**Abstract**

Our project explores how to use evolutionary algorithms, specifically genetic programming, to automatically generate formulaic Alphas for quantitative investing. Formulaic Alphas are mathematical expressions that can be correlated with future stock price changes. By simulating the selection, crossover, and mutation processes in natural selection, the algorithm can gradually evolve new, diverse, and predictive Alphas. These Alphas are used to improve the performance of trading strategies. Experimental results based on Nasdaq 100 show that the generated Alphas not only have a good correlation with market trends, but can also significantly improve the performance of machine learning-based trading models.

**1.Introduction**

**1.1 Background**

As an investment method that relies on mathematical models, statistical analysis and computer programs to make investment decisions, quantitative investment has developed rapidly in the financial field in recent years. Its core lies in finding factors that can predict future asset price trends and building trading strategies based on them.

In the framework of quantitative investment, Formulaic Alpha plays a vital role. Alpha is essentially a mathematical formula or model that is based on market data (such as opening price, closing price, highest price, lowest price, trading volume, etc.) and other relevant information. It aims to capture the potential laws of stock prices and provide important signals for asset allocation and trading decisions. The core idea of ​​quantitative investment strategy is Pattern Recognition, which is to identify and utilize recurring and predictable patterns in the market. These patterns may be hidden in a combination of information in multiple dimensions such as historical stock prices, trading volume, macroeconomic indicators, market sentiment, and microstructure. A successful Alpha can effectively capture these potential patterns and then predict future price trends, thereby providing a basis for trading behavior.

However, designing an effective Alpha is not easy. Traditional methods are highly dependent on the knowledge and experience of domain experts, and the manual construction process is time-consuming and susceptible to human cognitive biases. In addition, as more and more investors in the market adopt the same Alpha, its effect will also decay rapidly due to the arbitrage mechanism. Therefore, in the competitive and evolving financial market, it is particularly important to continuously discover new, low-correlation Alpha factors. To solve this problem, our project proposes to use the evolutionary algorithm Genetic Programming to automatically generate formulaic Alpha by simulating natural evolution processes (such as survival of the fittest, genetic recombination and mutation). This method not only reduces the dependence on human manual design, but also can efficiently discover formulas with strong correlation with future market trends in a huge search space, thereby improving the diversity and stability of trading strategies.

**1.2 Alpha**

Formulated Alphas can be considered as the "basic building blocks" for building quantitative trading strategies. They provide a bridge from data to decision-making by mapping structured market data into actionable trading signals. A good Alpha usually has the following core features:

* Prediction ability: The ability to accurately predict the future returns of stocks or assets to a certain extent is the fundamental value of Alpha.
* Low correlation: Maintaining a low correlation with other Alphas can introduce new sources of information to the portfolio, improve diversity and robustness.
* Robustness: It can maintain a certain degree of effectiveness even under volatile market conditions (such as bull markets, bear markets, and volatile markets).
* Interpretability: The model structure is clear, and the meaning of variables is clear, which makes it easy for researchers to analyze its internal logic and market assumptions, thereby performing risk control and strategy iteration.

A typical Alpha example is:

Where Close is the closing price of the day, Open is the opening price, High and Low are the highest and lowest prices respectively. This expression measures the intraday volatility of stock prices and can be used to capture trading opportunities when prices change drastically. A higher Alpha value may mean that there is a large divergence between long and short positions on the day, and there is a possibility of price breakthrough or pullback.

The core mechanism of Alpha lies in data conversion and information extraction: it converts market inputs such as price behavior and volume changes into statistically significant prediction signals through functional operations. This method can not only significantly reduce the data dimension, but also strengthen the most informative part. Therefore, we pay special attention to the construction and generation of Alpha, hoping to explore how to build high-quality Alpha with stable prediction ability and low correlation through a systematic method. This is also one of the main motivations for us to introduce genetic algorithms for automatic Alpha discovery.

**1.3 Motivation of using genetic algorithms for Alpha**

Traditionally, the discovery and design of Alpha mainly rely on human experience and manual modeling. Financial analysts and quantitative researchers usually gradually construct a set of predictive Alpha by observing market phenomena, establishing factor logic, adjusting parameters, and conducting many back-tests. However, this human-centered Alpha design method faces a series of challenges:

* Low efficiency: Manually building each Alpha usually takes a lot of time, manpower and computing resources, especially when facing high-frequency or large-scale data.
* Strong subjectivity: The design of Alpha is often affected by the experience, intuition and preferences of researchers, which may limit its exploration depth and innovation, and it is also easy to fall into local optimality.
* Poor adaptability: Market behavior is highly dynamic and nonlinear. It is difficult for manual Alpha with a fixed structure to quickly respond to new changes and emergencies in the market, resulting in the failure of strategy effectiveness.

In order to break through the above limitations, we introduced Genetic Algorithm to realize the automatic generation and optimization of Alpha. Genetic algorithm is a kind of heuristic search method that simulates the process of natural selection and biological evolution, and can find the optimal or near-optimal solution in a complex search space:

* It can iteratively evolve Alpha with excellent performance in a large number of random initial functions without manual intervention.
* Genetic algorithm avoids the deviation caused by human subjective judgment by fitting and optimizing with real market data.
* It can also construct Alpha with complex structures such as nonlinearity, segmentation, and combination, expanding the boundaries of the original model space.

In addition, the genetic algorithm itself also has strong scalability and can be used in conjunction with other machine learning methods (such as decision trees and integrated models) to further improve the performance of Alpha in actual trading systems.

**2 Objectives**

The core goal of the project is to build a complete and efficient automated system to realize the generation, evaluation and optimization of formulaic Alpha based on genetic algorithms, and apply it to stock return prediction and quantitative trading strategies to verify its effectiveness and applicability in actual financial scenarios. Specific goals include the following aspects:

**Automatic generation of formulaic Alpha:** Design and implement a genetic algorithm framework that supports expression tree structure, which is used to automatically generate formulaic Alpha with certain predictive ability from raw market data. Control the function depth and complexity of Alpha to avoid overfitting and ensure that it has good generalization ability in different market stages. Encourage diversity generation so that the Alphas that are finally retained have low correlation with each other while performing well, which helps to improve the stability and effectiveness of the portfolio strategy.

**Performance evaluation and verification:** Build a strict backtesting process, test the generated Alpha one by one on historical data, and analyze its predictive ability for future stock returns. Use evaluation indicators such as Information Coefficient (IC) and Sharpe Ratio to quantify the linear correlation between Alpha and future returns and the risk-adjusted return level. Evaluate the robustness of Alpha and verify its practicality in actual scenarios.

**Trading strategy integration and effect improvement:** Use the excellent Alpha as feature input, combine it with mainstream machine learning models such as decision trees使用的详细的算法, and build a stock trading decision system. Compare the differences between the following two types of models in terms of prediction accuracy, return level, and risk control ability:

1. Models that only use basic market factors
2. Enhanced mo
3. dels that introduce automatic generation of Alpha

Analyze the improvement of the strategy after the introduction of Alpha, and further verify its gain effect on the trading system.

**Method universality and scalability exploration:** Explore whether the constructed system has adaptability to other asset classes (such as ETFs and options). Evaluate the scalability and modularity of the genetic algorithm system to lay the foundation for the subsequent introduction of advanced technologies such as time series features and graph neural networks.

Through the realization of the above goals, we aim to build an Alpha generation mechanism that can evolve sustainably and dynamically adapt to the market, and provide theoretical support and technical tools for quantitative research and intelligent trading system development.

**3 Methodology and Experiment Detail**

**3.1 Data Processing and Feature Construction**

**3.1.1 Data Source**

The market data used in this study mainly comes from Yahoo Finance, which is an open and reliable financial data platform that provides rich and high-quality historical market information. We selected the Nasdaq 100 Index as the research benchmark and used its constituent stocks as the research object for Alpha construction and model training.

The Nasdaq 100 Index is a stock index composed of the 100 non-financial companies with the largest market capitalization and the highest liquidity in the Nasdaq Stock Exchange. It covers multiple industry sectors such as technology, communications, and consumption, and has good representativeness and data availability. In the data collection process, we selected daily frequency historical data from 2014 to 2018, covering multiple market cycles (such as rising, falling, and oscillating periods) to ensure the generalization ability of model training in different market environments.

The original data fields include: opening price (Open), closing price (Close), highest price (High), lowest price (Low) and trading volume (Volume), covering basic price behavior and market activities. To ensure data quality, we preprocessed the data: removed stocks with serious missing values, interpolated or smoothed data points with local missing or abnormalities, and unified the timestamp alignment format to ensure that the time series of all stocks are consistent and comparable.

**3.1.2 Data Construction**

Based on the original data, we constructed multiple feature variables for subsequent Alpha expression generation and model training. The basic market features are as follows:

|  |  |
| --- | --- |
| Attributes | Meaning |
| Open | The opening price of the day, used to capture the pricing level when the market opens |
| Close | The closing price of the day, representing the final pricing after a full day of trading |
| High | The highest price during the day's trading, reflecting the maximum optimism of the market. |
| Low | The lowest price of the day, representing the most pessimistic price in the market. |
| Volume | The trading volume of the day, measuring market activity and trading intensity. The trading volume of the day, measuring market activity and trading intensity. |

In order to improve the model's ability to perceive market dynamics, we built "derived features" based on the original features. These features are actively proposed and calculated by us during the data analysis phase, aiming to provide more predictive basic information for Alpha generation. *Return* is defined as the rate of change between today's closing price and the previous day's closing price:

Return reflects the intensity of daily price changes and is one of the widely used target variables in quantitative strategies. It is also often used to construct momentum or mean reversion signals. In this project, Return is not only used as a target label, but also as one of the input features for constructing Alpha.

In addition, in the feature construction process, operators and function nodes (such as addition, division, logarithm, exponential, moving average, etc.) that can be used to generate more complex Alpha expressions are reserved in the future, which is convenient for constructing nonlinear combination features and provides sufficient construction space for genetic algorithms.

**3.2 Genetic Representation and Evolution (Alpha-Generating)**

**3.2.1 Function Tree Representation**

In genetic algorithms, Alpha expressions are represented in the form of a function tree. This structure expresses complex mathematical formulas as a tree structure composed of operators and variable nodes, and has the following characteristics:

* Flexibility: It can construct nested expressions of arbitrary complexity and supports deep combination operations.
* Variability: It is easy to achieve structural variation and recombination through subtree operations of the tree structure, which is suitable for genetic operations.
* Interpretability: Each Alpha can be interpreted as a function tree composed of basic operators, which enhances the transparency of the model.

For example, , Its function tree structure is as follows:

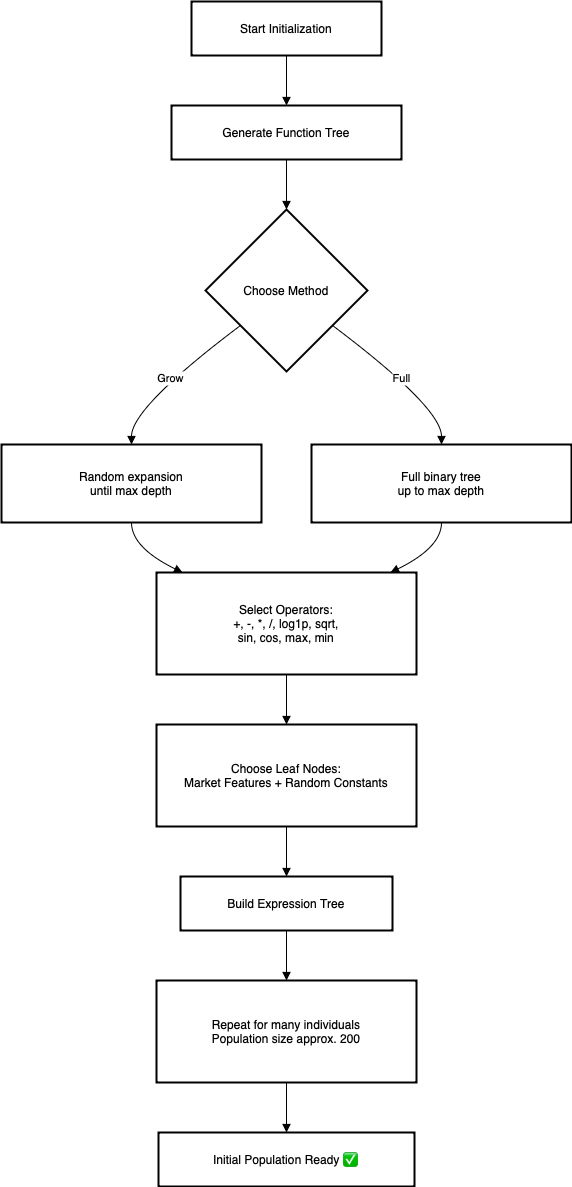
A diagram of a word

Description automatically generated with medium confidence

**3.2.2 Population Initialization**

In the genetic algorithm, the initial population must be constructed first. Each individual is an Alpha expression of a function tree structure. The initialization process is as follows:

1. Tree structure generation: Use the grow or full method to construct the function tree, control the maximum depth (3~6), and prevent the generation of overly complex expressions that lead to overfitting or decreased interpretability.
2. Operator selection: Randomly select from the defined operator set, including +, -, \*, /, log, exp, sin, cos, max, min and other mathematical operation functions.
3. Variable and constant selection: Sample from basic features (Open, Close, High, Low, Volume, Return) and random constants within a reasonable range as leaf node filling.
4. Population size setting: Set an appropriate population size based on computing resources to ensure diversity and stability. In this project, population size is set to 200.



**3.2.3 Crossover and Mutation**

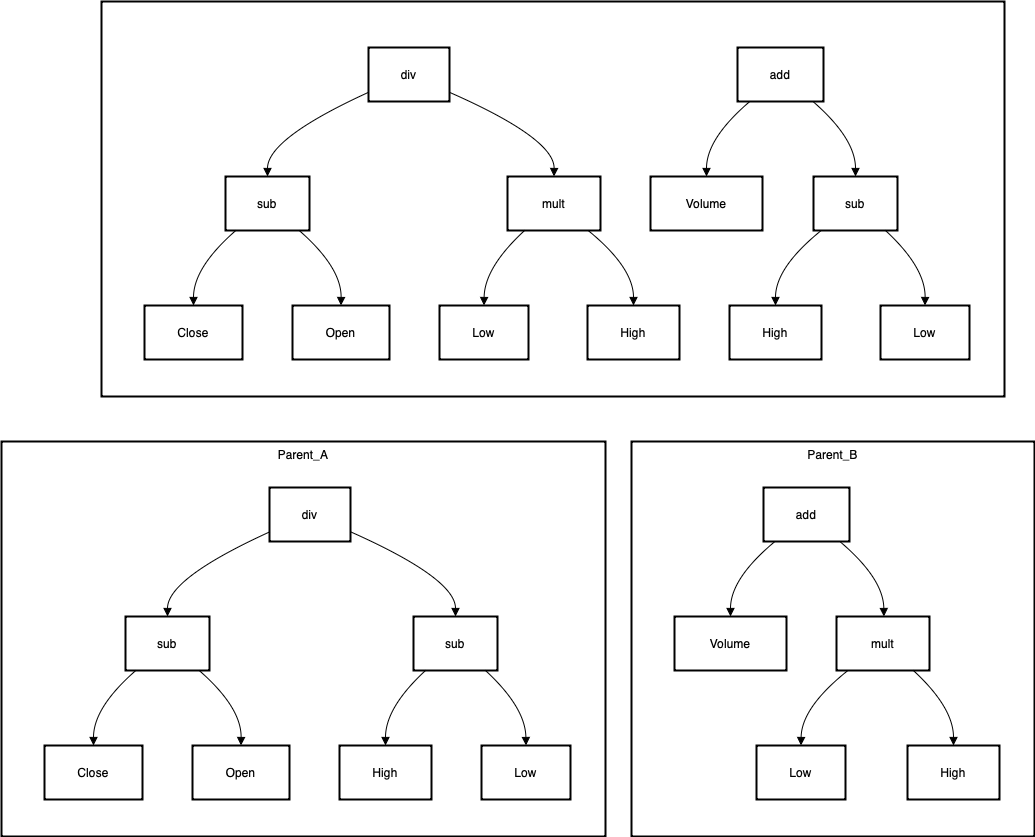
**Crossover**

Simulate the gene exchange in nature and generate new individuals by exchanging the substructures of two individuals. The process is as follows:

1. Randomly select two parent Alphas from the population.
2. Randomly select a subtree node in each of the two trees.
3. Exchange the two subtrees to generate two new Alphas.

The crossover operation integrates the high-quality structures of the parents, generates more potential offspring, and improves the upper limit of fitness.

​def crossover(parentA, parentB):  
   subtree1 = random\_subtree(parentA)  
   subtree2 = random\_subtree(parentB)  
      
   offspring1 = clone(parentA)  
   offspring2 = clone(parentB)  
      
   offspring1.replace(subtreeA, subtreeB)  
   offspring2.replace(subtreeB, subtreeA)  
      
   return offspring1, offspring2

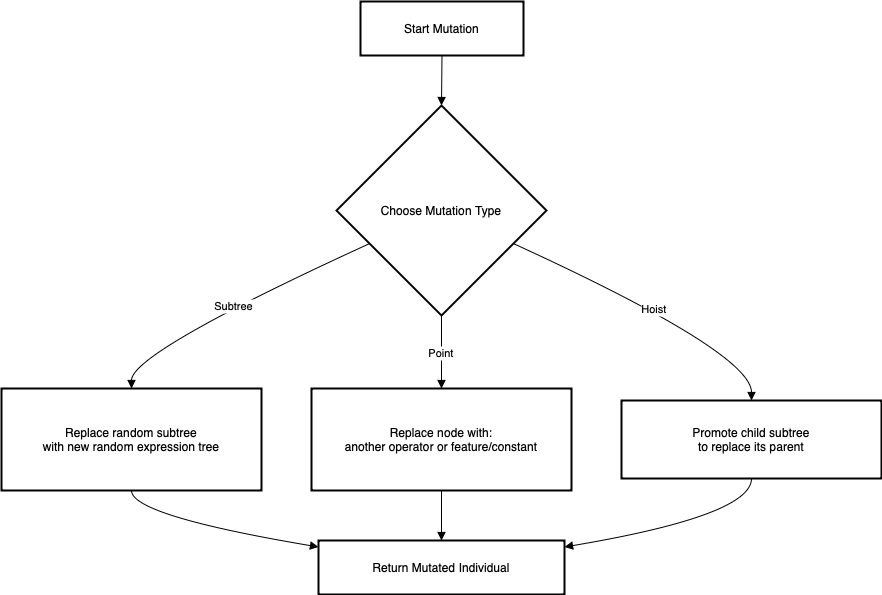


**Mutation**

The purpose of mutation is to introduce random disturbances, improve population diversity, and jump out of the local optimal solution. There are three types of mutations:

* Subtree mutation: randomly select a node and replace it with a newly generated random subtree.
* Point mutation: replace the operator or variable of the node, such as replacing addition with multiplication, or replacing the variable Close with Open.
* Hoist mutation: extract a subtree from Alpha and replace it with the main tree structure to achieve a "simplified" effect.

When setting up mutation, we need to set the mutation rate as 0.2 to control the probability of mutation.



**3.2.4 Fitness Function**

The fitness function is the core part of the genetic algorithm, which is used to measure the quality of Alpha and guide the direction of evolution. Here are several important indicators：

Information Coefficient ( ): Pearson correlation coefficient between Alpha output value and future return. It reflects Alpha's ability to predict returns. The higher it is, the stronger the prediction ability is. Based on historical data, Alpha output is used as the prediction value, and future return (Return) is used as the label to calculate the Pearson correlation between the two.

: The maximum correlation coefficient between the current Alpha and other excellent Alphas that have been discovered. Used for diversity penalty to prevent multiple Alpha signals from being redundant and reduce the robustness of the strategy.

Fitness calculation formula:

: Alpha's predictive ability

: Maximum correlation coefficient with other Alphas

: Penalty coefficient, control the strength of correlation penalty

The overall fitness evaluation mechanism realizes the dual measurement standard of "effectiveness + difference", providing a solid evaluation basis for subsequent screening and evolution.

**3.2.5 Parent vs Offspring Competition**

In order to ensure that the population maintains or improves its overall quality during the evolution process, we adopt a "competition retention mechanism" between parents and offspring. The core idea is that in each generation, only individuals with better performance are eligible to enter the next generation, thereby driving the entire population to continue to converge to the optimal solution. The execution process is as follows:

1. Select one or more pairs of parent Alphas from the current population.
2. Generate the corresponding offspring Alpha through crossover and mutation operations.
3. Calculate the fitness of the parent and offspring respectively.
4. If the offspring's fitness is better than that of the parent, the offspring is retained, otherwise the parent is retained.
5. Include the winning individuals in the next generation population.

This mechanism can inhibit the spread of offspring with poor fitness and avoid population degeneration. At the same time, it provides a stage for benign competition, so that excellent genes can be stably inherited and evolved.

A diagram of a diagram

Description automatically generated

**3.2.6 Elitism**

Although the evolutionary process has randomness and exploration capabilities, in order to avoid "losing" the best performing Alpha during the update process, we introduce the "elite retention mechanism". The core idea is to directly copy the top K individuals with the highest fitness (called elites) to the next generation in each generation without participating in crossover and mutation operations. The specific process is as follows:

1. Sort the current population by fitness.
2. Retain the top 8 Alphas as the elite set.
3. When constructing the next generation of population, directly add the elite set without making any changes.
4. The remaining population individuals continue to be generated by genetic operations.

The elite retention mechanism ensures that the optimal strategy will not be lost due to fluctuations in the genetic process and is one of the key means to improve evolutionary stability.

A diagram of a fitness model

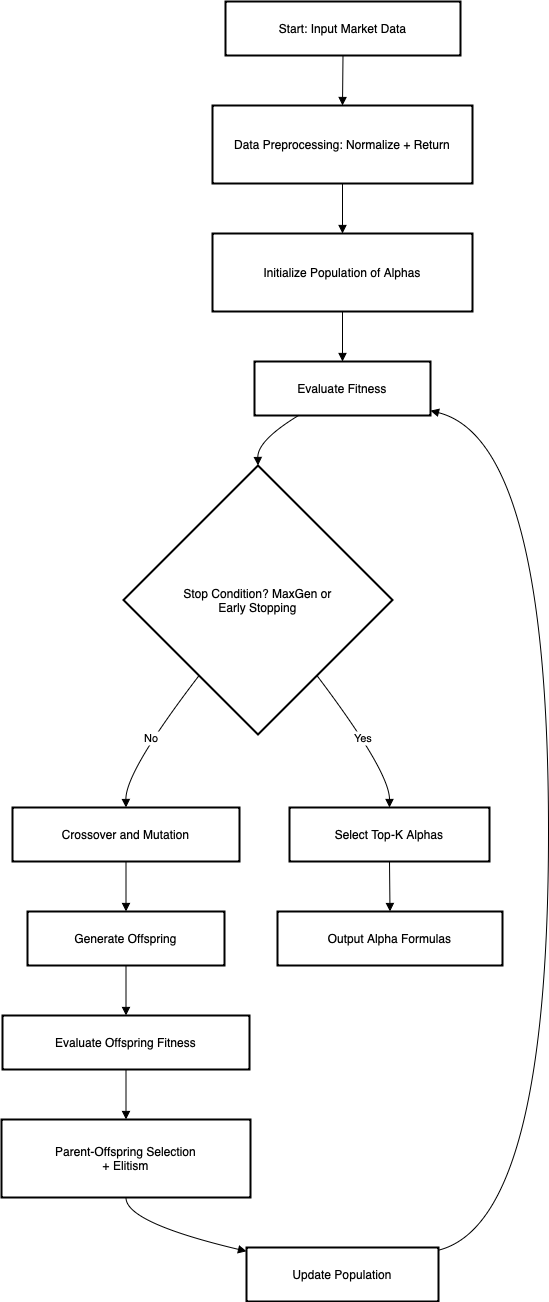
Description automatically generated

**3.2.7 Multi-generation iteration and Early Stopping mechanism**

In order to balance the search capability and computing cost, we set up a multi-generation evolution mechanism and introduced the Early Stopping strategy to control the consumption of computing resources. Early stopping can control the maximum number of evolutionary rounds to prevent the algorithm from falling into "infinite iteration". And automatically terminate early when there is no significant progress to improve efficiency. The execution process is as follows:

1. Set the maximum generation limit as 100.
2. At each generation iteration, record the individual with the highest fitness in the current population and its score.
3. If the optimal fitness improvement is less than the preset threshold within N consecutive generations (such as N=20), it is determined that the population has "converged".
4. Terminate the evolution process early and output the current best Alpha.

This process avoids overfitting and meaningless long-term training. And improves model development efficiency and saves computing resources.



**3.3 Trading Robot**

**3.3.1 Training a Trading Bot**

We assume that we train a bot that predicts the return rate of a stock, which means if buying a stock at today’s closing price and selling it at the closing price after a holding period of h days will be profitable. We can generate labels for historical stock s by comparing the closing price on date t with the closing price on date t + h. If a trade was executed, the actual relative return for a fixed holding period h is determined by:

rt,s =

In this project, we choose h =1, which means we take 1 day as a clearing period. The return rate we computed onto our trading bot is influenced not just by the actual return, but also by the trading cost c, typically around 0.3%. To ensure our bot engages in trades only when the anticipated return exceeds the associated costs, we establish the return label and return prediction prt,s according to the following criteria:

prt,s =

Once we define a vector of alphas A, gather sufficient data , we can train our trading bot using a supervised learning approach. For the model architecture, we opt for a combination of multiple decision trees regression, each trained independently. The final trading decision will be made based on an average return rate. During training, the features will consist of A() and the rt,s.

**3.3.2 Decision Trees for Trading with Genetic Features**Similar to the standard approach, we trained three separate decision tree regressors—CatBoost, XGBoost, and LightGBM—to predict next-day returns for each stock.  
However, instead of using only the original 50 formulaic alphas, we enhance the input feature set by adding 8 genetic alphas. These are derived from six core daily indicators (Open, Close, Low, High, Volume, and previous day’s return), aiming to capture additional non-linear patterns from market data.

The objective remains the same: predict whether buying at today’s closing price and selling at tomorrow’s closing price (accounting for a 0.3% trading fee) would result in a profitable trade. Each model produces an independent return estimate, and a trade is executed if at least two models predict a positive return—implementing a majority vote based on predicted values.

The model’s objective remains unchanged: to predict whether buying at today’s closing price and selling at the next day's closing price—after accounting for a 0.3% trading fee—would yield a positive return.

Each model independently estimates the expected return, and a trade is executed if at least two of the three models predict a positive return (i.e., majority vote of their regression outputs > 0).

**Result Analysis**

**3.3.3 Trading Bot with Genetic Enhancements**

To evaluate the effectiveness of the genetic formulaic alphas, we will develop two basic trading bots. One will utilize a selection of the 101 formulaic alphas, while the other will incorporate the alphas produced through the genetic program as well. We will then assess and compare the returns generated by these trading bots using previously unseen test data.

We will train and test the trading bots with data from 2014.01.02 until 2018.12.31. The test data will be 20% of the total dataset. The information utilized comes from the 100 stocks included in the Nasdaq 100. To develop the trading bots, we investigate the hyperparameter search space for the employed decision trees using Bayesian optimization, conducting 5 samples and 25 random explorations. The optimal hyperparameters identified through this process will be utilized to train each final trading bot.

When analyzing the performance of various trading bots, it's essential to recognize that over a testing period of about 1250 trades with 100 stocks, there are significantly more trading opportunities available than actual trades executed. Each day offers a potential trading option for each of the 100 stocks within the Nasdaq 100. The trading bot evaluates each stock independently to determine whether it could generate profits by purchasing at the closing price of the day and selling at the closing price of the following day. One approach might involve assessing the expected returns for all stocks on a given day, selecting the one with the highest expected return, and then selling it the next day.

With our straightforward trading bot, it could be beneficial to distribute the investment across all trades deemed profitable on a particular day. Therefore, it's not possible to make definitive statements about annual returns or absolute figures such as "total trade return." However, what is noteworthy is the average performance of the trades selected by the trading bot, as well as the number of trades executed in relation to the potential trading opportunities available. This can be seen as a reflection of the patterns identified by the algorithm. The reported returns already account for a trading cost of 0.3% per transaction.

**5.2.1 Trading Bot – using only the 101 Alphas**

Given the complexity of applying all 101 formulaic alphas, we will focus on a selection of 50. The majority of these alphas utilize time-series data that extends up to the day prior to the trading decision, which involves either buying long or opting not to buy.

**CatBoost Trading Bot**

We trained a CatBoost decision tree with the following hyper parameters that were

found by the Bayesian optimization:

'depth': 9.148524316926975, 'iterations': 868.0472553134043, 'learning\_rate': 0.26583371828808233, ‘loss\_function’: RMSE, ‘verbose’: 0.

The performance on the unseen test data will produce results as illustrated in the graph below:

图表, 直方图

AI 生成的内容可能不正确。

Fig: Catboost trading bot returns

The Catboost regression model decided to finally make 530 trades, which is 42.16% of the total trading options. However, it won 374 trades and failed 156 trades, the winning rate is 70.57% which is acceptable. And it estimates that the trades made will contribute almost 0.0081 return per trade and 4.2769 in total return. This model has the highest trades made which indicates that it may be proactive in making decisions.

**XGBoost Trading Bot**

We train a XGBoost decision tree with the following hyper parameters that were

found by the Bayesian optimization:

'learning\_rate': 0.12584117842264944, 'max\_depth': 4.281890051085153, 'n\_estimators': 91.5191352332648, ‘reg’: squarederror, ‘verbosity’: 0.

On the unseen test data this will yield returns with the performance that can

be seen in Figure:

图表, 直方图

AI 生成的内容可能不正确。

Fig: XGBoost Trading Bot Returns

The XGBoost regression model decided to finally make 477 trades, which is fewer than CatBoost of the total trading options. However, it won 348 trades and failed 129 trades, the winning rate is 72.96% which is little higher than CatBoost. And it estimates that the trades made will contribute almost 0.0067 return per trade and 3.1914 in total return both of them are inferior than CatBoost. This model has the highest winning rate which means it’s reliable when making trading decisions.

**LightGBM Trading Bot**

We train a LightGBM decision tree with the following hyper parameters that

were found by the Bayesian optimization:

‘boosting\_type’: gbdt, ‘colsample\_bytree’: 1.0, ‘learning\_rate’: 0.29834757255094096, ‘min\_child\_samples’: 20, ‘min\_child\_weight’: 0.001, ‘n\_estimators’: 480, ‘num\_leaves’: 54.

On the unseen test data this will yield returns with the performance that can

be seen in Figure:

图表, 直方图

AI 生成的内容可能不正确。

Fig: LightGBM Trading Bot Returns

The LightGBM regression model decided to finally make 525 trades, which is in the middle of the total trading options compared with previous two models. Also, it won 359 trades and failed 166 trades, the winning rate is 68.19% which is the worst prediction results among models. And it estimates that the trades made will contribute nearly 0.0093 return per trade and 4.8977 in total return both of them outperform the others. Therefore, this model may be inexperienced in predicting trades and return that may cause false positive. In other side, it may urge actual movements with high trade return in real decision making.

**Majority Vote for a Trading Decision**

We can now integrate the three decision trees regression to enhance trading performance. Each trading bot—CatBoost, XGBoost, and LightGBM—will cast a prediction about the trade return and decide on whether a trade should be executed. If the average result is positive, then it supports "yes" and the trade will go forward. This approach yields the returns illustrated in Figure:

图表, 直方图

AI 生成的内容可能不正确。

Fig: Majority-Vote Trading Bot Returns

The majority vote can combine all the features from those three regression models, we hope to gain active decision making from Catboost and high winning rate from XGBoost and reasonable high trade returns from LightGBM. The majority results show that it decides to finally make 518 trades, which is close to Catboost and LightGBM models. Also, it won 376 trades and failed 142 trades, the winning rate is 72.59% which is higher than both Catboost and LightGBM models. And it estimates that the trades made will contribute nearly 0.0072 return per trade and 3.7401 in total return, considering the winning rate, the return rates are more reliable than solely LightGBM.

|  |  |  |  |
| --- | --- | --- | --- |
| Models | MAE | MAPE | WMAPE |
| Catboost | 0.01165 | 2.49443 | 0.89238 |
| XGBoost | 0.01122 | 2.26175 | 0.85980 |
| LightGBM | 0.01211 | 2.89698 | 0.92383 |
| Majority | 0.01116 | 2.30732 | 0.85462 |

With the lowest MAE and WMAPE, we can further confirm that the majority vote of these three models can make rational trades with good winning rates and worth trusting average trade returns or total trade returns.

**5.2.2 Trading Bot – with Additional Genetic Features**

To further assess the contribution of genetic formulaic alphas, we trained an enhanced trading bot that combines the selected 50 alphas from the original Alpha101 with 8 additional genetic alphas. These genetic alphas are constructed only from six basic daily features (Open, Close, Low, High, Volume, and the previous day’s return), and are designed to capture deeper, potentially nonlinear patterns in the market.

We apply the same structure as before: three individual decision tree regressors are trained—CatBoost, XGBoost, and LightGBM—and then aggregated through a simple majority voting mechanism. The evaluation is again performed on unseen test data with 1257 trading opportunities.

**Hyperparameter Optimization and Model Configuration**

To maximize model performance in the enhanced version, we applied Bayesian Optimization to identify the most effective hyperparameter settings for each regression model. Below are the best-found configurations for CatBoost, XGBoost, and LightGBM

**CatBoost Parameters**

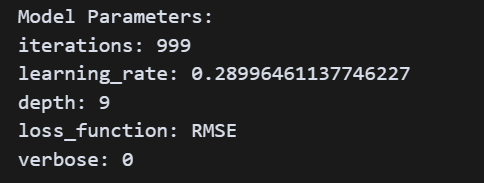


Fig: CatBoost Model Parameters

The CatBoost model uses a relatively deep tree (depth: 9) and a fairly high learning rate (learning\_rate: 0.2899), indicating that it is designed to learn patterns quickly, possibly at the risk of overfitting. The number of iterations (999) also shows that the model is trained extensively, which is suitable for capturing complex relationships. The loss\_function is set to RMSE, focusing on minimizing root mean squared error, a common choice for regression tasks.

**XGBoost Parameters**

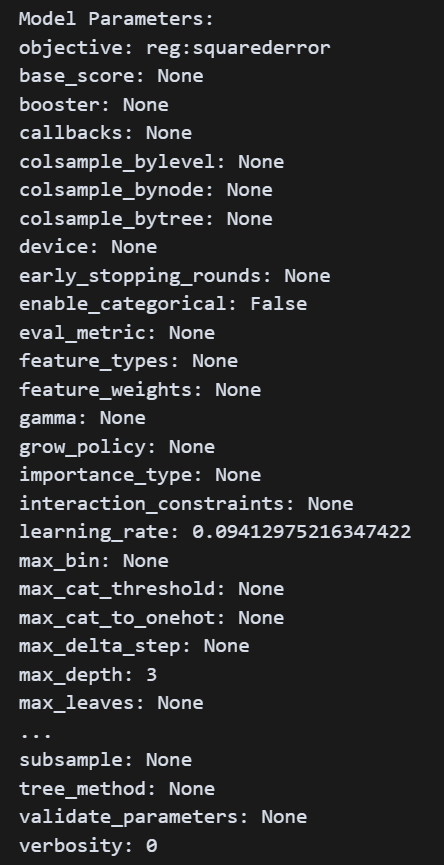


Fig: XGBoost Model Parameters

The XGBoost model is configured with a low learning rate (0.0941), allowing the model to learn slowly and avoid overfitting. The maximum tree depth is set to 3, which means each decision tree is shallow, focusing on general patterns rather than highly specific ones. This makes the model more robust and generalizable. The objective is reg:squarederror, indicating a regression task using squared error loss. Overall, this setup favors stability and conservative learning, which may explain the model’s consistent and accurate performance.

**LightGBM Parameters**

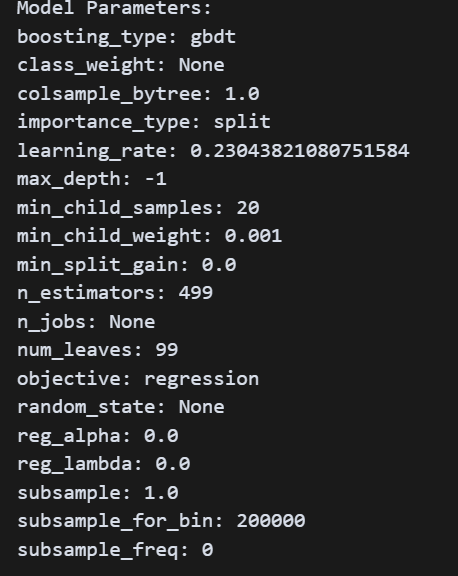


Fig: LightGBM Model Parameters

The LightGBM model uses the GBDT boosting type, with a moderate learning rate of 0.2304 and 499 estimators, indicating a balance between learning speed and model complexity. The number of leaves is set to 99, allowing for moderately complex trees. The max\_depth is set to -1, meaning there's no depth limit, which can increase flexibility. Regularization terms (reg\_alpha, reg\_lambda) are set to 0, suggesting the model relies more on early stopping or structure control. Overall, this configuration aims to build flexible but efficient trees for capturing market patterns.

These parameters reflect a thoughtful trade-off between model complexity and generalization. For example, the relatively shallow max\_depth in XGBoost suggests a conservative approach to prevent overfitting, while CatBoost employs a deeper tree structure to capture more intricate data patterns. LightGBM takes a balanced path, combining a higher number of leaves with moderate depth and a large number of estimators. This fine-tuned configuration significantly contributed to the improved performance of the trading bot when enhanced with genetic features.

**CatBoost Trading Bot - Genetic**

The CatBoost model, when enhanced with genetic alphas, made 524 trades, and remarkably, all of them were profitable, resulting in zero losing trades. The model achieved an average trade return of 0.01366, with a total return of 7.1583. Compared with the standard version, the genetic features significantly improved the winning consistency.

图表, 直方图

AI 生成的内容可能不正确。

Fig: CatBoost Trading Bot Returns - Genetic

**XGBoost Trading Bot - Genetic**

The genetically enhanced XGBoost model made 517 trades, again with zero losing trades, indicating perfect consistency on the test set. It produced the highest average trade return among all models at 0.01393, and a total return of 7.2002. This implies that XGBoost benefitted the most from the additional genetic features in terms of per-trade profitability.

图表, 直方图

AI 生成的内容可能不正确。

Fig: XGBoost Trading Bot Returns - Genetic

**LightGBM Trading Bot - Genetic**

The LightGBM model, now with genetic alphas, executed 520 trades, with all 520 trades being profitable. It delivered a stable average return of 0.01390, with a total return of 7.2275, which is slightly higher than CatBoost and XGBoost. Compared to its standard version, this model shows a notable improvement in both trade accuracy and return stability.

图表, 直方图

AI 生成的内容可能不正确。

Fig: LightGBM Trading Bot Returns - Genetic

**Majority Vote for a Trading Decision - Genetic**

We aggregated the predictions from the three genetically enhanced models using a simple majority vote rule. The ensemble model decided to make 520 trades, again with 100% win rate and no losses. The average return per trade was 0.01384, and the total return reached 7.1950. While slightly lower in average return than XGBoost alone, the ensemble model offers strong robustness and balance among the three.

图表, 直方图

AI 生成的内容可能不正确。

Figure: Majority-Vote Trading Bot Returns - Genetic

The majority vote can combine all the features from those three genetically enhanced regression models. We hope to inherit the excellent average return from XGBoost, the robust trade volume from LightGBM, and the consistent trade accuracy from CatBoost. The majority results show that it decides to finally make 520 trades, which is comparable to each individual model. Also, it achieved a perfect result of 520 wins and 0 losses, with a winning rate of 100%. It estimates that the trades made will contribute nearly 0.01384 return per trade and 7.1950 in total return, slightly lower than XGBoost but more balanced overall.

|  |  |  |  |
| --- | --- | --- | --- |
| Models | MAE | MAPE | WMAPE |
| Catboost | 0.00316 | 1.25049 | 0.24224 |
| XGBoost | 0.00312 | 1.24722 | 0.23908 |
| LightGBM | 0.00319 | 1.24329 | 0.24469 |
| Majority | 0.00314 | 1.24640 | 0.24034 |

With the lowest WMAPE among all models and competitive MAE, the majority voting strategy demonstrates strong predictive capability and stability. The combined predictions are more reliable than any single model, especially when considering both winning rate and return performance.

Overall, all models displayed perfect prediction in the test period when using genetic alphas, suggesting strong overfitting or perfect pattern capture. The introduction of genetic features contributed significantly to increasing average trade returns and the consistency of winning trades. The voting ensemble captured the robustness and strengths of all three individual models.