Sentiment Analysis for Foreign Exchange Trading

Stockpulse GmbH
Marvin Sohn

December 2021

Abstract

This text discusses the implementation of sentiment analysis into the production of foreign exchange trading strategies. Statistical significance testing for financial performance measures is used to compare generic foreign exchange strategies against sentiment analysis-strategies generated by Stockpulse GmbH. The strategy generated by Stockpulse outperform the generic strategies across all presented performance measures.

1 Introduction

Founded in 2011, Stockpulse specializes in using emotional data intelligence to improve decision making for financial markets. Employing deep learning strategies to analyze social media sentiments, Stockpulse seeks to enhance the asset allocation of their customers based upon not only traditional financial benchmarks, but also on emotional values connected to the assets.

This paper firstly further introduces Stockpulse's sentiment analysis and the process of generating a trading strategy based off of the analyzed data. In the second part of this paper, Stockpulse's foreign exchange trading strategy for the EUR/USD pair is compared with traditional trading strategies currently employed in the foreign exchange market. Stockpulse's strategy outperforms most of the traditional trading strategies on a statistically significant level and thereby validates its value for an improved asset allocation.

2 Strategy Generation

2.1 Deep Learning

In this section, we briefly describe our approach of finding a well-working deep neural network and the setup of the training and validation process. Neural networks have many parameters which influence their learning capacity and predictive power. It is important to make "good" choices for these parameters in order to find models which produce well-working signals. Input data can also be structured in different ways, adding another layer of complexity to the optimization problem at

hand.As the number of possible configurations for neural networks is very large, a suitable method is required to find "good" configurations in an acceptable amount of time. While our data center is equipped with nVidia GPUs to maximize training speed, it is still necessary to reduce the number of configurations significantly to bring down the necessary amount of time.

2.1.1 Genetic Algorithm

To do that, we use a heuristic approach called genetic algorithm. A genetic algorithm is inspired by Charles Darwin's theory of evolution. It follows the principle of natural selection where the fittest individuals are selected to produce offspring for the next generation. The following items are four essential processes in a genetic algorithm:

- Selection: a process of finding fittest individuals in a population. The purpose is to keep genes from these individuals through inheritance to the next generation.
- Fitness: a rating of how individuals perform. In our experiment, the fitness score is determined by financial key statistics like Sharpe ratio, volatility, or return on investment.
- Reproduction: a process of producing offspring. Pairs of individuals (parents), chosen during the selection step, are creating offspring from their genes. In our experiment, optimized parameters are the genes.
- Mutation: a process of mutating genes during reproduction to avoid falling into local optima. In our experiment, the possibility of mutation is 20%.

2.1.2 Implementation

We apply the genetic algorithm to optimize the neural net's structure. We implement the entire procedure in Python with machine learning library Tensorflow. The genetic algorithm runs through 10 generations with a population of 50 networks per generation. The initial population of neural networks is created with randomly chosen layouts, which can also have different kind of input parameters, e.g. the length of input history can vary. The following parameters are tuned by the genetic algorithm:

- Size of history fed into the model
- Type of neural network layer (dense, SimpleRNN, LSTM)
- Number of neurons per layer
- Number of layers
- Activation of layer (linear, elu, relu, tanh, sigmoid)

```
Layer (type)                 Output Shape              Param #
recurrent_stage (LSTM)       (None, 1, 24)             2688
recurrent_stage_1 (SimpleRNN) (None, 1, 128)           19584
recurrent_stage_3 (SimpleRNN) (None, 8)                1096
dense_stage (Dense)          (None, 8)                 72
prediction_stage (Dense)     (None, 1)                 9
Total params: 23,449
Trainable params: 23,449
Non-trainable params: 0
```

Figure 1: Tensorflow Output

2.1.3 Network Layout

In general, a neural network consists of nodes (neurons) and edges (connections between nodes), so it can be seen as a graph. There are many different configurations of neural networks. We build neural networks via Tensorflow's Keras module, which constructs the networks by defining layers. Each layer passes its output into the next layer, thus defining the graph. For more information about Tensorflow and Keras refer to `https://www.tensorflow.org`.

2.1.4 Recurrent Neural Networks

Different types of layers are available in Keras, for instance the Recurrent Neural Network layer (SimpleRNN). Recurrent neural networks (RNNs) are able to learn and represent temporal sequences, thus this type of layer is particularly fit for solving time-related problems, such as forecasting weather or periodic demands. In addition to SimpleRNN, we also make use of Long short-term memory (LSTM) layers, which is another architecture based on RNN. Conceptually, we have defined four stages in our neural network: recurrent stage, squeeze stage, dense stage and prediction stage. These stages may have individual layer types, number of layers, numbers of neurons, as well as activation and regularization functions. The following table is an example of the model summary output from Tensorflow (Keras):

2.1.5 Training

Each neural network is trained using the Nadam optimizer with mean squared error as the loss function. We apply dropout and regularization at various stages in the network to avoid overfitting. We train until loss is stable for a certain number of epochs. Once training is finished, the performance is evaluated by producing signals for the validation data and feed them into our proprietary backtest platform. Model outputs are floating point values, predicting the relative price change for a given feature vector. Thus, we need to derive the actual signal label (buy, sell, flat) from these continuous values. To do this, we apply threshold based classification, so that for values smaller than the

threshold, signal is sell, and for values larger than the threshold, signal is buy. Optionally, we define a quantile around the threshold. Values falling inside the quantile will be labeled as flat. We optimize the threshold through sensitivity analysis.

2.1.6 Validation

During sensitivity analysis of the threshold, each set of signals is run through the Stockpulse backtester. The backtester performs a complete trading simulation, considering transaction cost, dividends, trading hours, etc. The result is a comprehensive set of financial performance keys, such as cummulative return, return p.a., volatility, maximum drawdown, Sharpe ratio, Sortino ratio, and more. In addition to these financial metrics we also calculate the hit ratio for each signal set. We keep the threshold with the best result during the sensitivity analysis. When evaluating model performance, we look at different performance criteria which, when all fulfilled together, will label a model as a "good" model. Models not fulfilling the criteria are considered "bad". The criteria are as follows:

- Sharpe Ratio ≥ 1
- Hit ratio of buy signals ≥ 0.51
- Hit ratio of sell signals ≥ 0.51
- Overall hit ratio ≥ 0.53
- Each year's return ≥ 5

Inside the good and bad model classes, models are ranked by Sortino ratio. In the genetic algorithm, we make use of the model classes during selection. This is done by keeping the best 40 percent of the population in the selection step, with good models being considered before bad models.

2.1.7 Testing

Only when a model has found to be in the class of good models, by backtesting on the validation dataset, we run an additional backtest, but this time on the test dataset. In case the signals produced for the test dataset fulfill the criteria for good models as well, the model is stored and marked as being in the class "good". That means, that models have to fulfill all criteria with signals from both datasets, validation and test, in order to be considered as "good".

2.2 Generic Foreign Exchange Strategies

Based upon [TM11] and [Vau18], the most widely used forex strategies that can be assigned to the family of traditional trading strategies are the following: momentum, moving average convergence divergence (MACD), relative strength index, trending and filter rules. [CCCH16] adds the strategies of using the forward rate as a predictor and using carry trade rules.

Additionally, a Monte Carlo simulation is performed, which should mimic the investment decision of an unskilled investor. For every day in the test phase, the unskilled investor chooses randomly from a uniform distribution with the possible outcomes $\in \{buy, sell, hold\}$. The outcome of the Monte Carlo simulation is gonna be utilized as a bottom-line-benchmark for all other trading strategies.

3 Comparing Trading Strategies

Because the simplified goal of generating new trading strategies is to earn (more) money, it might seem reasonable to focus on the average profitability or the overall volatility of the different trading strategies. But as [HL14] already expressed, this could lead to an alarmingly high number of false-positive discoveries. Instead, by following the guidelines of modern hypothesis testing, the probability of a false discovery can be controlled by adjusting α, the level of significance. Keeping in line with the majority of the academic literature, we test for the two-sigma significance level of $\alpha = 5$. The mentioned trading strategies are all applied, based upon the trading rules stated in the literature, to the test phase starting from 01.01.2020 and ending on 17.11.2021.

Let $\mathcal{S}$ denote the Stockpulse strategy.

3.1 Tests of Normal Distribution

To start off our analysis of trading strategies, we first analyze whether the Stockpulse strategy follows a normal distribution. Therefore we set the null hypothesis to $\mathcal{H}_{0,0} :=$ $\mathcal{S}$ is normal distributed. Following the test of [**?**], $\mathcal{H}_{0,0}$ can be rejected on the α-level. This is caused by a skew of 0.0652 and more importantly a kurtosis of 2.3012.

Even though we rejected the normality assumption, it could still be the case that the true mean of $\mathcal{S}$ is equal to zero. Since we are looking at a trading strategy, it is very important that the true mean is unequal from zero, otherwise the strategy would on average not generate money. Because we already rejected $\mathcal{H}_{0,0}$, we cannot use a t-test to investigate the true mean. Instead, we are using the nonparametric Wilcoxon signed-rank test from [**?**], as this test does not assume normality of the tested data.

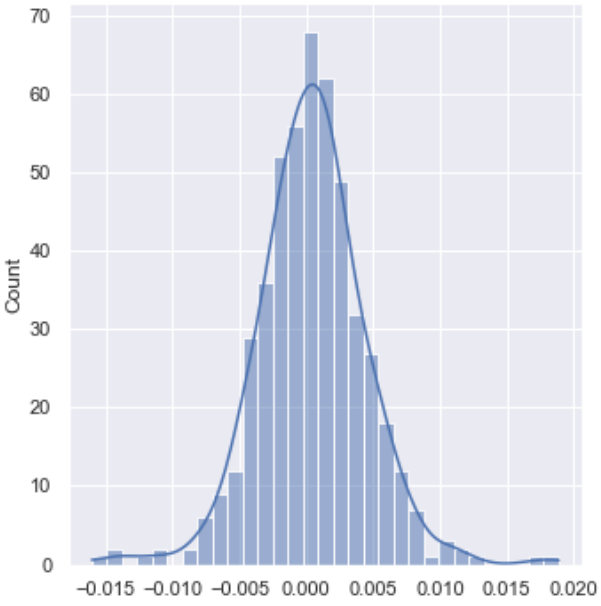

Figure 2: Distribution of $\mathcal{S}$ Daily Returns

We set $\mathcal{H}_{0,1}$:= the true mean of $\mathcal{S}$ is equal to zero. $\mathcal{H}_{0,1}$ can again be rejected at the α-level. Therefore we know that $\mathcal{S}$ is statistically significant in terms of generating money.

3.2 Testing Difference in Means

The result from Chapter 3.1 is valuable as a stand-alone result, but can be weakened if the true means of all trading strategies are the same, thereby making all strategies equally financially interesting. To test whether the true means of the strategies indeed differ from each other, we are using the Z-test to account for the violated normality assumption $\mathcal{H}_{0,0}$.

We define the null hypothesis as $\mathcal{H}_{0,2}$:= true mean of $\mathcal{S}$ is equal to true mean of the generic trading strategy. We do this for every trading strategy listed in Chapter 2.2. Displayed in Figure 3.2, we can reject $\mathcal{H}_{0,2}$ for all strategies except for buy and hold and RSI at the α-level. Even though we would have liked to also reject $\mathcal{H}_{0,2}$ for the RSI strategy, the failure to statistically reject $\mathcal{H}_{0,2}$ can be explained economically: $\mathcal{S}$ is build upon the inputs of the RSI strategy, therefore the distribution of $\mathcal{S}$ inherits the features of the RSI strategy and $\mathcal{H}_{0,2}$ cannot be rejected. On the contrary, we are yet to find an explanation for the failed rejection of $\mathcal{H}_{0,2}$ in regards to the buy and hold strategy.

3.3 Probabilistic Sharpe Ratio

A commonly used measurement of financial strategies is the Sharpe Ratio (SR) from [**?**]. As described by [BLdP12], a potential pitfall of using the SR as a comparison between different strategies is that we focus on point-wise estimation. [Mer03] stated that the standard deviation of the SR is greatly affected by the underlying distribution, which thereby diminishes the statistical significance of the SR results. Therefore [BLdP12] introduced the Probabilistic Sharpe Ratio (PSR), which can be expressed as

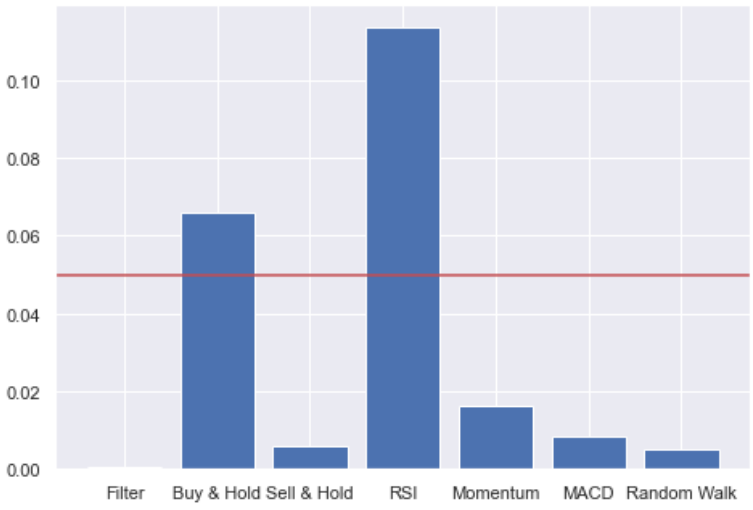

Figure 3: Significance Level for Z-test

$$\widehat{\mathrm{PSR}}\left(SR^*\right) = \mathrm{Prob}[SR \leq \widehat{SR}],$$

with SR^* being a theoretical SR and $\widehat{SR}$ being the estimated SR of the distribution. Put simply, PSR gives us a confidence level with which $\widehat{SR}$ is greater than a SR benchmark.

We are using three different benchmarks: firstly, we are testing each trading strategy against a SR of zero to determine if the respective strategy has a positive impact at all and we are testing each

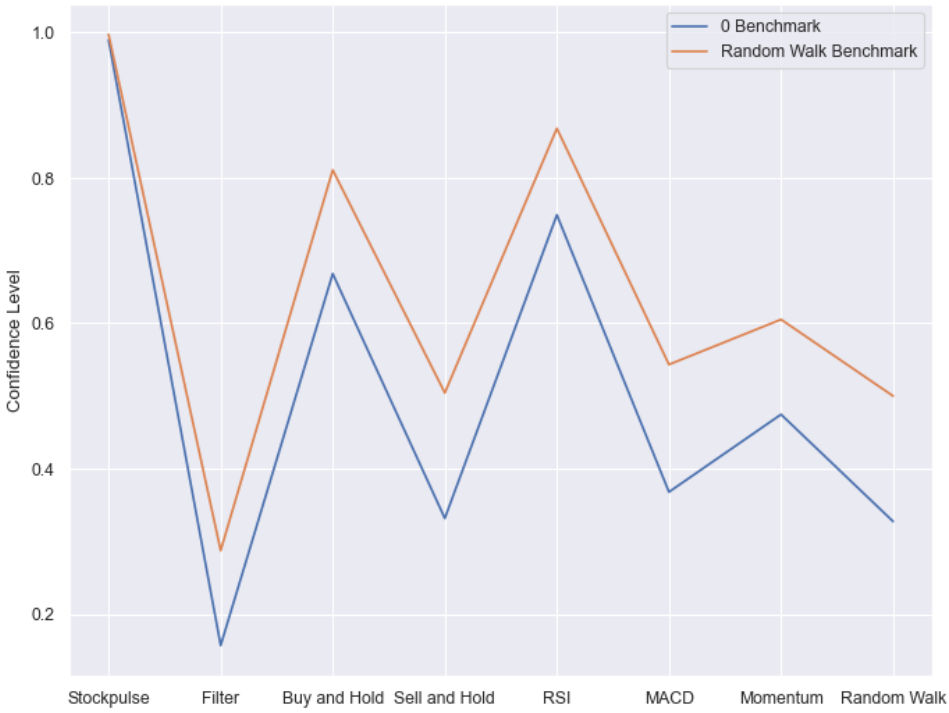

Figure 4: Probabilistic Sharpe Ratios of Trading Strategies with different Benchmarks

strategy against the SR of the random walk. Secondly, $\mathcal{S}$ is tested against the SR of all other generic strategies to compare their head-to-head performance.

The results for the first two benchmarks are shown in Figure 4. $\mathcal{S}$ beats with a confidence of 98.94% the SR of zero and with a confidence of 99.71% the SR of the random walk benchmark. This makes $\mathcal{S}$ the only strategy to surpass the 90% barrier for each benchmark, with the second best strategy being RSI with confidence levels of 74.90% and 86.79% respectively.

In Figure 5, the PSR of $\mathcal{S}$ is illustrated benchmarked to the generic trading strategies from Chapter 2.2. The result can be derived from looking at Chapter 3.2, where we unable to reject $\mathcal{H}_{0,2}$ for the RSI and the buy and hold strategy. This continues on in Figure 5 with RSI and buy and hold strategy leading to the two lowest PSR values for $\mathcal{S}$. But even though the RSI and the buy and hold strategy lead to lower PSR values, these values are still very high with a confidence level of 96.85% and 94.70% respectively. Therefore we can say at the α-level, that $\mathcal{S}$ has a higher SR than a buy and hold, sell and hold, MACD, Momentum and random walk strategy. If we loosen up the α-level by just one percent-point, we can also extend this statement to the RSI strategy.

4 Conclusion

Stockpulses strategy $\mathcal{S}$ is constructed by combining social media sentiment analyses with RSI and Momentum strategies. By doing this, Stockpulse creates a trading strategy for which both the

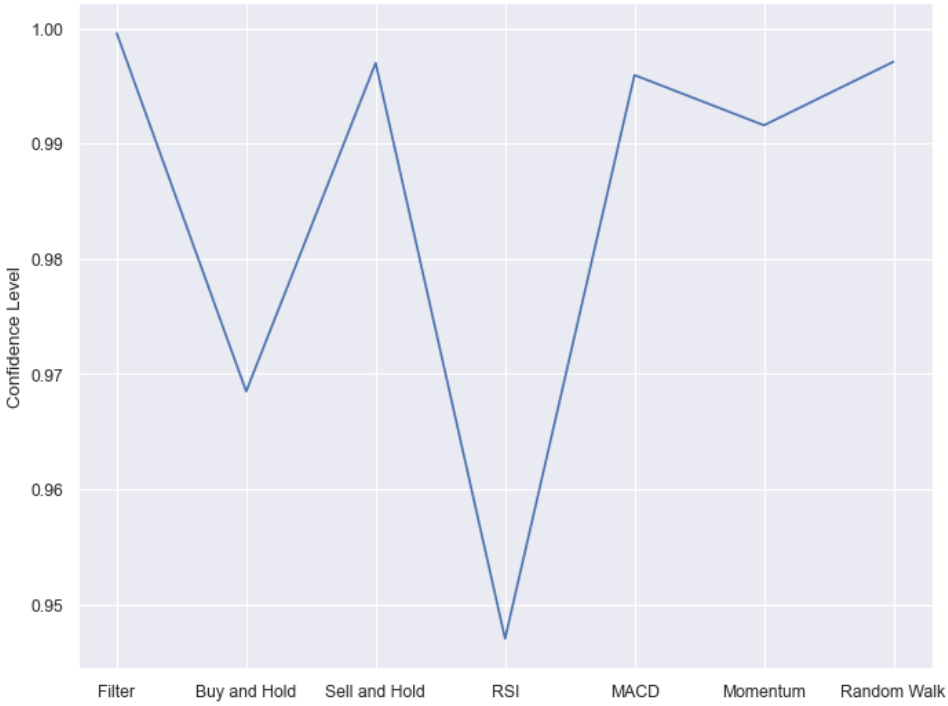

Figure 5: Probabilistic Sharpe Ratios of $\mathcal{S}$ with Trading Strategies as Benchmark

normality assumption and the assumption of a zero mean can be rejected, which indicates that $\mathcal{S}$ has on average an economic impact. Comparing the true means of $\mathcal{S}$ and generic trading strategies, it can be rejected for the majority of generic trading strategies that their true means are equal. Expanding these results to the return-risk combined measurement of PSR, $\mathcal{S}$ has a confidence of over 94% of beating the SRs of all generic trading strategies. Keeping in mind, that $\mathcal{S}$ is made up out of a combination of RSI and momentum strategy, the high confidence level at beating these generic trading strategies which were used as input parameters emphasizes the tremendous role of Stockpulses sentiment analysis in improving the financial significance of a trading strategy.

References

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu,

and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[BG18] Eric Benhamou and Beatrice Guez. Incremental sharpe and other performance ratios. *Journal of Statistical and Econometric Methods*, 7:19–37, 04 2018.

[BLdP12] David Bailey and Marcos Lopez de Prado. The sharpe ratio efficient frontier. *The Journal of Risk*, 15:3–44, 12 2012.

[CCCH16] Mauro Costantini, Jesus Crespo Cuaresma, and Jaroslava Hlouskova. Forecasting errors, directional accuracy and profitability of currency trading: The case of eur/usd exchange rate. *Journal of Forecasting*, 35(7):652–668, 2016.

[HL14] Campbell Harvey and Yan Liu. Evaluating trading strategies. *SSRN Electronic Journal*, 40, 09 2014.

[HZC+17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[Lo03] Andrew Lo. The statistics of sharpe ratios. *Financial Analysts Journal*, 58, 02 2003.

[Mer03] Elmar Mertens. Variance of the iid estimator in lo (2003). *Working Paper*, 2003.

[SVI+15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[TM11] Hannah Thinyane and Jonathan Millin. An investigation into the use of intelligent systems for currency trading. *Computational Economics*, 37:363–374, 04 2011.

[Vau18] Brock Vaughters. An examination of alternative trading techniques using intraday eur/usd currency prices. 2018.

[ZVSL17] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017.