Seminar: Topics in Machine Learning and Applied Econometrics

Kevin Stefan Meyer

Abstract—**In this paper we're going to examine a time series of the water level of the German river Rhine. Standard techniques like the econometric standard approach of Box-Jenkins and ARIMA models shall function as a baseline and be evaluated against more sophisticated machine learning methods. In this paper we're focusing on the CNN and LTSM models, where we're going to show that they don't outperform other regression models on this problem. Only when taken into account multiple variables, we can achieve an tremendous increase in accuracy. We always turn to machine learning methods when the classical methods show the possibility to be improved upon. The ability to model unknown nonlinear relationships in time series data and some methods may result in better performance when working with non-stationary observations or some mixture of stationary and non-stationary views of the problem. Furthermore several additional predictors, the general approach towards model selection, their data gathering and processing parts will be explained.**

I. First-step analysis

This section starts by looking at the plot of the Rhine river level time series. First of all one can already see, through movements around the mean and approximately constant variance, that the underlying stochastic process might be mean-reverting and thus be stationary. This will later be tested with the Augmented-Dickey-Fuller test [1].

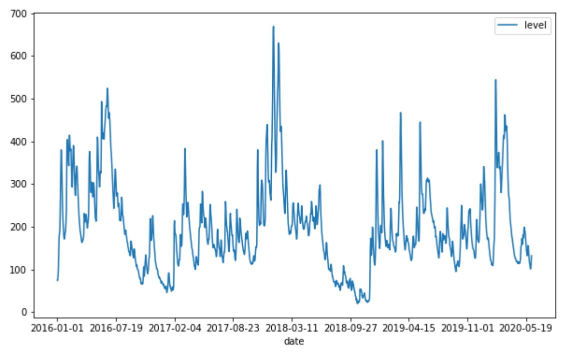


Fig. 1. Our basic dataset: Level of the Rhine river from 2016 to mid-2020.

When looking at the bigger machine learning picture, time series analysis has special properties that make it different from traditional classification and regression perspectives. The temporal structure adds an order to the observations, where classical models have to be adjusted to accurately reflect that fact. Aforementioned difference will become especially clear in Section 4, where we're applying methods like MLPs, CNNs as well as LSTMs, an advanced Recurrent neural network architecture. While MLP's and CNN's do not handle a specific internal state to account for the temporal structure of time series, LSTMs have been explicitly designed for that matter and thus present a state-of-the-art methodology to handle these specific problems. Nevertheless, all of the aforementioned methodologies depend on the specifics of the time series under investigation and it shall be proven that in our case, the multivariate LSTM model clearly outperforms all other approaches. Another important general approach to mention is the fact that our goal in mind is to outperform on a seven day forecast, which in practice comes down to us testing models with smaller test sets than in a setting with a different goal in mind (i.e. forecasting two weeks, three weeks or even larger time periods). It shall be brought up that all the models perform equally on larger test sets than the ones discussed here, while for ease of reproducability and fitting the correct model for the forecasting challenge this amount has been narrowed down to seven days.

Advisor: Dr. Arne Warnke

A. Check for Autocorrelation

In a first step, we look at possible autocorrelation amongst past values of the time series itself. Achieving that, we can plot the correlation coefficient for each lag variable. Our goal is to very quickly get an idea of which lag variables might be candidates for usage in a predictive model and how the relationship between the observation and its historic values changes over time. We're using pandas built-in plot function called the autocorrelation_plot() [6].

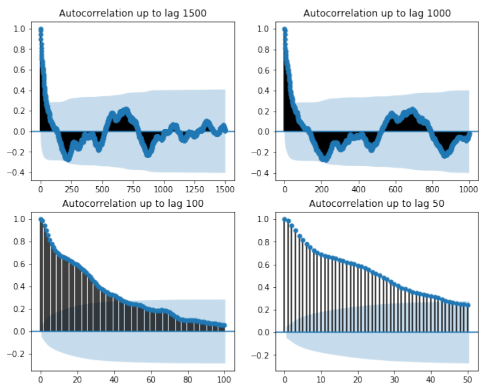


Fig. 2. Taxonomy of modelling decisions based on ACF & PACF plots.

The plot provides the lag number along the x-axis and the correlation coefficient value between -1 and 1 on the y-axis.

The plot also includes solid and dashed lines that indicate the 95% and 99% confidence interval for the correlation values. Correlation values above these lines are more significant than those below the line, providing a threshold or cutoff for selecting more relevant lag values. In this plot, through its very nature of including correlations with large lag values, we can see interesting patterns. Namely we can see that there is substantial negative correlation to the 200 day lag, which from a domain expertise standpoint tells us that this can express the cyclical, seasonal relationship between summer/winter. Furthermore, in the lower row plots we can see that there is significant correlation until up to a lag of about 50.

B. *Check for stationarity*

Next, we examine our time series on stationarity. The rationale for this step lies in the required differencing steps to enhance our modeling capabilities in the ARIMA models. Observations in a stationary time series are not dependent on time in the sense of a constant variance and mean over different periods. Furthermore, stationary time series don't exhibit trends or seasonal effects, which still has to be proven for our time series. We're going to test this initial guess of non-stationarity by the Augmented Dickey-Fuller Test [2]. This is a type of unit root test, which determines how strong a time series is influenced by trends and a shift of mean and variance over time [3]. It achieves that examination by constructing an autoregressive model and optimizing an information criterion (like AIC or BIC) across multiple different lag values. Formally, the null hypothesis of the test is that the time series can be represented by a unit root, which is equivalent to a non-stationary, time-dependent structure. The alternate hypothesis (rejecting the null hypothesis) claims that the time series is stationary.

Formally, we're testing:

- Null Hypothesis (H0): Time-dependent structure. If one fails to reject, it suggests the time series has a unit root, meaning it is non-stationary and it exhibits a time dependent structure.
  vs.
- Alternate Hypothesis (H1): If the null hypothesis is rejected, the test suggests the time series does not have a unit root, meaning it is stationary and it does exhibit have time-dependent structure.

Below the ADF-statistic for our time series is depicted.

```
ADF Statistic: -4.199761
p-value: 0.000660
Critical Values:
        1%: -3.434
        5%: -2.863
        10%: -2.568
```

Fig. 3. Augmented Dickey-Fuller Test Statistic with corresponding critical values.

Running the example prints the test statistic value of around -4. The more negative this statistic, the more likely we are to reject the null hypothesis (we have a stationary dataset). As part of the output, we get a look-up table to help determine the ADF statistic [2]. We can see that our statistic value of -4.199761 is less than the value of -3.434 at 1%. This suggests that we can reject the null hypothesis with a significance level of less than 1% (i.e. a low probability that the result is a statistical fluke). Rejecting the null hypothesis means that the process has no unit root, and in turn that the time series is stationary or does not have time-dependent structure.

Our first glimpse and naive tests indicated the time series to be non-stationary, the ADF-test claims the contrary - we're going to stick to the technical analysis, but keep our intuition in mind when applying ARIMA models to the time series.

II. Naive and Box-Jenkins Method

The Box-Jenkins method was proposed by George Box and Gwilym Jenkins in their seminal 1970 textbook Time Series Analysis: Forecasting and Control [7]. The approach starts with the assumption that the process that generated the time series can be approximated using an ARMA model if it is stationary or an ARIMA model if it is non-stationary.

Before following that approach, establishing a baseline is essential on any time series forecasting problem first. A baseline in performance gives an idea of how well all other models will actually perform on the problem. The technique used to generate a forecast to calculate the baseline performance must be easy to implement and naive of problem-specific details.

A. *Baseline models: naive and average methods*

The simplest model that we could use to make predictions would be to persist the last observation. We can call this a persistence model and it provides a baseline of performance for the problem that we can use for comparison with an autoregression model and more sophisticated methods like the CNN/RNN (LSTM) approach. We can develop a test harness for the problem by splitting the observations into training and test sets, with only the last 7 observations in the dataset assigned to the test set as unseen data that we wish to predict. The predictions are made using a walk-forward validation model so that we can persist the most recent observations for the next day. This means that we are not making a 7-day forecast, but seven 1-day forecasts. There are two main themes to simple forecast strategies, namely naive (or using observations values directly) and average (or using a statistic calculated on previous observations) methods. A naive forecast involves using the previous observation directly as the forecast without any change. It is often called the persistence forecast as the prior observation is persisted. This simple approach can be adjusted slightly for seasonal data. In this case, the observation at the same time in the previous cycle may be persisted instead. This can be further generalized to testing each possible offset into the historical data that could be used to persist a value for a forecast. This was tested for all natural numbers in the range up to 365 days (which corresponds to the hypothesis of a yearly cycle).

The grid search over the most suitable values for our forecasts enables to average over seasonal data, respecting the seasonal offset. An offset argument can be added to the function that when not set to 1 will determine the number of prior observations backwards to count before collecting values from which to include in the average.

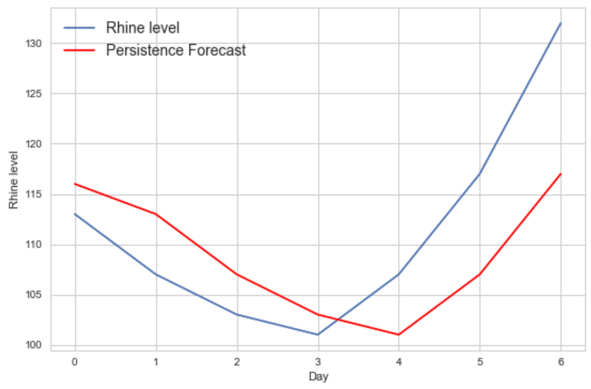


Fig. 4. Plot of the persistence forecast (red) vs. actual values (blue).

The best performing persistence model has a RMSE of 7.801, which will in the following be the metric to outperform. For comparison, if we extend our training set to 10% of the overall data[1], we get an RMSE of 20.535. This will also be evaluated against more sophisticated methods, especially regarding the general fit and not running the danger of overfitting our models by only testing it on the last seven days, for which we want to maximize performance in order to increase the probability of mastering the forecasting challenge.

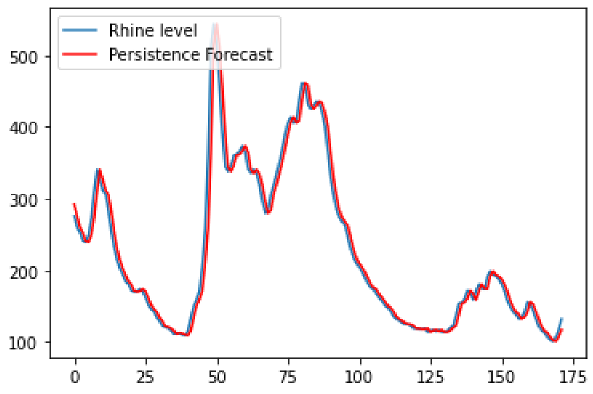


Fig. 5. Plot of the persistence forecast (red) vs. actual values (blue) at a larger time frame (172 values, which corresponds to 10% of the dataset.

The Rhine river dataset is a univariate time series dataset that describes the level of the Rhine river over a period of four and a half years. The standard econometric methods of looking at ACF and PACF plots to find out what the appropriate amount of lags is shall be shortly evaluated. Selecting candidate Auto Regressive Moving Average (ARMA) models for time series analysis and forecasting, understanding Autocorrelation function (ACF), and Partial autocorrelation function (PACF) plots of the series are necessary to determine the order of AR and/ or MA terms. Though ACF and PACF do not directly dictate the order of the ARMA model, the plots can facilitate understanding the order and provide an idea of which model can be a good fit for the time-series data.

[1]We won't follow the path of doing a two-third-one-third train-test split, because we only have 1720 values available. With larger datasets, that would be the better choice.

B. Autocorrelation

We can calculate the correlation for time series observations with observations with previous time steps, called lags. Because the correlation of the time series observations is calculated with values of the same series at previous times, this is called a serial correlation, or an autocorrelation. For our method of grid searching this is extremly important to limit our search space for hyperparameters to a considerable space.

```
from pandas import read_csv
from matplotlib import pyplot
from statsmodels.graphics.tsaplots import
    plot_acf
series = read_csv('basicDataset.csv',
    header=0, index_col=0)
plot_acf(series)
pyplot.show()
```

Running the example creates a 2D plot showing the lag value along the x-axis and the correlation on the y-axis between -1 and 1.

Confidence intervals are drawn as a cone. By default, this is set to a 95% confidence interval, suggesting that correlation values outside of this code are very likely a correlation and not a statistical fluke.

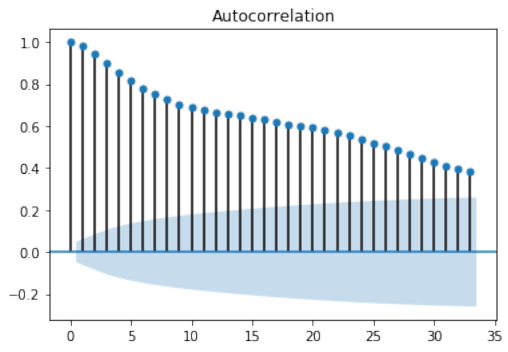


Fig. 6. Autocorrelation of the Rhine river dataset up to a lag of 35 days.

If we include lags bigger than the ones depicted in the figure above (like in Fig. 2), we can deduce other relevant patterns (see discussion there).

C. Partial Autocorrelation

A partial autocorrelation is a summary of the relationship between an observation in a time series with observations at prior time steps with the relationships of intervening observations removed.

```
from pandas import read_csv
from matplotlib import pyplot
from statsmodels.graphics.tsaplots import
    plot_pacf
```

```
series = read_csv('basicDataset.csv',
    header=0, index_col=0)
plot_pacf(series, lags=50)
pyplot.show()
```

Running the example creates a 2D plot of the partial autocorrelation for the first 35 lags.

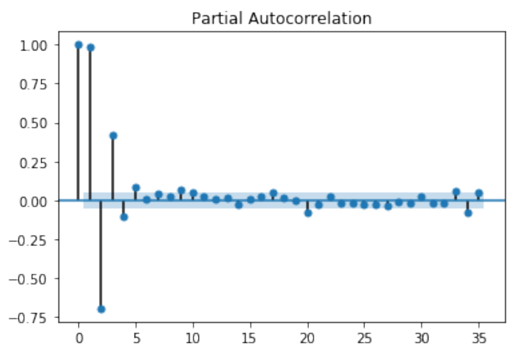
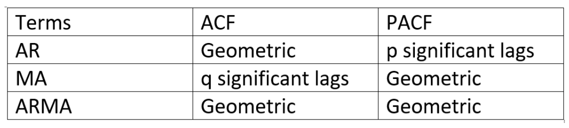


Fig. 7. Partial Autocorrelation of the Rhine river dataset up to a lag of 35 days.

The ACF and PACF plots should be considered together to define the process. For the AR process, we expect that the ACF plot will gradually decrease and simultaneously the PACF should have a sharp drop after p significant lags. To define a MA process, we expect the opposite from the ACF and PACF plots, meaning that: the ACF should show a sharp drop after a certain q number of lags while PACF should show a geometric or gradual decreasing trend. On the other hand, if both ACF and PACF plots demonstrate a gradual decreasing pattern, then the ARMA process should be considered for modeling.



Terms	ACF	PACF
AR	Geometric	p significant lags
MA	q significant lags	Geometric
ARMA	Geometric	Geometric

Fig. 8. Taxonomy of modelling decisions based on ACF & PACF plots.

Following the taxonomy we have a strong inclination to assume an AR(4) process (ACF plot showing a geometric decrease while PACF has four significant lags). This will define our baseline model against which we're going to evaluate our sophisticated machine learning methods, i.e. the LSTM and CNN model approaches.

Both plots are drawn as bar charts showing the 95% and 99% confidence intervals as horizontal lines. Bars that cross these confidence intervals are therefore more significant and worth noting. Some useful patterns you may observe on these plots are: The model is AR if the ACF trails off after a lag and has a hard cut-off in the PACF after a lag. This lag is taken as the value for p. The model is MA if the PACF trails off after a lag and has a hard cut-off in the ACF after the lag. This lag value is taken as the value for q. The model is a mix of AR and MA if both the ACF and PACF trail off.

III. ARIMA MODELS

An autoregression model is a linear regression model that uses lagged variables as input variables. An ARMA model [1] is specified as

$$y_i = c + \sum_{i=1}^{p} \varphi_i y_{t-i} + \sum_{i=1}^{p} \theta_i \varepsilon_{t-i}, \tag{1}$$

which depicts the fact that it takes both lag values of the variable itself as well as previous error terms into account, weighing them according to the estimated influence on the current value. We could calculate the linear regression model manually using the LinearRegession class in scikit-learn and manually specify the lag input variables to use. Alternately, the statsmodels library provides an autoregression model where you must specify an appropriate lag value and trains a linear regression model. It is provided in the AutoReg class [8].

Following our intuition of examining the ACF and PACF plots, we fit an AR(4) model to our dataset and examine the results. When once again doing a 90-10 split for the respective train and test percentage, we get an RMSE of 12.815. This outperforms the naive approach of persisting values and can thus be considered skillful.

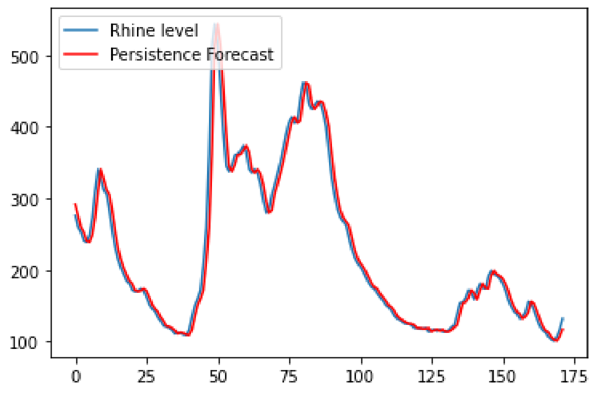


Fig. 9. AR(4) model, plotted in red against the realized Rhine water level. Both graphs overlap and can't be distinguished by looking at the plot.

Below is the plot using only a seven day test window, yielding an error rate of 4.258.

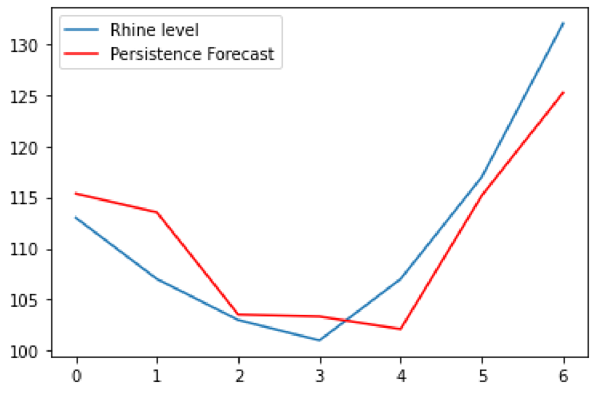


Fig. 10. AR(4) model, plotted against the realized Rhine water level.

As we can see, we can indeed improve upon our naive forecast model. Grid search regarding other lag and differencing parameters led to no improvement of performance, so in this model class (with only considering our univariate time series) we can consider it the most skillful model.

Next, we're going to go deeper on the seasonal structure of our data. The intuition that the Rhine level follows a periodical pattern leads directly to the application of the SARIMA model (which has proven not to yield better results than the ARIMA model; thus, a Grid Search of various SARIMA hyperparameters has been outsourced to the Jupyter notebook file). As just stated, we're focusing on the ARIMA model in this paper instead. The configuration of ARIMA models requires careful analysis and domain expertise in order to configure the hyperparameters. We are circumventing said approach by configuring the model with grid searching a suite of hyperparameter configurations in order to discover what works best. Often, this process can reveal non-intuitive model configurations that result in lower forecast error than those configurations specified through careful analysis. The results for Grid Searching the ARIMA parameters is depicted in Fig. 11. As we see, the only model that slightly outperforms the AR(4) model is the ARIMA(4, 0, 1) configuration, again convincing us, analogously to our findings from the Augmented Dickey-Fuller test, that we don't have to take differencing into account[2].

```
ARIMA(0, 0, 0) RMSE=83.056
ARIMA(0, 0, 1) RMSE=43.204
ARIMA(0, 1, 0) RMSE=20.580
ARIMA(0, 1, 1) RMSE=14.318
ARIMA(0, 1, 2) RMSE=13.152
ARIMA(0, 2, 0) RMSE=16.210
ARIMA(0, 2, 1) RMSE=15.250
ARIMA(0, 2, 2) RMSE=14.773
ARIMA(1, 0, 0) RMSE=20.475
ARIMA(1, 0, 2) RMSE=12.980
ARIMA(1, 1, 0) RMSE=14.915
ARIMA(1, 1, 1) RMSE=13.215
ARIMA(1, 1, 2) RMSE=13.089
ARIMA(1, 2, 0) RMSE=15.747
ARIMA(1, 2, 1) RMSE=14.936
ARIMA(2, 0, 0) RMSE=14.470
ARIMA(2, 0, 1) RMSE=12.940
ARIMA(2, 0, 2) RMSE=12.850
ARIMA(2, 1, 0) RMSE=13.052
ARIMA(2, 1, 1) RMSE=12.996
ARIMA(2, 1, 2) RMSE=12.909
ARIMA(2, 2, 0) RMSE=14.677
ARIMA(2, 2, 1) RMSE=13.069
ARIMA(4, 0, 0) RMSE=12.837
ARIMA(4, 0, 1) RMSE=12.801
ARIMA(4, 1, 0) RMSE=12.944
ARIMA(4, 1, 1) RMSE=12.913
ARIMA(4, 2, 0) RMSE=14.326
```

Fig. 11. RMSE's of different configurations of the ARIMA model.

[2]For model selection purposes, here we tried a 2/3-1/3 split, yielding the same ranking as a 90-10 split.

IV. MLP, CNN and LSTM approaches

Now we're going to focus on different pure machine learning methods, which have historically proven to yield great results on time series forecasting problems. We begin our investigation by configuring Multilayer Perceptron (MLP) models [4]. Our approach will be to work our way up from rather simple approaches like the MLP network do more advanced structures like CNN's or LSTM approaches. An MLP is a feedforward neural network model that can be, similarly to the ARIMA methods used before, take multiple lag observations and using them as input features to be able to predict one or more time steps from those observations. The input parameters for our model are[3]

- n_input: The number of lag observations to use as input to the model.
- n_nodes: The number of nodes to use in the hidden layer.
- n_epochs: The number of times to expose the model to the whole training dataset.
- n_batch: The number of samples within an epoch after which the weights are updated

After trying different combinations for the number of inputs, I used the specification of the tuple of [36, 50, 100, 100]. Unintuitively, after running the algorithm for several times, 36 lags seemed to be more accurate than simpler configurations. After fitting the model and repeatedly evaluating it to get in parts rid of the inherent randomness, the algorithm yielded the following result.

```
mlp: 5.936 RMSE (+/- 2.019)
```

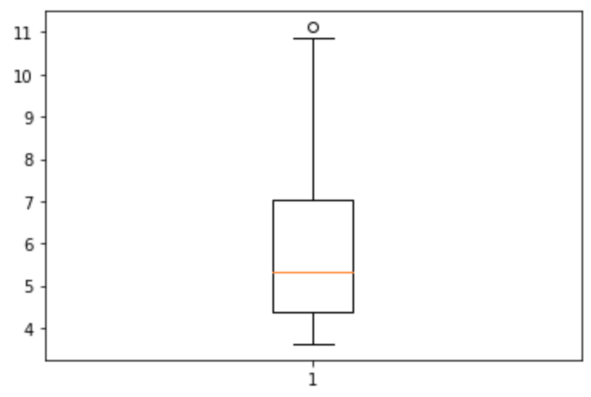


Fig. 12. Box and Whisker Plot of Multilayer Perceptron RMSE Forecasting the Rhine Level.

Next, we're going to explore CNNs and LTSMs, as well as combinations of the two. CNNs have traditionally been developed for two-dimensional image data, although its usage can be expanded on time series forecasting poblems. In its simplest form, the one-dimensional CNN, the network is configured with a convolutional hidden layer that operates over a 1D sequence [5]. This is followed by a second convolutional layer followed by a pooling layer which distills the output of the convolutional layer to the most salient elements. Finally,

[3]This methodology follows the state-of-the-art in the industry. See Jason Brownlee - Deep Learning for Time Series Forecasting, 2020.

the network consists of a dense fully connected layer that interprets the features extracted by the convolutional part of the model. A flatten layer is used between the convolutional layers and the dense layer to reduce the feature maps to a single one-dimensional vector. One has to carefully construct the input to the network, because of the fact that the specification requires a specific input format, i.e. [samples, timesteps, features]. Since we only have one feature (namely the Rhine water level itself) this is straightforward using basic numpy functionality. For our problem, I chose a convolutional layer with 256 filter maps and a kernel size of 4, followed by a max pooling layer and a dense layer to interpret the input features. The model was fit using the efficient Adam version of stochastic gradient descent and optimized using the mean squared error oss function, for the sake of comparability to the models tested above. The performance is depicted in Fig. 13, where we can see a clear improvement to the existing models.

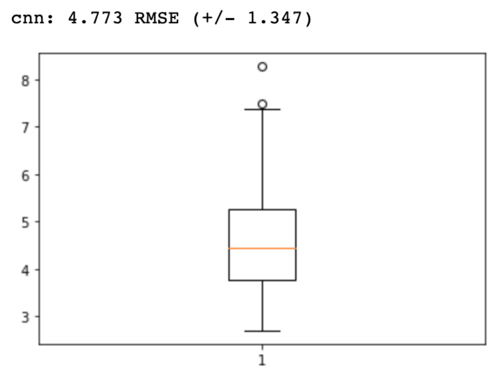


Fig. 13. Box and Whisker Plot of the CNN RMSE Forecasting the Rhine Level.

A CNN model can also be used in a hybrid model with an LSTM backend. In a first step the CNN inteprets subsequences of the input that are then passed on as a sequence to an LSTM model to further process it This hybrid model is called a CNN-LSTM. The CNN-LSTM approach yielded an RMSE of 5.097, with a standard deviation of +/- 1.366. Compared to our AR-approach earlier we didn't improve with that approach.

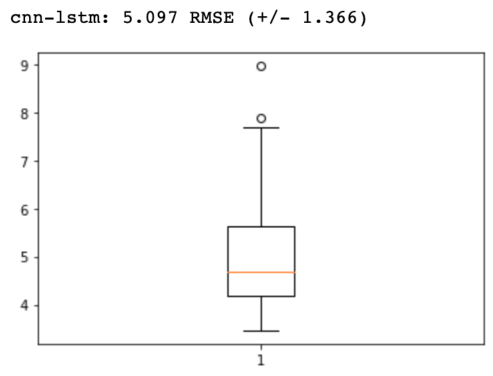


Fig. 14. Box and Whisker Plot of the CNN-LSTM RMSE Forecasting the Rhine Level.

In our final model of this section, we're going to test a standalone LSTM approach. The LSTM, initially proposed by [9], can be seen as an alternative to standard neural networks. Due to its nature be applied to time series in a straightfoward way. The hyperparameters for the LSTM model are the same as in the MLP model; they are:

- n_input: The number of prior inputs to use as input for the model (e.g. 7 days).
- n_nodes: The number of nodes to use in the hidden layer (e.g. 50).
- n_epochs: The number of training epochs (e.g. 1000).
- n_batch: The number of samples to include in each minibatch (e.g. 32).
- n_diff: The difference order (e.g. 0 or 180).

The average RMSE of the LSTM model is rather disappointing, leading to the conclusion that an additional predictor is necessary.

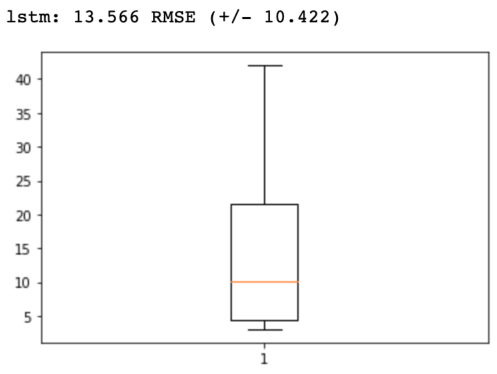


Fig. 15. Box and Whisker Plot of the CNN-LSTM RMSE Forecasting the Rhine Level.

V. Multivariate Forecasting

Our approach now is to use other data than the plain vanilla time series of the Rhine level to test if further enhancement of influential variables increases our capability of predicting. The decision process involved taking into account the Rhine level of stations tracking the precipitation level along the Rhine river. To accomplish this, the first step was in marking over 400 coordinates along the Rhine via the Google Maps API and fetch stations located near said coordinates (to be exact, the distance had to be strictly less than 10km). To calculate the distance, the Haversine formula [10] was used. The haversine formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes, which makes it perfect for our purpose. The next step was in getting data from the "Deutsche Wetterdienst" regarding historical and recent precepitation data [11]. We achieved this by using the Keras/TensorFlow Libraries and their implementation of LSTM networks.

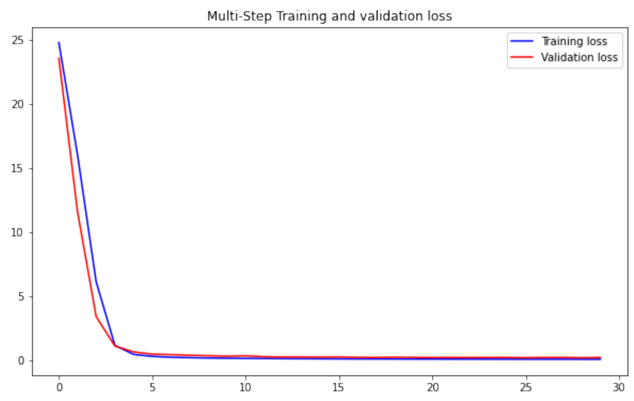


Fig. 16. Training and validation loss for our training and validation set.

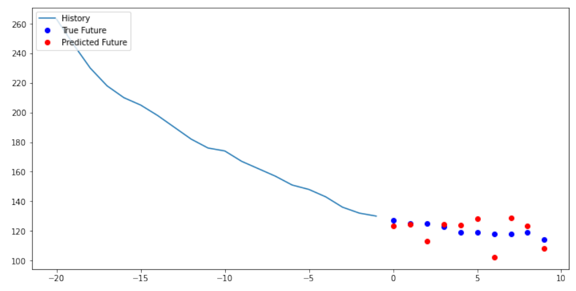


Fig. 17. Plot of the forecast against the last seven true Rhine level values.

The additional predictor was thus constructed by taking into account the amount of precipitation in the proximity of the fetching stations, which should get clear in the additional Jupyter notebook. We tried different number of neurons for the following network, 32 and 16 are the most suitable ones, whereas a greater number or neurons has the tendency to overfitting. The results of this approach were impressive: after a training of 4 epochs, the MSE was around 1.1407, which clearly outperforms all previous methods. One can critize this low error by stating the guess of an overfitted model, but since our challenge was to predict a seven day forecast, this fact was materialized in our thought process. Further increasing the accuracy would definitely lead to the model just memorizing the time series, depicted in the fact that the error rate quickly converges to zero after the fourth epoch. The final forecasts for our challenge are listed below:

	Forecasts
2020-06-08	132.549759
2020-06-09	127.396751
2020-06-10	118.665489
2020-06-11	122.375198
2020-06-12	128.689163
2020-06-13	120.729950
2020-06-14	108.064194
2020-06-15	137.830338
2020-06-16	116.712555
2020-06-17	124.005730

VI. Conclusion

In this seminar we examined several methods, beginning by standard time series analysis techniques, i.e. the ARIMA model. We established a baseline forecast and proved, that going the path of an univariate forecast, these methods barely outperform naive models. We proceeded to investigate modern architectures like MLP, CNN's, LSTM and hybrid approaches, where our result showed that a multivariate version can significantly improve the model performance. Further investigations regarding more advanced grid searches and the carefully selected inclusion of additional parameters could, given the time and resources, again improve the forecasting performance, especially regarding forecasting intervals longer than seven days and incorporating multiple steps into one forecast.

References

[1] Citation: Stock J, Watson MW. Introduction to Econometrics. New York: Prentice Hall; 2003.
[2] Cheung, Y.-W., & Lai, K. S. (1995). Lag order and critical values of the augmented Dickey–Fuller test. Journal of Business & Economic Statistics, 13(3), 277–280.
[3] Hamilton, J. D. (1994), Time Series Analysis , Princeton University Press.
[4] Almeida, L.B. Multilayer perceptrons, in Handbook of Neural Computation, IOP Publishing Ltd and Oxford University Press, 1997.
[5] LeCun, Y., Bengio, Y. and Hinton, G., 2015. Deep learning. Nature, 521(7553), pp.436-444.
[6] http://pandas.pydata.org/pandas-docs/stable/visualization.html#autocorrelation-plot
[7] George Edward Pelham Box and Gwilym Jenkins. 1990. Time Series Analysis, Forecasting and Control. Holden-Day, Inc., USA.
[8] https://www.statsmodels.org/devel/generated/statsmodels.tsa.ar_model.AutoReg.html.
[9] Sepp Hochreiter, Jürgen Schmidhuber: Long short-term memory In: Neural Computation (journal), vol. 9, issue 8, S. 1735–1780, 1997.
[10] R. W. Sinnott, "Virtues of the Haversine", Sky and Telescope 68 (2), 159 (1984).
[11] https://opendata.dwd.de/climate_environment/CDC/observations_germany/climate/daily/kl/recent/