Backtesting a Momentum Trading Strategy

Fernando Lozano

***Abstract*—In this note we explore a cross-sectional momentum trading strategy based only on the history of past prices: buys the previous winners (i.e., stocks with the highest returns in the recent past) and sells the losers (i.e., stocks with the lowest past returns). The data set has approximately 1500 stock instruments listed on the US stock exchange, and it contains daily data from February 2008 to July 2017. We build a random forest and deep learning model to back-test our results against a baseline strategy using logistic regression. In terms of returns, we find that the deep learning model outperformed the other models, since holding a portfolio that selected stocks based on their past performance and then held it for one month, generated an extra average annual return of about 5.79% above what would have been expected. However, the baseline model has the highest Sharpe ratio. We believe that a tuning-intensive search to find the optimal hyperparameters for neural networks could generate beneficial results.**

I. Introduction

Numerous studies have shown that a profit can be realized by buying outperformed stocks and selling the underperforming stocks (Jegadeesh & Titman, 1993). This strategy also is known as momentum or relative strength. In other words, a stock's relative strength over the previous month to three months typically predict its relative performance for the following month to three months.

In this note, we show how machine learning techniques can be applied to replicate the momentum trading strategy, the strategies we examine include portfolios with overlapping holding periods. The objective is to compare three different methods and describes the advantages and disadvantages of each of them. The fundamental question is how to address backtest overfitting, if there was an easy answer then hedge funds would reach higher returns with certainty. Therefore, the goal of this note is to present experimental results using machine learning techniques rather than present a winner strategy.

The vast majority of our work is testing the hyperparameters, since small adjustment can have outsized impacts, which why it is important to have a robust testing process and methodology.

The rest of this note is organized as follows: Section II describes the data set and provides a decomposition of the frequency of bets during the overlapping training periods. Section III includes the methodology and the reasons we choose the model. Section IV presents the training procedure and Section V the results.

II. Data

We must start this discussion by firstly introducing the details of the cross-sectional training and testing periods. The feature space is given as follows: in the month m we select a series of portfolios based upon the performance of 12 monthly return for the month $m-2$ through $m-13$, and 20 daily returns corresponding to month m. We have a total of 33 input variables for each stock, including an indicator if the holding period falls in January. The features are fed into the models to calculate the probability for each stock to outperform the cross-sectional median return of all stocks in the holding month $m+1$. Our target variable is equal to 1 if the normalized cumulative sum of past 20 trading days return is above the median value of the trading universe for that day, otherwise is 0.

The data comprises 1500 stock instruments listed on the US stock exchange, and it contains daily data from February 2008 to July 2017. There are 3,547,259 samples in the dataset. The overlapping training set is defined as $t-(t-4)$ where $t \in \{2012, 2013, 2014, 2015, 2016\}$, and the testing set is $t+1$.

Next, we compute the ratio of longs per year that shows the proportion of bets involved in long positions (Figure 1). For a neutral strategy the value is 50%. Note that all values oscillate around 50%, reflecting a representative market condition. Additionally, a decrease in the ratio is associated with higher volatility, as it is depicted for 2009 and 2010.

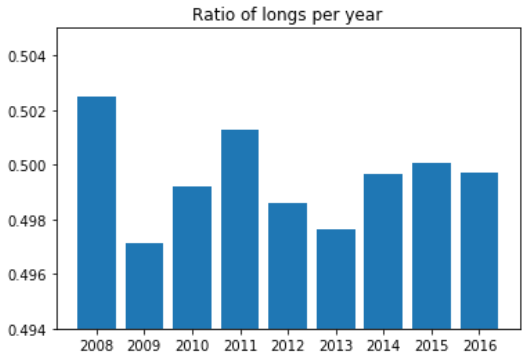


Fig. 1. Ratio of longs over the cross-sectional training periods

III. Machine Learning Models

Due to the complexity of the stock market dynamics, stock price data is often filled with noise and is known to be prone to overfitting. In our case, overfitting means that our strategy can generate a Sharpe ratio above 1.0 when the actual Sharpe ratio may be zero.

To address this concern, we use a random forest model to reduce the variance of the accuracy and limit overfitting. We build experiments with 5-fold cross-validation during hyperparameter optimization, implementing the RandomSearchCV function. Once we get a narrow selection of hyperparameters, we use a 10% hold-out strategy over the training set to evaluate their performance and choose the model with the highest accuracy.

Deep learning experiments were carried out by trial and error, using a for loop to select the best hyperparameters, while controlling for overfitting with L2 regularization, and then testing the hyperparameters using a 10% hold-out validation strategy with an early stopping policy.

A. Random forest classifier

We use the random forest model because it is a tractable model and easy to train, as well as it helps to reduce overfitting. Moreover, some studies have shown the advantages of deploying random forest to develop trading strategies similar to what we are implementing (Moritz & Zimmermann, 2014., and Krauss, Christopher & Do, Xuan Anh & Huck, Nicolas, 2016), arguing the explanatory power attributed to random forest to explain the performance of a stock based on its most recent returns. Given the extensive data we are analyzing, random forest is one of the fastest algorithms to train and also the hyperparameters are straightforward to understand and tune. By exploiting the benefits of random forest we are looking for a general behavior to perform well when presented with new data.

One of the main limitations of random forest is the computation time it takes to get the optimal hyperparameters, as well as the low interpretability of some hyperparameters for overlapping training sets.

B. Deep learning model

We follow the network architecture presented by Takeuchi and Lee (2013), using 3 hidden layers and a number of neurons per layer between 40 and 50. The first step that comes into consideration while building a neural network is the optimization of the parameters, and this stage is characterized by being computationally expensive. Instead, we opt for a pragmatic approach by using trail an error to determine the learning rate, a reasonable number of neurons, and see if the validation accuracy increases or decreases as we change the number of hidden layers. According to Geron (2017), 2 layers could tackle complex relationships in data.

Our model tests three different activation function: ELU, RELU and Leaky RELU. In our case, ReLU activation function outperformed the ELU and Leaky ReLU functions and has the advantage of being fast to compute; nonetheless, RELU slope abruptly jumps from 0 to 1 and it could slow down the algorithm. The default initialization strategy is Xavier and He, because it can speed up the training considerably.

The model uses L2 regularization to ostensibly prevent overfitting. We choose to include L2 instead of dropout, according to Laarhoven (2017) when L2 regularization is used together with batch normalization we are making adaptive adjustments of the learning rate. Moreover, previous studies have shown the disharmony between dropout and batch normalization (Li, Xiang, Shuo Chen, Xiaolin Hu, and Jian Yang, 2018). The L2 regularization scale is hardcoded to 0.00001. Lastly, the batch size is equal to 3000, the smaller the batch the less accurate the estimate, and the number of epochs is set to 9.

IV. Training procedure

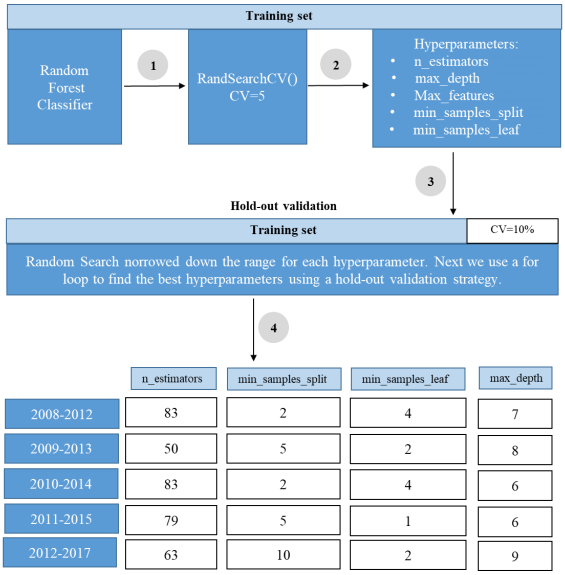
Hyperparameter tuning is an essential step in fitting a machine learning algorithm, if we don't

apply the correct approach we will end up with a poor performance in our models. The task to find optimal hyperparameters is an exhaustive one, especially when you are dealing with a vast amount of data. The training data was partitioned as follows to choose hyperparameters: 90% of the data for training and 10% for validation.

A. *Random forest classifier*

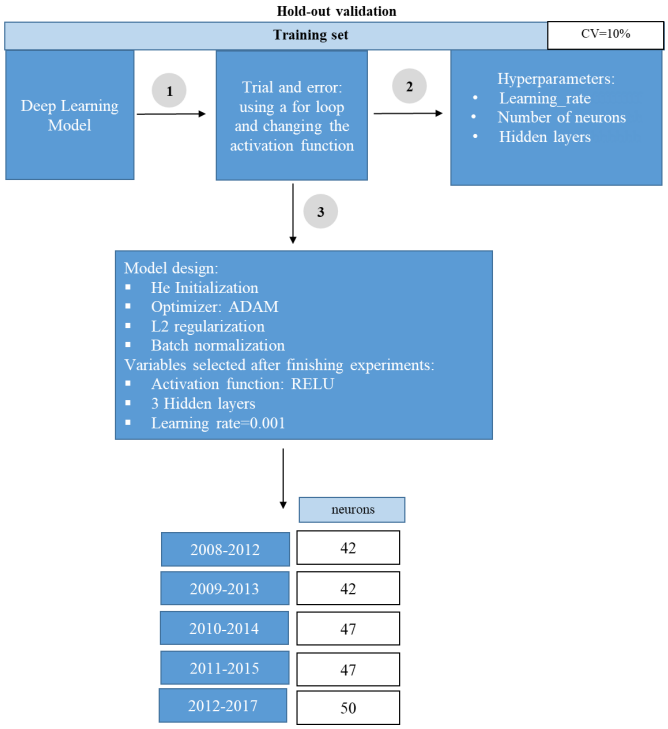
We will control for the number of combinations to optimize our random forest model, regardless of the dimensionality of the problem. Two of the most crucial parameters to train the model are the depth of the three and the number of trees. If we increase the depth of the tree, we increase the complexity of the model, and we know random forest is prone to overfitting on noisy stock data, therefore the model should not be too deep. We restrict the depth level between 4 and 8. Additionally, we set the number of trees between 45 and 105.

Figure 2 describes the training process of the random forest model. For each year, we employed the RandomSearchCV function to narrow our search for hyperparameters. The entire task is computationally expensive and time-consuming. We got an insight into the behavior of the model across a range of values for each hyperparameter. Next, we use a hold-out validation strategy, splitting the training set into 90/10, to compute the validation score using selected values for the hyperparamters that we got from the previous step. Finally, we chose the combination of parameters that achieved the highest validation score. The set of hyperparameters is presented in Figure 2 and it varies each year.



	n_estimators	min_samples_split	min_samples_leaf	max_depth
2008-2012	83	2	4	7
2009-2013	50	5	2	8
2010-2014	83	2	4	6
2011-2015	79	5	1	6
2012-2017	63	10	2	9

Fig. 2. Random forest training process



	neurons
2008-2012	42
2009-2013	42
2010-2014	47
2011-2015	47
2012-2017	50

Fig. 3. Deep learning training process

B. *Deep learning model*

The design of an deep neural network is more of an art than a science (Zhang et al., 1998), extreme care must be taken to find the optimal hyperparameters. Since there are many hyperparameters to tune, and since training a neural network on a large dataset takes a lot of time, I explore a tiny part of the hyperparameter space that is updated over time. The hyperparameters such as the number of layers, learning rate and the number of neurons in each layer are empirically tuned based on experiments. The training process is depicted in Figure 3.

We noticed the models work better using a low L2 regularization value, which helps in the optimization procedure so that the weights do not get saturated during training. Additionally, the model can perform better with 3 layers than one with 4 or 2 hidden layers. We performed some tests which yielded a higher accuracy when used 3 hidden layers.

To choose the hyperparameters we always test the validation accuracy and apply early stopping to prevent overfitting. We select the parameters

that give us the highest validation accuracy. When it concerns the number of epochs, we found that above 20 epochs the model was overfitted, in some experiments, it seems that even fewer epochs give better results. To make training manageable we choose to keep 9 epochs.

We test three activation functions, each one together with a set of values of the number of neurons between 40 and 50 and the learning rate between 0.001 and 0.01. In all the cases, RELU activation outperformed the other functions, and the lower the learning rate the highest the accuracy. Concerning the optimizer and weight initialization, we selected He initialization and ADAM optimizer given that they are appropriate for noisy stock data.

V. Results

At first, we analyze the performance of the models on the out sample data. Figure 4 depicts the results, in blue it is the highest out sample accuracy per testing period. We can observe the neural network and random forest model outperformed the baseline during 2013-2016, but the highest accuracy of 51.83% belongs to the baseline model. We presume that this outperformance is driven by the optimization of hyperparameters. Regarding the in sample accuracy in the deep learning model, the training accuracy increases with the number of epochs and leading to overfitting.

Model accuracy*

Year	Baseline Model: Logistic Regression		Random Forest		Deep Learning	
	In sample	Out sample	In sample	Out sample	In sample	Out sample
2013	51.76%	49.49%	53.64%	50.06%	54.65%	49.94%
2014	51.69%	50.39%	53.71%	50.15%	54.57%	50.60%
2015	51.62%	51.46%	52.76%	51.74%	54.34%	51.27%
2016	51.80%	49.36%	52.99%	49.29%	54.58%	49.72%
2017	51.48%	51.83%	54.77%	51.07%	54.49%	50.38%

*/ For each year, the training set corresponds to the previous 5 years.

Fig. 4. Out sample and In sample accuracy of each model

Now we analyze the performance of the cross-sectional trading strategies in terms of the yearly return and Sharpe ratio. For each testing period from 2013 to 2017, we see that deep neural networks model produces an average annual return of 5.79%, followed by the logistic regression with 5.76%, and the random forest with 4.23%. However, the baseline model has the highest Sharpe ratio. If we exclude the testing period for the year 2016 from our results, the deep learning model outperforms the other models on both metrics. Furthermore, it is interesting to note that returns start to deteriorate since 2012 in every model, indicating that beating the market is becoming more difficult.

Figure 7 and 8 in the Appendix illustrates higher yearly returns during 2008-2012 for the deep learning and random forest model. We can attribute this behavior to the power of each model to train the data compared to the baseline model.

Overall, the deep neural network model has more parameters compared with the random forest model and the logistic regression, so the performance of deep neural network model tends to increase as the amount of data grows. However, deep learning has high requirements to tune the hyperparamters and uses nested hierarchy structure to perform representation learning, so deep learning algorithms are less interpretable. On the other hand, random forest model performs well with high dimensional data, but when we work with large data the size of the trees can take up a lot of memory and time to compute. We can conclude from this section that neural networks can indeed be used in defining a profitable momentum trading strategy.

Yearly Return and Sharpe Ratio

Year	Baseline Model: Logistic Regression		Random Forest		Deep Learning	
	Return	Sharpe	Return	Sharpe	Return	Sharpe
2008	-0.056674	-0.692299	0.199600	4.654672	0.112119	1.942865
2009	0.361200	3.676245	0.465463	6.327644	0.465010	4.805855
2010	0.409331	3.224571	0.366234	4.120010	0.454794	4.241634
2011	0.134093	2.118230	0.252835	5.558153	0.173923	3.438949
2012	0.092953	1.429364	0.204690	5.753753	0.203762	3.840502
2013	0.114847	1.680922	0.084900	1.294486	0.167185	2.433415
2014	0.098261	1.818474	0.097934	1.860062	0.096976	1.435444
2015	0.097750	2.223340	0.082738	1.887559	0.115605	2.608061
2016	-0.038390	-1.367387	-0.060479	-1.658501	-0.115591	-2.614055
2017	0.015641	1.587530	0.006680	1.662077	0.025171	1.593722
Average 2013-2017	0.057622	1.188576	0.042355	1.009137	0.057869	1.091317

Fig. 5. Yearly return and Sharpe ratio

VI. Conclusion

Backtesting is not a research tool, you end up with very little insight into the reason why a particular model works better than others. Instead, backtesting is useful to improve or discard bad models. Additionally, trying to adjust your hyperparameters to beat the baseline model is time-consuming and perhaps dangerous. Therefore it is convenient to backtest until your model is fully specified. Moreover, financial data comprises a series of events unlikely to occur again, and we should expect in some extend to overfit the model.

We believe that a tuning-intensive search to find the optimal hyperparameters for neural networks could generate beneficial results compared to the baseline model.

APPENDIX

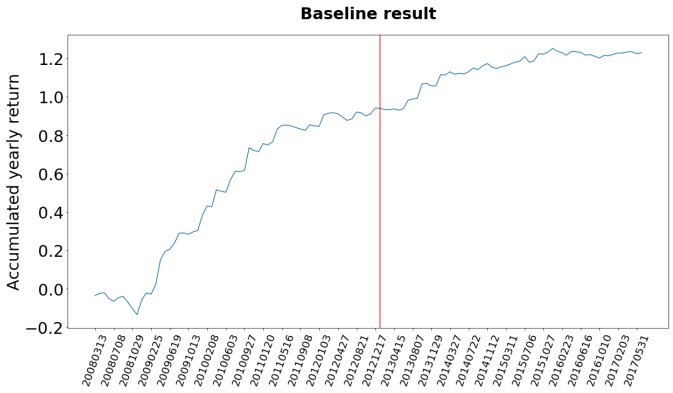


Fig. 6. Baseline result: accumulative yearly return

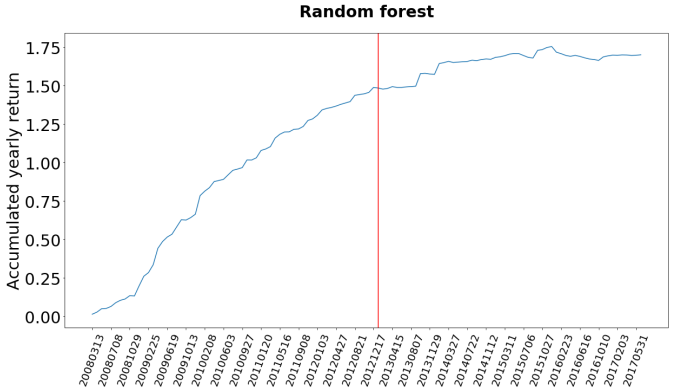


Fig. 7. Random forest: accumulative yearly return

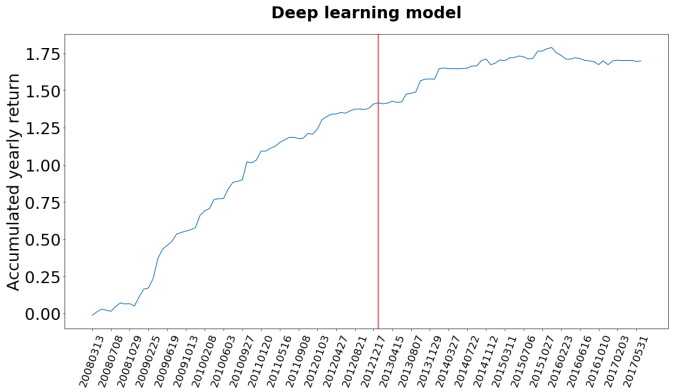


Fig. 8. Deep learning model: accumulative yearly return

REFERENCES

Geron, Aurlien. Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. ” O’Reilly Media, Inc.”, 2017.

Jegadeesh, Narasimhan, and Sheridan Titman. ”Returns to buying winners and selling losers: Implications for stock market efficiency.” The Journal of finance 48, no. 1 (1993): 65-91.

Krauss, Christopher, Xuan Anh Do, and Nicolas Huck. ”Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500.” European Journal of Operational Research 259, no. 2 (2017): 689-702.

Li, Xiang, Shuo Chen, Xiaolin Hu, and Jian Yang. ”Understanding the disharmony between dropout and batch normalization by variance shift.” arXiv preprint arXiv:1801.05134 (2018).

Moritz, Benjamin, and Tom Zimmermann. ”Deep conditional portfolio sorts: The relation between past and future stock returns.” In LMU Munich and Harvard University Working paper. 2014.

Takeuchi, Lawrence, and Yu-Ying Applying Deep Learning Lee. ”to Enhance Momentum Trading Strategies in Stocks.” (2013).

Zhang, G., Patuwo, B. E., and Hu, M. Y. (1998). Forecasting with artificial neural networks: The state of the art. International Journal of Forecasting, 14(1):3562.