# Morfeo protocol

## Abstract

## Introduction

The goal of the Morfeo protocol is to run a bot which operates with the stock market based on machine learning models.

## Infrastructure

The bot which operates in the stock market is able to buy or sell assets depending on certain conditions. These conditions are adjustable by parameters. Thus, these parameters can be optimized depending on the situation of the market. The bot uses a machine learning model to find the most optimal parameters and maximize gains.

The protocol uses a software infrastructure as describes in the figure 1.0.

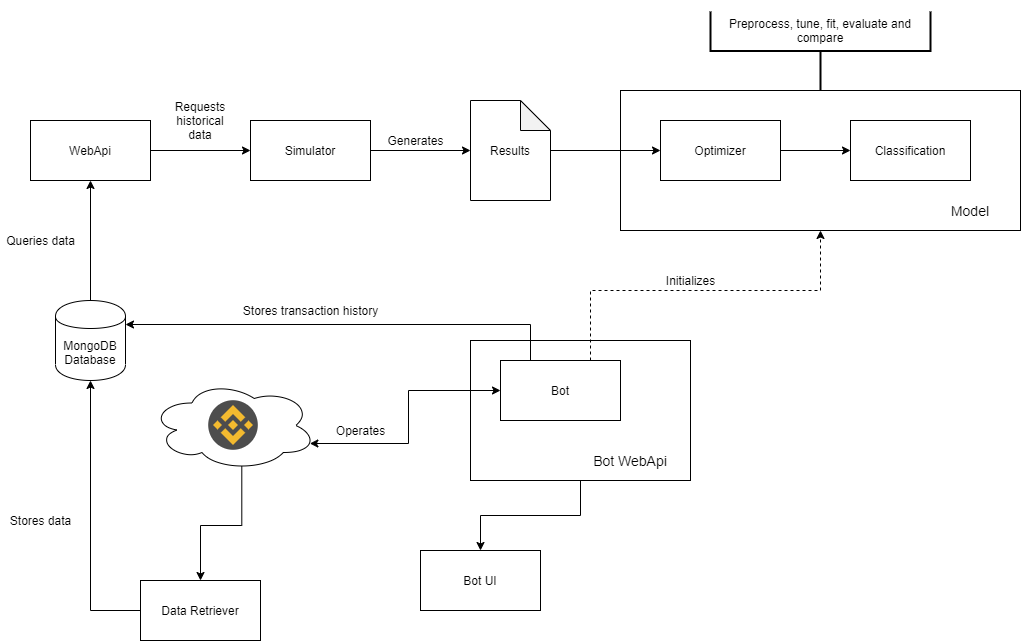


Figure 1.0 system components diagram

Such an infrastructure aims to satisfy a fluid data flow between the components. Its implementation ensures that no dependencies would lock the execution of a component. For example, if a new model is being generated with the most recent data obtained from the markets, the bot would use a previous version of the model without waiting until the update completes. Obviously, excluding the initialization of the protocol.

A requirement that the protocol follows is the scalability. That means that different models can be generated, and different instances of the bot can use them in parallel.

## Simulations

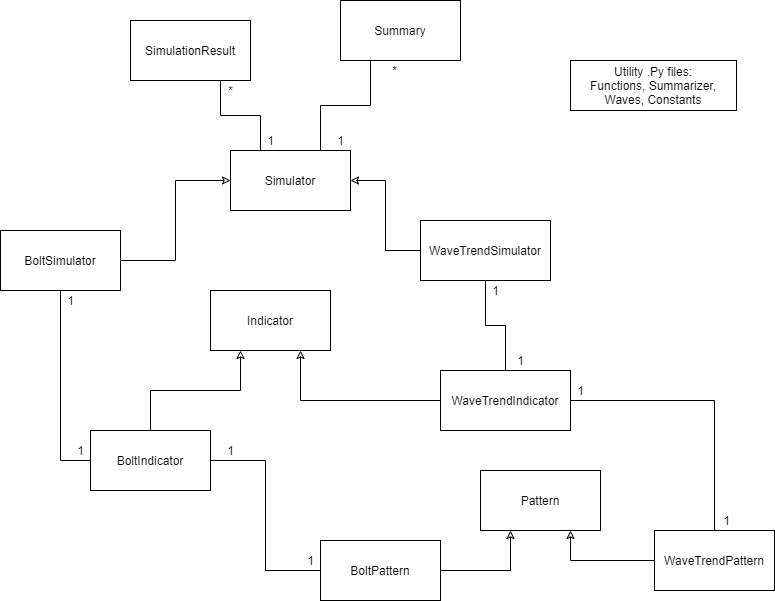
Any model that can be created is based on variables which describe statistically the market data of a certain time period, and results from simulating operations under certain conditions. The time periods used to make the simulation do never overlap to avoid overfitting when building the model. The way in which these operations are structured is described in the figure 2.0.

Figure 2.0 class diagram used to do simulations

This is a representation of the software classes used to make simulations on the marked data. It is important to note that the same classes used to make simulations, are later used to keep the state of the bot. The are three main types. The Indicator type is the central one. It is responsible of ingesting the market data and keep the state of a simulation in numeric attributes (e.g. initial budget, money spent for a buy order, the calculated gains, etc). Moreover, it has an attribute Pattern. Pattern classes have two methods which return Boolean values. That methods are buy and sell conditions. Whenever a buy or sell condition is matched, the Indicator class updates its attributes depending on the executed order. Finally, this is encapsulated in a Simulator class. The Simulator class simply generates combinations of different versions of the patterns and ingest the same data to all of them via a generated indicator class. As a result, all the simulation results are stored in a dataframe. In the figure 2.0, there are several classes which extend the super classes Simulator, Indicator and Pattern. These classes are just examples. More subtypes can be added in the future, which makes the system scalable in that sense.

Running simulations might be time consuming due to the number of calculations that the program needs to do. Moreover, different time ranges are simulated, which multiplies the total execution time for generating simulation results. To make the process more dynamic, on top of the architecture described in the figure 2.0, runner classes were created to execute simulations in parallel via multithreading. The figure 3.0 shows how this is implemented.

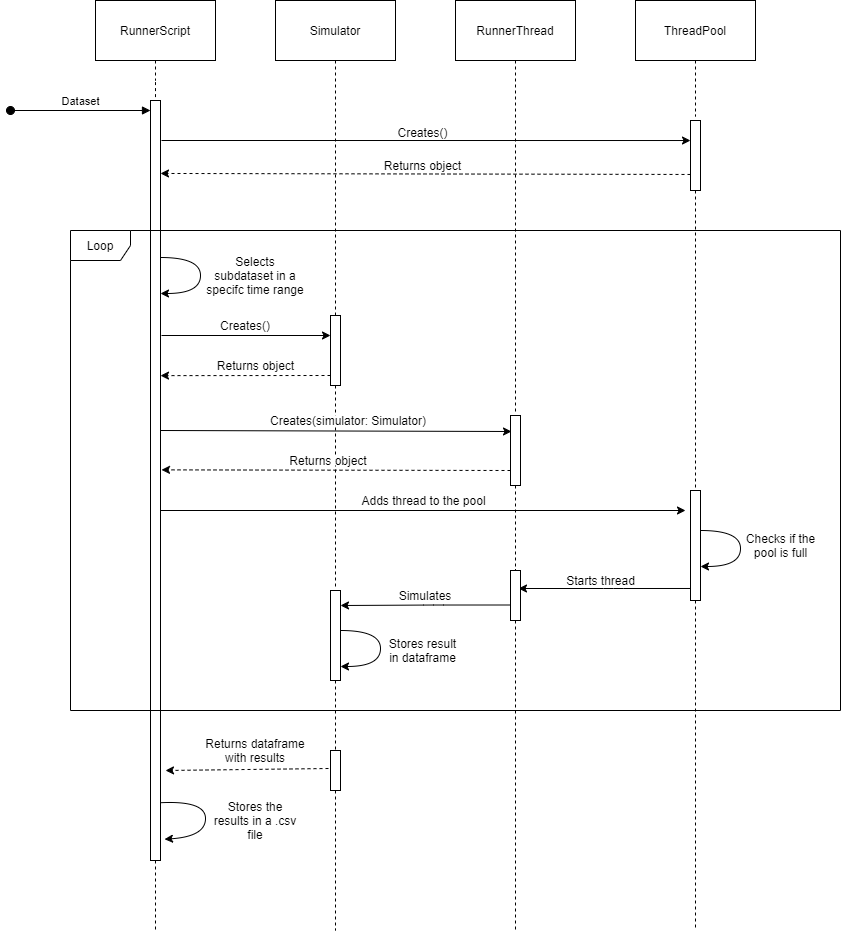


Figure 3.0 multithreading sequence diagram

The simulator object represented in the figure 3.0 can be of any of its subtypes.

## Optimization of parameters

The results of the simulations are stored in a .csv file containing the following information:

* Statistical descriptive values
* Parameters used in the simulations
* Results obtained in terms of net gains

In the optimization phase, a machine learning model will be generated in order to find the best parameters of a certain Indicator or Pattern object. A different script must be generated for each individual case due to the differences in their implementations. This means that in some cases a classification model will be needed, while in some others a regression model would be more convenient. Nevertheless, the model would always be trained using a supervised algorithm. Several ML algorithms are selected which will be compared when building the model. The process of building a ML model goes through the following steps:

1. Load the results dataset
2. Remove unnecessary columns
3. Prepare train and test data sets
4. Do hyperparameters tunning for each of the models
5. Select best configuration
6. Fit each of the models
7. Select the best model performance based evaluation metrics
8. Model deployment

By model deployment it is meant that a new software component is created which enables to process any statistical descriptive values and then find the most optimal parameters for that situation.

**Statistical descriptive values**

The statistical descriptive values used to train the model which optimizes the parameters used by the indicator are calculated from previous normalization of the market dataset. The normalized data takes values between 0 and 1. The descriptive values are:

* Mean: for central tendency
* Standard deviation: for dispersion
* Skeweness: for symetry
* Kurtosis: for outlier detection
* Entropy: for surpriseness (indicates volatility of the market)

Question Mark ‘?’

Additionally, some other values are used. These values are not strictly statistically descriptive but might contribute in information gain to train the models. These values are:

## Conclusions

## Discussion

The indicators added to the system do not necessarily perform well. However, the infrastructure implemented is designed to test them by simulating real scenarios. Then, it is possible to optimize the parameters which they use depending on certain situations.

## References