Technical University of Munich

# Predictive Process Monitoring

## Development of machine learning and deep learning models to predict the remaining time of process instances

Report for the Predictive Process Monitoring project in the course

Advanced Practical Course – Solving Information System Problems Trough Data Science

at Technical University of Munich.


**Submitted by** Eric Näser

Lukasstraße 6

81735 Munich

+49 1575 0155003


**Submitted at** Munich, 17.01.2023

# Table of Contents

# List of Figures

# List of Abbreviations

| | |
|---|---|
| **LSTM** | Long Short Term Memory |
| **RNN** | Recurrent Neural Network |
| **FNN** | Feedforward Neural Network |
| **MAE** | Mean Absolute Error |
| **RMSE** | Root Mean Squared Error |

# List of Tables

# 1. Introduction

Data contains valuable information about events in the past. For me, it is always particularly interesting to use this past information to gain insight into future developments. Because of this personal interest I have focused my courses at university and practical experiences as a working student or intern on Machine Learning. For example, I have already worked in artificial intelligence research at Siemens and in Data Science at Infineon. Current technological trends in business also increasingly involve the use of Machine Learning [1, S. 1]. Complex correlations can thus be analysed in a shorter time and with greater accuracy [2, S. 4]. Especially in the medical field, this even makes developments possible that not only increase profit, but also the health of people [3]. The practical relevance and my interest in creating an insight into the future using Machine Learning led me to choose the project in Predictive Process Modelling.

# 2. Methodology and implementation

For predicting the remaining time of running process instances, the ***machine learning based approach*** and the ***deep learning based approach*** of the lecture are used as guide. To sustain the quality, maintainability, and reusability of the developed Python code, each approach utilizes the same classes and functions for data engineering and preparation.

The implemented process for remaining time prediction starts with ***importing*** the event log as a Dataframe of the library Pandas[1]. Furthermore, the target attribute, remaining time, is calculated by subtracting a traces duration with the duration since trace start for an event. In addition to that, the last event of a trace, the patients age and temporal information such as month, week and hour are added depending on the parameter configuration. Next, the whole dataset is ***splitted*** into a training, validation, and testing dataset[2] with shares of 70%, 20% and 10%. To improve the performance of machine learning and deep learning models, the

[1] https://pandas.pydata.org/

[2] This report and the code refers to the dataset for hyperparameter optimization as validation set and the dataset for an unbiased estimate as test set as it is most common in research, literature, and other lectures.

numerical attributes like remaining time and age are standardized. For standardization, the StandardScaler class of Scikit-learn[3] is fitted to the training set and applied to all three splitted sets. Furthermore, the traces in the splitted sets are transformed from an event log representation to an event stream representation for an easier extraction of the prefixes.

Now, that the traces are in an event stream representation, they can be extracted, bucketed, and encoded. Essential for these steps are the parameters “bucketing_lower_bound” and “bucketing_upper_bound” in the parameter configuration. These parameters define the range of prefix lengths that are extracted. The ***prefix extraction*** is realized by cutting off a trace at the defined length. If the trace is shorter than the defined prefix length, the extracted prefix is filled up with None values until it has reached the desired length. For each number in the range between “bucketing_lower_bound” and “bucketing_upper_bound”, a Dataframe with traces of the defined length is created to ***bucket the prefixes***. These Dataframes get stored in a dictionary, where each key represents the length of the prefixes, and each value represents the corresponding Dataframe with extracted prefixes of that length. For bucketing, the three techniques Single Length, Prefix Length and Clustering are implemented. To use Single Length and Clustering bucketing, “bucketing_lower_bound” and “bucketing_upper_bound” must be of equal size. Contrary to that, “bucketing_upper_bound” must be larger than “bucketing_lower_bound” for Prefix Length bucketing. The **prefix encoding** is available in both the aggregation and the last state technique. All activities get one hot encoded by the OneHotEncoder class of Scikit-learning. Since longer prefixes contain the same activity multiple times, the OneHotEncoder class creates multiple columns for encodings of the same activity. To reduce the dimensionality and complexity of the data to train on, all columns with encodings for the same activity get added together. Since machine learning models require encoded attributes for their usage, bucketing with the Clustering technique is done after encoding by applying the KMeans algorithm from Scikit-learn to the data. The implemented process for the data preparation is visualized in the following figure. The visualization of the splitted datasets was omitted for reasons of clarity.

[3] https://scikit-learn.org/stable/

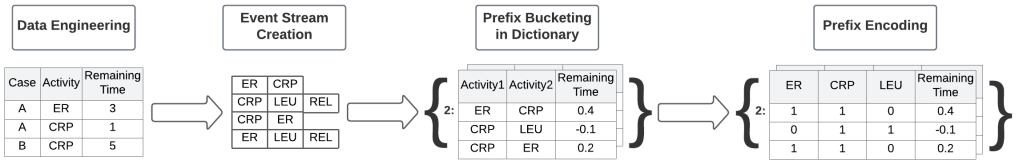


Figure 1: Data Preparation Process

Recent ***machine learning based approaches*** for predictive process monitoring apply Support Vector Machines, K-Nearest Neighbors and Gradient Boosted Trees [4] [5]. The implementation of Support Vector Machines and K-Nearest Neighbors is realized by the classes SVR and KNeighborsRegressor of Scikit-learn. For the utilization of Gradient Boosted Tress, the XGBoost[4] library is used. During the training phase, one algorithm is fitted for each bucket in the training dictionary of Dataframes. For each bucket label, a trained machine learning model is stored in a dictionary. If a running trace must be predicted, its bucket is determined by its length or the fitted KMeans model, which was used for clustering. Subsequently, a trained machine learning models of the same bucket is selected from the dictionary of machine learning models and used for predicting the remaining time of the running trace. The predictions get scored with the Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE).

To support the traceability and trust of remaining time predictions, an ***explainability method*** has been realized for the machine learning models. The machine learning based approach does not consist of one machine learning model but instead of multiple machine learning models for multiple buckets. Thus, the whole workflow is seen as one model with multiple parameters from configuration, which needs to be explained. For explainability, a dashboard with diagrams visualising the influence of parameters on the predictions has been developed. MAE has been selected as an accuracy metric because of its intuitive interpretability. To make the MAE scores more tangible, the values have been converted from minutes to hours.

More advanced methods for remaining time prediction are found at ***deep learning based approaches***. This approach makes use of neural networks, which can be created in different architectures such as feedforward (FNN), recurrent (RNN)

[4] https://xgboost.readthedocs.io/en/stable/#

and convolutional neural networks. Suitable for remaining time prediction are FNNs and RNNs. FNNs are useful for tabular data, but don't consider dependencies between activities like in event logs. However, they are rather simple architecture and a good entry to deep learning based remaining time predictions. RNNs are good at leveraging dependencies between activities and time series data like event logs. Since RNNs suffer from the vanishing gradient problem, their subtype of Long Short Term Memory networks (LSTM) is often used. [6, S. 5–6] Pytorch Lightning[5] is used for the implementation of both network architectures, FNNs and LSTMs.

# 3. Analysis

The analyses of the machine learning and deep learning based approaches are performed on the "sepsis_cases_1" dataset. To investigate the effect of different design choices for the machine learning based approach, a search space is defined. Ray Tune[6] is used to optimize the design choices in the search space by minimizing the resulting RMSE-Score. Table 1 shows the search space for design choices in the machine learning based approach. Each grid search point is repeated ten times.

Table 1: Search space for design choices in the machine learning based approach

| **Parameter** | **Distribution** | **Values** |
|---|---|---|
| temporal_split_sort_by | Choice | x ∈ ["timesincecasestart", "time:timestamp"] |
| add_age | Choice | x ∈ [True, False] |
| add_temporal_information | Choice | x ∈ [True, False] |
| bucketing_technique | Grid | x ∈ ["SingleBucket", "PrefixLength", "Clustering"] |
| bucketing_lower_bound | Uniform integer | x ∈ [2, 20] |
| bucketing_upper_bound | Uniform integer | x ∈ [2, 20] |
| kmeans_n_clusters | Uniform integer | x ∈ [5, 15] |
| encoding_technique | Grid | x ∈ ["Aggregation", "LastState"] |

[5] https://www.pytorchlightning.ai/

[6] https://docs.ray.io/en/latest/tune/index.html

| Parameter | Distribution | Values |
|---|---|---|
| ml_algorithm | Grid | x ∈ ["SVR", "KNN", "XGB"] |
| svr_c | Uniform float | x ∈ [0.5, 1.5] |
| svr_epsilon | Uniform float | x ∈ [0.1, 1.0] |
| knn_n_neighbors | Uniform integer | x ∈ [5, 15] |
| knn_leaf_size | Uniform integer | x ∈ [25, 45] |
| xgb_eta | Uniform float | x ∈ [0.1, 1.0] |
| xgb_max_depth | Uniform integer | x ∈ [5, 15] |

The results show that the machine learning based approach performs best with the use of Support Vector Machine, aggregation encoding, and long prefixes. While storing all prefixes in one bucket achieved the lowest RMSE score, Prefix Length bucketing reaches slightly higher score but shows a smaller variance. Sorting the dataset by timestamp instead of the time since case start before splitting also resulted in a better outcome as well. No differences are evident by including age and temporal attribute. For the best configuration from validation, the model achieves a RMSE score of 0.02 and a MAE score of 762.94 on the test data.

For the analysis of the deep learning based approach, the FNN and LSTM are trained for 50 epochs with a batch size of 50. The age, month, week, and hour attributes are added to the data for leveraging the benefits of LSTMs with temporal dependencies. Prefix padded and the frequency of activities are selected as encoding techniques for sequences and events to include as much information as possible but also keep the dimensionality of the data low. Furthermore, the learning rate is set to a value of 0.01, dropout to a value of 0.2 and hidden_dim to 100. Both neural networks use Mean Squared Error as a loss metric, which is optimized by Adam. Additional configuration choices are listed below in Table 2.

Table 2: Configuration of neural networks

| FNN | | LSTM | |
|---|---|---|---|
| **Parameter** | **Value** | **Parameter** | **Value** |
| # layers | 5 | # layers | [2 lstm, 2 fully connected] |

| FNN | | LSTM | |
|---|---|---|---|
| # neurons per layer | [input_dim, hidden _dim, hidden_dim * 1.2, hidden_dim * 0.6, hidden_dim * 0.3, output_dim] | # neurons per layer | [input_dim, hidden_dim, hidden_dim, hidden_dim * 0.3, output_dim] |
| # dropout layer | 5 | # dropout layer | 2 |
| Activation functions | [relu, relu, relu, relu, tanh] | Activation functions | [relu, tanh] |

The validation (left) and test loss (right) of the FNN (orange) and LSTM (green) are visualized in Figure 2. Further figures use the same colour scheme and arrangement for validation and test loss curves as Figure 2.

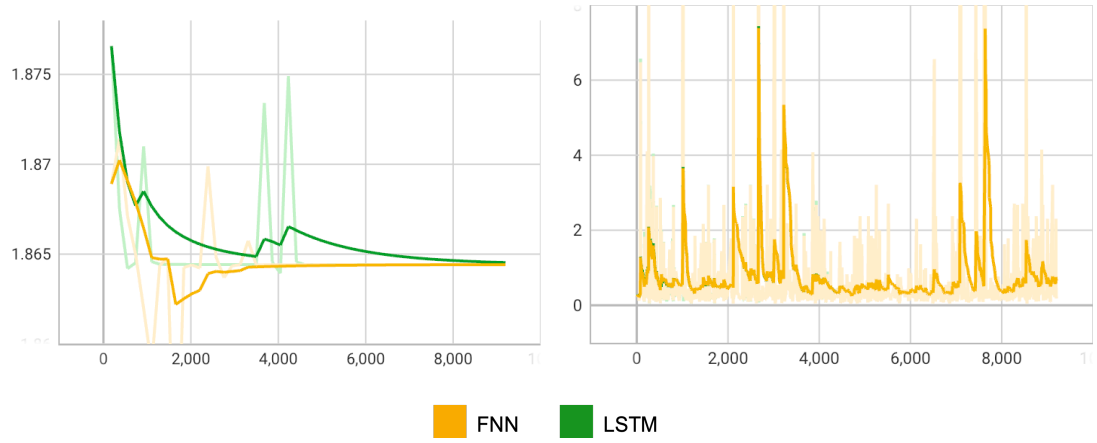


Figure 2: Validation and test loss of FNN and LSTM

The execution of the proposed FNN resulted in an RMSE score of 7.14 and an MAE score of 19190.72. Also considering the Tensorboard graph with a decline of the validation loss from 1.87 to only 1.85, one can conclude, that the FNN did not learn the prediction of the remaining time successfully. A similar pattern for the validation loss and learning behaviour is also evident for the LSTM. It achieves the same scores as the FNN with a RMSE score of 7.14 and a MAE score of 19190.72.

Following the disappointing results, manual hyperparameter optimization as well as changes in the network architectures and data engineering steps are undertaken. Figure 3 shows the validation and test loss of the FNN and LSTM with One-hot encoding.

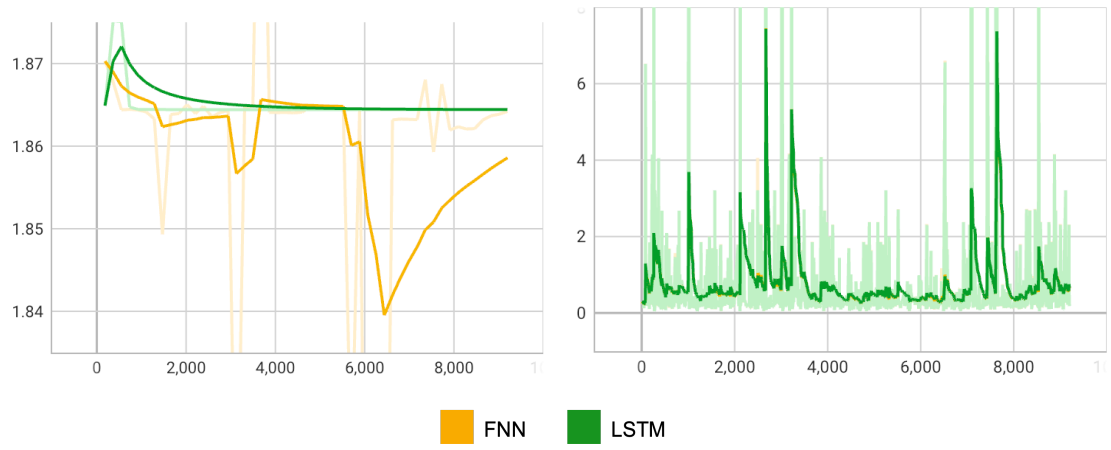


Figure 3: Validation and test loss with One-hot encoding

Both network architectures show a different learning pattern with One-hot encoding but result in the same RMSE and MAE scores on test data as with frequency encoding. Thus, no improvements of the results are evident.

Since the neural networks demonstrate a large generalization gap between the validation and test loss, additional regularization is needed. Thus, batch normalization is applied after every LSTM block and linear layer except the ones for output. Figure 4 visualizes the validation and test loss for both network architectures with batch normalization.

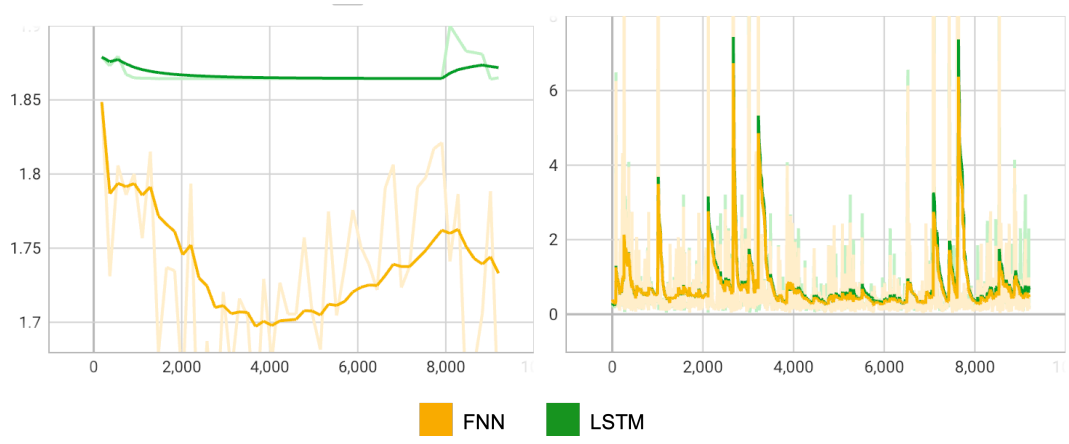


Figure 4: Validation and test loss with batch normalization

By applying batch normalization, FNNs RMSE score decreased to 6.62 and a MAE score of 18106.46. While the FNN achieved better results with batch normalization, scores of LSTMs remained constant. Batch normalization stays implemented since it's improved results.

Additional experiments are underdone with a decrease in the network's complexity. A decrease in complexity is realized for the FNN by deleting the fully connected layer with size "hidden_dim" * 0.6 as well as for the LSTM by reducing the number of layers from two to one.

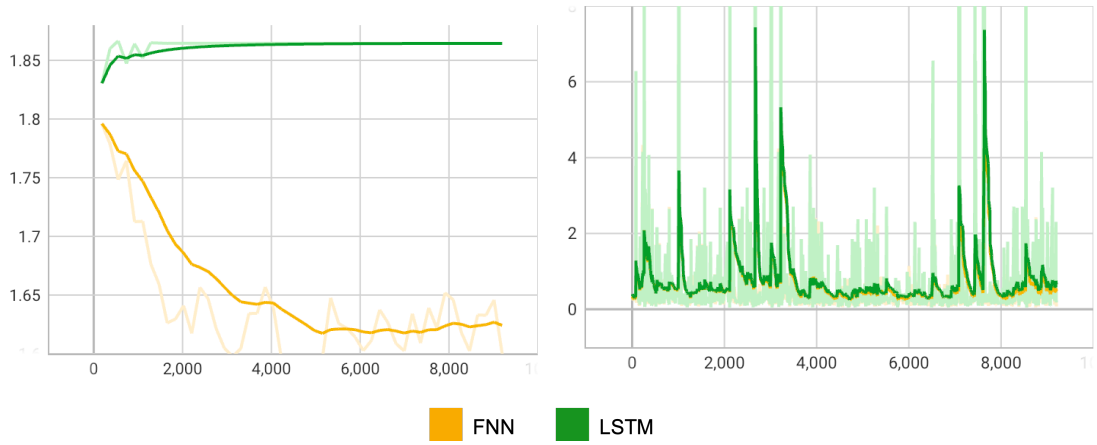


Figure 5: Validation and test loss with decrease in number of layers

Once more, an improvement is observable for the FNN, but not for the LSTM. The FNN scores a RMSE of 5.64 and an MAE of 16677.14 following the decrease in the number of layers.

Since the results are still not useful and satisfying, additional experiments are realized. Figure 6 visualizes the validation losses for a decrease in batch size to 20 (top left), an increase in batch size to 70 (top right), a decrease in "hidden_dim" to 50 (bottom left) and an increase in "hidden_dim" to 150 (bottom right). All experiments result in worse RMSE and MAE scores than achieved before and thus won't be analysed further.

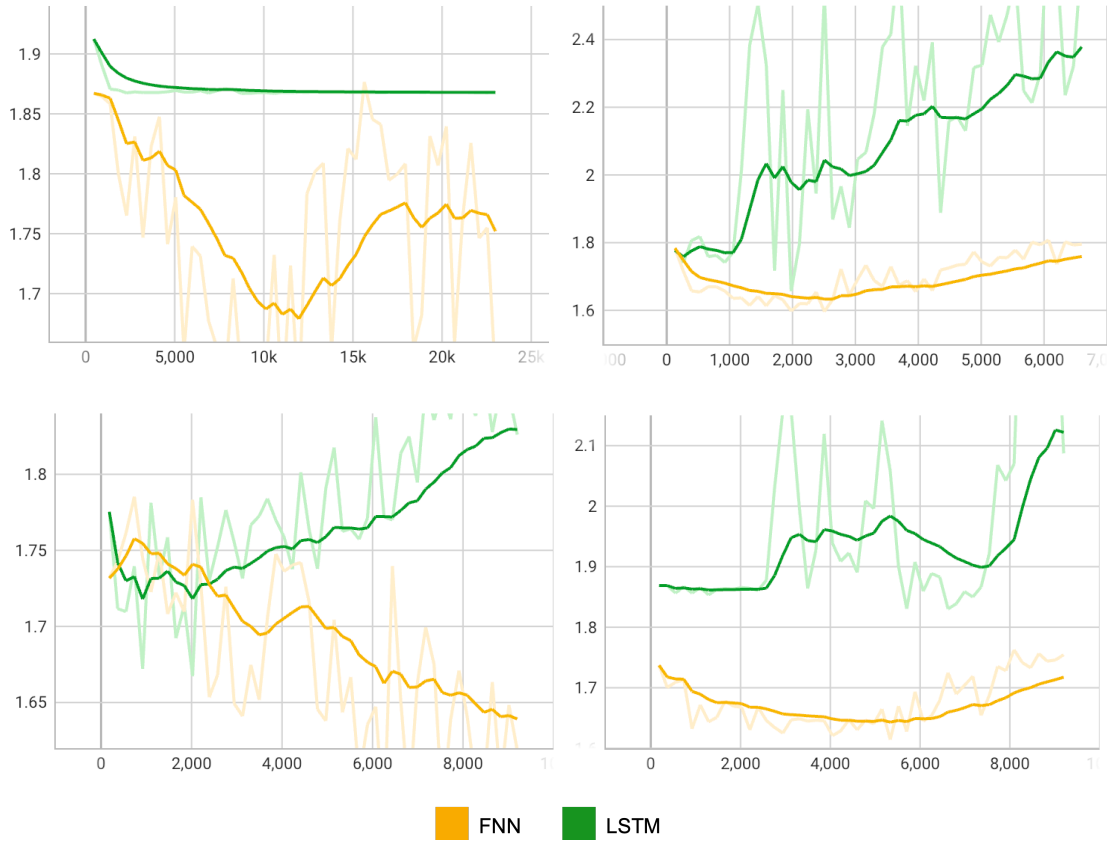


Figure 6: Validation loss for changes in batch size and "hidden_dim"

# 4. Results

In this project, remaining time prediction has been realized with machine learning and deep learning based approaches. Looking at the results of the machine learning against the deep learning based approaches, it can be concluded, that the machine learning based approaches achieved better results. While machine learning models reached an RMSE score of 0.02 and a MAE score of 762.94 on the test data, neural networks were not able to learn the prediction of remaining times and performed way worse. Especially Support Vector Machines, which are trained on aggregated prefixes in a single bucket, achieve good results. Thus, time intensive treatments of sepsis can be detected with a high accuracy and resources for the patient's treatment can be increased to enhance their chances of survival. Additional experiments with the deep learning based approach could focus on different data preparation techniques as well as architectures beside neural networks, like Markov models and Automaton based approaches.

# Bibliography


[1] S. Ransbotham, S. Khodabandeh, R. Fehling, B. LaFountain, und D. Kiron, „Pioneers Combine Strategy, Organizational Behavior, and Technology“.

[2] A. V. Joshi, *Machine Learning and Artificial Intelligence*. Cham: Springer International Publishing, 2020. doi: 10.1007/978-3-030-26622-6.

[3] „Chasing Value as AI Transforms Health Care“, *BCG Global*, 22. Dezember 2020. https://www.bcg.com/publications/2019/chasing-value-as-ai-transforms-health-care (zugegriffen 9. Januar 2023).

[4] J. Vandenabeele, G. Vermaut, J. Peeperkorn, und J. De Weerdt, „Enhancing Stochastic Petri Net-based Remaining Time Prediction using k-Nearest Neighbors“. arXiv, 27. Juni 2022. Zugegriffen: 6. Januar 2023. [Online]. Verfügbar unter: http://arxiv.org/abs/2206.13109

[5] A. Leontjeva, R. Conforti, C. Di Francescomarino, M. Dumas, und F. M. Maggi, „Complex Symbolic Sequence Encodings for Predictive Monitoring of Business Processes“, in *Business Process Management*, Bd. 9253, H. R. Motahari-Nezhad, J. Recker, und M. Weidlich, Hrsg. Cham: Springer International Publishing, 2015, S. 297–313. doi: 10.1007/978-3-319-23063-4_21.

[6] E. Rama-Maneiro, J. C. Vidal, und M. Lama, „Deep Learning for Predictive Business Process Monitoring: Review and Benchmark“. arXiv, 29. Oktober 2021. Zugegriffen: 9. Januar 2023. [Online]. Verfügbar unter: http://arxiv.org/abs/2009.13251