to evaluate the results through the SPA test, which is a much less expensive task feasible for a single computer. The logging mechanism of the library was extended to include a string version of the mathematical expression to simplify the reconstruction of the population.

6.5 Execution Hyperparameters

The SR problem was instantiated with a restricted library consisting of:

- Addition
- Subtraction
- Product
- Division
- Minimum operator
- Maximum operator

The limited set was chosen to simplify the solution space, reduce the complexity of the final solution and computational costs. Even when disposing of a limited set of items, previous studies proven that SR can converge to an approximation of a complex function, such as the Taylor expansions of sine and cosine [12].

The population was initialized with 100 individuals, with a tournament size of 3, which defines low competition, allowing for the survival of a wide range of alternative solutions.

Numerical constants are initially sampled from an interval of [0.1] following a uniform distribution. The optimization of constants applies a Gaussian Process, which samples 50 different points to approximate the real function: a compromise value which sacrifices accuracy for a lighter process.

For each type of fitness function, random-subperiods, and bootstrap, the Sharpe ratio is computed with an annual risk-free rate of 0.002, a common value for experimental scenarios. For random-subperiod functions, the training set was divided into intervals of 3 months each.

The genetic operations used by the process are crossover (between two trees), mutation (of the tree structure), insertion of a new node, deletion of a node, and mutation of a leaf (input feature or constant), and mutation of an operator (internal node). Each operator can occur with the same probability, but the library allows to assign different weights to bias the selection.

In this project, the complexity of an individual is controlled using a probabilistic approach. The parsimony is defined as the ratio in which a new token for a node is chosen to be a terminal node instead of a token from the library. This ratio decays as the tree depth increases by a rate equal to another parameter called parsimony-decay. The model instantiates parsimony, with a value of 0.8, and parsimony-decay, with a value of 0.85, as hyperparameters. The expected depth of a tree is given by

$$ExpectedTreeDepth = \frac{Parsimony}{1 - ParsimonyDecay}$$

Each probability of the input dataset can be selected with the same probability as the others. Eventually, the library allows for biasing the choice with a set of weights for each feature.

7 Data and anomaly set

Market anomalies capture information of different nature, some with large values, others with numbers close to 0. To better aggregate each anomaly, and avoid SR to balance numbers on different magnitudes, all values were normalized to have a mean 0 and a unitary standard deviation.

Following [5], we combine monthly market data with annual accounting information. [2] Our sample spans January 2000 to December 2023 and includes U.S. common stocks listed on major exchanges (NYSE, AMEX, and NASDAQ). We use the French data library as a benchmark for the excess market return and the risk-free rate.

To ensure data quality, we exclude stocks with fewer than 24 observations and those with more than 90% missing observations. Monthly returns from CRSP (RET) are adjusted for delisting returns. We use the French data library[3] as a benchmark for the excess market return and the risk-free rate.

Our anomaly set consists of stock characteristics capturing documented cross-sectional return patterns. We include: accruals (acc), market capitalization (size), momentum (mom), book-to-market ratio (bm), gross profitability (gp), asset growth (ag), and net stock issues (ns). In addition, we incorporate four characteristics commonly associated with trading frictions and risk exposures: market beta (β), dollar trading volume (dvol), illiquidity (ill), and return volatility (vol). Each characteristic is computed at the stock level using the definitions in [5].

The dataset has been reduced to the subperiod from January 2014 to December 2020, to simplify the training and force the model to exploit only relatively recent information. The training set runs from 2014 to 2018, while the test set runs from 2019 to 2020.

[2] `https://www.crsp.org/`

[3] `http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html`

8 Results

The output of the process is, for each type of fitness function, a collection of the monthly returns and Sharpe ratios for 100 different portfolio formation strategies derived from as many mathematical expressions. The solutions achieve a wide spectrum of results; this isn't a surprise because large population size and low tournament size created a low competitive environment where many different solutions survived. The final populations are composed equally from strategies that produce negative, stable, and positive cumulative returns. The complete list of individuals (A) and the summary of the results (B) are reported in the corresponding appendices.

8.1 Returns

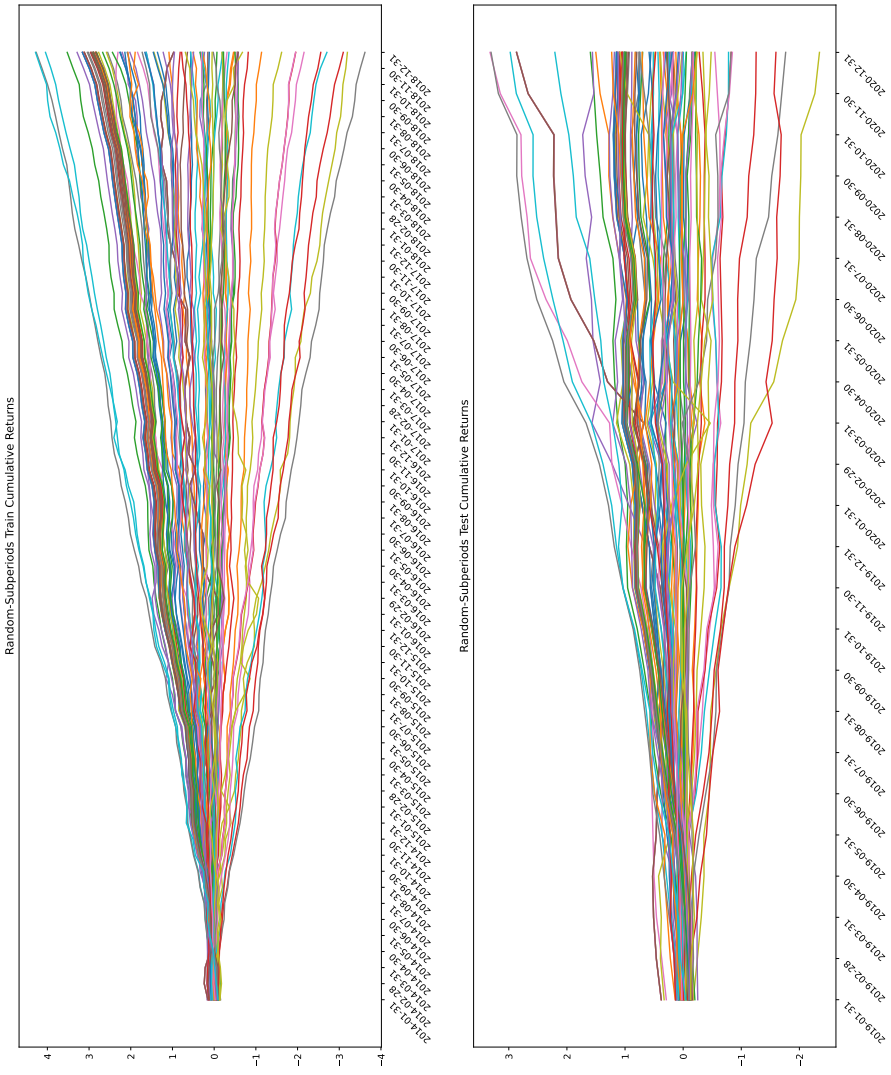


Figure 5: Random-Subperiods population cumulative returns over training period (left) and test period (right).

An immediate observation is that, between the two types of fitness function, stationary bootstrap controls better overfitting. Random-subperiod fitness functions achieve higher cumulative returns over the training set, but performance is worse on the test set.

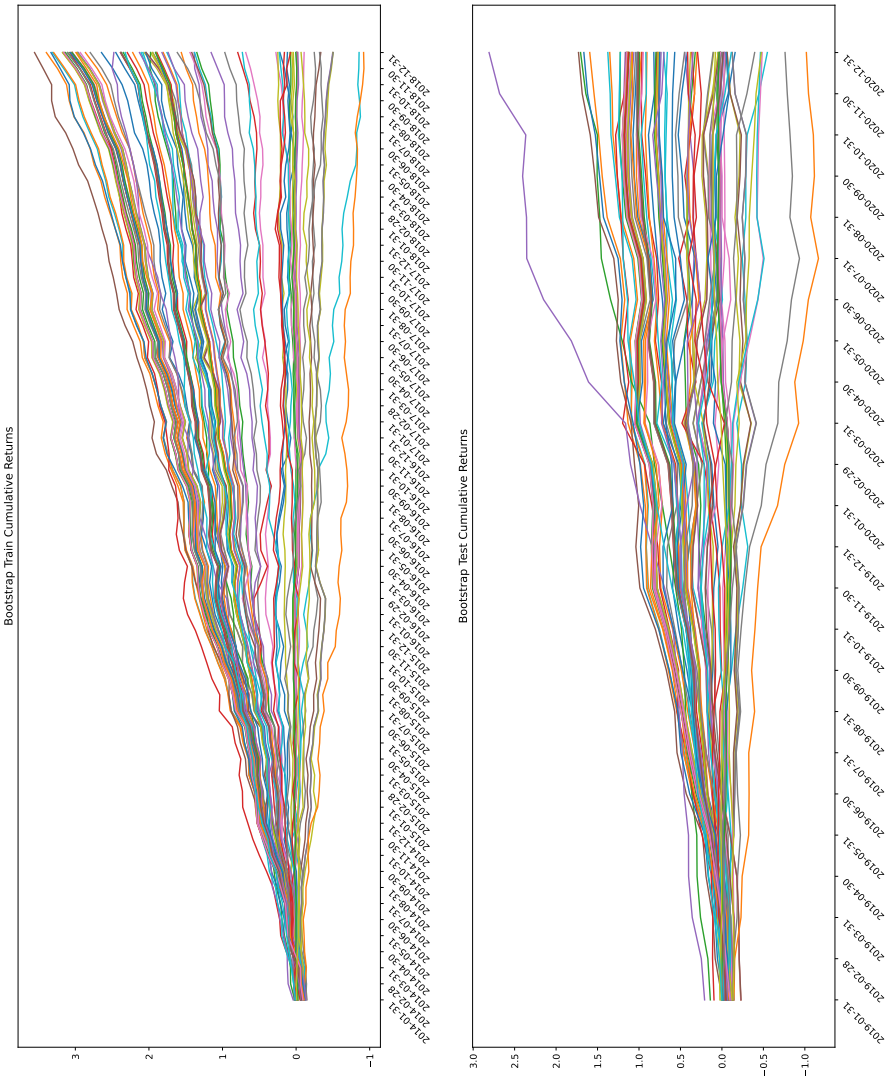


Figure 6: Bootstrap population cumulative returns over training period (left) and test period (right).

Furthermore, random-subperiod also evolved individuals that reach the worst negative performance among the two populations. Bootstrap fitness functions yield a population with a higher cumulative returns on average, and with lower variance, indicating that the method prevents divergence of the strategies to negative results.

The average cumulative returns over the training set of the bootstrap methods is 0.54 with a standard deviation of 0.64, with a minimum value of -1.01.

Table 1: Mean, standard deviation, quartiles, minimum and maximum of the bootstrap population

Bootsrap population summary						
	TrAvg	TrC	TrSP	TesAvg	TesC	TesSP
count	100.0	100.0	100.0	100.0	100.0	100.0
mean	0.02728	1.63708	2.42281	0.02268	0.54429	1.34193
std	0.02139	1.28342	1.90756	0.02671	0.64101	1.52528

min	-	-	-	-	-	-
	0.01529	0.91728	1.63893	0.04243	1.01828	2.01518
25%	0.00341	0.20473	0.41307	0.00076	0.01832	0.07
50%	0.03214	1.92845	3.15461	0.02558	0.61383	1.76758
75%	0.04901	2.94042	4.09275	0.04445	1.06682	2.48808
max	0.05924	3.55452	5.13641	0.11701	2.80815	4.61998

The average cumulative returns over the training set of the random-subperiods method is 0.56 with a standard deviation of 0.97, with a minimum value of -2.34

Table 2: Mean, standard deviation, quartiles, minimum and maximum of the random-subperiods population

Random-Subperiods population summary						
	TrAvg	TrC	TrSP	TesAvg	TesC	TesSP
count	100.0	100.0	100.0	100.0	100.0	100.0
mean	0.01928	1.15685	1.44652	0.02358	0.56587	0.94303
std	0.02998	1.79877	2.69213	0.04051	0.97213	1.86213
min	-	-	-	-	-2.3456	-
	0.06026	3.61589	6.82435	0.09773		5.74707
25%	-0.0016	-	-	0.00035	0.00836	0.01803
		0.09619	0.18515			
50%	0.02151	1.29054	1.90929	0.02226	0.53425	1.40457
75%	0.04743	2.84588	3.68325	0.04395	1.05487	2.12206
max	0.07139	4.28355	7.50422	0.13843	3.32227	5.80808

. In a real-world scenario, where only training data are available, the bootstrap method is preferred because it produces individuals with more reliable results and therefore a better population from which to select a portfolio formation strategy 2.

8.2 Complexity

Another observation is taken by relating the statistics of the dataset to the complexity of the individual. In both populations, the majority of individuals present a value below 40. It is also observable how, for each statistic, alternative solutions with different complexity values obtain similar results. These two observations suggest that it is possible to reduce the complexity of the population during the training process without reducing the quality of the solutions.

Another suggestion in this direction comes from plotting the trajectory of the variance of the average returns by the complexity of the populations sorted into deciles. In both types of fitness functions, the variance of the cumulative returns reaches the higher values between 10 and 20 complexity. The simplest individuals produce portfolio formation strategies that summarize the whole population.

A simpler population would help to produce solutions that are more easily interpretable for a human expert, easier to simplify, and reduce the search space of the problem.

8.3 SPA Rank

For both fitness types, the spa rank procedure sorted the strategies into three positions: the first two levels for a set of the best performing individuals in terms of cumulative

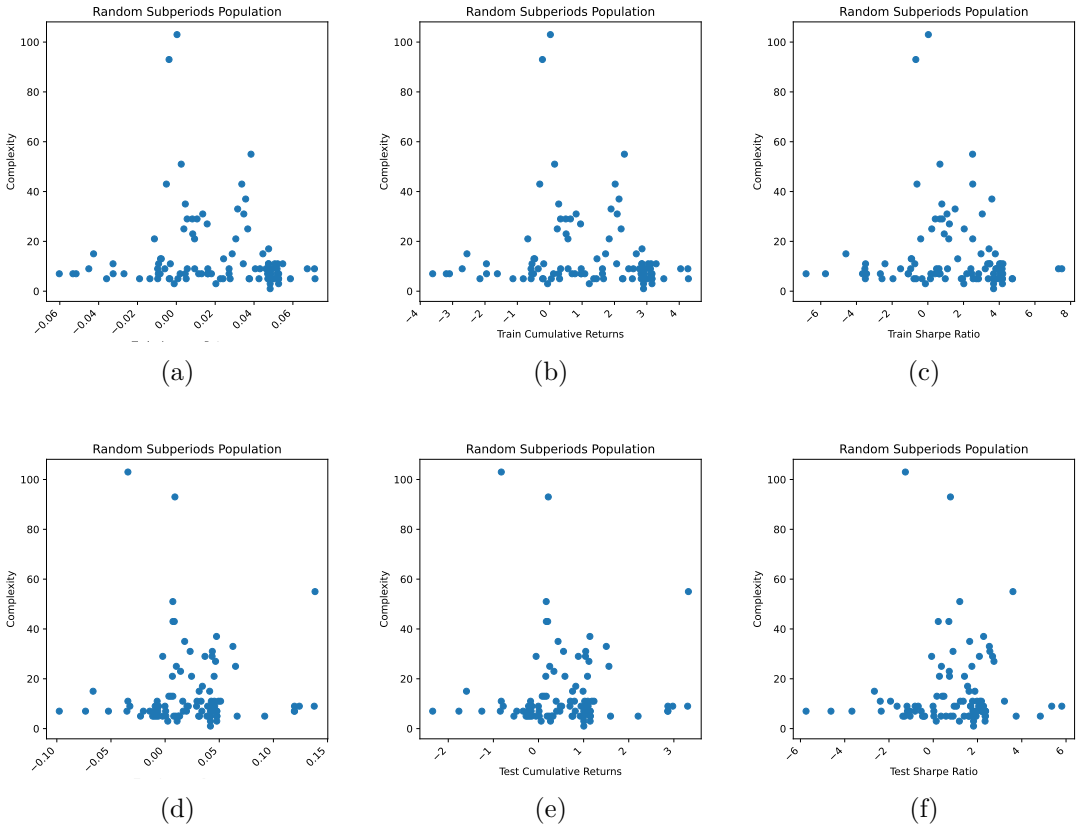


Figure 7: Complexity of the Random-Subperiods population, on y axis, with respect of (a)TrAvg, (b)TrC, (c)TrSP, (d)TeAvg, (a)TeC, (a)TeSP

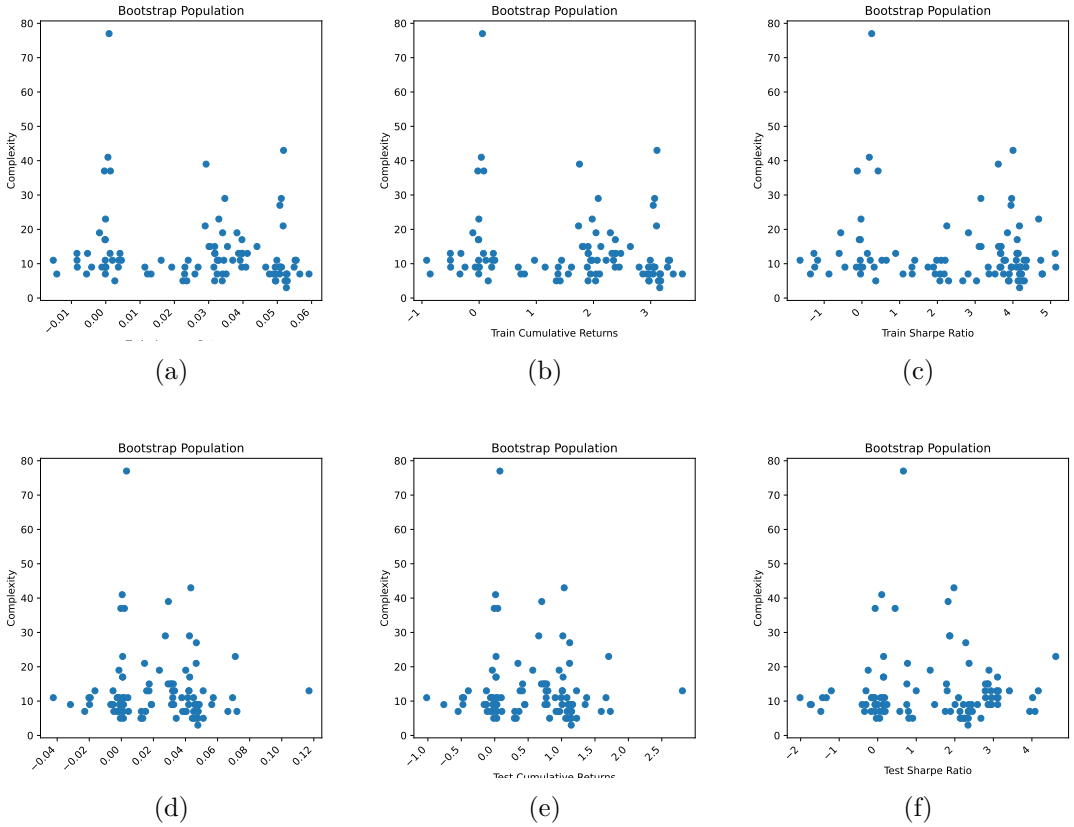
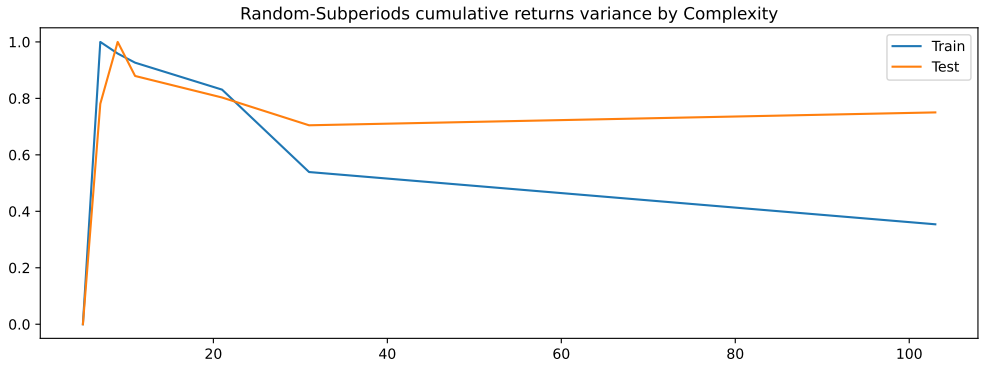
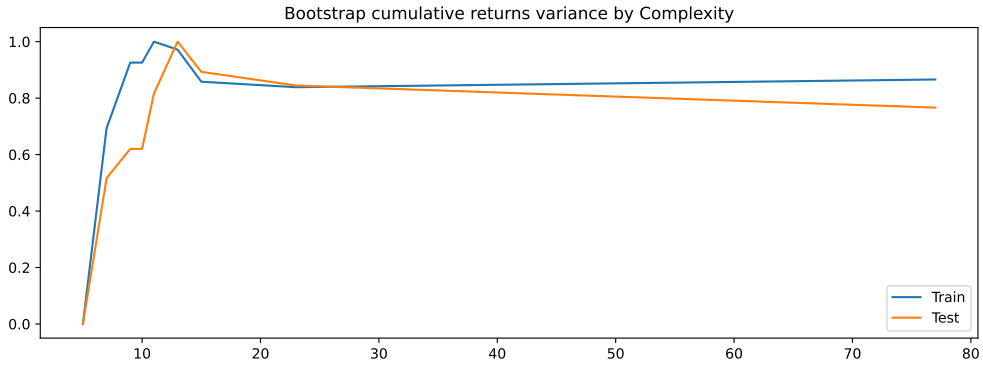


Figure 8: Complexity of the Bootstrap population, on y axis, with respect of (a)TrAvg, (b)TrC, (c)TrSP, (d)TeAvg, (a)TeC, (a)TeSP



(a)



(b)

Figure 9: Variance by complexity deciles over the training and test periods of (a)Random-Subperiods population and (b)Bootsrap population.

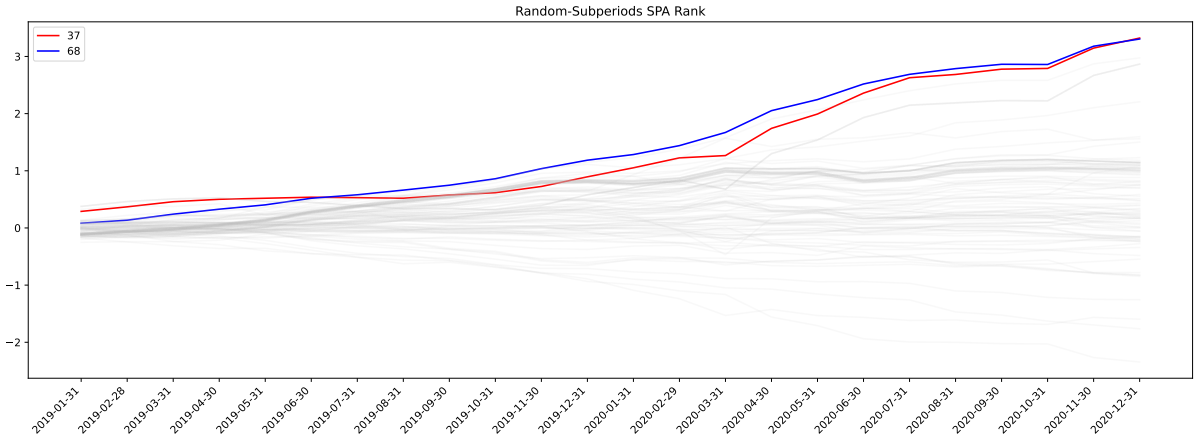
returns. All other strategies are put into the third rank. For reproducibility, for both populations the algorithm had a fixed seed of 42 for the SPA test it involves.

For the Random Subperiod population, the first positions are occupied by:

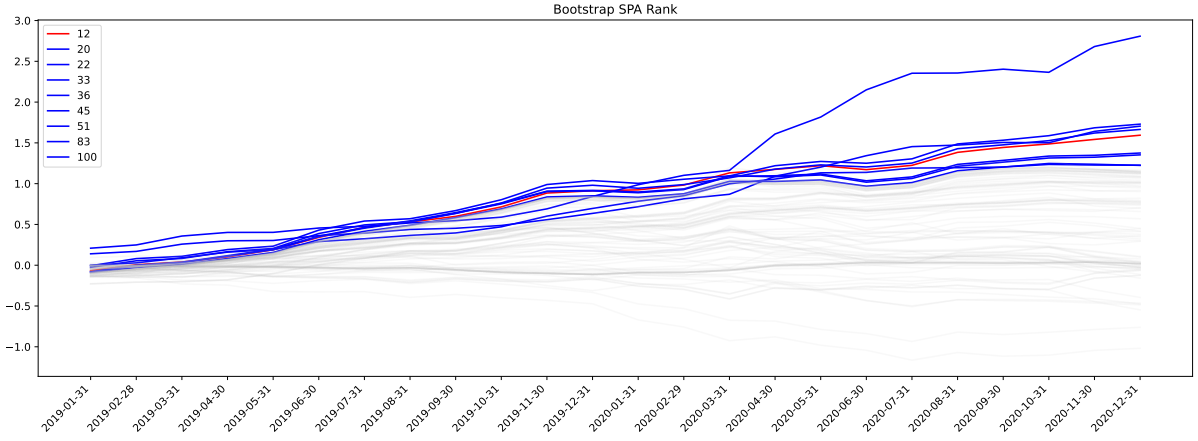
1. Individual 37.
2. Individual 68.

For the Bootstrap population, the first positions are occupied by:

1. Individual 12.
2. Individuals 20, 22, 33, 36, 45, 51, 83, 100.



(a)



(b)

Figure 10: Cumulative returns over the test period of (a)Random-Subperiods population and (b)Bootstrap Populations, colored following the SPA rank: red for the first rank, blue for the second and gray for the third

For the bootstrap version of the algorithm, the best individual is not the equation that produced the highest returns, but the one that consistently provided the best average returns over the time blocks. A ranking built on the spa test selects the best individual not only in terms of global performance, mitigating an important data spoofing bias common

in finance research [2]. The specific human interpretation of the solutions requires further analysis and isn't in the scope of this thesis, which focuses on symbolic regression model.

The final elaboration is an algorithm, with a wide space for improvements, which can generalize over a dataset of market anomalies and produce a ranking score which can be explained better than a classical black box model.

9 Conclusions

Symbolic Regression has proven to be an effective way to produce a ranking score which can be easily explained compared to a black box model. The final populations produced a wide range of alternative solutions, with mixed results, leaving space to inquire a higher competition would lead to a better quality for the individuals. Deploying Symbolic Regression to more Learning to Rank scenarios would help understand if such technology is capable of improving the understanding of a problem, rather than relying on deep learning models, effective but complex to interpret.

Gaussian process has proven to be a useful tool for optimizing the ranking score used to build a portfolio formation strategy, capable of dealing with multiple fitness functions. This method can be deployed with other methods to build a portfolio and study how well it performs compared to alternatives to compose multiple single-factor strategies.

The combined work of Symbolic Regression and Gaussian process applied to finance presented in this work has a large space for improvement. The current algorithm doesn't account for transaction taxes, which heavily influence the performance of a strategy. More elaborate formation strategies can be deployed to build better portfolios, comparing their performance, the costs, and evaluating different strengths and weaknesses.

Finally, the current project was entirely tested with in-sample data over a limited universe of anomalies. To effectively evaluate the algorithm, it is necessary to conduct a more extensive study following a useful backtesting protocol principle [2] to gain a better understanding of it and how it performs in different settings, closer to reality.

References

[1] Dimitrios Angelis, Filippos Sofos, and Theodoros E Karakasidis. Artificial intelligence in physical sciences: Symbolic regression trends and perspectives. *Archives of Computational Methods in Engineering*, page 1, 2023.

[2] Robert D Arnott, Campbell R Harvey, and Harry Markowitz. A backtesting protocol in the era of machine learning. *Available at SSRN 3275654*, 2018.

[3] Charles Audet, Jean Bigeon, Dominique Cartier, Sébastien Le Digabel, and Ludovic Salomon. Performance indicators in multiobjective optimization. *European journal of operational research*, 292(2):397–422, 2021.

[4] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[5] Veronica Guidetti and Alessandra Insana. Trust anomalies' judgment: Social ranking theories for portfolio construction. *European Journal of Operational Research*, 2025.

[6] Peter Reinhard Hansen. A test for superior predictive ability. *Journal of Business & Economic Statistics*, 23(4):365–380, 2005.

[7] Po-Hsuan Hsu and Chung-Ming Kuan. Re-examining the profitability of technical analysis with white's reality check and hansen's spa test. *Available at SSRN 685361*, 2005.

[8] Nour Makke and Sanjay Chawla. Interpretable scientific discovery with symbolic regression: a review. *Artificial Intelligence Review*, 57(1):2, 2024.

[9] Eero Pätäri, Ville Karell, Pasi Luukka, and Julian S Yeomans. Comparison of the multicriteria decision-making methods for equity portfolio selection: The us evidence. *European Journal of Operational Research*, 265(2):655–672, 2018.

[10] Daniel Poh, Bryan Lim, Stefan Zohren, and Stephen Roberts. Building cross-sectional systematic strategies by learning to rank. *arXiv preprint arXiv:2012.07149*, 2020.

[11] Dimitris N Politis and Joseph P Romano. The stationary bootstrap. *Journal of the American Statistical association*, 89(428):1303–1313, 1994.

[12] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *science*, 324(5923):81–85, 2009.

[13] Kevin Sheppard, Stanislav Khrapov, Gábor Lipták, Rick van Hattem, mikedeltalima, Joren Hammudoglu, Rob Capellini, alejandro cermeno, Snyk bot, Hugle, esvhd, Alex Fortin, JPN, Matt Judell, Ryan Russell, Weiliang Li, 645775992, Austin Adams, jbrockmendel, LGTM Migrator, M. Rabba, Michael E. Rose, Nikolay Tretyak, Tom Rochette, UNO Leo, Xavier RENE-CORAIL, Xin Du, and Burak Çelik. bashtage/arch: Release 7.2, November 2024.

[14] Alexander Swade, Matthias X Hanauer, Harald Lohre, and David Blitz. Factor zoo (. zip). *The Journal of Portfolio Management, Quantitative Special*, (2024):50, 2023.

[15] Wikipedia contributors. Factor investing — Wikipedia, the free encyclopedia, 2025. [Online; accessed 29-December-2025].

[16] Wikipedia contributors. Investment strategy — Wikipedia, the free encyclopedia, 2025. [Online; accessed 29-December-2025].

[17] Wikipedia contributors. Modern portfolio theory — Wikipedia, the free encyclopedia, 2025. [Online; accessed 29-December-2025].

[18] Wikipedia contributors. Pareto front — Wikipedia, the free encyclopedia, 2025. [Online; accessed 29-December-2025].

A Population table

Due to the complexity of some of the individuals of both populations, their expression form has been reported in a dedicated sections for the sake of readability of this elaborate. For each type of population, the index points to the corresponding table summary.

To help reading the equations, we report a table where we report each anomaly and the acronym used in the equations.

Anomaly	Equation acronym
Accruals	acc
Market Capitalization	size
Momentum	mom
Book-to-Market Ratio	bm
Gross Profitability	gp
Asset Growth	ag
Net Stock Issues	ns
Beta	beta
Dollar trading volume	dvol
Illiquidity	ill
Return volatility	vol

Random-Subperiods population	
IDX	Equation
1	(size + (acc * gp))
2	Max((size / fbeta), Min(ns, Min(volatility, Max(bm, (bm * ns)))))
3	(ill + Min(0.9490757914447615, size))
4	(ill + Min(size, (0.972490490654147 + acc)))
5	(0.09040506077503353 + (ill + (0.9528116957176468 * size)))
6	Min(((Abs(size) ** 2.090429361669492) / (0.31396310010155215 + gp)), (Max((bm + Min(acc, bm)), Min((0.5848133324797552 + ill), (Max(gp, Min(ill, Max(size, volatility))) / gp))) / Min((bm * (acc + gp)), (0.5329390597355033 + (ns + ((0.5594336550017787 * fbeta) + ((0.3868477305518944 * cumret) + Max(acc, (cumret * size))))))))
7	(0.7155913782653468 + (ill + size))
8	(gp + (-2.1244266523425273 * bm))
9	((2.5558318376986593 * size) + ((-0.981164120519507 * bm) + (-0.45470514784281935 * ill)))
10	(ill + (0.9595548416512084 * size))
11	Max(size, (0.03325561952297856 * acc))
12	Max((size / fbeta), Min(ns, Min(volatility, Max(gp, (bm * ns)))))
13	(gp / bm)
14	(-0.8797515292910892 + ((0.6508038753319242 * ill) + (2.783629441826239 * size)))
15	((-0.01034452269308119 * (bm * size)) + Min((ag + (1.6361930881680915 * cumret)), (fbeta + Min(size, ((2.435183355505421 * Min(acc, cumret)) + (ag * size))))))
16	((1.3895545596437746 + ag) * (-0.5296253911570291 + volatility))
17	((0.9823150481497496 * size) + (0.8162808323484239 * ill))
18	size
19	((cumret + (2.4321303170340163 * size)) / (2.69653767210521 + gp))
20	(gp * Max(Min(0.0296266213401384, acc), Min(0.644527235628260 8, size)))
21	(dolvol + (2.136801815061434 * Min((size / (0.8845515297383033 + gp)), (Max((bm + Min(acc, bm)), Min((-1.590082837344085 + ill), (Max(gp, Min(ill, Max(size, volatility))) / gp))) / ag))))
22	(size * Max(ag, Max(cumret, dolvol)))
23	((0.1161064865493383 * dolvol) + (ag * size))

24	((0.5427856321515107 * ill) + (0.7144020925025364 * size))
25	((-1.3369201907058366 * bm) + (-0.43468905811095104 * Min((ag + (1.112331100460724 * cumret)), (fbeta + Min(size, ((0.16112060509331835 * Min(acc, cumret)) + (ag * size)))))))
26	(size + (0.0138106690477776 * gp))
27	((0.762617090992174 + Max(0.8092002735894995, ag)) * (0.30987892463081074 + volatility))
28	(ag * ((((Abs(((0.4348049928974863 * Max(0.3627709616114918, (fbeta + (0.5624805318888797 * Max(0.4395601414427321, (volatility + (gp * size))))))) + (ag * fbeta))) ** 0.47466620409343796) * (Abs(((0.5344002856813526 * ((Abs(ill) ** 0.41166740912243577) * ((Abs(((0.46267706394916497 * Max(0.4909204817091791, (fbeta + (0.4826027414370153 * Max(0.5012445330901026, (volatility + (gp * size))))))) + (ag * fbeta))) ** 0.5390100714245044) * (Abs(Max(0.550516655617916, gp)) ** 0.5342003778480349)))) + (fbeta * (ill * (volatility * (((0.4927343610789038 * Max(0.3750262430514525, (fbeta + (0.511371393241545 * Max(0.42161677207485126, (volatility + (gp * size))))))) + (ag * fbeta)) * Max(0.5671057499173185, gp))))))) ** 0.5386755063130667)) / ill) / fbeta))
29	Max(size, (size * (-2.260614780624575 + volatility)))
30	Min(size, (0.10159597788570407 / ag))
31	(0.8712394087552324 * (gp * ((cumret + size) * (bm + (fbeta + (gp + ((0.9153456423783971 * volatility) + (0.8228262772103362 * ag))))))))
32	(size / Max(ill, ns))
33	(size * Max(dolvol, Min(ag, cumret)))
34	(0.4569859685220275 / Min(acc, bm))
35	(size * (dolvol + (-0.05679795690935606 * Min(ag, cumret))))
36	Min(((Abs(size) ** 2.346917335383067) / (-0.04079217219162739 + gp)), (Max((bm + Min(acc, bm)), Min((2.2865511570449 + ill), (Max(gp, Min(ill, Max(size, volatility))) / gp))) / ag))
37	(-0.500577854933416 * (volatility * (((0.08824085430740995 * (bm * volatility)) + ((-0.6786999099065623 * ((ag + gp) * (gp + Max(volatility, (bm + (0.5136758714489728 * dolvol)))))) + (volatility * ((gp + Max(volatility, (bm + (0.538802860728221 * dolvol)))) * ((0.48148762754295293 * bm) + (-0.6041613879425344 * volatility)))))) / (gp + Max(volatility, (bm + (0.8388412883037091 * dolvol)))))))

38	(0.643876380199865 * size)
39	(size / Max(dolvol, Min(cumret, (ag + (0.011723621350441684 * volatility)))))
40	(size * Max(Min(3.2123681775155064, size), Min(ag, cumret)))
41	(ill + size)
42	((-0.2165993830677704 * Min(ill, (dolvol + ns))) + ((acc + (0.34575012786383613 * fbeta)) * Min(size, (acc + (size * (0.7183615629606217 + ag))))))
43	(size + Max(0.7414763838042436, Min(ag, cumret)))
44	(ill * Min(size, (0.8198462352325009 + acc)))
45	(size * Max(ag, Max(cumret, Max(size, (((1.4087032228885343 * Min(ill, volatility)) + (1.9223831112078233 * ((volatility + Min(cumret, ((Abs(size) ** 0.11044618366398663) / volatility))) * Max(acc, fbeta)))) / (volatility + Min(cumret, (1.0911376410472027 * (((Abs(size) ** 0.5034891564541887) / volatility) / fbeta))))))))
46	(size + (acc * ill))
47	(0.05840855637142147 * (size / ill))
48	(size + (bm * dolvol))
49	(size * Min(ill, (dolvol + ns)))
50	((0.672642031314699 * size) + (0.47523678201540737 * Max(ill, volatility)))
51	((-0.18271373770304386 * bm) + (ill * size))
52	(size + (Min(ill, (dolvol + ns)) * Min(size, (acc + (size * (0.8733157461153014 + ag))))))
53	Max(size, ((0.7081739580117742 + volatility) * Min(ag, size)))
54	(size * (dolvol + Min(cumret, (ag + (0.11567796003840297 * volatility)))))
55	Min(((Abs(size) ** 0.8898390883668258) / (0.6670969593004598 + gp)), (Max((bm + Min(acc, bm)), Min((0.9079385327449617 + ill), (Max(gp, Min(ill, Max(ns, size))) / gp))) / ag))
56	((0.741547080614152 + volatility) * Max(3.2497842890609356, ag))
57	Min(((Abs(size) ** 0.8823818144570453) / (gp + ns)), (Max((bm + Min(acc, bm)), Min((0.5232783924319211 + ill), (Max(gp, Min(ill, Max(size, volatility))) / gp))) / Min((bm * (acc + gp)), ((0.5372060705927668 * cumret) + Max(acc, (cumret * size))))))
58	(ill * Min(size, (0.8887067719344013 + ns)))
59	(size * (0.06884204024870365 + ill))

60	Min((size / (0.6494332796409124 + gp)), (Max((bm + Min(acc, bm)), Min((0.717318823736522 + ill), (Max(gp, Min(ill, Max(size, volatility))) / gp))) / Min(ag, volatility)))
61	Min(Max(cumret, size), ((cumret + (0.9200979644693508 * size)) / (0.5987882452569812 + gp)))
62	((-0.9521042757745839 * bm) + Min((ag + (0.2754765208815269 * cumret)), (fbeta + Min(size, ((0.5334981653764301 * cumret) + ((0.5604169542493667 * volatility) + ((0.4535612918686553 * Min(acc, cumret)) + (0.31970511652258227 * (ag * size))))))))
63	(size * Max(ag, dolvol))
64	((cumret * ill) + Min(0.9805742487107273, size))
65	Max(Min(0.053912264956797025, acc), Min(0.5246706771950538, size))
66	(0.06837581513104861 + (ill + (size + (0.016463664239430596 * bm))))
67	((0.1433752278559327 * Min(ill, (dolvol + ns))) + ((0.23012479726861093 + ill) * Min(size, (acc + (size * (0.9770115489669183 + ag))))))
68	((2.542632041835783 * size) + (2.0292110531149867 * (Abs(volatility) ** 1.4299349467935527)))
69	(((0.5006531202361935 * (fbeta * (-1.1914106500198087 + cumret))) + (-0.7858738098542283 * (ag * ((0.2242192768502955 * bm) + (-0.8102875657476043 * size))))) / ag)
70	(size * Min(gp, Min(ill, (dolvol + ns))))
71	((-0.7883843493990284 * bm) + Max(Min(1.5946776307535608, acc), Min(0.4622825761308222, ((Abs(ag) ** 0.8509967575849613) * ((Max(ns, (0.25607794540052164 / bm)) / ((0.8259672064457638 * Max(0.8750892788171302, (dolvol * ill))) + (bm * size))) / (dolvol + (0.6468519762040522 * ns)))))))
72	(size * (1.0 + Min((dolvol + ns), (ill + ns))))
73	Min(ag, ((Abs(size) ** 2.637990011056692) / (1.6581401605585555 + gp)))
74	(-1.3740163252988409 * (bm * (dolvol * ((0.11903332258736575 + volatility) * (-0.36085185933235703 + Max(1.3533000238574324, ag))))))
75	Min((size / (0.6562076806786202 + gp)), (Max((bm + Min(acc, bm)), Min((0.8214031996841029 + ill), (Max(gp, Min(ill, Max(size, volatility))) / gp))) / ag))
76	((0.5461202423576773 * size) + (-0.18822993619898445 * (bm * ill)))

77	(size / (dolvol + Min(cumret, (ag + (0.07457267123964009 * volatility)))))
78	((0.3037411947662843 + Min((dolvol + ns), (ill + ns))) * (ag + (0.9823548076349681 * size)))
79	Min(size, Max(Min(ag, cumret), (gp / ill)))
80	(size * ((size + (ag * dolvol)) / dolvol))
81	(size + (gp * ns))
82	(size * (ag + Max(dolvol, size)))
83	(size * (1.277148939934714 + dolvol))
84	Min(((cumret + (0.9761368869106535 * size)) / (0.5097961400421744 + gp)), (Max((bm + Min(acc, bm)), Min((0.8213152438259782 + ill), (Max(gp, Min(ill, Max(size, volatility))) / gp))) / ag))
85	(0.49270268734396816 + (ill + ((2.3000615569193528 * size) + (-1.1142437195213541 * bm))))
86	(0.08352026294670419 * (gp / ill))
87	(size * (dolvol + Min(ag, cumret)))
88	(0.38667493961477784 * ((((Abs(((0.6100283009148457 * Max(0.3549050372625257, (fbeta + (0.45616662634029126 * Max(0.32650356278000103, (volatility + (gp * size))))))) + (ag * fbeta))) ** 0.8626272150909513) * (Abs(((0.5821143022743784 * ((Abs(fbeta) ** 0.5725318391331153) * (Abs(volatility) ** 0.6772983753061055))) + ((0.3637486876476194 * ((Abs(ill) ** 0.563506052993503) * ((Abs(((0.6750359897837666 * Max(0.41340862587706934, (fbeta + (0.6333952822300472 * Max(0.5551032278554947, (volatility + (gp * size))))))) + (ag * fbeta))) ** 0.7863922870065181) * (Abs(Max(0.6726619389812827, gp)) ** 0.44808133549062135)))) + (fbeta * (ill * (volatility * (((0.5925822235468342 * Max(0.6250939824356385, (fbeta + (0.5147349358670577 * Max(0.34991930986593106, (volatility + (gp * size))))))) + (ag * fbeta)) * Max(0.6378663965958412, gp)))))))) ** 0.7949647599673781)) / ill) / fbeta))
89	(acc * (1.930636813971994 + ill))
90	Max(Min(1.1502530449446577, acc), Min(1.6297703822645506, ((Abs(ag) ** 0.888374906108051) * ((Max(ns, (0.9336789690253585 / bm)) / ((0.5391515373122556 * Max(1.165583358414438, (cumret * (dolvol * ill)))) + (bm * size))) / (dolvol + (0.602933188753248 * ns))))))
91	(size / Max(0.09091972885288098, ns))
92	(size * (1.5840550484413698 + Min(cumret, (ag + (2.9639066389931914 * volatility)))))

93	((0.960964967070303 * size) + Max(dolvol, (gp * Min((0.22390317021080397 + fbeta), Max(gp, ill)))))
94	(size * Max(dolvol, (ag / cumret)))
95	Max(0.03772821066317762, size)
96	(size * (3.18347519058651 + Min((dolvol + ns), (fbeta + ill))))
97	Min(size, (3.2115948625744792 / cumret))
98	(ill + (size + ((0.0305001681725706 * dolvol) + (0.05173574623893758 * bm))))
99	((1.4830003880552267 + ill) * (-1.9045521157985652 + gp))
100	(ill + Min(ill, size))

Bootstrap population	
Index	Equation
1	(size + (acc * gp))
2	Max((size / fbeta), Min(ns, Min(volatility, Max(bm, (bm * ns)))))
3	(ill + Min(0.9490757914447615, size))
4	(ill + Min(size, (0.972490490654147 + acc)))
5	(0.09040506077503353 + (ill + (0.9528116957176468 * size)))
6	Min(((Abs(size) ** 2.090429361669492) / (0.31396310010155215 + gp)), (Max((bm + Min(acc, bm)), Min((0.5848133324797552 + ill), (Max(gp, Min(ill, Max(size, volatility))) / gp))) / Min((bm * (acc + gp)), (0.5329390597355033 + (ns + ((0.5594336550017787 * fbeta) + ((0.3868477305518944 * cumret) + Max(acc, (cumret * size))))))))
7	(0.7155913782653468 + (ill + size))
8	(gp + (-2.1244266523425273 * bm))
9	((2.5558318376986593 * size) + ((-0.981164120519507 * bm) + (-0.45470514784281935 * ill)))
10	(ill + (0.9595548416512084 * size))
11	Max(size, (0.03325561952297856 * acc))
12	Max((size / fbeta), Min(ns, Min(volatility, Max(gp, (bm * ns)))))
13	(gp / bm)
14	(-0.8797515292910892 + ((0.6508038753319242 * ill) + (2.783629441826239 * size)))
15	((-0.01034452269308119 * (bm * size)) + Min((ag + (1.6361930881680915 * cumret)), (fbeta + Min(size, ((2.435183355505421 * Min(acc, cumret)) + (ag * size))))))
16	((1.3895545596437746 + ag) * (-0.5296253911570291 + volatility))
17	((0.9823150481497496 * size) + (0.8162808323484239 * ill))
18	size
19	((cumret + (2.4321303170340163 * size)) / (2.69653767210521 + gp))
20	(gp * Max(Min(0.0296266213401384, acc), Min(0.644527235628260 8, size)))
21	(dolvol + (2.136801815061434 * Min((size / (0.8845515297383033 + gp)), (Max((bm + Min(acc, bm)), Min((-1.590082837344085 + ill), (Max(gp, Min(ill, Max(size, volatility))) / gp))) / ag))))
22	(size * Max(ag, Max(cumret, dolvol)))
23	((0.1161064865493383 * dolvol) + (ag * size))

24	((0.5427856321515107 * ill) + (0.7144020925025364 * size))
25	((-1.3369201907058366 * bm) + (-0.43468905811095104 * Min((ag + (1.112331100460724 * cumret)), (fbeta + Min(size, ((0.16112060509331835 * Min(acc, cumret)) + (ag * size)))))))
26	(size + (0.0138106690477776 * gp))
27	((0.762617090992174 + Max(0.8092002735894995, ag)) * (0.30987892463081074 + volatility))
28	(ag * ((((Abs(((0.4348049928974863 * Max(0.3627709616114918, (fbeta + (0.5624805318888797 * Max(0.4395601414427321, (volatility + (gp * size))))))) + (ag * fbeta))) ** 0.47466620409343796) * (Abs(((0.5344002856813526 * ((Abs(ill) ** 0.41166740912243577) * ((Abs(((0.46267706394916497 * Max(0.4909204817091791, (fbeta + (0.4826027414370153 * Max(0.5012445330901026, (volatility + (gp * size))))))) + (ag * fbeta))) ** 0.5390100714245044) * (Abs(Max(0.550516655617916, gp)) ** 0.5342003778480349)))) + (fbeta * (ill * (volatility * (((0.4927343610789038 * Max(0.3750262430514525, (fbeta + (0.511371393241545 * Max(0.42161677207485126, (volatility + (gp * size))))))) + (ag * fbeta)) * Max(0.5671057499173185, gp))))))) ** 0.5386755063130667)) / ill) / fbeta))
29	Max(size, (size * (-2.260614780624575 + volatility)))
30	Min(size, (0.10159597788570407 / ag))
31	(0.8712394087552324 * (gp * ((cumret + size) * (bm + (fbeta + (gp + ((0.9153456423783971 * volatility) + (0.8228262772103362 * ag))))))))
32	(size / Max(ill, ns))
33	(size * Max(dolvol, Min(ag, cumret)))
34	(0.4569859685220275 / Min(acc, bm))
35	(size * (dolvol + (-0.05679795690935606 * Min(ag, cumret))))
36	Min(((Abs(size) ** 2.346917335383067) / (-0.04079217219162739 + gp)), (Max((bm + Min(acc, bm)), Min((2.2865511570449 + ill), (Max(gp, Min(ill, Max(size, volatility))) / gp))) / ag))
37	(-0.500577854933416 * (volatility * (((0.08824085430740995 * (bm * volatility)) + ((-0.6786999099065623 * ((ag + gp) * (gp + Max(volatility, (bm + (0.5136758714489728 * dolvol)))))) + (volatility * ((gp + Max(volatility, (bm + (0.538802860728221 * dolvol)))) * ((0.48148762754295293 * bm) + (-0.6041613879425344 * volatility)))))) / (gp + Max(volatility, (bm + (0.8388412883037091 * dolvol)))))))

38	(0.643876380199865 * size)
39	(size / Max(dolvol, Min(cumret, (ag + (0.011723621350441684 * volatility)))))
40	(size * Max(Min(3.2123681775155064, size), Min(ag, cumret)))
41	(ill + size)
42	((-0.2165993830677704 * Min(ill, (dolvol + ns))) + ((acc + (0.34575012786383613 * fbeta)) * Min(size, (acc + (size * (0.7183615629606217 + ag))))))
43	(size + Max(0.7414763838042436, Min(ag, cumret)))
44	(ill * Min(size, (0.8198462352325009 + acc)))
45	(size * Max(ag, Max(cumret, Max(size, (((1.4087032228885343 * Min(ill, volatility)) + (1.9223831112078233 * ((volatility + Min(cumret, ((Abs(size) ** 0.11044618366398663) / volatility))) * Max(acc, fbeta)))) / (volatility + Min(cumret, (1.0911376410472027 * (((Abs(size) ** 0.5034891564541887) / volatility) / fbeta))))))))
46	(size + (acc * ill))
47	(0.05840855637142147 * (size / ill))
48	(size + (bm * dolvol))
49	(size * Min(ill, (dolvol + ns)))
50	((0.672642031314699 * size) + (0.47523678201540737 * Max(ill, volatility)))
51	((-0.18271373770304386 * bm) + (ill * size))
52	(size + (Min(ill, (dolvol + ns)) * Min(size, (acc + (size * (0.8733157461153014 + ag))))))
53	Max(size, ((0.7081739580117742 + volatility) * Min(ag, size)))
54	(size * (dolvol + Min(cumret, (ag + (0.11567796003840297 * volatility)))))
55	Min(((Abs(size) ** 0.8898390883668258) / (0.6670969593004598 + gp)), (Max((bm + Min(acc, bm)), Min((0.9079385327449617 + ill), (Max(gp, Min(ill, Max(ns, size))) / gp))) / ag))
56	((0.741547080614152 + volatility) * Max(3.2497842890609356, ag))
57	Min(((Abs(size) ** 0.8823818144570453) / (gp + ns)), (Max((bm + Min(acc, bm)), Min((0.5232783924319211 + ill), (Max(gp, Min(ill, Max(size, volatility))) / gp))) / Min((bm * (acc + gp)), ((0.5372060705927668 * cumret) + Max(acc, (cumret * size))))))
58	(ill * Min(size, (0.8887067719344013 + ns)))
59	(size * (0.06884204024870365 + ill))

60	Min((size / (0.6494332796409124 + gp)), (Max((bm + Min(acc, bm)), Min((0.717318823736522 + ill), (Max(gp, Min(ill, Max(size, volatility))) / gp))) / Min(ag, volatility)))
61	Min(Max(cumret, size), ((cumret + (0.9200979644693508 * size)) / (0.5987882452569812 + gp)))
62	((-0.9521042757745839 * bm) + Min((ag + (0.2754765208815269 * cumret)), (fbeta + Min(size, ((0.5334981653764301 * cumret) + ((0.5604169542493667 * volatility) + ((0.4535612918686553 * Min(acc, cumret)) + (0.31970511652258227 * (ag * size))))))))
63	(size * Max(ag, dolvol))
64	((cumret * ill) + Min(0.9805742487107273, size))
65	Max(Min(0.053912264956797025, acc), Min(0.5246706771950538, size))
66	(0.06837581513104861 + (ill + (size + (0.016463664239430596 * bm))))
67	((0.1433752278559327 * Min(ill, (dolvol + ns))) + ((0.23012479726861093 + ill) * Min(size, (acc + (size * (0.9770115489669183 + ag))))))
68	((2.542632041835783 * size) + (2.0292110531149867 * (Abs(volatility) ** 1.4299349467935527)))
69	(((0.5006531202361935 * (fbeta * (-1.1914106500198087 + cumret))) + (-0.7858738098542283 * (ag * ((0.2242192768502955 * bm) + (-0.8102875657476043 * size))))) / ag)
70	(size * Min(gp, Min(ill, (dolvol + ns))))
71	((-0.7883843493990284 * bm) + Max(Min(1.5946776307535608, acc), Min(0.4622825761308222, ((Abs(ag) ** 0.8509967575849613) * ((Max(ns, (0.25607794540052164 / bm)) / ((0.8259672064457638 * Max(0.8750892788171302, (dolvol * ill))) + (bm * size))) / (dolvol + (0.6468519762040522 * ns)))))))
72	(size * (1.0 + Min((dolvol + ns), (ill + ns))))
73	Min(ag, ((Abs(size) ** 2.637990011056692) / (1.6581401605585555 + gp)))
74	(-1.3740163252988409 * (bm * (dolvol * ((0.11903332258736575 + volatility) * (-0.36085185933235703 + Max(1.3533000238574324, ag))))))
75	Min((size / (0.6562076806786202 + gp)), (Max((bm + Min(acc, bm)), Min((0.8214031996841029 + ill), (Max(gp, Min(ill, Max(size, volatility))) / gp))) / ag))
76	((0.5461202423576773 * size) + (-0.18822993619898445 * (bm * ill)))

77	(size / (dolvol + Min(cumret, (ag + (0.07457267123964009 * volatility)))))
78	((0.3037411947662843 + Min((dolvol + ns), (ill + ns))) * (ag + (0.9823548076349681 * size)))
79	Min(size, Max(Min(ag, cumret), (gp / ill)))
80	(size * ((size + (ag * dolvol)) / dolvol))
81	(size + (gp * ns))
82	(size * (ag + Max(dolvol, size)))
83	(size * (1.277148939934714 + dolvol))
84	Min(((cumret + (0.9761368869106535 * size)) / (0.5097961400421744 + gp)), (Max((bm + Min(acc, bm)), Min((0.8213152438259782 + ill), (Max(gp, Min(ill, Max(size, volatility))) / gp))) / ag))
85	(0.49270268734396816 + (ill + ((2.3000615569193528 * size) + (-1.1142437195213541 * bm))))
86	(0.08352026294670419 * (gp / ill))
87	(size * (dolvol + Min(ag, cumret)))
88	(0.38667493961477784 * ((((Abs(((0.6100283009148457 * Max(0.3549050372625257, (fbeta + (0.45616662634029126 * Max(0.32650356278000103, (volatility + (gp * size)))))))) + (ag * fbeta))) ** 0.8626272150909513) * (Abs(((0.5821143022743784 * ((Abs(fbeta) ** 0.5725318391331153) * (Abs(volatility) ** 0.6772983753061055))) + ((0.3637486876476194 * ((Abs(ill) ** 0.563506052993503) * ((Abs(((0.6750359897837666 * Max(0.41340862587706934, (fbeta + (0.6333952822300472 * Max(0.5551032278554947, (volatility + (gp * size)))))))) + (ag * fbeta))) ** 0.7863922870065181) * (Abs(Max(0.6726619389812827, gp)) ** 0.44808133549062135)))) + (fbeta * (ill * (volatility * (((0.5925822235468342 * Max(0.6250939824356385, (fbeta + (0.5147349358670577 * Max(0.34991930986593106, (volatility + (gp * size)))))))) + (ag * fbeta)) * Max(0.6378663965958412, gp)))))))) ** 0.7949647599673781)) / ill) / fbeta))
89	(acc * (1.930636813971994 + ill))
90	Max(Min(1.1502530449446577, acc), Min(1.6297703822645506, ((Abs(ag) ** 0.888374906108051) * ((Max(ns, (0.9336789690253585 / bm)) / ((0.5391515373122556 * Max(1.165583358414438, (cumret * (dolvol * ill)))) + (bm * size))) / (dolvol + (0.602933188753248 * ns))))))
91	(size / Max(0.09091972885288098, ns))
92	(size * (1.5840550484413698 + Min(cumret, (ag + (2.9639066389931914 * volatility)))))

93	((0.960964967070303 * size) + Max(dolvol, (gp * Min((0.22390317021080397 + fbeta), Max(gp, ill)))))
94	(size * Max(dolvol, (ag / cumret)))
95	Max(0.03772821066317762, size)
96	(size * (3.18347519058651 + Min((dolvol + ns), (fbeta + ill))))
97	Min(size, (3.2115948625744792 / cumret))
98	(ill + (size + ((0.0305001681725706 * dolvol) + (0.05173574623893758 * bm))))
99	((1.4830003880552267 + ill) * (-1.9045521157985652 + gp))
100	(ill + Min(ill, size))

B Populations summary

Table summary of the random-subperiod 5 and bootstrap population 6. In both tables the first column contains the index of each individual, the second the complexity. The following three columns report, respectively, the average returns, the cumulative returns and the annual Sharpe ratio over the training set. The final three columns report the same three metrics for the test set

Table 5:

Random-Subperiods population summary							
IDX	CPX	TrAvg	TrC	TrSP	TesAvg	TesC	TesSP
1	5	0.03744	2.24637	3.38639	0.03098	0.74363	1.62671
2	13	-0.00814	-0.4885	-0.93099	0.00718	0.17224	0.48422
3	5	0.05873	3.52402	4.74311	0.06652	1.59649	3.74611
4	7	0.04854	2.91233	4.12514	0.04386	1.05254	2.22482
5	7	0.05267	3.1602	4.19568	0.0478	1.14726	2.35533
6	51	0.00252	0.15107	0.67709	0.00713	0.17107	1.19861
7	5	0.05265	3.15875	4.17834	0.0477	1.1449	2.34184
8	5	0.04255	2.55281	2.7203	0.03172	0.76129	1.39768
9	11	0.05216	3.12932	4.01989	0.04545	1.09083	1.77636
10	5	0.0526	3.15619	4.1738	0.0479	1.1495	2.36069
11	5	0.02266	1.35945	1.94547	-0.00812	-0.19495	-0.38912
12	13	-0.00774	-0.46433	-0.90849	0.00526	0.12633	0.40771
13	3	-0.00111	-0.06648	-0.13825	0.01107	0.26559	1.1818
14	9	0.04944	2.96666	3.84015	0.04424	1.06184	2.05471
15	25	0.00388	0.23259	0.2183	0.01034	0.24826	0.36933
16	7	0.01618	0.97064	0.62434	0.11935	2.86432	2.20818
17	7	0.05232	3.13932	4.10602	0.04763	1.1431	2.36559
18	1	0.0483	2.89778	3.68307	0.04184	1.00412	1.82092
19	9	0.04304	2.58251	3.21597	0.02017	0.48398	0.94456
20	9	-0.00593	-0.35555	-0.96027	-0.00611	-0.14675	-0.67793
21	31	0.03473	2.08403	3.04881	0.0231	0.55444	0.8933
22	7	0.01807	1.08399	2.46368	0.01675	0.40201	1.94507
23	7	-0.00929	-0.55715	-1.10404	-0.00679	-0.16299	-1.06761
24	7	0.05191	3.11456	4.05072	0.04732	1.13563	2.34797
25	25	0.03676	2.2053	2.03739	0.06503	1.56078	1.74623
26	5	0.04891	2.93472	3.7094	0.04083	0.98003	1.77652
27	9	0.0162	0.97225	0.62534	0.11946	2.86701	2.21221
28	93	-0.00376	-0.22544	-0.67453	-0.00904	-0.21703	-0.77676
29	7	-0.05325	-3.1951	-3.6626	-0.09773	-2.3456	-3.67397
30	5	-0.00337	-0.20211	-0.32585	0.00978	0.23476	0.89871
31	21	0.00939	0.56321	1.17289	0.02441	0.5859	1.41146
32	5	-0.01894	-1.1364	-2.56902	-0.00632	-0.1517	-0.43069
33	7	-0.00835	-0.50087	-1.09249	-0.00877	-0.21043	-1.16667

34	5	-	-	-	-	-	-
		0.01362	0.81706	1.95516	0.01148	0.27552	1.37348
35	9	-	-	-	-	-	-
		0.00963	0.57769	1.52842	0.00659	0.15824	1.18507
36	29	0.01067	0.64042	0.80756	0.0434	1.04153	2.67994
37	55	0.03844	2.30644	2.50625	0.13843	3.32227	3.60077
38	3	0.0483	2.89778	3.68307	0.04184	1.00412	1.82092
39	11	-	-0.1853	-	0.0163	0.39113	2.17784
		0.00309		0.73068			
40	9	0.00944	0.5661	1.10218	-	-	-
					0.00013	0.00321	0.00925
41	3	0.05265	3.15875	4.17834	0.0477	1.1449	2.34184
42	23	0.00844	0.50624	0.91773	0.01424	0.34167	0.7242
43	7	0.0483	2.89778	3.68307	0.04184	1.00412	1.82092
44	7	-	-	-	-	-	-
		0.05161	3.09648	5.73579	0.05233	1.25585	4.62121
45	43	0.03369	2.02141	2.5151	0.00723	0.17361	0.2273
46	5	0.04919	2.95133	3.82938	0.04027	0.96652	1.76937
47	5	-	-	-	-	-	-
		0.03592	2.15532	3.50269	0.02272	0.54521	1.29068
48	5	0.05027	3.01635	3.96638	0.03857	0.92568	1.59646
49	7	-	-	-	-	-	-
		0.02696	1.61752	2.66172	0.01421	0.34094	0.82181
50	9	0.06741	4.04453	7.50422	0.12396	2.97505	5.80808
51	7	0.00194	0.11655	0.17001	0.03395	0.81489	1.79277
52	17	0.04752	2.85099	3.42679	0.03442	0.82611	1.55141
53	9	0.04065	2.43901	2.41091	0.02904	0.69706	1.05171
54	11	-0.0326	-	-	-	-	-
			1.95574	3.49046	0.03439	0.82541	2.39353
55	29	0.00824	0.49452	0.67651	0.03678	0.88273	2.08469
56	7	0.0162	0.97225	0.62534	0.11951	2.86813	2.21283
57	43	-	-	-	0.00848	0.20362	0.70567
		0.00513	0.30779	0.60935			
58	7	-	-	-	-0.0735	-	-
		0.06026	3.61589	6.82435		1.76395	5.74707
59	5	0.02404	1.44264	1.8731	0.00068	0.01625	0.02712
60	29	0.00555	0.33296	0.41616	-	-	-
					0.00222	0.05335	0.07269
61	13	0.02433	1.45955	1.67352	0.00346	0.08296	0.16389
62	33	0.03157	1.89419	1.52651	0.0626	1.50249	2.53525
63	5	-	-	-	-	-	-
		0.00366	0.21974	0.77956	0.00234	0.05604	0.48343
64	7	0.04699	2.81952	4.104	0.04587	1.10083	2.17528
65	7	0.02715	1.62905	2.84393	0.00051	0.01222	0.02737
66	9	0.05185	3.111	4.16197	0.0458	1.0991	2.27302
67	21	0.03061	1.83677	2.51392	0.00673	0.16148	0.27942
68	9	0.07112	4.26712	7.32012	0.13771	3.30513	5.35302

69	21	-0.01129	-0.67729	-0.40009	0.04525	1.08592	0.73338
70	9	-0.04507	-2.70395	-3.52818	-0.03255	-0.78117	-1.63478
71	37	0.03568	2.14101	3.58683	0.04746	1.1391	2.27554
72	11	0.04745	2.84727	3.35286	0.02943	0.70632	1.2272
73	9	0.00562	0.33711	0.48899	-0.0092	-0.22091	-0.82108
74	15	-0.04261	-2.55669	-4.59073	-0.06652	-1.59643	-2.65903
75	27	0.01587	0.95223	1.20645	0.04658	1.11787	2.74345
76	9	0.04815	2.88877	3.80771	0.03927	0.94243	1.6957
77	11	-0.00912	-0.54749	-2.40256	-0.00734	-0.17608	-1.93006
78	15	0.02875	1.72525	2.97918	0.03144	0.75467	1.61717
79	9	0.04616	2.76975	3.92917	0.0433	1.03912	2.20945
80	9	0.02714	1.62855	4.02452	0.02142	0.51407	1.682
81	5	0.03769	2.26124	2.83607	0.00797	0.19125	0.34974
82	7	0.00457	0.27408	0.40162	-0.00991	-0.23776	-0.54533
83	5	0.04709	2.82539	3.68379	0.04208	1.01001	1.89401
84	31	0.01359	0.81517	1.07298	0.04359	1.04616	2.5582
85	11	0.05478	3.28697	4.20808	0.04929	1.18287	1.97827
86	5	-0.00958	-0.57488	-0.65738	0.01203	0.28873	0.78747
87	7	-0.03254	-1.95232	-3.43474	-0.03527	-0.8465	-2.35123
88	103	0.00034	0.02031	0.02768	-0.03436	-0.82455	-1.25931
89	5	0.0008	0.04825	0.14514	-0.00988	-0.23717	-1.04661
90	35	0.00462	0.27693	0.7843	0.01805	0.43332	1.64639
91	5	0.0277	1.66206	2.53592	0.01251	0.30019	0.57587
92	11	0.03445	2.06723	3.46002	0.05124	1.22983	3.21865
93	15	0.04456	2.67335	3.77941	0.04114	0.98726	1.88158
94	7	0.013	0.78013	2.18045	0.01972	0.4733	1.92803
95	3	0.02036	1.22164	2.01451	0.00245	0.05881	0.13458
96	11	0.04742	2.84542	3.47684	0.0332	0.79675	1.35501
97	5	0.00507	0.30435	0.97567	0.01341	0.3218	1.93589
98	11	0.05068	3.04076	4.08341	0.04265	1.02372	2.10432
99	7	0.01149	0.68947	0.68829	-0.02008	-0.48198	-1.09019
100	5	0.07139	4.28355	4.72352	0.09197	2.20731	4.8384

Table 6:

Bootsrap population summary							
IDX	CPX	TrAvg	TrC	TrSP	TesAvg	TesC	TesSP
1	5	0.03744	2.24637	3.38639	0.03098	0.74363	1.62671
2	13	-0.00814	-0.4885	-0.93099	0.00718	0.17224	0.48422
3	5	0.05873	3.52402	4.74311	0.06652	1.59649	3.74611
4	7	0.04854	2.91233	4.12514	0.04386	1.05254	2.22482
5	7	0.05267	3.1602	4.19568	0.0478	1.14726	2.35533
6	51	0.00252	0.15107	0.67709	0.00713	0.17107	1.19861
7	5	0.05265	3.15875	4.17834	0.0477	1.1449	2.34184
8	5	0.04255	2.55281	2.7203	0.03172	0.76129	1.39768
9	11	0.05216	3.12932	4.01989	0.04545	1.09083	1.77636
10	5	0.0526	3.15619	4.1738	0.0479	1.1495	2.36069
11	5	0.02266	1.35945	1.94547	-0.00812	-0.19495	-0.38912
12	13	-0.00774	-0.46433	-0.90849	0.00526	0.12633	0.40771
13	3	-0.00111	-0.06648	-0.13825	0.01107	0.26559	1.1818
14	9	0.04944	2.96666	3.84015	0.04424	1.06184	2.05471
15	25	0.00388	0.23259	0.2183	0.01034	0.24826	0.36933
16	7	0.01618	0.97064	0.62434	0.11935	2.86432	2.20818
17	7	0.05232	3.13932	4.10602	0.04763	1.1431	2.36559
18	1	0.0483	2.89778	3.68307	0.04184	1.00412	1.82092
19	9	0.04304	2.58251	3.21597	0.02017	0.48398	0.94456
20	9	-0.00593	-0.35555	-0.96027	-0.00611	-0.14675	-0.67793
21	31	0.03473	2.08403	3.04881	0.0231	0.55444	0.8933
22	7	0.01807	1.08399	2.46368	0.01675	0.40201	1.94507
23	7	-0.00929	-0.55715	-1.10404	-0.00679	-0.16299	-1.06761
24	7	0.05191	3.11456	4.05072	0.04732	1.13563	2.34797
25	25	0.03676	2.2053	2.03739	0.06503	1.56078	1.74623
26	5	0.04891	2.93472	3.7094	0.04083	0.98003	1.77652
27	9	0.0162	0.97225	0.62534	0.11946	2.86701	2.21221
28	93	-0.00376	-0.22544	-0.67453	-0.00904	-0.21703	-0.77676
29	7	-0.05325	-3.1951	-3.6626	-0.09773	-2.3456	-3.67397
30	5	-0.00337	-0.20211	-0.32585	0.00978	0.23476	0.89871
31	21	0.00939	0.56321	1.17289	0.02441	0.5859	1.41146
32	5	-0.01894	-1.1364	-2.56902	-0.00632	-0.1517	-0.43069
33	7	-0.00835	-0.50087	-1.09249	-0.00877	-0.21043	-1.16667

34	5	-	-	-	-	-	-
		0.01362	0.81706	1.95516	0.01148	0.27552	1.37348
35	9	-	-	-	-	-	-
		0.00963	0.57769	1.52842	0.00659	0.15824	1.18507
36	29	0.01067	0.64042	0.80756	0.0434	1.04153	2.67994
37	55	0.03844	2.30644	2.50625	0.13843	3.32227	3.60077
38	3	0.0483	2.89778	3.68307	0.04184	1.00412	1.82092
39	11	-	-0.1853	-	0.0163	0.39113	2.17784
		0.00309		0.73068			
40	9	0.00944	0.5661	1.10218	-	-	-
					0.00013	0.00321	0.00925
41	3	0.05265	3.15875	4.17834	0.0477	1.1449	2.34184
42	23	0.00844	0.50624	0.91773	0.01424	0.34167	0.7242
43	7	0.0483	2.89778	3.68307	0.04184	1.00412	1.82092
44	7	-	-	-	-	-	-
		0.05161	3.09648	5.73579	0.05233	1.25585	4.62121
45	43	0.03369	2.02141	2.5151	0.00723	0.17361	0.2273
46	5	0.04919	2.95133	3.82938	0.04027	0.96652	1.76937
47	5	-	-	-	-	-	-
		0.03592	2.15532	3.50269	0.02272	0.54521	1.29068
48	5	0.05027	3.01635	3.96638	0.03857	0.92568	1.59646
49	7	-	-	-	-	-	-
		0.02696	1.61752	2.66172	0.01421	0.34094	0.82181
50	9	0.06741	4.04453	7.50422	0.12396	2.97505	5.80808
51	7	0.00194	0.11655	0.17001	0.03395	0.81489	1.79277
52	17	0.04752	2.85099	3.42679	0.03442	0.82611	1.55141
53	9	0.04065	2.43901	2.41091	0.02904	0.69706	1.05171
54	11	-0.0326	-	-	-	-	-
			1.95574	3.49046	0.03439	0.82541	2.39353
55	29	0.00824	0.49452	0.67651	0.03678	0.88273	2.08469
56	7	0.0162	0.97225	0.62534	0.11951	2.86813	2.21283
57	43	-	-	-	0.00848	0.20362	0.70567
		0.00513	0.30779	0.60935			
58	7	-	-	-	-0.0735	-	-
		0.06026	3.61589	6.82435		1.76395	5.74707
59	5	0.02404	1.44264	1.8731	0.00068	0.01625	0.02712
60	29	0.00555	0.33296	0.41616	-	-	-
					0.00222	0.05335	0.07269
61	13	0.02433	1.45955	1.67352	0.00346	0.08296	0.16389
62	33	0.03157	1.89419	1.52651	0.0626	1.50249	2.53525
63	5	-	-	-	-	-	-
		0.00366	0.21974	0.77956	0.00234	0.05604	0.48343
64	7	0.04699	2.81952	4.104	0.04587	1.10083	2.17528
65	7	0.02715	1.62905	2.84393	0.00051	0.01222	0.02737
66	9	0.05185	3.111	4.16197	0.0458	1.0991	2.27302
67	21	0.03061	1.83677	2.51392	0.00673	0.16148	0.27942
68	9	0.07112	4.26712	7.32012	0.13771	3.30513	5.35302

69	21	-0.01129	-0.67729	-0.40009	0.04525	1.08592	0.73338
70	9	-0.04507	-2.70395	-3.52818	-0.03255	-0.78117	-1.63478
71	37	0.03568	2.14101	3.58683	0.04746	1.1391	2.27554
72	11	0.04745	2.84727	3.35286	0.02943	0.70632	1.2272
73	9	0.00562	0.33711	0.48899	-0.0092	-0.22091	-0.82108
74	15	-0.04261	-2.55669	-4.59073	-0.06652	-1.59643	-2.65903
75	27	0.01587	0.95223	1.20645	0.04658	1.11787	2.74345
76	9	0.04815	2.88877	3.80771	0.03927	0.94243	1.6957
77	11	-0.00912	-0.54749	-2.40256	-0.00734	-0.17608	-1.93006
78	15	0.02875	1.72525	2.97918	0.03144	0.75467	1.61717
79	9	0.04616	2.76975	3.92917	0.0433	1.03912	2.20945
80	9	0.02714	1.62855	4.02452	0.02142	0.51407	1.682
81	5	0.03769	2.26124	2.83607	0.00797	0.19125	0.34974
82	7	0.00457	0.27408	0.40162	-0.00991	-0.23776	-0.54533
83	5	0.04709	2.82539	3.68379	0.04208	1.01001	1.89401
84	31	0.01359	0.81517	1.07298	0.04359	1.04616	2.5582
85	11	0.05478	3.28697	4.20808	0.04929	1.18287	1.97827
86	5	-0.00958	-0.57488	-0.65738	0.01203	0.28873	0.78747
87	7	-0.03254	-1.95232	-3.43474	-0.03527	-0.8465	-2.35123
88	103	0.00034	0.02031	0.02768	-0.03436	-0.82455	-1.25931
89	5	0.0008	0.04825	0.14514	-0.00988	-0.23717	-1.04661
90	35	0.00462	0.27693	0.7843	0.01805	0.43332	1.64639
91	5	0.0277	1.66206	2.53592	0.01251	0.30019	0.57587
92	11	0.03445	2.06723	3.46002	0.05124	1.22983	3.21865
93	15	0.04456	2.67335	3.77941	0.04114	0.98726	1.88158
94	7	0.013	0.78013	2.18045	0.01972	0.4733	1.92803
95	3	0.02036	1.22164	2.01451	0.00245	0.05881	0.13458
96	11	0.04742	2.84542	3.47684	0.0332	0.79675	1.35501
97	5	0.00507	0.30435	0.97567	0.01341	0.3218	1.93589
98	11	0.05068	3.04076	4.08341	0.04265	1.02372	2.10432
99	7	0.01149	0.68947	0.68829	-0.02008	-0.48198	-1.09019
100	5	0.07139	4.28355	4.72352	0.09197	2.20731	4.8384