Deep Learning for Finance - Deep Portfolio

T.Becker, M.Bergerot, P.de Kerdrel, A.Mascia, N.Tachet

Columbia University

Deep Learning - IEORE4720

Professor: Ali Hirsa
Teacher Assistant: Francois Fagan

May 2018

Abstract

This paper is based on the work of J.Heaton, N.Polson and J.Witte and their article *Deep Learning for Finance: Deep Portfolios*. This paper let us explore the use of deep learning models for problems in financial prediction and classification. Our goal is to show how applying deep learning methods to these problems can produce better outcomes than standard methods in finance or in Machine Learning.

1. Introduction and Objectives

In this paper, we explore the use of Deep Learning and more specifically autoencoders to learn about latent factors that drive security returns. We used Deep Learning techniques in order to find hidden structure, detect and exploit interactions in the data that are currently invisible to any existing financial theory. Our main goal was to build a portfolio i.e. find the right selections of investments that reproduces an index. Firstly, we focused our efforts on replicating the IBB, a biotechnological index.

More accurately, we identified three objectives:

- Reproduce the IBB index;
- Reproduce the IBB index with anti-correlation in periods of large drawdowns to outperform this index. If the IBB decreases by more than 5%, our *modified benchmark* should increase by 5% or more (fig. 1);
- Extend our technique to another index (S&P500).

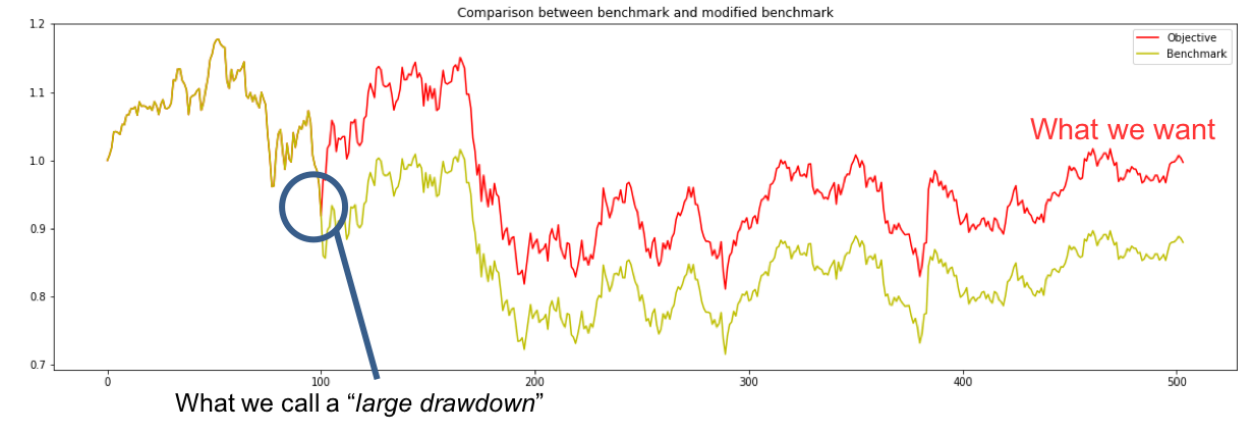

Fig. 1. IBB index vs. modified IBB index

The rest of the paper continues as follows. Section 2 introduces our deep learning framework divided into 4 phases (i.e. subsections). Section 3 is an extension of our technique to the S&P500. Section 4 concludes.

2. The Deep Learning framework

Assume that the available market data has been separated into two disjoint sets for training and validation. This four steps deep learning algorithm proceeds via auto-encoding, calibrating, validating, and verifying.

2.1. Step 1: Autoencoding

In the process of creating a deep portfolio, we first need to map the market. For finance applications, one of the most useful deep learning applications is an auto-encoder. An auto-encoder is a deep learning routine which trains the architecture to replicate X itself, namely X = Y , via a bottleneck structure. The architecture of the autoencoder is as follows:

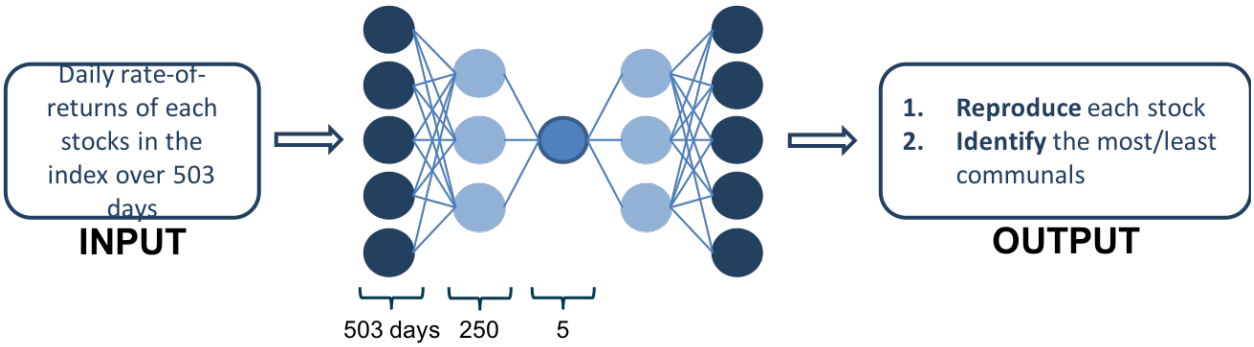

Fig. 2. Autoencoder Process

It is a fully connected feed forward neural network made of:

- an *encoder* with two layers: the input layer and one hidden layer
- a *bottleneck* layer made of 5 neurons
- a *decoder* with two layers: one hidden layer and one output layer

We feed our Neural Network with daily log rate-of-returns of each stocks in the index over 503 days. This is why our input layer has 503 neurons because we feed the autoencoder stock by stock. By doing this, we create an autoencoder that replicates the behavior of the market - as he sees every stock the same number of time.

We want our output to be exactly equal to our input - because it is an autoencoder. We can then reproduce each stock and compare those replica to their original version. The proximity of a stock to its auto-encoded version provides a measure for the similarity of a

stock with the stock universe.

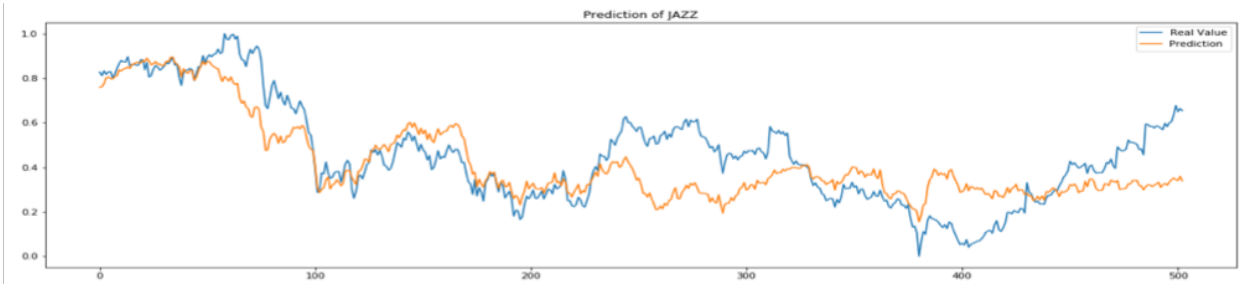

Fig. 3. Least Communal

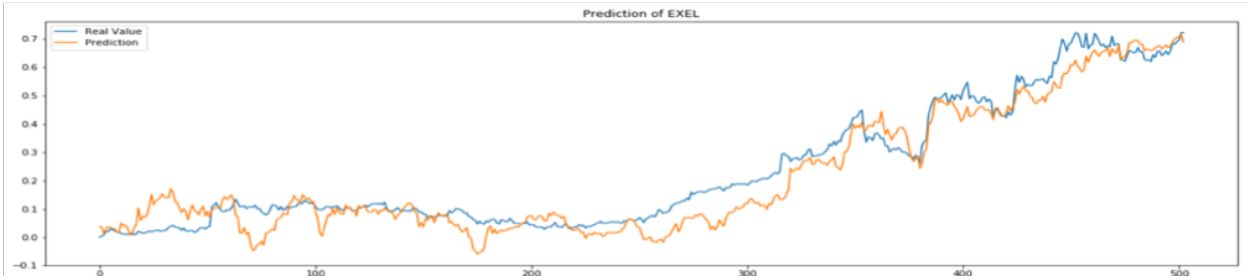

Fig. 4. Most Communal

Those who are similar to the original stock are called *most communals*, other are called *least communals*. The *most communals* are the stocks that are most highly correlated to our autoencoder and therefore to the market.

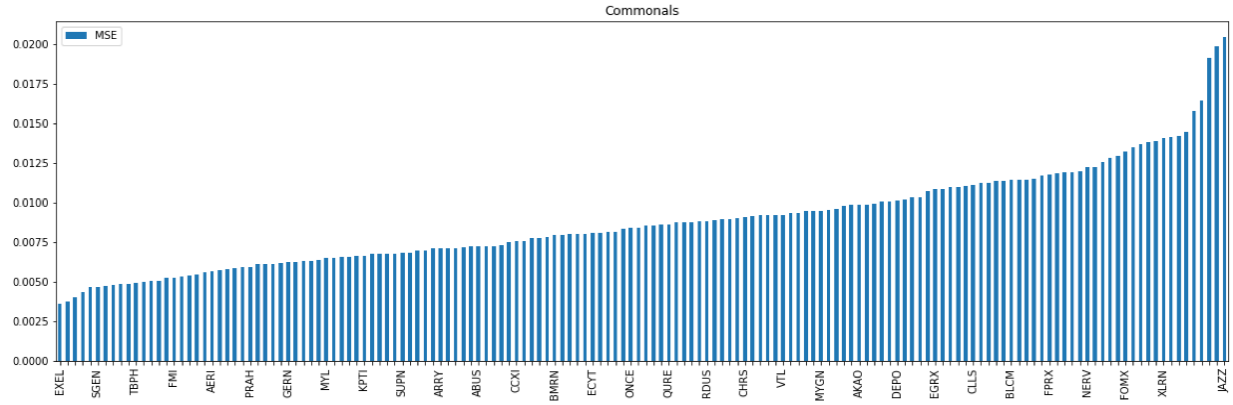

Fig. 5. Accuracy on replicating the stocks - communals

In other words, *most communals* (i.e. stocks on the left in Fig. 5) are more in the family of the IBB whereas other are less part of the co-factors driving the index.

We need to choose S stocks to replicate our index. As there is no benefit in having multiple stocks contributing the same information, we define the number of stocks in our deep portfolio by using the 10 most communal stocks plus x-number of most non-communal stocks (as we do not want to add unnecessary communal information); e.g S = 25 stocks means 10 plus 15 (where X = 15). One important fact we need to emphasize is that we prefer the least communals, as they explain the unusual moves of the markets, and will allow the creation of alpha when we will tackle the calibration part.

Before analyzing the results of our autoencoder, we thought beneficial to process a grid search to get the best parameters for the autoencoder. We thought it was a crucial part of the replication part since we want to assess the "financial" most and least communals. If we reduce the systematic error for the replication, we will end with only the idiosyncratic error of each stocks, and we will be able to interpret this as how much our stock is correlated with the market (financial most / least communals).

learning_rate	num_steps	batch_size	lambd	val_dropout	cross_len	num_encode_1	n_code	MSE
0.001	1000	10	0.2	0.85	3	250	5	0.01531761
0.001	1500	8	0.2	0.8	4	250	5	0.01637979
0.001	1500	9	0.2	0.8	4	250	5	0.01669362
0.001	1500	10	0.2	0.8	4	250	5	0.01540838
0.001	1500	11	0.2	0.8	4	250	5	0.01484443
0.001	1500	12	0.2	0.8	4	250	5	0.01463671
0.001	1500	13	0.2	0.8	4	250	5	0.01530727
0.001	1500	14	0.2	0.8	4	250	5	0.01516682
0.001	1500	15	0.2	0.8	4	250	5	0.01535057
0.001	1500	16	0.2	0.8	4	250	5	0.01558387
0.001	1500	17	0.2	0.8	4	250	5	0.01498939
0.001	1500	18	0.2	0.8	4	250	5	0.01497427
0.001	1500	19	0.2	0.8	4	250	5	0.01477058
0.001	1500	20	0.2	0.8	4	250	5	0.0152149
0.001	1500	21	0.2	0.8	4	250	5	0.01475504
0.001	1500	22	0.2	0.8	4	250	5	0.01440499
0.001	1500	23	0.2	0.8	4	250	5	0.01430068
0.001	1500	24	0.2	0.8	4	250	5	0.01451658
0.001	1500	25	0.2	0.8	4	250	5	0.01414756
0.001	1500	26	0.2	0.8	4	250	5	0.01454983
0.001	1500	27	0.2	0.8	4	250	5	0.01465802
0.001	1500	28	0.2	0.8	4	250	5	0.01476936
0.001	1500	29	0.2	0.8	4	250	5	0.01571872
0.001	1500	30	0.2	0.8	4	250	5	0.01556248
0.0004	3000	25	0.12	0.85	4	250	5	0.01300547
0.0004	5000	25	0.12	0.85	3	250	5	0.01139948
0.0004	20000	25	0.12	0.85	3	250	5	0.01188237
0.0006	1000	10	0.2	0.85	4	250	5	0.01408296
0.0004	1000	10	0.2	0.85	4	250	5	0.01354178
0.0002	1000	10	0.2	0.85	4	250	5	0.01603417
0.0001	1000	10	0.2	0.85	4	250	5	0.01837431
0.00045	1000	10	0.2	0.85	4	250	5	0.01410418
0.00035	1000	10	0.2	0.85	4	250	5	0.0141247
0.00041	1000	10	0.2	0.85	4	250	5	0.01383883
0.00039	1000	10	0.2	0.85	4	250	5	0.01450793
0.0008	1000	10	0.2	0.85	4	250	5	0.01441391
0.0008	1000	10	0.22	0.8	4	250	5	0.01569552
0.0008	1000	10	0.22	0.85	4	250	5	0.01457783
0.0008	1000	10	0.25	0.85	4	250	5	0.01459609
0.0008	1000	10	0.18	0.85	4	250	5	0.01436209
0.0008	1000	10	0.16	0.85	4	250	5	0.01428247
0.0008	1000	10	0.14	0.85	4	250	5	0.01349476
0.0008	1000	10	0.12	0.85	4	250	5	0.01343946
0.0008	1000	10	0.1	0.85	4	250	5	0.01420758
0.0008	1000	10	0.1	0.87	4	250	5	0.01295696
0.0008	1000	10	0.12	0.91	4	250	5	0.01300525
0.0008	1000	10	0.12	0.89	4	250	5	0.01336471

Fig. 6. Example of results obtained performing a grid search - AutoEncoder

We used the following parameters:

- $learning\_rate = 0.0004$
- $num\_steps = 5000$
- $batch\_size = 30$
- $lambda = 0.12$
- $val\_dropout = 0.89$

The learning_rate variable corresponds to the hyper parameter we used in the stochastic gradient descent (SGD) algorithm during the training phase. It is an iterative method for minimizing an objective function that is written as a sum of differentiable functions ; in other words SGD tries to find minima and maxima by iteration. The learning rate controls the size of the step at which we lead the function towards a minima/maxima. Choosing a proper learning rate can be difficult. A learning rate that is too small leads to painfully slow convergence, while a learning rate that is too large can hinder convergence and cause the loss function to fluctuate around the minima or even to diverge.

The batch_size variable indicates the size of the batch for one computation of the gradient for SGD. Larger mini-batches allow us to reduce the variance of the stochastic gradient updates because we consider the average of the gradients in the mini-batch. A direct consequence of this is that it allows us to take bigger step-sizes, which means the optimization algorithm will make progress faster.

Regularization is also performed to our autoencoder in order to reduce overfitting during the training phase. For our Neural Network we chose to use L2 Regularization and the lambda variable is the value of our regularization parameter. The effect of regularization is to penalize large weights and make the network to prefer to learn small weights.

Another regularization technique we use is the dropout technique. Unlike L2 regularization, dropout doesn't rely on modifying the cost function. Instead, in dropout we modify the network itself. At each training stage, individual nodes are either dropped out of the network with probability 1-p or kept with probability p. The value of p is our val_dropout parameter and num_steps is the number of iterations of the training stage.

2.2. *Step 2: Calibration*

By using another Neural Network, we output the weights of the stocks in our portfolio that best reproduce the index. We feed the Neural Network with all the stock prices day by day. Hence the S input neurons. In output, we used a *softmax* activation function to make sure our weights are included in the interval $[0, 1]$. Therefore, in this strategy, we are only long. Note that allowing a long-short strategy would require an activation function with outputs within $[-1, 1]$. We did not create such a strategy but we are convinced this might be an interesting extension to this project.

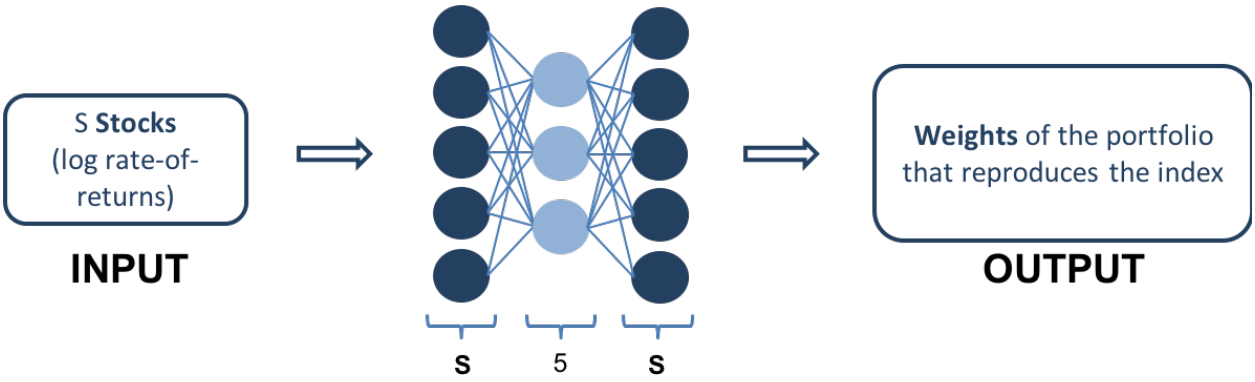

Fig. 7. Calibration Process

On the graphic below, we plotted the returns of the *calibration*, the *benchmark* (i.e. index) and the *modified benchmark* (the modified IBB in periods of large drawdowns).

We therefore used the following parameters:

- $learning\_rate = 0.0004$
- $num\_steps = 1000$
- $batch\_size = 10$
- $lambda = 0.1$
- $val\_dropout = 0.88$

2.3. *Step 3: Validation*

Using the test set, we computed the results for different portfolios (i.e. different values of S).

Fig. 8. Results out-of-sample for different values of S - IBB index

In each cases, we are able to outperform the benchmark. For the current example (beating the IBB index), we have amended the target data during the calibration phase by replacing all returns smaller than 5% by exactly 5%, which aims to create an index tracker with anti-correlation in periods of large draw-downs. This might explain why we are performing better than the benchmark - or it might be luck, we show in section 3 that we are able to replicate those results with another index.

In order to convince ourselves of the results, we decided to implement this portfolio on a Quantopian algorithm. This way, we backtested our take position and hold strategy over 6 months. Here are the results:

Fig. 9. Results out-of-sample for S = 65 stocks

As we can see, the risk metrics are proving us right: the alpha is positive over this period of time (we actually outperform the IBB) and the beta is a little bit greater than 1, meaning that we're reproducing the amplitude of the benchmark's moves.

2.4. Step 4: Vizualisation of the Deep Frontier

We plot below the efficient deep frontier of the considered example, which plots the number of stocks used in the deep portfolio against the achieved validation accuracy. The result is counter-intuitive and different than the one presented in the paper *Deep Learning for Finance: Deep Portfolio*: the more we increase the number of stocks in portfolio, the worst our accuracy is. We have not been able to explain this result, in the light of the paper's results.

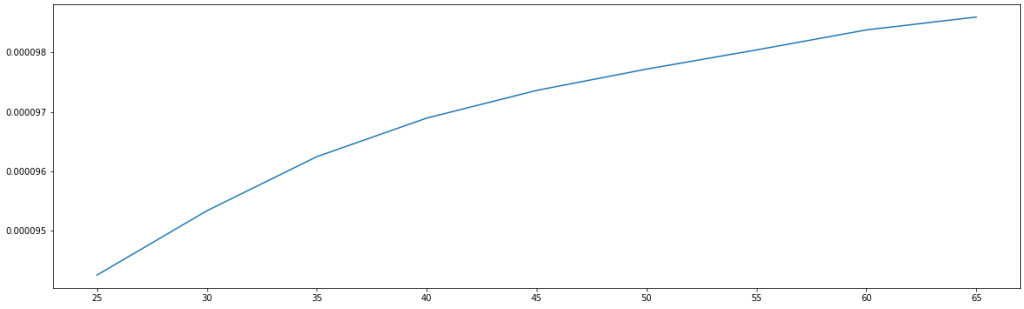

Fig. 10. Deep Frontier

3. Extension to S&P500

Our last objective was to extend this paper to other indexes - to make sure we were not just having luck with the IBB. We chose to work with the S&P500. The method is the same and we use the same set of parameters for both Autoencoder and Calibration. Again, our out-of-sample results are pretty satisfying. We are able to follow the benchmark (which is similar to the modified benchmark as there is no large drawdowns over the period we studied).

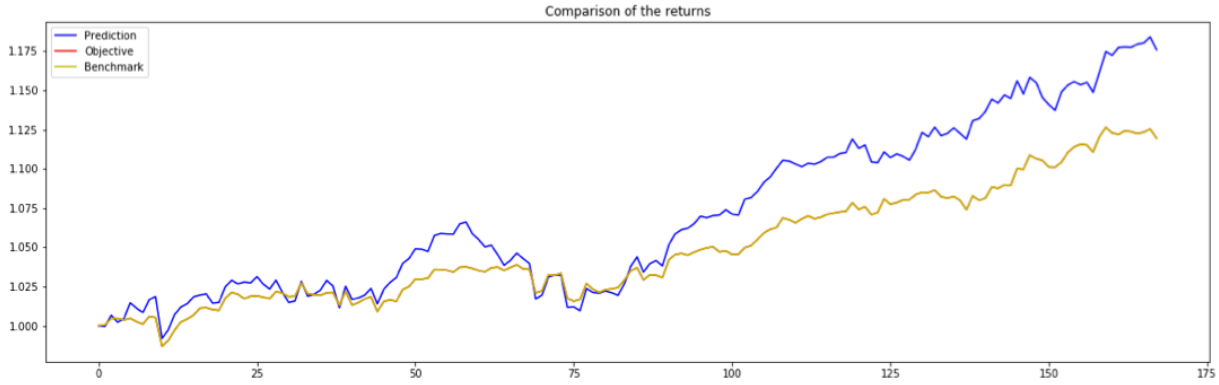

Fig. 11. Results out-of-sample on S&P500

An analysis of our strategy using Quantopian also proves that we outperform the index - the β is close to 1 and α is positive.

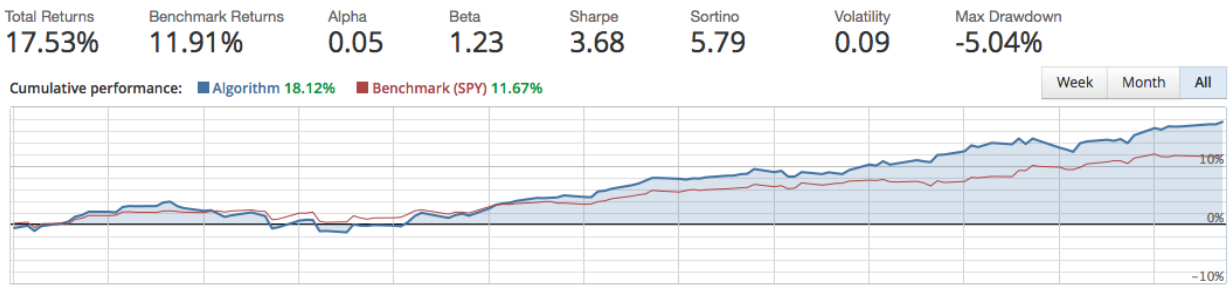

Fig. 12. Analysis of the S&P500 strategy out-of-sample - Quantopian

4. Conclusion

Our examples on Deep Portfolio present just one way to implement deep learning models in finance. However, we also wanted to prove that Deep Learning was making a difference in our case. Are we really achieving better performances than with Machine Learning techniques for instance? This is the question we asked ourselves. Can we achieve the same results with simpler techniques?

In order to answer this question, we tried to do exactly the same as before but using ML techniques. Our method was the following:

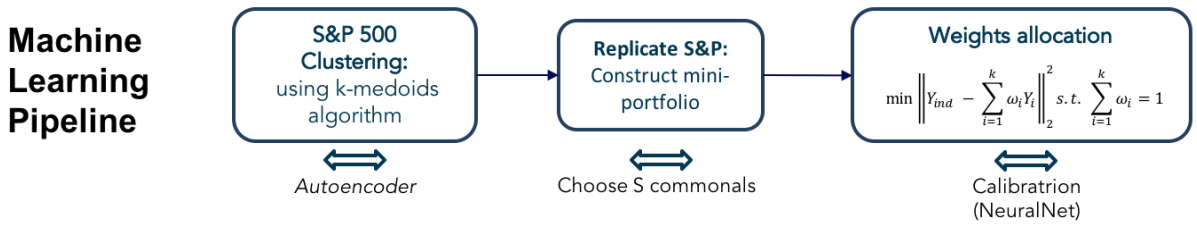

Fig. 13. Our Machine Learing Process

- A clustering of the S&P to construct a mini-portfolio. This is similar to what we did with the AE - but simpler. The idea is again to measure the similarity of a stock with the stock universe. The larger a cluster is, the better it represents the index. The *most communals* are therefore stocks at the center of these large clusters;
- A decision process on the stocks that will compose our portfolio. Here we chose to take only the medoid (stock closest to the center of a cluster) of each cluster.
- A weight allocation using optimization techniques similar to the calibration we did using a Neural Network.

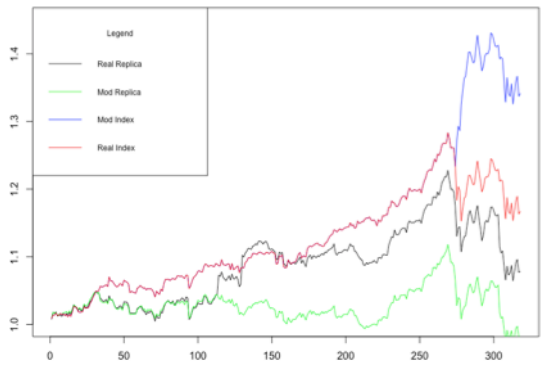

Fig. 14. Results using ML techniques on the S&P500 with 100 stocks

These results are definitely not satisfying. We have not been able to reproduce nor outperform the S&P500 using this simpler technique. This lets us think the use of Deep Learning techniques can help achieving better results.

Last, we would say that if we would have done only Machine Learning with this project, we might never have validated the intuition of the paper. Even though reproducing an index with fewer stocks is already satisfying (because we can take the same directions we would have for indexes but with less exposure), our Deep Learning method is powerful enough to fit a curve with no financial meaning - the so-called calibration curve - with financial tools - the stocks of the IBB or S&P.