Master degree

in Economics and Finance

Final thesis

Financial time series forecasting using hybrid ARIMA and Deep Learning models

Supervisor
Professor Claudio Pizzi

Graduant
Pavlo Koliadenko
883029

Academic Year
2020/2021

Abstract:

Machine Learning is a trending area of research in many sciences. For a long time, it has had its place in the study of time series forecasting, being constantly compared to traditional methods of Econometrics. This paper proposes and compares alternative architectures of these methods and is dedicated to studying their relative performance on various financial data sets, and studies its possible applications to trading.

The paper studies the ARIMA model specifically, as well as variations of Multilayer perceptrons and Long short-term memory networks and their hybrid modifications. Compared with the single LSTM model, a Multilayer perceptron model, or a classical ARIMA model, the experimental results show that in general, hybrid models display better performance. Nevertheless, the results depend on the time series type.

Keywords:

Time Series Forecasting, ARIMA model, Artificial Neural Network, Hybrid model, Long short-term Memory, Deep Learning, Multilayer Perceptron.

Introduction:

Forecasting is a very topical application of modern Mathematics and Data Science. Over the last 100 years, the methods and models for forecasting improved drastically. First, science went from the simple and intuitive naive methods to advanced statistical methods, and then starting from the nineties, we observed the introduction and the rapid development of Machine Learning Methods as a strong alternative to the more traditional methods.

One of the most fundamental and well-studied models for forecasting is ARIMA (Autoregressive Integrated Moving Average model). It elegantly combines the strong sides of more basic Autoregressive and Moving average models in itself. It owes its popularity to its statistical properties, as well as the famous Box-Jenkins methodology, which provides the full modeling procedure for individual time series.

The mathematical theory of ANN (Artificial Neural Networks) was developed quite a long time ago, though it was not able to take off from the theoretical field due to a lack of computational power. Since suitable computers appeared, the theory was introduced and applied to many fields, and forecasting was one of them.

The introduction of ANN and following ML (Machine Learning) methods naturally produced competition between the available methods and opened a scientific debate on areas of the suitability of these methods. Soon it was discovered that it is not a competition to define a "better" model. The ARIMA model is extremely helpful in detecting linear patterns of the model, which, at the same time, is its weak side because the model pre-assumes a linear form of the data together with the linear structure of correlation between the variables forming the Data Generating Process (DGP). So it is unable to detect any non-linear patterns in the data. Such an assumption is unrealistic and it is a major limitation of the ARIMA model. At the same time, despite their rapid development and reasonably successful application to real-world problems, ML methods are quite complex and currently may lack the transparency needed to receive widespread acceptance. Classical ANN models also are not good enough to capture both linear and nonlinear patterns equally well.

This paper will review and test the performance of the available traditional methods for forecasting together with the well-known hybrid method proposed by Zhang in 2003, and in the end, we will try to propose some alternative methods and compare them to existing ones.

The paper will be structured as follows. In the next sections, there is a review of the MLP, LSTM, ARIMA, and Zhang's methods. After it, the mosel's specifications and the considered metrics are going to be introduced. Then a section that consists of empirical results would be followed by a section with conclusions.

Algorithmic trading

Algorithmic trading has been actively developing in recent decades due to a combination of factors: the rapid development of machine learning methods, the development of technologies for working with data and its analysis, the growth of storage and processing capabilities for large amounts of data. In addition, the complexity of trading system algorithms used by market participants is growing, since they compete not only with those who do not use automated systems, but also with each other. In connection with these trends, the study of the applicability of various machine learning algorithms to algorithmic trading problems is an urgent task. In such studies, there is interest not only from companies engaged in algorithmic trading, such as hedge funds, but also from the scientific community, since the application of machine learning algorithms to the area under consideration can bring new knowledge to the development of machine learning as a branch of computer science, which can as well be applied to other subject areas.

The theoretical value of the work is the development of research on the application of machine learning methods to algorithmic trading. In addition, similar methods can be used in other subject areas when searching for a strategy for the most accurate forecast of the next value of the time series to maximize profits. As a practical value of the work, one can consider the possibility of creating software products for trading on the stock exchange based on the algorithms studied in the work.

Machine learning for the stock market

The stock market is a public trading platform for buying and selling securities of various companies. Attempts to accurately predict future values of the price of a particular stock is an important economic task for a wide range of companies, since an accurate forecast can result in great financial success. Since the inception of stock markets, people have searched and developed ways to predict stock prices as accurately as possible. As a result, three main approaches were formed: fundamental analysis, technical analysis, and usage of technological methods. Despite the sometimes mutually exclusive results of using these approaches in the framework of one task (some predict future value, others — the direction of movement for a given period), they are all used for trading in stock markets. At the same time, the task of improving the forecast accuracy continues to be relevant for all companies operating on stock markets.

This work is devoted to the application of machine learning (including deep learning) to the problem of forecasting in the stock market. In stock market trading, neural networks can also imitate the actions of an agent performing certain tasks in the stock market. Both classical machine learning and neural networks are used as predictive tools, however, reinforcement learning and genetic algorithms have not gained the popularity and degree of sophistication as many classical algorithms (random forest, support vector machine) have. One of the well-known types of neural networks is a feedforward neural network combined with a backpropagation method (as a training method).

An important task in the construction of neural networks is data preprocessing, and many researchers solve it their own way, proposing new methods since there are no established practices for unambiguous preprocessing of input data arrays due to the wide variety of both the data itself and its types. The output data within the framework of a specific task, on the contrary, can be unambiguously determined, taking the form of financial or economic indicators. As for the evaluation of the results, the most frequently used method is to compare the results of the algorithm with the values from the test sample.

Methodology

ARIMA

The ARIMA model is a famous generalization of the AutoRegressive (AR) and Moving Average (MA) models with the addition of the option of the initial differencing the time series (corresponding to the "integrated" part of the model). It is widely used in the modeling of financial time series.

Let y_t be an initial time series. Then $\widehat{y}_t$ is a time series that was differenced a suitable amount of times d.

Then the ARIMA model of the order (p,d,q) has a form:

$$\widehat{y}_t = \theta_0 + \phi_1 \widehat{y}_{t-1} + \phi_2 \widehat{y}_{t-2} + ... + \phi_p \widehat{y}_{t-p} + \varepsilon_t - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - ... - \theta_q \varepsilon_{t-q}$$

Here, ϕ_i are the coefficients that correspond to the Autoregressive component, and θ_i are the coefficients that correspond to the Moving average component.

Where $\widehat{y}_t$ is the actual value measured at time t, ε_t is the residual error at time t.

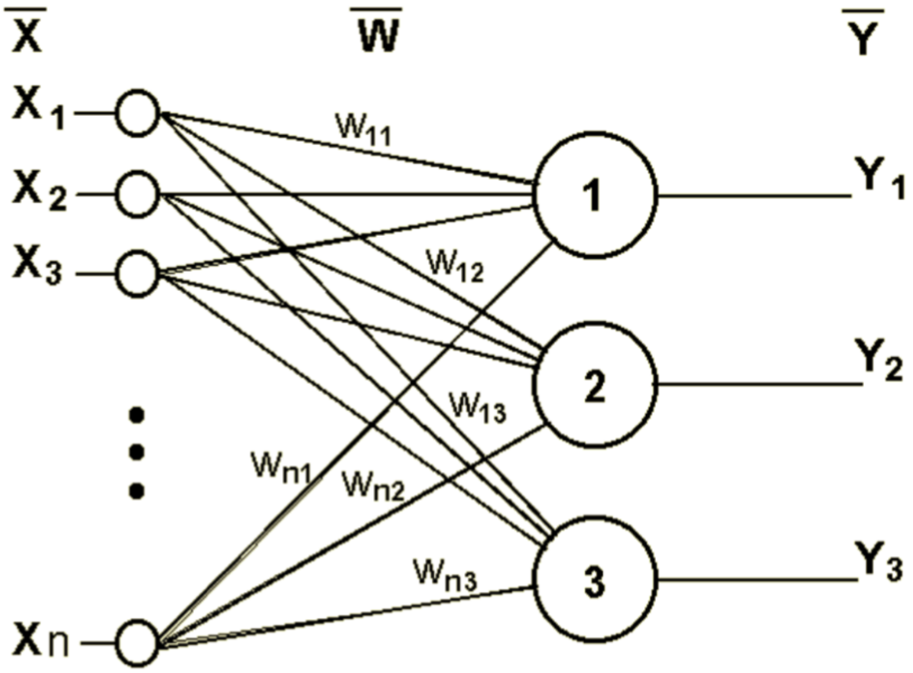
Multilayer Perceptron (MLP)

The perceptron is based on a mathematical model of perception of information by the brain. Different researchers define it differently. In its most general form, the perceptron is a system of elements of three different types: sensors, associative elements, and reactive elements.

A single-layer perceptron is a single-layer neural network, all neurons of which have a rigid threshold activation function.

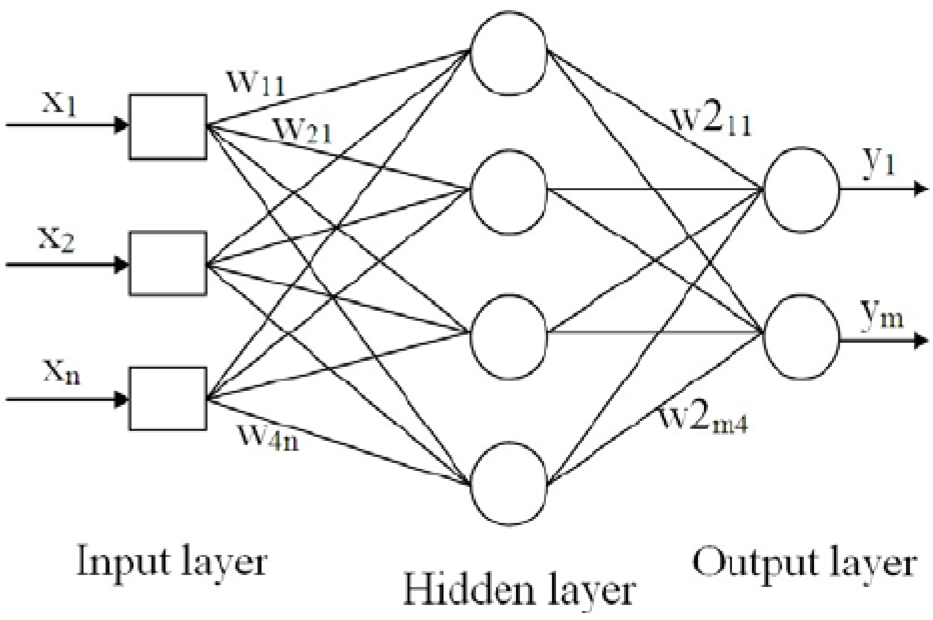
The single-layer perceptron has a simple learning algorithm and can solve only the simplest problems. This model aroused great interest in the early 1960s and gave impetus to the development of artificial neural networks.

A classic example of such a neural network - a single-layer perceptron is shown in the figure.



The network shown in the figure has n-inputs (X), which receive signals that go through the synapses of 3 neurons. These three neurons form a single layer of this network and have three output signals (Y).

An MLP is a neural network of direct signal propagation (without feedback), in which the input signal is converted into output, passing sequentially through several layers. The first of these layers is called the input, the last — the output. These layers contain so-called degenerate neurons, and sometimes the number of layers is not taken into account. In addition to the input and output layers, a multilayer perceptron has one or more intermediate layers, which are called hidden layers. This perceptron model must have at least one hidden layer. The presence of several such layers is justified only in the case of using nonlinear activation functions. An example of a two-layer perceptron is shown in the figure.



The network shown in the figure has 3 inputs. They receive signals that go further along the synapses of the 4 neurons that form the first layer. The output signals of the first layer are transmitted to two neurons of the second layer. The latter, in turn, emits two output signals.

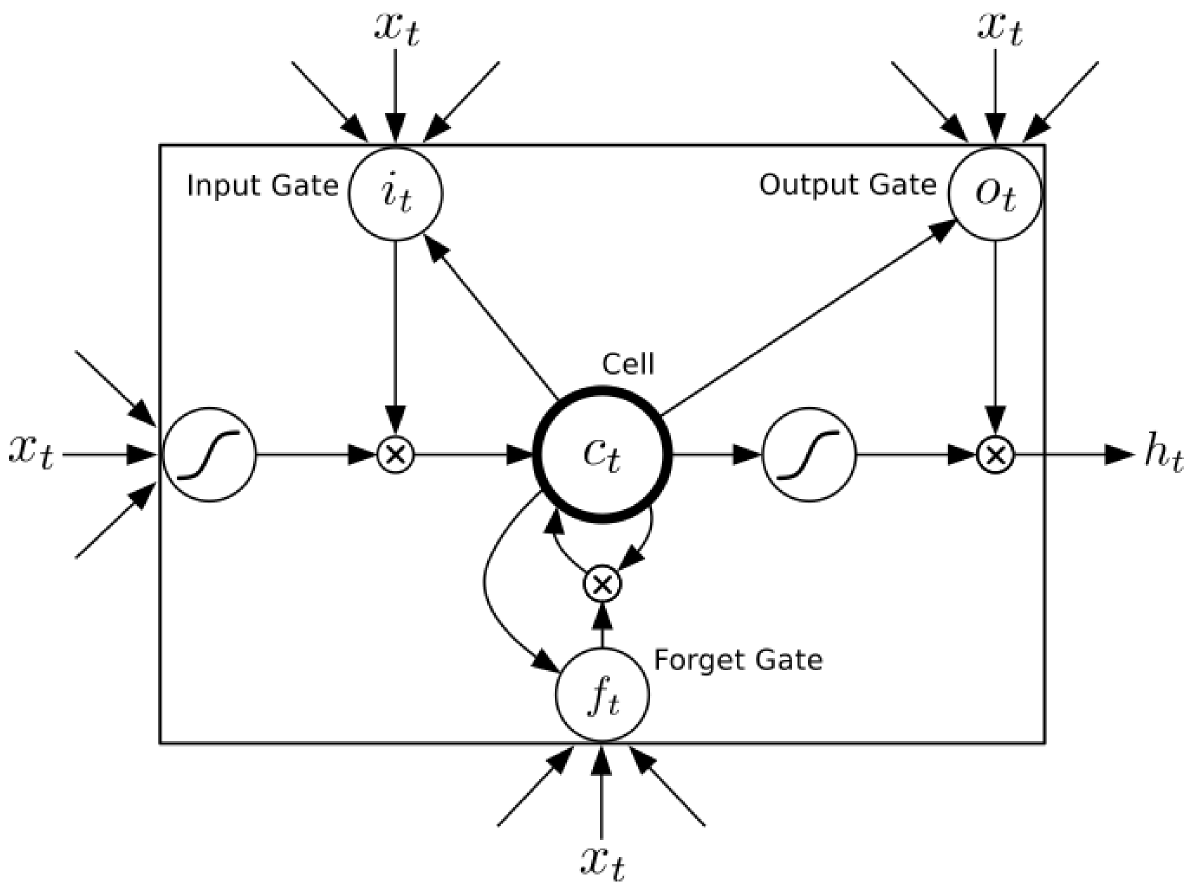
According to the universal approximation theorem, MLPs are universal function approximators (Cybenko) so they can be used to create mathematical models by regression analysis. As classification is a particular case of regression when the response variable is categorical, MLPs make good classifier algorithms.

MLPs were a popular machine learning solution in the 1980s, being applied in various fields such as speech recognition, machine translation software image recognition.

LSTM

A recurrent neural network (RNN) is a type of neural network where connections between elements form a directed sequence. This makes it possible to process a series of events in time or sequential spatial chains. Unlike multilayer perceptrons, recurrent networks can use their internal memory to process sequences of arbitrary length. Therefore, RNNs are applicable in tasks where something holistic is divided into parts, for example, handwriting recognition or speech recognition. Many different architectural solutions, from simple to complex, have been proposed for recurrent networks. Currently, the most widespread are the networks with long- and short-term memory (LSTM) and the controlled recurrent unit (GRU).

Forecasting one step ahead of the financial time series requires not only up-to-date data but also previous data. The RNN model, which is an improvement of the hidden layer self-feedback mechanism, has advantages in dealing with long-term dependency problems, but it is difficult to apply in practice. To solve the problem of RNN gradient disappearance, Sepp Hochreiter and Jurgen Schmidhuber (1997) proposed an LSTM model, which was recently improved and promoted by Alex Graves. The LSTM unit consists of memory cells that store information updated by three special gates: the input gate, the forget gate, and the output gate.



At time t, x_t is the input data of the LSTM cell, h_{t-1} is the output of the LSTM cell at the previous moment, c_t is the value of the memory cell, and h_t is the output of the LSTM cell. The LSTM unit calculation process is performed following the formulas:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f)$$

$$c_t = f_t c_{t-1} + i_t tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-1} + b_o)$$

$$h_t = o_t tanh(c_t)$$

Benefiting from three control gates and memory cells, LSTMs easily retain, read, reset and update long-term information. It is important to note that the LSTM internal parameter sharing mechanism allows control of the dimensions of the output by setting the dimensions of the weight matrix. LSTM establishes a long time delay between input and feedback. The internal state of memory cells in this architecture maintains a continuous error flow, so the gradient does not explode or disappear.

Hybrid model

There are 5 unique forecasting methods that were looked up in this work:

- ARIMA
- MLP
- LSTM
- Hybrid ARIMA based on MLP
- Hybrid ARIMA based on LSTM

All models are suitable for one-step ahead forecasting.

The idea of a hybrid network is not new, as it has its roots in the previously mentioned Zhang paper (2003).

The idea is to apply a neural network to the residuals of the ARIMA model, so that ARIMA would catch all linear patterns of the data, while a neural network is supposed to catch nonlinear patterns, thus improving the forecast accuracy.

$$r_t = log(\frac{p_t}{p_{t-1}})$$

$$r_t = R_t * arima(p, q, d) + e_t$$

$$e_t = E_t * NN + u_t$$

$$R_t = \{r_i : i < t\}$$

$$E_t = \{e_i : i < t\}$$

$$\Rightarrow$$

$$r_t = R_t * arima(p, q, d) + E_t * NN + u_t$$

Numerically, the idea is to:

1. Estimate the ARIMA model on the training data.
2. Train ANN on the residuals of the ARIMA model.

And to obtain the forecast, the ARIMA model forecast and ANN forecast should be summed up.

It should be noted that during the evaluation procedure, the ARIMA model was refitted based on new data on each step of the simulation, thus approximating the real-world situation. At the same time, due to high training time, the ANN models were trained just once and using only training data.

Models’ specifications

In our application, the ARIMA model was semi-automatically selected by the smallest BIC value on the training set for each specific time series. When the best suitable model was a white noise model, the restricted Autoregressive model was selected based on the Partial AutoCorrelation Function and BIC values.

MLP stands for Multilayer Perceptron.

MLP8 is a model that has 3 input neurons, 8 neurons in the second layer, and an out neuron.

MLP12 is a model that has 5 input neurons, 12 neurons in the second layer, and an out neuron.

LSTM stands for long short-term memory.

Both studied LSTM models consist of 20 LSTM neurons.

The difference is only in input size: for LSTM 3 it is 3 neurons, while for LSTM 5 is 5 neurons.

Optimization of the number of epochs is very important in the training process.

This is needed to avoid overfitting.

Overfitting is a situation when the model starts to learn “noise” or irrelevant information within the training set due to a long training process. When the model memorizes the noise and fits too closely to the training set, the model becomes “overfitted”.

To solve this, semi-manual optimization of the number of epochs was performed for each pair of stock and the network architecture.

	Amazon	Bitcoin	Dow Jones	PTFS
ARIMA	AR(2)	AR(6)	ARIMA(1, 0, 1)	ARIMA(2, 0, 0)
MLP 8 epochs	425	25	17	7
MLP 12 epochs	30	12	7	8
LSTM 3 epochs	270	57	100	3
LSTM 5 epochs	150	19	95	7

	Input	First layer	Second layer	Number of trainable parameters
MLP 8	3	3	8	41
MLP 12	5	5	12	85
LSTM 3	3	20	-	1781
LSTM 5	5	20	-	1781

Results

There were 4 financial time series that were studied:

- Amazon stock price (AMZN)
- The exchange rate of the Bitcoin cryptocurrency to USD (BTC-USD)
- Dow Jones stock market index (^DJI)
- PFTS index.

PFTS is a benchmark index of Ukrainian leading bourse - PFTS Ukraine stock exchange.

For each time series, the whole available period was considered. (With an exception of AMZN due to the abnormal volatility at the beginning of the observations. For AMZN it starts from 2004-01-01)

The time series were split into training and test sets in the 7 : 3 proportion.
Test sets were used as validation sets.

Due to the varying sizes of the data samples, it is difficult to compare the performance of methods between the different time series. Because of this, 3 naive algorithms were introduced to be used as a benchmark: White Noise, Random Walk and Buy&Sell. White noise is a randomly generated time series that is normally distributed with the mean of the sample mean of the training set and the variance that is equal to the variance of the training set. The Random Walk time series follows the martingale assumption, thus treating the last value as the best forecast for the future value. Buy&Sell replicates the strategy of buying a stock (or index) for a fixed amount of money each day and selling it at the end of the day.

Scalar product is a useful metric, as it simulates the trading strategy where the agent uses the “Buy&Sell” strategy, but the sum of money is proportional to the forecasted signal.

Custom score is calculated using the formula:

sum(abs(actual)*sign(forecast*actual))

Due to the stochastic nature of the ANN, the results that rely on ANN use are the aggregate average of multiple simulations (from 50 to 200 simulations for each pair of method and time series)

The analysis is applied to the first-differenced time series, so the resulting tables also relate to them.

Amazon													
	MAPE	ME	MAE	MPE	RMSE	CORR	MINMAX	Scalar product	Score	long	short	sign	sign+	sign-
WN	8.391	-0.000566	0.025041	-0.879042	0.031891	0.000596	2.207596	0.001712	0.062448	0.220868	0.158420	50.2%	51.3%	48.9%
RW	5.278420	-0.0000183	0.019179	-0.878804	0.027504	-0.019092	2.085152	-0.010335	0.297932	0.825793	-0.527860	50.3%	54.7%	45.1%
Buy&Sell									2.154529	13.060777	-10.90624	54.7%	100.0%	0.0%
ARIMA(AR2)	1.092	-0.001389	0.012785	-1.005568	0.018361	-0.00316	6.472477	-0.000158	0.253511	-0.09445	**0.347966**	49.8%	44.7%	56.4%
MLP8	**1.374845**	-0.000325	0.012746	-0.91405	0.018268	0.096837	0.682059	0.005974	1.163602	3.889756	-2.726154	52.9%	72.3%	28.2%
MLP12	1.61316	-0.000409	**0.007202**	-0.998752	**0.012815**	0.024784	4.909425	0.000992	0.012598	0.488843	-0.476244	50.9%	55.3%	45.3%
LSTM 3	1.433725	-0.000148	0.012747	-0.945721	0.018217	**0.122715**	-4.13255	**0.008138**	1.538967	3.914431	-2.375464	53.2%	73.7%	27.3%
LSTM 5	1.457233	-7.008542	0.012795	-0.950759	0.018345	0.068824	-1.586255	0.005615	**1.540239**	4.015975	-2.475735	53.4%	75.0%	26.1%
MLP 8	1.680982	-6.502361	0.012736	-1.116019	0.018254	0.104914	-0.076942	0.005714	1.293569	4.797642	-3.504072	53.2%	78.5%	22.3%
MLP12	2.396499	0.001462	0.012891	-1.185416	0.018515	0.011777	-6.071824	0.001755	0.433894	**5.455071**	-5.021177	51.9%	85.2%	14.0%
LSTM 3	1.808353	0.000136	0.012768	-1.095803	0.018232	0.115294	2.109369	0.007611	1.493537	4.555804	-3.062266	53.3%	78.5%	22.4%
LSTM 5	1.756975	0.000228	0.012781	-0.948926	0.018344	0.05602	-0.766744	0.004785	1.461823	4.636841	-3.175018	53.5%	79.8%	21.2%

Bitcoin														
	MAPE	ME	MAE	MPE	RMSE	CORR	MINMAX	Scalar product	Score	long	short	sign	sign+	sign-
WN	9.939764	-0.000459	0.042527	-1.027941	0.056615	0.000362	1.729692	0.002941	0.039846	0.351483	-0.311636	50.0%	51.6%	48.2%
RW	6.30161	-0.000043	0.040748	0.812226	0.061021	-0.088154	3.205476	-0.107537	-0.763481	-0.481248	-0.282232	46.6%	49.1%	43.8%
Buy&Sell									1.542012	10.55310	-9.011097	52.5%	100.0%	0.0%
ARIMA(AR6)	3.299886	0.007081	0.027627	-0.674121	0.041979	0	2.116765	0.014158	1.542012	**10.55310**	-9.011097	**52.5%**	100.0%	0.0%
MLP8	1.643680	-0.000394	0.026886	-0.790514	0.041659	0.031125	4.245179	0.008539	1.201264	3.662717	-2.461453	51.2%	68.2%	32.6%
MLP12	1.970193	-6.391621	0.027198	-0.58057	0.042097	0.019785	-5.744791	0.008888	1.142096	3.063023	-1.920927	51.2%	64.2%	37.0%
LSTM 3	1.762403	3.157977	**0.026678**	-0.605615	**0.041281**	**0.093369**	1.115954	**0.016874**	1.427731	4.197535	-2.769803	51.8%	71.1%	30.6%
LSTM 5	1.503251	0.000291	0.026684	-0.792311	0.041384	0.049726	0.182529	0.005835	1.084225	5.792253	-4.708027	51.2%	76.3%	23.6%
MLP 8	1.559381	-0.004581	0.027199	-0.857408	0.041960	0.025956	-7.267669	0.003472	0.524458	-1.499954	**2.024413**	49.5%	42.6%	58.6%
MLP12	1.733203	-0.004085	0.027413	-0.916918	0.042256	0.009751	1.799782	0.003021	0.353146	-0.221937	0.575084	50.0%	48.5%	52.1%
LSTM 3	1.492158	-0.00394	0.026936	-0.914490	0.041489	0.089125	-4.145971	0.014063	**1.549098**	0.475498	1.073599	50.9%	51.4%	50.3%
LSTM 5	**1.312797**	-0.004406	0.026994	-0.975777	0.041618	0.049425	-2.045053	0.002408	0.324896	-0.981631	1.306526	48.8%	42.8%	56.7%

Dow Jones														
	MAPE	ME	MAE	MPE	RMSE	CORR	MINMAX	Scalar product	Score	long	short	sign	sign+	sign-
WN	10.436366	-0.000163	0.011697	-1.179166	0.015714	0.001025	0.481892	0.000547	0.032143	0.153954	-0.121811	50.1%	51.0%	49.1%
RW	5.219212	5.817767	0.009839	-1.596347	0.0166686	-0.173325	-1.169498	-0.045128	-0.369245	-0.322097	-0.047148	48.6%	53.2%	43.0%
Buy&Sell								0.97106	0.97106	7.842431	-6.87137	54.9%	100.0%	0.0%
ARIMA(2,0,0)	**1.214764**	-0.000157	**0.006589**	-0.999388	**0.010821**	**0.109011**	2.640241	0.002408	**0.888637**	**3.482931**	-2.594294	**52.5%**	70.9%	30.1%
MLP8	1.61191	-6.655272	0.00666	-1.068739	0.01093	0.071665	15.834615	0.003132	0.38716	2.133788	-1.746627	51.4%	62.7%	37.6%
MLP12	1.558549	0.000248	0.006685	-1.02562	0.011031	0.028378	7.957586	0.001119	0.300161	1.724192	-1.424031	51.2%	60.7%	39.5%
LSTM 3	1.658071	-1.545279	0.006654	-1.059439	0.010899	0.108865	1.855079	**0.005443**	0.213262	1.663135	-1.449872	51.1%	58.3%	42.2%
LSTM 5	1.666705	7.376781	0.006643	-1.070174	0.010881	0.101176	2.869313	0.004264	0.398177	2.985397	-2.587219	52.0%	68.6%	31.8%
MLP 8	2.626531	0.000291	0.006662	-0.743787	0.010907	0.058803	-1.124154	0.00273	-0.140136	2.002115	-2.142251	50.2%	61.8%	36.8%
MLP12	2.686753	7.302359	0.006687	-0.995935	0.011007	0.014890	4.358777	0.000695	-0.264292	0.965901	-1.230193	49.9%	55.0%	43.9%
LSTM 3	2.624753	1.421214	0.006676	-1.214403	0.010938	0.019629	-3.846109	0.000704	-0.582651	0.633249	**-1.2159**	49.3%	50.5%	47.9%
LSTM 5	2.521787	4.076151	0.00667	-1.189977	0.010964	-0.005397	1.028285	-0.000188	-0.427831	1.124324	-1.552156	49.8%	54.4%	44.6%

PTFS														
	MAPE	ME	MAE	MPE	RMSE	CORR	MINMAX	Scalar product	Score	long	short	sign	sign+	sign-
WN	39.055473	5.923431	0.019054	-1.642348	0.024941	0.001301	-3.324888	0.00081	0.024145	0.097352	-0.073207	50.0%	50.8%	49.2%
RW	9.898918	1.222130	0.009096	-5.041073	0.017105	0.137821	5.584702	0.036988	2.557515	1.544854	1.012661	56.9%	59.4%	54.1%
Buy&Sell								0.553001	0.553001	5.525985	-4.972984	53.0%	100.0%	0.0%
ARIMA(1,0,1)	**2.447762**	7.170024	**0.006718**	1.370627	**0.013019**	0.058309	20.287096	0.001893	**1.359567**	**2.089654**	-0.730087	**54.5%**	70.6%	36.5%
MLP8	4.011532	-6.704623	0.006985	-1.262855	0.013266	0.040281	36.518946	0.00232	0.810141	0.853131	-0.042989	52.4%	58.7%	45.3%
MLP12	4.574602	-8.620665	0.00711	-1.369087	0.013295	0.047438	3.023154	**0.002839**	0.7859975	0.644893	**0.141103**	52.1%	57.5%	46.0%
LSTM 3	4.179311	0.000232	0.006919	-1.020327	0.01308	**0.092913**	-2.789665	0.002461	0.770165	1.364829	-0.594664	52.4%	63.1%	40.3%
LSTM 5	4.456117	0.000255	0.006996	-1.125265	0.013142	0.060175	3.185651	0.001852	0.599519	1.160114	-0.560595	52.1%	61.4%	41.6%
MLP 8	3.367579	0.000345	0.007194	-0.245313	0.013417	-0.028258	-4.233138	-0.001343	-0.102171	0.548976	-0.651148	48.4%	55.5%	42.1%
MLP12	3.482664	0.000306	0.007191	-0.202168	0.013353	-0.000787	-0.210186	0.000086	0.108876	0.512272	-0.403396	48.6%	55.0%	42.9%
LSTM 3	3.105948	0.000458	0.007139	-0.487451	0.013253	-0.037994	2.265849	-0.001165	-0.078066	0.75361	-0.831676	48.4%	55.7%	41.9%
LSTM 5	2.804261	0.000256	0.007069	-0.618289	0.013216	-0.041959	0.782121	-0.001211	-0.165605	0.610274	-0.775880	48.1%	54.6%	42.4%

Conclusions

There are several comparisons to be made in this section.
First of all, it is important to note that even though a lot of metrics are considered, the neural networks are optimized by MSE, thus RMSE should be considered as a key metric.

First of all, all of the proposed models perform better than the benchmarks.

Constructed models perform well with predicting the sign of the value, as the percentage of the guessed positive signs together with the negative ones sum up to a value above 100% in most of the cases.

Empirical data suggests that the ARIMA model outperforms ANN models on the index time series. This may be related to the nature of the time series.

Hybrid models do not outperform their pure ANN counterparts. This may be due to the fact that there is a lot of noise in the residuals time series, so the hybrid models become overfitted by the noise.

Hybrid models rarely outperform the linear model. The reason for that may again lay in the fact that by training on the residuals, they catch more noise than actual nonlinear patterns in the data. It also may be the case that there are just no nonlinear patterns in given data.

It is worth noting that the overfitted models perform better on the given metrics related to trading. Thus with more training, MSE is growing, but Scalar product, Score metric and the percentage of the guessed signs are growing as well. (Performance results of the overfitted models are available in Appendix 2)

Neural networks with more inputs don’t perform better than ones with less inputs. The reason for that likely lies in the fact that short-term dependencies are much stronger, and by introducing more lag terms, more noise is embedded.

The LSTM model with 3 input nodes seems to be the best among all the ANN models that were considered.

References

W. Gilchrist, Statistical Forecasting, London, 1976.

G.P. Zhang Time series forecasting using a hybrid ARIMA and neural network model, Atlanta, 2003.

G.E.P. Box, G. Jenkins, Time Series Analysis: Forecasting and Control, San Francisco, 1970.

I. Kaastra, M. Boyd, Designing a neural network for forecasting financial and economic time series, 1996

T. Hill, L. Marquez, M. O'Connor, Artificial neural network models for forecasting and decision making, 1994

S. Makridakis, V. Assimakopoulos, E. Spiliotis, The M5 Accuracy competition: Results, findings, and conclusions, 2020

S. Makridakis, V. Assimakopoulos, E. Spiliotis, The M4 Competition: 100,000-time series and 61 forecasting methods, 2020

G. Cybenko "Approximation by superpositions of a sigmoidal function". Mathematics of Control, Signals, and Systems. 1989

K. Hornik, M. Tinchcombe, H. White, Multilayer Feedforward Networks are Universal Approximators. 1989

A. Graves, Generating sequences with recurrent neural networks. 2013

M. Kumar, M. Thenmozhi, Forecasting stock index returns using ARIMA-SVM, ARIMA-ANN, and ARIMA-random forest hybrid models. 2014

S. Siami-Namini, N. Tavakoli, A. Siami-Namin, A Comparison of ARIMA and LSTM in Forecasting Time Series. 2018

D. Krukovets, Short-run forecasting of core inflation in Ukraine: A disaggregated approach. 2019

Y. Deng, H. Fan, S. Wu, A hybrid ARIMA-LSTM model optimized by BP in the forecast of outpatient visits. 2020

Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult. 1994

S. Hochreiter, J. Schmidhuber, Long short-term memory. 1997

A. Graves, Generating sequences with recurrent neural networks. 2013
H. Sak, A. Senior, F. Beaufays, Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling 2014

machinelearningmastery.com

https://pfts.ua/en/1-market-data/1-pfts-index

Appendix 1

This part consists of the listing of the working code.

Intro¶

In [ ]:

```
x



#!pip install yfinance
#!pip install yahoofinancials

import numpy as np
import pandas as pd
import yfinance as yf
import datetime as dt
import matplotlib.pyplot as plt
from yahoofinancials import YahooFinancials

#import statsmodels.tsa.stattools as ts
import statsmodels.formula.api as smf
import statsmodels.tsa.api as smt
import statsmodels.api as sm
import scipy.stats as scs

from sklearn.metrics import mean_squared_error
from statsmodels.tools.eval_measures import rmse

#from pandas import datetime
from math import sqrt
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.stattools import acf
import pmdarima as pm
from numpy import inf


import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

```python
def forecast_accuracy(forecast, actual):
    #clear zeros in actual
    #print(forecast, actual)
    forecast = np.array([a for a,b in zip(forecast,actual) if b !=0])
    actual = np.array([b for b in actual if b !=0])
    #print(forecast, actual)


    mape = np.mean(np.abs(forecast - actual)/np.abs(actual))  # MAPE
    me = np.mean(forecast - actual)             # ME
    mae = np.mean(np.abs(forecast - actual))    # MAE
    mpe = np.mean((forecast - actual)/actual)   # MPE
    mse = np.mean((forecast - actual)**2)       # MSE
    rmse = mse**.5                              # RMSE
    temp_cor1 = forecast.reshape(len(forecast))
    temp_cor2 = actual.reshape(len(forecast))
    corr = np.corrcoef(temp_cor1, temp_cor2)[0,1]   # corr
    mins = np.amin(np.hstack([forecast[:,None],
                             actual[:,None]]), axis=1)
    maxs = np.amax(np.hstack([forecast[:,None],
                             actual[:,None]]), axis=1)
    #clean zeros in maxs
    mins = np.array([a for a,b in zip(mins,maxs) if b !=0])
    maxs = np.array([b for b in maxs if b !=0])
    minmax = 1 - np.mean(mins/maxs)             # minmax
    acf1 = acf(forecast-actual)[1]              # ACF1
    scalar_product = np.sum(forecast*actual)
    custom1 = np.sum(np.abs(actual)*np.sign(forecast*actual))
    long_income = np.sum(np.abs(actual)*np.sign(forecast*actual)*(np.sign(actual)>0))
    short_income = np.sum(np.abs(actual)*np.sign(forecast*actual)*(np.sign(actual)<0))
    guess_sign = np.mean(np.sign(forecast*actual)>0)
    guess_sign_pos = np.mean((np.sign(forecast*actual)>0)*(np.sign(actual)>0))
    guess_sign_pos /= np.mean(np.sign(actual)>0)
    guess_sign_neg = np.mean((np.sign(forecast*actual)>0)*(np.sign(actual)<0))
    guess_sign_neg /= np.mean(np.sign(actual)<0)

    accuracy = {'mape': mape, 'me': me, 'mae': mae,
            'mpe': mpe, 'rmse': rmse,
            'corr': corr, 'minmax': minmax, 'scalar product': scalar_product,
            'custom1': custom1, 'long_income': long_income, 'short_income': short_income,
            'guess_sign': '{:.1%}'.format(guess_sign),
            'guess_+': '{:.1%}'.format(guess_sign_pos),
            'guess_-': '{:.1%}'.format(guess_sign_neg)}

    for i in accuracy:
      print (i,':',accuracy[i])
     result = np.array([mape, me, mae, mpe, rmse, corr, minmax, scalar_product, custom1, long_income,
short_income,
                      guess_sign, guess_sign_pos, guess_sign_neg])
    return result
```

In [ ]:

```
#end = dt.datetime.today()
end = "2021-06-01"
start="1990-01-01"
#start="2004-01-01"

amazon_df = pd.DataFrame(yf.download("^DJI", start=start, end = end)['Adj Close'])

col_names = ["TradeDate","PFTS Index","Previous PFTS Index","Variation","Max 52 Weeks Index","Max 52 Weeks Index Date","Min 52 Weeks Index","Min 52 Weeks Index Date"]
amazon_df = pd.read_csv('pfts.csv', names=col_names, sep=';', index_col='TradeDate', skiprows=50, skipfooter = 510)
#amazon_df = amazon_df['PFTS Index']
amazon_df['PFTS Index'] = amazon_df['PFTS Index'].apply(lambda x: float(x.split()[0].replace(',', '')))
print(amazon_df['PFTS Index'])
amazon_df['PFTS Index']=amazon_df['PFTS Index'].astype(float)
amazon_df = pd.DataFrame(amazon_df['PFTS Index'])
#amazon_df = amazon_df.values
#amazon_df = [item for sublist in amazon_df for item in sublist]

#amazon_df = amazon_df.values

amazon_df.plot.line(figsize=(20, 8), color = "darkblue")
```

In [ ]:

```
amazon_df_chng = np.log(amazon_df / amazon_df.shift(1))
amazon_df_chng = amazon_df_chng.dropna()
amazon_df_chng.plot(figsize=(20, 8), color='orange')
```

In [ ]:

```
#print(len(amazon_df_chng))
train, test = np.split(amazon_df_chng, [int(.7 *len(amazon_df_chng))])
#traino, testo = np.split(amazon_df, [int(.7 *len(amazon_df_chng))])
#print(train)
#print(test)
rw = amazon_df_chng[-2-int(.3 *len(amazon_df_chng)):-1]
#print(rw)
np.array(rw).reshape(len(test),1)
#print(rw)
#temp = [item for sublist in test for item in sublist]
temp = test.values
#print(temp)
tempo = [np.sign(i) for i in temp]
tempo = np.array(tempo)
tempo.reshape(len(test),1)
tempo = np.ones((len(test), 1))
#print(tempo)
#tempo = [[el] for el in tempo]
#print(len(test), len(train), type(tempo), type(temp.values), tempo.shape, temp.values.shape)
#tempo[0] = [0]
tempo[1] = [1]
#tempo[tempo == 0] = inf
#print(temp.values, tempo)
forecast_accuracy(tempo, temp)
trainu, testu = np.split(amazon_df, [int(.7 *len(amazon_df_chng))])
print(testu)
```

In [ ]:

```
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(train, lags=40, ax = ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(train, lags=40, ax = ax2)
```

White noise¶

In [ ]:

```
noise = np.random.normal(np.mean(train.values),np.std(train.values),len(test)).reshape(len(test),1)
print(np.mean(train.values), np.mean(noise), np.mean(test.values))
print(np.std(train.values), np.std(noise), np.std(test.values))
print(noise.shape)
print(type(noise))
print(noise.shape)
forecast_accuracy(noise, test.values)
```

In [ ]:

```
temp_train = train.values
temp_train = [item for sublist in temp_train for item in sublist]
temp_test = test.values
#temp_test = [item for sublist in temp_test for item in sublist]
history = [x for x in temp_train]
predict_wn_test = list()
```

In [ ]:

```
import time
start_time = time.time()

for t in range(len(temp_test)):
    noise = np.random.normal(np.mean(history),np.std(history),1)[0]
    #print(noise)
    predict_wn_test.append(noise)
    obs = temp_test[t]
    history.append(obs)
    if t%100 == 0:
        print(t)
    #print('predicted=%f, expected=%f' % (yhat, obs))

print("--- %s seconds ---" % (time.time() - start_time))
```

In [ ]:

```
#print(predict_wn_test)
#predict_wn_test = [item for sublist in predict_wn_test for item in sublist]
print(predict_wn_test)
```

```
# evaluate forecasts
temp_test = np.array(temp_test).reshape(len(temp_test),1)
predict_wn_test = np.array(predict_wn_test).reshape(len(temp_test),1)
forecast_accuracy(predict_wn_test, temp_test)
forecast_accuracy(tempo, predict_wn_test)
# plot forecasts against actual outcomes
plt.plot(temp_test)
plt.plot(predict_wn_test, color='red')
plt.show()
```

In [ ]:

```
result_list = []
repetition = 1000
for i in range(repetition):

    noise = np.random.normal(np.mean(train.values),np.std(train.values),len(test)).reshape(len(test),1)
    #forecast_accuracy(noise, test.values)
    result_list.append(forecast_accuracy(noise, test.values))

result_av = []
#print(result_av)
for k in range(14):
    temp = 0
    for j in range(repetition):
        temp += result_list[j][k]
    result_av.append(temp/repetition)
print(result_av)

accuracy = {'mape': result_av[0], 'me': result_av[1], 'mae': result_av[2],
            'mpe': result_av[3], 'rmse': result_av[4],
            'corr': result_av[5], 'minmax': result_av[6], 'scalar product': result_av[7],
                'custom': result_av[8], 'long income': result_av[9], 'short_income': result_av[10], 'guess_sign':
'{:.1%}'.format(result_av[11]),
```

```
        'guess_+': '{:.1%}'.format(result_av[12]), 'guess_-': '{:.1%}'.format(result_av[13])}
for i in accuracy:
    print (i,':',accuracy[i])
```

Commented¶

¶

In [ ]:

```
'''
from statsmodels.tsa.arima_model import ARIMA
import itertools
# Grid Search
p = d = q = range(0,3) # p, d, and q can be either 0, 1, or 2
pdq = list(itertools.product(p,d,q)) # gets all possible combinations of p, d, and q
combs = {} # stores bic and order pairs
bics = [] # stores bics
# Grid Search continued
for combination in pdq:
    try:
        model = ARIMA(amazon_df_chng, order=combination) # create all possible models
        model = model.fit()
        combs.update({model.bic : combination}) # store combinations
        bics.append(model.bic)
    except:
        continue

best_bic = min(bics)

# Model Creation and Forecasting
model = ARIMA(amazon_df_chng, order=combs[best_bic])
model = model.fit()
model.forecast(7)[0]
print(model.bic)
print(model.summary())
'''
```

Arima¶

¶

In [ ]:

```python
from itertools import product
# setting initial values and some bounds for them
ps = range(0, 8)
ds = range(0, 2)
qs = range(0, 8)


# creating list with all the possible combinations of parameters
parameters = product(ps, ds, qs)
parameters_list = list(parameters)
len(parameters_list)
```

In [ ]:

```python
def optimizeARIMA(input, parameters_list):
    """
        Return dataframe with parameters and corresponding AIC

        parameters_list - list with (p, q, P, Q) tuples
        d - integration order in ARIMA model
        D - seasonal integration order
        s - length of season
    """

    results = []
    best_bic = float("inf")

    for param in tqdm_notebook(parameters_list):
        # we need try-except because on some combinations model fails to converge
        try:
            model=ARIMA(input, order=(param[0], param[1], param[2])).fit(disp=1)
        except:
            continue
        bic = model.bic
        # saving best model, BIC and parameters
        if bic < best_bic:
            best_model = model
            best_bic = bic
            best_param = param
        results.append([param, model.bic, model.aic])
    print(results)
```

```
    result_table = pd.DataFrame(results)

    result_table.columns = ['parameters', 'bic', 'aic']

    # sorting in ascending order, the lower BIC is - the better
    result_table = result_table.sort_values(by='bic', ascending=True).reset_index(drop=True)

    return result_table
```

In [ ]:

```
from tqdm import tqdm_notebook

#%%time
result_table = optimizeARIMA(train, parameters_list)
```

In [ ]:

```
print(result_table)
```

In [ ]:

```
model_arima_train = ARIMA(train, order=(1,0,1))
model_fit_arima_train = model_arima_train.fit(disp=0)
print(model_fit_arima_train.aic)
print(model_fit_arima_train.summary())
```

In [ ]:

```
residuals_arima_train = pd.DataFrame(model_fit_arima_train.resid)
```

```
fig, ax = plt.subplots(1,2)
residuals_arima_train.plot(title="Residuals", ax=ax[0])
residuals_arima_train.plot(kind='kde', title='Density', ax=ax[1])
plt.show()
```

In [ ]:

```
model_arima_test = ARIMA(test, order=(0,0,0))
model_fit_arima_test = model_arima_test.fit(disp=0)
print(model_fit_arima_test.aic)
print(model_fit_arima_test.summary())
```

In [ ]:

```
residuals_arima_test = pd.DataFrame(model_fit_arima_test.resid)

#print(residuals_arima_test)
residuals_arima_test = residuals_arima_test.to_numpy()
print(residuals_arima_test)
residuals_arima_test = [item for sublist in residuals_arima_test for item in sublist]
print(residuals_arima_test)
```

In [ ]:

```
model_fit.plot_predict(dynamic=False)

leg = plt.legend()
# get the individual lines inside legend and set line width
for line in leg.get_lines():
    line.set_linewidth(1)
plt.rcParams["figure.figsize"] = (40,30)
plt.show()
```

Restricted ARIMA¶

¶

AR 2¶

¶

In [ ]:

```
#model_restricted_train = sm.tsa.statespace.SARIMAX(train, order=((0,0,0,0,0,1),0,0))
model_restricted_train = ARIMA(history, order=(1,0,1))
model_fit_restricted_train = model_restricted_train.fit(disp=0)
print(model_fit_restricted_train.bic)
print(model_fit_restricted_train.summary())
```

In [ ]:

```
residuals_restricted_train = pd.DataFrame(model_fit_restricted_train.resid)
fig, ax = plt.subplots(1,2)
residuals_restricted_train.plot(title="Residuals", ax=ax[0])
residuals_restricted_train.plot(kind='kde', title='Density', ax=ax[1])
plt.show()
```

In [ ]:

```
temp_train = train.values
#temp_train = [item for sublist in temp_train for item in sublist]
temp_test = test.values
#temp_test = [item for sublist in temp_test for item in sublist]
history = [x for x in temp_train]
predict_restricted_test = list()
residuals_restricted_test = []
```

In [ ]:

```
import time
start_time = time.time()

for t in range(len(temp_test)):
    #model = sm.tsa.statespace.SARIMAX(train, order=((0,0,0,0,0,1),0,0))
    model = ARIMA(history, order=(1,0,1))
    model_fit = model.fit()
    output = model_fit.forecast()
    yhat = output[0]
    predict_restricted_test.append(yhat)
    obs = temp_test[t]
    history.append(obs)
    if t%100 == 0:
        print(t)
    #print('predicted=%f, expected=%f' % (yhat, obs))
    residuals_restricted_test.append(obs-yhat)

print("--- %s seconds ---" % (time.time() - start_time))
```

In [ ]:

```
# evaluate forecasts
residuals_restricted_test = np.array(residuals_restricted_test).reshape(len(temp_test),1)
temp_test = np.array(temp_test).reshape(len(temp_test),1)
predict_restricted_test = np.array(predict_restricted_test).reshape(len(temp_test),1)
forecast_accuracy(predict_restricted_test, temp_test)
# plot forecasts against actual outcomes
plt.plot(temp_test)
plt.plot(predict_restricted_test, color='red')
plt.show()
```

In [ ]:

```
#residuals_restricted_test = np.subtract(test, predictions)
print(test)
#print(predictions)
print(residuals_restricted_test)
```

```
#residuals_restricted_test = residuals_restricted_test.to_numpy()
#residuals_restricted_test = [item for sublist in residuals_restricted_test for item in sublist]
```

Machine Learnin'¶

In [ ]:

```
from numpy import array
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM

# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
# hybrid
raw_seq = residuals_restricted_train.to_numpy()
#print(residuals_restricted_train)
# vanilla
#raw_seq = train.to_numpy()
raw_seq = [item for sublist in raw_seq for item in sublist]
```

MLP 3 - 8 - 1¶

¶

In [ ]:

```
# choose a number of time steps
n_steps = 5
# split into samples
X, y = split_sequence(raw_seq, n_steps)

# define model
model_3_8 = Sequential()
model_3_8.add(Dense(12, activation='relu', input_dim=n_steps))
model_3_8.add(Dense(1))
model_3_8.compile(optimizer='adam', loss='mse')

import time
start_time = time.time()

# fit model
model_3_8.fit(X, y, epochs=20, verbose=0)

print("--- %s seconds ---" % (time.time() - start_time))
print(model_3_8.summary())
```

Validation¶

¶

In [ ]:

```
# RABOTAET!!!

# vanilla
tail_test = temp_test
# hybrid
#tail_test = residuals_restricted_test

tail_test = [item for sublist in residuals_restricted_test for item in sublist]

temp_resid = raw_seq[-n_steps:]+tail_test

X, y = split_sequence(temp_resid, n_steps)
mlp_8_predict = model_3_8.predict(X, verbose=0)
mlp_8_predict_list = [item for sublist in mlp_8_predict for item in sublist]
```

```
mlp_8_predict_list = np.array(mlp_8_predict_list).reshape(len(temp_test),1)

tail_test = np.array(tail_test).reshape(len(temp_test),1)

# vanilla
result_8_predict = mlp_8_predict_list
# hybrid
#result_8_predict = np.add(mlp_8_predict_list, predict_restricted_test)

forecast_accuracy(result_8_predict, tail_test)

#print(temp_test, residuals_restricted_test, predict_restricted_test, mlp_8_predict_list, hybrid_mlp_8_predict_list)
#print(temp_test.shape, forecast_restricted_test.shape, residuals_restricted_test.shape,
# mlp_8_predict_list.shape, mlp_8_predict_list+forecast_restricted_test.shape, hybrid_mlp_8_predict_list.shape)



plt.plot(tail_test)
plt.plot(result_8_predict, color='red')
plt.show()
```

In [ ]:

```
from __future__ import division
def mean(a):
    return sum(a) / len(a)
a = [[240, 240, 239],
     [250, 249, 237],
     [242, 239, 237],
     [240, 234, 233]]


opti_repetition = 10
aggregate = []

for i in range(opti_repetition):
    # choose a number of time steps
    n_steps = 5
    # split into samples
    X, y = split_sequence(raw_seq, n_steps)
    n_features = 1
    #X = X.reshape((X.shape[0], X.shape[1], n_features))
```

```
# define model
model_3_8 = Sequential()
model_3_8.add(Dense(12, activation='relu', input_dim=n_steps))
model_3_8.add(Dense(1))
model_3_8.compile(optimizer='adam', loss='mse')

#tail_test = temp_test
tail_test = residuals_restricted_test

tail_test = [item for sublist in tail_test for item in sublist]

temp_resid = raw_seq[-n_steps:]+tail_test

X_test, Y_test = split_sequence(temp_resid, n_steps)
#X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], n_features))

import time
start_time = time.time()

# fit model
train_history = model_3_8.fit(X, y, epochs=20, verbose=0, validation_data=(X_test, Y_test))

print("--- %s seconds ---" % (time.time() - start_time))
#print(model_3_8.summary())

losses_lstm = model_3_8.history.history['loss']
#plt.figure(figsize=(12,4))
#plt.xlabel("Epochs")
#plt.ylabel("Loss")
#plt.xticks(np.arange(0,len(losses_lstm)+1,1))
#plt.plot(range(len(losses_lstm)),losses_lstm);

loss = train_history.history['loss']
val_loss = train_history.history['val_loss']
print(type(train_history.history['val_loss']))
t = pd.Series(val_loss)
aggregate.append(list(t))


#plt.plot(loss)
#plt.plot(t)
#plt.legend([ 'val_loss'])
#plt.show()
```

In [ ]:

```
print(len(temp_test))
avg = [float(sum(col))/len(col) for col in zip(*aggregate)]
t = pd.Series(avg)
t = t.rolling(window=3, center=True).mean()
plt.plot(t)
'''from statistics import mean

aggregate_ready = map(mean, zip(*aggregate))
#print(*aggregate_ready)
r = np.array(*aggregate_ready)
print(r)
#print(*map(mean, zip(*aggregate)))'''
```

MLP 5 - 12 - 1¶

¶

In [ ]:

```
# choose a number of time steps
n_steps = 5
# split into samples
X, y = split_sequence(raw_seq, n_steps)
# define model
model_5_12 = Sequential()
model_5_12.add(Dense(12, activation='relu', input_dim=n_steps))
model_5_12.add(Dense(1))
model_5_12.compile(optimizer='adam', loss='mse')

import time
start_time = time.time()

# fit model
model_5_12.fit(X, y, epochs=2000, verbose=0)

print("--- %s seconds ---" % (time.time() - start_time))
```

Validation¶

¶

In [ ]:

```
# not really
#residuals_test = test.to_numpy()
#residuals_test = [item for sublist in residuals_test for item in sublist]
temp_resid = raw_seq[-n_steps:]+residuals_restricted_test
print(raw_seq[-n_steps:])
X, y = split_sequence(temp_resid, n_steps)
print(X)
mlp_12_predict = model_5_12.predict(X, verbose=0)
mlp_12_predict_list = [item for sublist in mlp_12_predict for item in sublist]
temp_test = np.array(temp_test).reshape(len(temp_test),1)
mlp_12_predict_list = np.array(mlp_12_predict_list).reshape(len(temp_test),1)
forecast_accuracy(mlp_12_predict_list, temp_test)

print(mlp_12_predict_list.shape)
plt.plot(temp_test)
plt.plot(mlp_12_predict_list, color='red')
plt.show()
```

LSTM¶

¶

In [ ]:

```
# choose a number of time steps
n_steps = 3
# split into samples
X, y = split_sequence(raw_seq, n_steps)
# reshape from [samples, timesteps] into [samples, timesteps, features]
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))

# define model
model_lstm = Sequential()
model_lstm.add(LSTM(20, activation='relu', input_shape=(n_steps, n_features)))
model_lstm.add(Dense(1))
model_lstm.compile(optimizer='adam', loss='mse')
```

```
import time
start_time = time.time()

# fit model
model_lstm.fit(X, y, epochs=5, verbose=0)

print("--- %s seconds ---" % (time.time() - start_time))
```

In [ ]:

```
losses_lstm = model_lstm.history.history['loss']
plt.figure(figsize=(12,4))
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.xticks(np.arange(0,len(losses_lstm)+1,1))
plt.plot(range(len(losses_lstm)),losses_lstm);
```

In [ ]:

```
print(model_lstm.summary())
```

Validation¶

¶

In [ ]:

```
# not really
#residuals_test = test.to_numpy()
#residuals_test = [item for sublist in residuals_test for item in sublist]
temp_resid = raw_seq[-n_steps:]+residuals_restricted_test
#print(raw_seq[-n_steps:])
X, y = split_sequence(temp_resid, n_steps)
X = X.reshape((X.shape[0], X.shape[1], n_features))
```

```
mlp_lstm_predict = model_lstm.predict(X, verbose=0)
print(type(mlp_lstm_predict))
mlp_lstm_predict_list = [item for sublist in mlp_lstm_predict for item in sublist]
mlp_lstm_predict_list = np.sum(mlp_lstm_predict_list,forecast_restricted_test)
temp_test = np.array(temp_test).reshape(len(temp_test),1)
mlp_lstm_predict_list = np.array(mlp_lstm_predict_list).reshape(len(temp_test),1)
#print(forecast_accuracy(mlp_lstm_predict_list, temp_test))

print(mlp_lstm_predict_list.shape)
plt.plot(temp_test)
plt.plot(mlp_lstm_predict_list, color='red')
plt.show()
```

In [ ]:

Average¶

¶

In [ ]:

```
result_list = []
repetition = 20
for i in range(repetition):
    # choose a number of time steps
    n_steps = 5
    # split into samples
    X, y = split_sequence(raw_seq, n_steps)

    # define model
    model_5_12 = Sequential()
    model_5_12.add(Dense(12, activation='relu', input_dim=n_steps))
    model_5_12.add(Dense(1))
    model_5_12.compile(optimizer='adam', loss='mse')
```

```
    import time
    start_time = time.time()

    # fit model
    model_5_12.fit(X, y, epochs=2000, verbose=0)

    print("--- %s seconds ---" % (time.time() - start_time))
    # RABOTAET!!!

    # vanilla
    tail_test = temp_test
    # hybrid
    #tail_test = residuals_restricted_test

    tail_test = [item for sublist in residuals_restricted_test for item in sublist]

    temp_resid = raw_seq[-n_steps:]+tail_test

    X, y = split_sequence(temp_resid, n_steps)
    mlp_12_predict = model_5_12.predict(X, verbose=0)
    mlp_12_predict_list = [item for sublist in mlp_12_predict for item in sublist]
    mlp_12_predict_list = np.array(mlp_12_predict_list).reshape(len(temp_test),1)

    tail_test = np.array(tail_test).reshape(len(temp_test),1)

    # vanilla
    result_12_predict = mlp_12_predict_list
    # hybrid
    #result_8_predict = np.add(mlp_8_predict_list, predict_restricted_test)

    forecast_accuracy(result_12_predict, tail_test)

                    #print(temp_test,     residuals_restricted_test,     predict_restricted_test,     mlp_8_predict_list,
hybrid_mlp_8_predict_list)
    #print(temp_test.shape, forecast_restricted_test.shape, residuals_restricted_test.shape,
                 #                         mlp_8_predict_list.shape,     mlp_8_predict_list+forecast_restricted_test.shape,
hybrid_mlp_8_predict_list.shape)
    result_list.append(forecast_accuracy(result_12_predict, tail_test))

    print(model_lstm.summary())
    plt.plot(tail_test)
    plt.plot(result_12_predict, color='red')
    plt.show()

result_av = []
#print(result_av)
for k in range(14):
    temp = 0
    for j in range(repetition):
        temp += result_list[j][k]
```

```
    result_av.append(temp/repetition)
print(result_av)

accuracy = {'mape': result_av[0], 'me': result_av[1], 'mae': result_av[2],
            'mpe': result_av[3], 'rmse': result_av[4],
            'corr': result_av[5], 'minmax': result_av[6], 'scalar product': result_av[7],
                 'custom': result_av[8], 'long income': result_av[9], 'short_income': result_av[10], 'guess_sign':
'{:.1%}'.format(result_av[11]),
           'guess_+': '{:.1%}'.format(result_av[12]), 'guess_-': '{:.1%}'.format(result_av[13])}
for i in accuracy:
    print (i,':',accuracy[i])
```

In [ ]:

```
print(result_list)
result_av = []
#print(result_av)
for k in range(9):
    temp = 0
    for j in range(20):
        temp += result_list[j][k]
    result_av.append(temp/20)
print(result_av)

accuracy = {'mape': result_av[0], 'me': result_av[1], 'mae': result_av[2],
            'mpe': result_av[3], 'rmse': result_av[4],
            'corr': result_av[5], 'minmax': result_av[6],
            'custom': result_av[7], 'guess_sign': '{:.1%}'.format(result_av[8]),
           'guess_+': '{:.1%}'.format(result_av[9]), 'guess_+': '{:.1%}'.format(result_av[10])}
for i in accuracy:
    print (i,':',accuracy[i])
```

In [ ]:

```
n_steps = 5
repetition = 20

t_t = test.values
t_t = [item for sublist in t_t for item in sublist]
```

```
for kakaya_epokha_prashla in range(100, 200, 15):
    result_list = []

    for i in range(repetition):

        # choose a number of time steps
        # split into samples
        X, y = split_sequence(raw_seq, n_steps)
        # reshape from [samples, timesteps] into [samples, timesteps, features]
        n_features = 1
        X = X.reshape((X.shape[0], X.shape[1], n_features))

        # define model
        model_lstm = Sequential()
        model_lstm.add(LSTM(20, activation='relu', input_shape=(n_steps, n_features)))
        model_lstm.add(Dense(1))
        model_lstm.compile(optimizer='adam', loss='mse')

        import time
        start_time = time.time()

        # fit model
        model_lstm.fit(X, y, epochs=kakaya_epokha_prashla, verbose=0)

        print("--- %s seconds ---" % (time.time() - start_time))

        # vanilla
        tail_test = temp_test
        # hybrid
        #tail_test = residuals_restricted_test

        tail_test = [item for sublist in tail_test for item in sublist]

        temp_resid = raw_seq[-n_steps:]+tail_test
        X, y = split_sequence(temp_resid, n_steps)
        X = X.reshape((X.shape[0], X.shape[1], n_features))

        mlp_8_predict = model_lstm.predict(X, verbose=0)
        mlp_8_predict_list = [item for sublist in mlp_8_predict for item in sublist]
        mlp_8_predict_list = np.array(mlp_8_predict_list).reshape(len(temp_test),1)

        tail_test = np.array(tail_test).reshape(len(temp_test),1)

        # vanilla
        result_8_predict = mlp_8_predict_list

        #forecast_accuracy(result_8_predict, tail_test)
```

```
                    #print(temp_test,   residuals_restricted_test,   predict_restricted_test,   mlp_8_predict_list, 
hybrid_mlp_8_predict_list)
        #print(temp_test.shape, forecast_restricted_test.shape, residuals_restricted_test.shape,
                    #               mlp_8_predict_list.shape,   mlp_8_predict_list+forecast_restricted_test.shape, 
hybrid_mlp_8_predict_list.shape)
        result_list.append(forecast_accuracy(result_8_predict, tail_test))

        #print(model_3_8.summary())
        plt.plot(tail_test)
        plt.plot(result_8_predict, color='red')
        plt.show()

        #print(result_list)
    #print(result_list[9][8])
    result_av = []
    #print(result_av)
    for k in range(14):
        temp = 0
        for j in range(repetition):
            temp += result_list[j][k]
        result_av.append(temp/repetition)
    print(kakaya_epokha_prashla)
    print(result_av)

    accuracy = {'mape': result_av[0], 'me': result_av[1], 'mae': result_av[2],
            'mpe': result_av[3], 'rmse': result_av[4],
            'corr': result_av[5], 'minmax': result_av[6], 'scalar product': result_av[7],
                'custom': result_av[8], 'long income': result_av[9], 'short_income': result_av[10], 'guess_sign': 
'{:.1%}'.format(result_av[11]),
           'guess_+': '{:.1%}'.format(result_av[12]), 'guess_-': '{:.1%}'.format(result_av[13])}
    for i in accuracy:
        print (i,':',accuracy[i])
    print(result_av)
```

Appendix 2

Here you can find the results on overfitted models

	Amazon	Bitcoin	Dow Jones	PTFS
ARIMA	AR(2)	AR(6)	ARIMA(1,0,1)	ARIMA(2,0,0)
MLP 8 epochs	350	30	350	350
MLP 12 epochs	350	30	350	350
LSTM 3 epochs	150	50	150	150
LSTM 5 epochs	150	50	150	150

Amazon													
	MAPE	ME	MAE	MPE	RMSE	CORR	MINMAX	Scalar product	Score	long	short	sign	sign+	sign-
WN	8.391	-0.000566	0.025041	-0.879042	0.031891	0.000596	2.207596	0.001712	0.062448	0.220868	0.158420	50.2%	51.3%	48.9%
RW	5.278420	-0.0000183	0.019179	-0.878804	0.027504	-0.019092	2.085152	-0.010335	0.297932	0.825793	-0.527860	50.3%	54.7%	45.1%
Always buy									2.154529	13.060777	-10.906248	54.7%	100.0%	0.0%
ARIMA(AR2)	1.092	-0.001389	0.012785	-1.005568	0.018361	-0.00316	6.472477	-0.000158	0.253511	-0.09445	**0.347966**	49.8%	44.7%	56.4%
MLP8	**0.723**	0.000027	**0.006373**	-0.446914	0.009141	0.045144	1.610939	0.00347	0.639249	2.467603	-1.828354	26.8%	39.4%	10.7%
MLP12	1.571	-0.000096	0.012887	-0.890703	0.018526	0.046023	1.367782	0.005994	1.110646	3.746914	-2.636268	52.9%	72.7%	27.7%
LSTM 3	1.286	-0.000432	0.012721	-0.930975	**0.018294**	0.081185	-22.759246	0.003237	1.433385	3.774426	-2.341041	53.5%	73.2%	28.6%
LSTM 5	1.366	-0.000193	0.012763	-0.919318	0.018353	0.037128	0.368273	0.003017	1.354131	4.129648	-2.775516	53.6%	76.0%	25.2%
MLP 8	1.318	-0.000672	0.012810	-1.030269	0.018339	0.077563	2.410217	0.005076	0.894608	3.199848	-2.305239	52.1%	68.3%	31.3%
MLP12	1.522	-0.000111	0.012917	-1.083471	0.018541	0.052494	1.789515	**0.006762**	1.151668	3.676481	-2.524812	53.1%	72.5%	28.2%
LSTM 3	1.338	-0.000408	0.012773	-1.000452	0.018300	**0.083701**	2.496375	0.005933	1.370122	3.800503	-2.430381	53.3%	74.5%	26.0%
LSTM 5	1.382	0.000078	0.012795	-1.017472	0.018404	0.034257	-1.112917	0.004558	**1.45009**	**4.928475**	-3.478385	**53.8%**	80.9%	19.2%

Bitcoin														
	MAPE	ME	MAE	MPE	RMSE	CORR	MINMAX	Scalar product	Score	long	short	sign	sign+	sign-
WN	9.939764	-0.000459	0.042527	-1.027941	0.056615	0.000362	1.729692	0.002941	0.039846	0.351483	-0.311636	50.0%	51.6%	48.2%
RW	6.30161	-0.000043	0.040748	0.812226	0.061021	-0.088154	3.205476	-0.107537	-0.763481	-0.481248	-0.282232	46.6%	49.1%	43.8%
									1.542012	10.553109	-9.011097	52.5%	100.0%	0.0%
ARIMA(AR 6)	3.299886	0.007081	0.027627	-0.674121	0.041979	0	2.116765	0.014158	**1.542012**	**10.553109**	-9.011097	**52.5%**	100.0%	0.0%
MLP8	1.674255	-0.000364	0.026899	-0.772597	0.041681	0.036399	-3.102849	0.009977	1.269885	3.615129	-2.345243	51.2%	67.8%	33.0%
MLP12	2.005285	-1.576976	0.027255	-0.595456	0.042178	0.019247	-1.43765	0.008988	1.082695	3.034403	-1.951707	51.2%	64.2%	37.0%
LSTM 3	1.651541	-0.000442	-0.667594	-0.667594	**0.041312**	**0.084275**	1.057012	0.013671	1.409956	3.652451	-2.242495	51.6%	68.3%	33.3%
LSTM 5	1.899144	0.000313	0.026923	-0.496478	0.0415612	0.064488	3.031175	**0.016505**	1.305758	3.907992	-2.602233	51.1%	68.2%	32.3%
MLP 8	1.578748	-0.004142	0.027144	-0.816022	0.041880	0.031384	0.026424	0.005940	0.835895	-0.663624	1.499519	50.2%	46.7%	54.8%
MLP12	1.80003	-0.004289	0.027481	-0.899591	0.042305	0.025313	2.701471	0.007413	0.562591	-0.560401	1.122992	50.0%	46.0%	55.1%
LSTM 3	1.465356	-0.004772	0.027045	-0.957214	0.041584	0.083911	-2.788749	0.009592	0.97423	-1.628127	**2.602357**	49.6%	41.1%	60.6%
LSTM 5	**1.4903**	-0.003929	0.027118	-0.874379	0.041721	0.059088	8.800898	0.011571	1.117299	-0.287658	1.404958	49.3%	46.3%	53.2%

Dow Jons:

White Noise

mape : 10.436366814736921
me : -0.00016321305579858824
mae : 0.011697662965691196
mpe : -1.1791668245088522
rmse : 0.015714680430213783
corr : 0.0010250523350075198
minmax : 0.4818926805545867
scalar product : 0.0005479326678447218
custom : 0.0321430764933239
long income : 0.15395457814979802
short_income : -0.1218115016564742
guess_sign : 50.1%
guess_+ : 51.0%
guess_- : 49.1%

Random Walk

mape : 5.219212537539019
me : 5.817767977946852e-06
mae : 0.009839714763280875
mpe : -1.5963471470669697
rmse : 0.016668670838248783
corr : -0.17332538757461607
minmax : -1.1694982679921408
scalar product : -0.04512800442347348
custom1 : -0.36924561711010045
long_income : -0.32209737213890427
short_income : -0.04714824497119627
guess_sign : 48.6%
guess_+ : 53.2%
guess_- : 43.0%

Buy&Hold*

scalar product : 0.9710600618328693
custom1 : 0.9710600618328693
long_income : 7.842430560780054
short_income : -6.871370498947185
guess_sign : 54.9%
guess_+ : 100.0%
guess_- : 0.0%

ARIMA

mape : 1.2147642497214883
me : -0.00015713553572341125
mae : 0.006589229878467969
mpe : -0.9993889567714689
rmse : 0.010821992083327145
corr : 0.10901069444108524
minmax : 2.6402416533533515
scalar product : 0.002408015817102362
custom1 : 0.8886370345377217
long_income : 3.48293130851832
short_income : -2.594294273980598
guess_sign : 52.5%
guess_+ : 70.9%
guess_- : 30.1%

MLP_8

mape : 1.6861911557781533
me : -0.00040059004261623924
mae : 0.006735756777042994
mpe : -0.9700979945800899

rmse : 0.011430473470921815
corr : 0.016297448552937497
minmax : -10.102443992266187
scalar product : 0.0006111508786195919
custom : -0.01577543525213609
long income : -0.7213070464582896
short_income : 0.7055316112061532
guess_sign : 49.5%
guess_+ : 43.5%
guess_- : 56.8%

MLP_12

mape : 1.601350321246132
me : -9.745296427657553e-05
mae : 0.007002678901303272
mpe : -1.0494256011853573
rmse : 0.015449996658949932
corr : -0.042638081107970384
minmax : -25.238481331384136
scalar product : -0.012480792879165757
custom : 0.058172827342257254
long income : 0.7442392773583574
short_income : -0.6860664500161
guess_sign : 50.5%
guess_+ : 55.0%
guess_- : 44.9%

LSTM_3

mape : 1.5762348640401842
me : -7.150372357002891e-05
mae : 0.0066604182566833625
mpe : -1.0304196141565545
rmse : 0.010975981834237179
corr : 0.11375021060016756
minmax : -1.6122456583447646
scalar product : 0.007659423293130826
custom : 0.2572907471778018
long income : 1.6916815319047749
short_income : -1.434390784726973
guess_sign : 51.2%
guess_+ : 59.3%
guess_- : 41.3%

LSTM 5

mape : 1.600563042861508
me : -3.495914813869219e-05
mae : 0.006650021051131446
mpe : -1.0615863809705146
rmse : 0.01092393796581103
corr : 0.11330011786373313
minmax : 0.8689967900925352
scalar product : 0.006529383798091409
custom : 0.3755639288375545
long income : 2.399394553433426
short_income : -2.02383062459587
guess_sign : 51.7%
guess_+ : 64.7%
guess_- : 35.8%

MLP_8 hybrid

mape : 2.464349952629985
me : 0.00038902247336294345
mae : 0.006681550329626107
mpe : -0.655105940492198
rmse : 0.011502513252687686
corr : -0.013005238834791606
minmax : 4.186113283916864
scalar product : -0.0032344888913085385
custom : -0.14946285369790632
long income : 2.349551680034563
short_income : -2.4990145337324687
guess_sign : 50.5%
guess_+ : 65.3%
guess_- : 33.3%

MLP_12 hybrid

mape : 2.5043654112974814
me : 0.00013940308644677562
mae : 0.006933181006466366
mpe : -0.7735627311047033
rmse : 0.014297192912049048
corr : -0.013570516230645131
minmax : 17.671008108507714
scalar product : -0.003916260540699872
custom : -0.03749843346353996
long income : 1.4380707731098434
short_income : -1.4755692065733839
guess_sign : 50.2%
guess_+ : 59.1%
guess_- : 40.0%

LSTM 3 hybrid

mape : 2.5956935381475477
me : 0.0002979044935630514
mae : 0.006665169556863377
mpe : -0.8138064044279525
rmse : 0.010968352109580057
corr : 0.03540429917575422
minmax : 40.73465044598211
scalar product : 0.0017619936446797006
custom : -0.32345964741561617
long income : 2.4005029121229238
short_income : -2.7239625595385406
guess_sign : 50.4%
guess_+ : 64.5%
guess_- : 34.1%

LSTM 5 hybrid

mape : 2.3036114650140416
me : 0.0003435953642084869
mae : 0.00665304290285251
mpe : -0.7730425107153033
rmse : 0.01101332126950573
corr : 0.002948336024318808

minmax : -3.5627771729189965
scalar product : 0.00022437895604446973
custom : -0.4250292494321736
long income : 2.4618040891222766
short_income : -2.886833338554448
guess_sign : 50.2%
guess_+ : 64.6%
guess_- : 33.5%

PFTS:

White Noise

mape : 39.05547389674293
me : 5.923431744365488e-05
mae : 0.01905468638846994
mpe : -1.642348909285545
rmse : 0.024940857362265954
corr : 0.001301606784204048
minmax : -3.3248883789619286
scalar product : 0.0008102189412535656
custom : 0.02414528024454681
long income : 0.09735257581231968
short_income : -0.07320729556777279
guess_sign : 50.0%
guess_+ : 50.8%
guess_- : 49.2%

Random Walk

mape : 9.898918795627187
me : 1.2221309451149012e-05
mae : 0.009096997505961076
mpe : -5.041073746143001
rmse : 0.017105616088556954
corr : 0.13782093685604171
minmax : 5.584702861758743
scalar product : 0.03698811065578311
custom1 : 2.557515215752489
long_income : 1.5448540860430309
short_income : 1.0126611297094583
guess_sign : 56.9%
guess_+ : 59.4%
guess_- : 54.1%

Buy&Hold*

scalar product : 0.5530012518876806
custom1 : 0.5530012518876806
long_income : 5.525985658662016
short_income : -4.972984406774335
guess_sign : 53.0%
guess_+ : 100.0%
guess_- : 0.0%

ARIMA

mape : 2.447762466287323
me : 7.17002490082652e-06
mae : 0.006718253942909236
mpe : -1.3706273875194683
rmse : 0.013019871807140603
corr : 0.058309671819179336
minmax : 20.287096278854566
scalar product : 0.0018939617120650528
custom1 : 1.3595677088531735
long_income : 2.089654716360336
short_income : -0.7300870075071622
guess_sign : 54.5%
guess_+ : 70.6%
guess_- : 36.5%

MLP_8

mape : 4.2646740789681195
me : 6.256933959795482e-05
mae : 0.007016163521664131
mpe : -1.363642830184368
rmse : 0.013651024778063285
corr : 0.03701311906045517
minmax : 0.8261472344787688
scalar product : 0.0032994490866243478
custom : 1.1986511188879418
long income : 1.374795375367786
short_income : -0.17614425647984408
guess_sign : 54.0%
guess_+ : 65.3%
guess_- : 41.3%

MLP_12

mape : 4.887398213644209
me : -0.00035467560541374836
mae : 0.007249311874588565
mpe : -1.7624105222322237
rmse : 0.01409678492242013
corr : 0.03604750827837187
minmax : 0.9041770391877592
scalar product : 0.004051142440477986
custom : 1.071926067303646
long income : 0.46500577666987064
short_income : 0.6069202906337753
guess_sign : 53.1%
guess_+ : 56.8%
guess_- : 49.0%

LSTM_3

mape : 4.830326804656913
me : 4.622849075805623e-06
mae : 0.0071542117527238795
mpe : -1.3981429231132338

rmse : 0.013804276572825932
corr : 0.02124709272865036
minmax : 1.8207726720339903
scalar product : 0.0020191580253410544
custom : 1.1584710904858222
long income : 1.2081750190940912
short_income : -0.049703928608268895
guess_sign : 53.9%
guess_+ : 63.5%
guess_- : 43.0%

LSTM 5

mape : 4.852486360871328
me : -0.0002986890161212818
mae : 0.007192112664159893
mpe : -1.6551468720312632
rmse : 0.01372623776623984
corr : 0.044448929054593765
minmax : 1.1485482903540252
scalar product : 0.004125244270323949
custom : 1.1008208432507942
long income : 0.7998690886178391
short_income : 0.3009517546329554
guess_sign : 53.3%
guess_+ : 59.6%
guess_- : 46.3%

MLP_8 hybrid

mape : 4.04524446593055
me : 0.0003095247757429832
mae : 0.007243837471364233
mpe : 0.3735793280825107
rmse : 0.013632868932173216
corr : 0.006034196126230569
minmax : 5.1435230707048705
scalar product : 0.00047284567778319145
custom : 0.22242265086893895
long income : 0.6804886585766193
short_income : -0.45806600770768063
guess_sign : 49.1%
guess_+ : 58.0%
guess_- : 41.4%

MLP_12 hybrid

mape : 3.9917014057495965
me : 0.0003191570378191716
mae : 0.007284210400903838
mpe : 0.39441820109585835
rmse : 0.013865473986023714
corr : 0.03253047524706035
minmax : 2.3600189002293566
scalar product : 0.0033163481629450405
custom : 0.6051660135860791
long income : 0.6619461309291992
short_income : -0.05678011734312042
guess_sign : 49.5%
guess_+ : 57.9%
guess_- : 42.1%

LSTM 3 hybrid

mape : 3.5674541232847905
me : -4.98073718137567e-05
mae : 0.007211266154053154
mpe : -0.10621992375648044
rmse : 0.013499146302279896
corr : 0.007296627294748583
minmax : 1.8349901995243147
scalar product : 0.0005241159767973521
custom : 0.24386343907956803
long income : 0.08528527830484034
short_income : 0.15857816077472764
guess_sign : 50.0%
guess_+ : 51.1%
guess_- : 49.1%

LSTM 5 hybrid

mape : 3.5644770350677657
me : 0.00010621312465968316
mae : 0.007218726528339063
mpe : -0.18667380292037475
rmse : 0.013495996292448922
corr : 0.008519302744399024
minmax : -2.349255042412077
scalar product : 0.0007137489088937493
custom : 0.2746046374664949
long income : 0.2559583331134003
short_income : 0.01864630435309438
guess_sign : 49.5%
guess_+ : 53.4%
guess_- : 46.1%