# OPTION A

**Question:**

OK, let's assume that the future will be similar to the past, so that it is reasonable to use a machine learning approach to learn patterns from the historical data and apply those to determine the optimal investment strategy for a given set of input parameters.

We want to determine, given a dataset that contains return series for 26 indexes, which combination (selection and weighting) of indexes:

* (criterion 1) provides the best "lowest cumulative return" (comparing all cumulative returns of a specified time horizon; e.g., for all historic 10-year time periods), and
* (criterion 2) at no point (e.g., within any historic 10-year time period) drops below the minimum acceptable cumulative return (e.g., 75% of the initial value).

**Which AI / machine learning tools can we use in R to determine this combination of indexes as accurately and efficiently as possible? Please provide the code for this, given our below dataset "index\_daily\_returns\_CHF".**

**Suggestion:**

To accomplish your task, we could take advantage of several machine learning techniques available in R. **A genetic algorithm might be a good option** to search the space of potential index combinations, while **an ensemble learning technique such as random forest or gradient boosting could help** to predict the cumulative return and risk.

You'll need a few packages in R to perform this task. Here's a list of suggested packages:

1. dplyr: Used for data manipulation
2. tidyquant: To perform financial analysis – tidy time series analysis, financial statistics, etc.
3. GA: Genetic Algorithm package
4. xgboost: For using the XGBoost ML algorithm
5. purrr: For functional programming

The main challenge will be the computational cost, as the number of potential combinations of index portfolios could be very large. A genetic algorithm (GA) could be a suitable approach to handle this. GA mimics the process of natural evolution to solve optimization problems and is suitable for high-dimensional, non-convex search spaces. The GA will "evolve" a population of potential solutions over many iterations, selecting the "fittest" solutions (those with the highest returns and lowest risk) at each step.

**Here's some sample code for how you might set this up.** Note that this code is not tested and serves more as a guideline or starting point for your own implementation.

# Required Libraries

library(dplyr)

library(tidyquant)

library(GA)

library(xgboost)

library(purrr)

# Load Data

# Assuming index\_daily\_returns\_CHF is already loaded

# Function to calculate cumulative returns

cumulative\_return <- function(weights, returns) {

return(sum(weights \* returns))

}

# Function to calculate risk (here defined as negative cumulative return)

risk <- function(weights, returns) {

return(-cumulative\_return(weights, returns))

}

# The fitness function calculates the total return and risk for a given portfolio allocation

fitness\_function <- function(weights) {

returns <- index\_daily\_returns\_CHF %>%

select(-Dates) %>%

apply(1, cumulative\_return, weights = weights)

# Calculate risk for each time period

risks <- sapply(returns, risk, weights = weights)

# Calculate the total return and risk

total\_return <- sum(returns)

total\_risk <- sum(risks)

# Combine total return and risk into a single fitness score

# You might want to adjust this according to your own criteria

fitness <- total\_return - total\_risk

return(fitness)

}

# Set up the genetic algorithm

ga <- ga(type = "real-valued",

fitness = fitness\_function,

lower = rep(0, ncol(index\_daily\_returns\_CHF) - 1),

upper = rep(1, ncol(index\_daily\_returns\_CHF) - 1),

popSize = 100,

maxiter = 100)

# Run the genetic algorithm

ga <- run(ga)

# Print the optimal portfolio allocation

optimal\_weights <- ga@solution

# Function to predict returns using XGBoost

predict\_returns <- function(data, weights) {

# Convert data to matrix format

data\_matrix <- as.matrix(data)

# Train XGBoost model

model <- xgboost(data = data\_matrix,

label = weights,

nrounds = 100)

# Make predictions

predictions <- predict(model, data\_matrix)

return(predictions)

}

# Predict returns

predicted\_returns <- predict\_returns(index\_daily\_returns\_CHF, optimal\_weights)

# Print predicted returns

print(predicted\_returns)

This code first defines some utility functions for calculating cumulative returns and risk (here, simply defined as negative cumulative return). Then, it creates a fitness function which calculates the total return and risk for a given portfolio allocation.

Next, the script sets up the genetic algorithm (GA). The GA is run, and the optimal portfolio allocation (i.e., the weights applied to each index) is stored.

Finally, an XGBoost model is trained using the daily index returns and the optimal weights found by the GA as the target variable. This model is then used to predict future returns.

Remember that the genetic algorithm is not aware of any constraints you might have in terms of allocation rules (e.g., maximum weight for any single index), so you should check the solution and adjust it manually if needed.

Also, bear in mind that this is a simplistic approach to portfolio optimization and risk estimation. In a more realistic scenario, you might want to consider a more complex definition of risk, such as Value-at-Risk (VaR) or Conditional Value-at-Risk (CVaR), and you might also want to include transaction costs or other constraints in your optimization. For the prediction part, be aware that using historical returns to predict future returns can lead to overfitting, and it's usually recommended to use additional features that could have predictive power over the returns of your indexes.

In terms of the XGBoost model, it may not be the best choice for time-series data as it doesn't naturally take into account the time-dependence between observations. ARIMA, SARIMA, or LSTM models might provide better performance for such data.

Please also remember to split your data into a training and a test set when using machine learning models to avoid overfitting. The model should be trained on the training set and evaluated on the test set.

Also, the code above is not explicitly using ensemble methods. Ensemble methods could be used instead of (or in addition to) XGBoost for prediction. An example of an ensemble method that could be used is Random Forest, also available in R.

# OPTION B

OK, let's assume that the future will be similar to the past, so that it is reasonable to use a machine learning approach to learn patterns from the historical data and apply those to determine the optimal investment strategy for a given set of input parameters.

We want to determine, given a dataset that contains return series for 26 indexes, which combination (selection and weighting) of indexes:

* (criterion 1) provides the best "lowest cumulative return" (comparing all cumulative returns of a specified time horizon; e.g., for all historic 10-year time periods), and
* (criterion 2) at no point (e.g., within any historic 10-year time period) drops below the minimum acceptable cumulative return (e.g., 75% of the initial value).

Which AI / machine learning tools can we use in R to determine this combination of indexes as accurately and efficiently as possible? Please provide the code for this, given our below dataset "index\_daily\_returns\_CHF". **Consider backpacking problem and linear programming tools (heuristics-based, but will bring us close to the optimum)**