HFRX Global Hedge Fund Index Replication

Two Forecasting Approaches

Doldi Filippo
Nodari Alessandro
Proserpio Lorenzo
Venturi Simone

Università degli Studi di Milano & Politecnico di Milano

June 5, 2022

The Problem

- Index replication is a famous problem in finance. An Index is often made up of hundreds of equities or even other indexes. Our goal is to replicate its returns using a much smaller portfolio.
- The HFRXGL Index is designed to be representative of the overall composition of the hedge fund universe. Their strategies often involve non-standard markets such as real estate, private equities and many financial instruments that the *common folk* cannot trade.
- Wealthy people and mutual funds invest money in hedge funds because they want to diversify their portfolio more. However, in order to do that, they have to pay management fees (usually 2% plus performance fee).

Our Goal

- We want to determine the weights of a dynamic portfolio made up of 11 generic futures (all of them commonly tradable) which replicates the returns of the HFRX Global Hedge Fund Index since 2013 till 2022.
- We used the following futures (Bloomberg's tickers only): RX1, TY1, GC1, CO1, ES1, VG1, NQ1, LLL1, TP1, DU1, TU2.
- Our ultimate goal is to find a way to deploy a strategy with similar characteristics to the hedge funds' ones without paying so much fees. In order to do that, we want at each time t to predict effectively the weights of our portfolio at time $t + 1$.

The Dataset

- We obtained from Bloomberg the weekly prices of the 11 futures and the index from 2007-10-23 to 2021-04-20.
- In both of our approaches, we started to forecast the new weights after 300 weeks, so the first 300 observations were our training set (which is incremented each time we see new data) and the other 404 observations are our testing set.
- We computed the log-returns to achieve stationarity in the time series. Whenever we refer to returns in this presentation, we mean log-returns.

Notation

From now on, we use the following notation:

- the vector of the weights of each futures in our portfolio at time t: $\mathbf{w}_t = (w^1, ..., w^{11})_t$;
- the vector of the predicted weights of each futures in our portfolio at time t: $\hat{\mathbf{w}}_t = (\hat{w}^1, ..., \hat{w}^{11})_t$;
- the vector of the returns of each futures in our portfolio at time t: $\mathbf{r}_t = (r^1, ..., r^{11})_t$;
- the return of the index at time t: r_{ind_t}.

Assumptions

We make the following assumptions:

- to replicate the index at time t we suppose to know the returns of both the futures and the index at time t and the weights of our portfolio at time $t-1$.
- our entire wealth is 1;
- C is a fixed constant which represents the trading cost per 1 unit per operation (in our case 0.02%);
- the market has no slippage nor liquidity problems;
- each year has 36 trading weeks.

Metrics

We care about two metrics:

- the Annualized Tracking Error Volatility:

$$TEV = \sqrt{\frac{1}{403}\sum_{j=300}^{703}\left|\hat{\mathbf{w}}_j \cdot \mathbf{r}_j - r_{ind_j}\right|} \cdot \sqrt{36}$$

- the Mean Annual Trading Costs to deploy the strategy (it does not take into account the cost of the initial allocation):

$$TC = C \cdot \frac{36}{404} \cdot \sum_{j=300}^{702}\sum_{i=1}^{11}\left|\hat{w}_{j+1}^{i} - \hat{w}_{j}^{i}\right|$$

Two solutions

We propose two methods, the first one focuses on achieving the minimum *TEV*, while the second one is a compromise between minimizing *TEV* and *TC* while also caring about the deployability of the strategy in the real world. In brief:

1. a *State-Space Model* using the *Kalman Filter* to forecast the new weights;
2. our *Greedy-Minimization Algorithm* plus a calibrated exponential mean to forecast the new weights.

State-Space Model

We have one observable variable, i.e. target portolio return, which is a function of:

1. an unobservable variable, the vector of weights;
2. an observable variable, the vector of (candidate) asset returns.

We can represent this using the following State-Space Model:

$$\mathbf{w}_t = A \cdot \mathbf{w}_{t-1} + \mathcal{N}(0, Q)$$

$$r_t^{ind} = \mathbf{r}_t \cdot \mathbf{w}_t + \mathcal{N}(0, R)$$

where (reasonably):

- A is the identity matrix;
- Q is the covariance of the states;
- R is variance of the index returns.

Kalman Filter

The algorithm works as follow:

1. we set $\mathbf{w}_0 = (0, ..., 0)$ and $\Sigma_0 = 10^9 \cdot \mathbb{I}$ (as suggested in reference [6]);
2. apply the classical Kalman Filter algorithm, as written in reference [2], to obtain the $\mathbf{w}_1$ and Σ_1 given $\mathbf{w}_0$ and Σ_0 (the matrix of variance-covariance of the states at time 0);
3. iterate with the initial guess equals to $(\mathbf{w}_1, \Sigma_1)$ and obtain the next one and so on.

To be precise in our model $Q = Q_t$ and $R = R_t$, indeed, after 100 observation we start to estimate Q_t with the classical Maximum-Likelihood estimator of the covariance matrix of the weights and R_t using the sample variance. To forecast we simply exploit the estimation of the weights at the previous time-step.

The minimization problem

We want our strategy to be easily deployable in the real world. So we do not want to use leverage, have a long-only portfolio, minimize the trading costs and be, at least, near the returns of our index. Moreover we want, at each time t, to be fully invested (this means that the sum of our weights at each time t is equal to 1). In mathematical notation, if there are only two times $t-1$ and t we want to know $\mathbf{w}_t$ which solve:

$$(\star) \qquad \min_{\substack{\forall j\ 0<w_t^j<1 \\ \sum_{j=1}^{11} w_t^j=1 \\ (\mathbf{w}_t\cdot\mathbf{r}_t-r_{ind_t})^2<toll}} \left\{ C\sum_{j=1}^{11}\left(w_t^j - w_{t-1}^j\right)^2 \right\}$$

where *toll* is a constant (how much our returns are far from the index returns at most).

The Greedy-Minimization Algorithm

We aim to construct a sort of "optimal" portfolio at time t given that we know the starting weights $\mathbf{w}_0$, the weights at the precedent time step and the returns of the futures and the index at the time we are considering. We use a greedy approach in order to do that, working as follow:

1. knowing $\mathbf{w}_0$ and seeing $\mathbf{r}_1$ and r_{ind_1} compute $\mathbf{w}_1$ as the solution of $(\star)$;
2. iterate to obtain $\mathbf{w}_2, \mathbf{w}_3, ...$

Obviusly this algorithm is not suitable for our forecast, because we assume to know the returns at the current time. However, we have obtained a vector of "optimal" portfolio $(\mathbf{w}_0, \mathbf{w}_1, ...)$. Afterwards, we will use this, truncated at time $t-1$, as a training set to make our prediction of the weights at time t.

The Starting Portfolio

Now we have to solve another problem: determine the starting portfolio $\mathbf{w}_0$. In order to do that we choose the minimum variance portfolio that replicates the returns of the index of the first day. In mathematical terms we choose the $\mathbf{w}_0$ as the solution of:

$$(\star\star) \qquad \min_{\substack{\forall j\ 0<w_0^j<1 \\ \sum_{j=1}^{11} w_0^j=1 \\ (\mathbf{w}_0\cdot\mathbf{r}_0-r_{ind_t})=0}} \mathbf{w}_0\Sigma\mathbf{w}_0^T$$

where Σ is the matrix of variance-covariance of the futures estimated using the Oracle Approximation on the first 200 observations (see [4] in the references). Here it seems that we made a mistake since we used informations obtained from future observations, however, remember that we will start predicting from the 300th observation.

Calibrated Exponential Mean

In order to forecast the new weights at time t we used the exponential mean of the "optimal" portfolio weights until time $t-1$. The exponential mean, given a parameter $\lambda \in (0,1)$, is simply:

$$\tilde{\mathbf{w}}_{t,\lambda} = \sum_{j=0}^{t-1} \lambda(1-\lambda)^j \mathbf{w}_{t-1-j}$$

and then normalize them:

$$\hat{\mathbf{w}}_{t,\lambda} = \frac{\tilde{\mathbf{w}}_{t,\lambda}}{||\tilde{\mathbf{w}}_{t,\lambda}||}$$

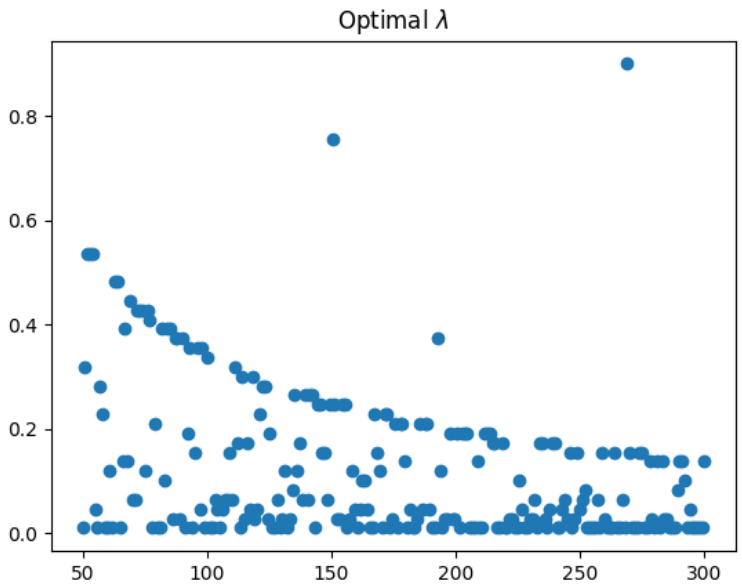
higher λ gives more importance to more recent weights, lower λ gives more importance to the trend.

Calibrating Lambda

However, we want λ to be time-dependent (more market dependent), so we utilize the observations from 50 to 300 to build a dataset with the "optimal lambdas" for each step. The optimal lambdas are the lambdas which solves, for each time t:

$$\min_{\lambda \in (0,1)} \left| \left(\hat{\mathbf{w}}_{t,\lambda} - \mathbf{w}_t \right) \cdot \mathbf{r}_t \right|$$

the following is the scatter plot of such "optimal lambdas":



We can see there are 2 "states":

1. μ^1 low lambda ($\sim$0.08);
2. μ^2 high lambda ($\sim$0.15-0.5).

With μ^2 that seems to decay over time. We try to modelize it as a Discrete-time Markov chain. First of all discretize the train dataset rounding to the nearest state. Then infer the transition matrix. So we want to infer, if λ_t is the state at time t:

$$P = (p_{ij})_{i,j}$$

where $p_{ij} = \mathbb{P}(\lambda_t = \mu^j | \lambda_{t-1} = \mu^i)$ and $i, j = 1, 2$.

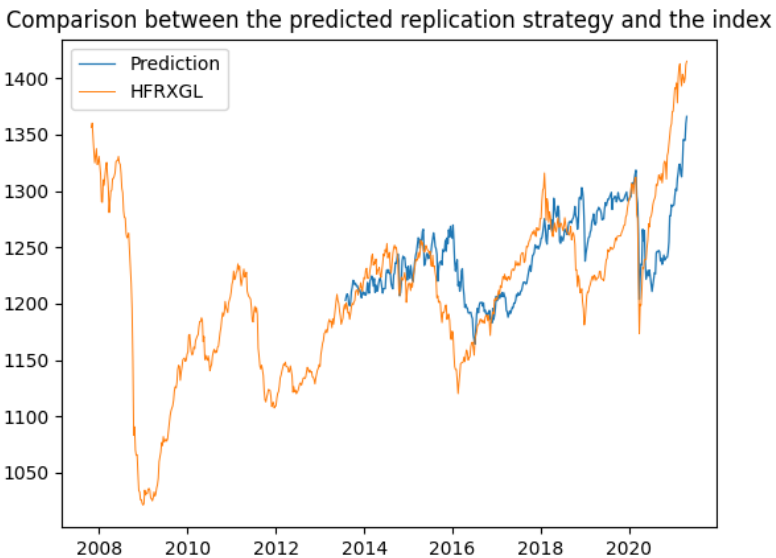
Forecasting algorithm

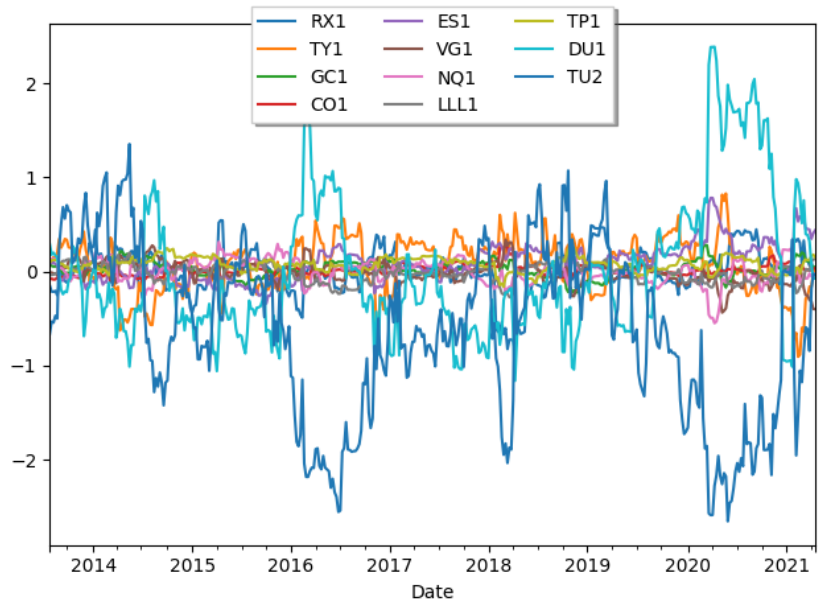
We assume that μ^2 remains constant at 0.15. Now we start to forecast from the 300th observation. To choose the λ_t and compute the prediction we use the following algorithm:

1. compute the previous state λ_{t-1} since we have already seen that data, this is simply done as we have constructed the train set in the previous paragraphs, suppose that $\lambda_{t-1} = \mu^j$;
2. if it is not the first step, update the transition matrix;
3. take the j-th row of the transition matrix, with that partitionate the interval $[0, 1]$ in a way that respects the probability in the transition matrix (see [7] in the references);
4. simulate a random uniform variable $Q \sim U[0, 1]$ and choose the state λ_t as the associated state to the interval in where Q realized;
5. compute the prediction using the exponential moving average with λ_t.

Results: State-Space Model plus Kalman Filter

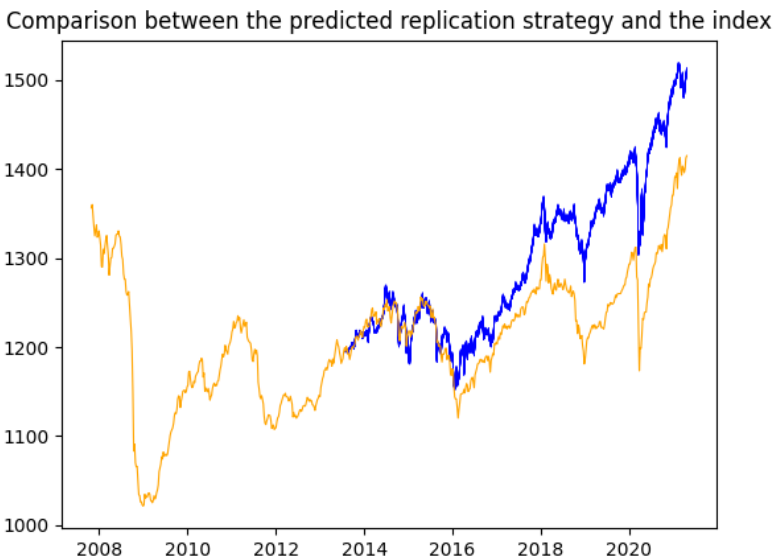
On the right side there is the visualization of the evolution of weights of the predicted portfolio, on the left there is the visualization of the value of our portfolio in comparison with the index.

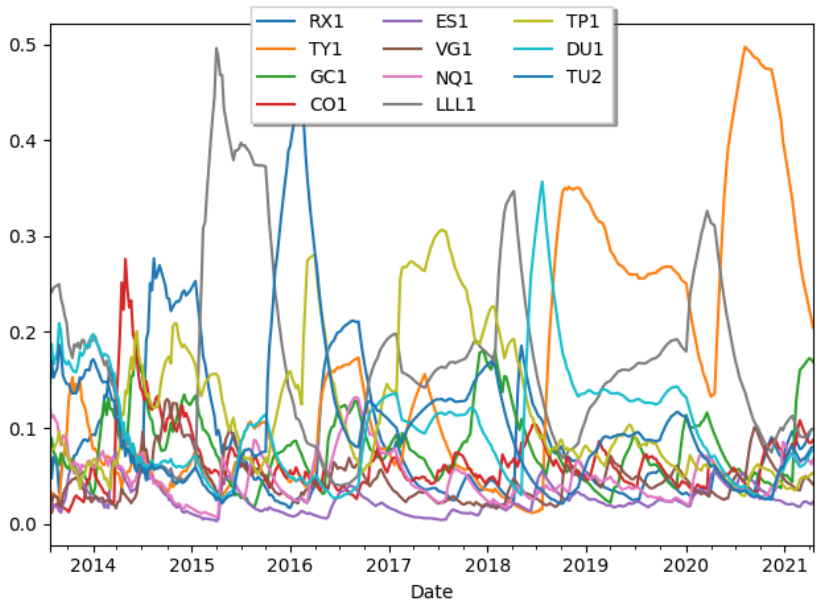




Results: Greedy-Minimization plus Calibrated ExpMean

Since it involves a stochastic component we run it 20 times to see if it is "stable". On the right side there is the visualization of the evolution of weights of only one simulation, on the left all the 20 simulations are plotted.





Analysis of TC&TEV and conclusion

The strategies obtained the following scores:

1. SSKF: $TC = 0.52\%$ and $TEV = 0.042$;
2. GMCEM: $TC = 0.039\%$ and $TEV = 0.046$.

We conclude that, GMCEM may be the better alternative since it has one tenth of the trading costs, a good TEV, use no leverage and does not require shorting futures (which sometimes may be difficult). We also calculated the IR of the GMCEM strategy using the HFRXGL as benchmark and obtained $IR = 0.13$ and the strategy has better "financial" properties than the index (higher mean, lower variance, higher skewness and lower kurtosis). As a last mathematical note we have that the problems $(\star)$ and $(\star\star)$ satisfy the KKT conditions and both the function to minimize and the constraints are convex, so we are sure to find a local minimum (may not be global however).

References

1. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html;
2. https://pykalman.github.io;
3. https://scikit-learn.org/stable/modules/generated/sklearn.covariance.OAS.html;
4. Chen et al. (2010), Shrinkage Algorithms for MMSE Covariance Estimation, IEEE Transactions on Signals Processing, 58(10), 5016-5029;
5. Ledoit, O., & Wolf, M. (2003). Honey, I Shrunk the Sample Covariance Matrix The Journal of Portfolio Management, 30(4), 110–119;
6. Analysis of Financial Time Series - Tsay, Ruey;
7. Markov Chains - Norris, JR.