Report Assigment 2 APML24/25@UU

Rens van Moorsel, Ruben de Groot, Ewoud ter Wee, Lucas van der Jagt
Group 17

December 13, 2024

Abstract

In this report, a dataset containing Sepsis cases in a hospital will be pre-precessed so it can be used to predict the remaining duration of the stay of a patient. The dataset will be pre-processed such that empty cells are filled without being disadvantageous for the machine learning algorithm. After this we will use aggregating state information encoding and bucketing with different prefix lengths. Then two different regression models, including a Random Forest regressor and a Multi-Layer Perceptron will be trained. Evaluations using MAE, MSE, RMSE, and R^2 show that the MLP model better captures complex relationships in the data than the Random Forest. We also add features to further improve the predictions. While none of the tested approaches achieves accurate predictions, the MLP combined with pre-processing and the time-based feature 'Elapsed time' shows a step in the right direction.

1 Introduction

This report is about training a machine learning algorithm to predict the remaining duration of patients with Sepsis in hospitals. For training the model, a dataset with a real life event log of Sepsis cases is used, but this dataset is not preprocessed yet and can not be used immediately. The task for our research is to preprocess the data and fit it to a Random Forest Regressor and a MLP Regressor. Predicting the duration of the patient's stay can be useful for hospital planning and scheduling employees. In addition, it is helpful for a patient to know their duration of a hospital stay, so they can plan around this.

Pre-processing is important as Regressor algorithms need numeric data. The dataset used contains a lot of empty values and parts of the features are strings. During pre-processing decisions must be made to convert this data into a dataset, so that it can be fed to a Regressor. To check the reliability of the Regressors, several errors are measured on predictions with the train data, and on predictions with the test data.

2 Data

Before building any predictive model it is crucial to thoroughly understand the data. A clear understanding of the data ensures that the underlying patterns, relationships, and characteristics are identified. Analyzing and exploring the data helps by a better understanding of the descriptions of the data, the amount of empty cells and the correlation between features. Only then can the data be used to train a model.

2.1 Descriptions of the data

The data used in the experiment is from a hospital and contains the event logs of patients with Sepsis. It does not contain the duration of how long a patient is in the hospital yet, but this will be added later in the experiment. The table has multiple rows for each case and each row has his own part of the data. The data set has to be pre-processed to be useful for training.

The data set has 13333 rows (samples) with 34 features. Each row is an event and corresponds to a case ID. Based on the type of activity of the event, certain columns are filled in. The dataset includes both numerical and categorical data and contains a lot of different datatypes like boolean, int64, float64, Char, TimeStamp and String.

Looking at the activities, there was something very interesting that caught our attention: there was an event called "Return ER". Sometimes there were months between release of a patient and the moment the "Return ER" activity was present. We presumed this meant that the patient got released, went home, then returned to the ER much later. This is definitely something that we're going to have to remove during pre-processing.

A quick descriptive summary of the activities is as follows:

- **ER Registration:** `Rows of this activity contain the following features: InfectionSuspected, DiagnosticBlood, DisfuncOrg, SIRSCritTachypnea, Hypotension, SIRSCritHeartRate, Infusion, DiagnosticArtAstrup, Age, DiagnosticIC, DiagnosticSputum, DiagnosticLiquor, DiagnosticOther, SIRSCriteria2OrMore, DiagnosticXthorax, SIRSCritTemperature, DiagnosticUrinaryCulture, SIRSCritLeucos, Oliguria, DiagnosticLacticAcid, Diagnosis, Hypoxia, DiagnosticUrinarySediment, DiagnosticECG.`
- **CRP:** `This activity gives a value for the CRP column, however occasionally the activity also gives an age and diagnose.`
- **LacticAcid:** `This activity contains a value for the LacticAcid column, but even as by CRP, it occasionally gives an age and diagnose.`
- **Leucocytes:** Rows of this activity contain data for the Leucocytes column. Even as by LacticAcid and CRP it sometimes contain values for age and diagnose.
- **ER Sepsis Triage, ER Triage, IV Liquid:** `Rows that are of one of these activities contain occasionally an age and diagnose, but most of the time they do not have any additional data.`
- **Other activities:** `All other activities do not provide any additional data.`

The data is very inconsistent, as it has multiple rows that can be combined and the age and diagnose seems to be filled in for random rows. Furthermore, sometimes it happens that a row misses a value that it should have according to its activity. This are missing values. These inconsistencies should be handled correctly in the pre-processing.

2.2 Visualisations of the data

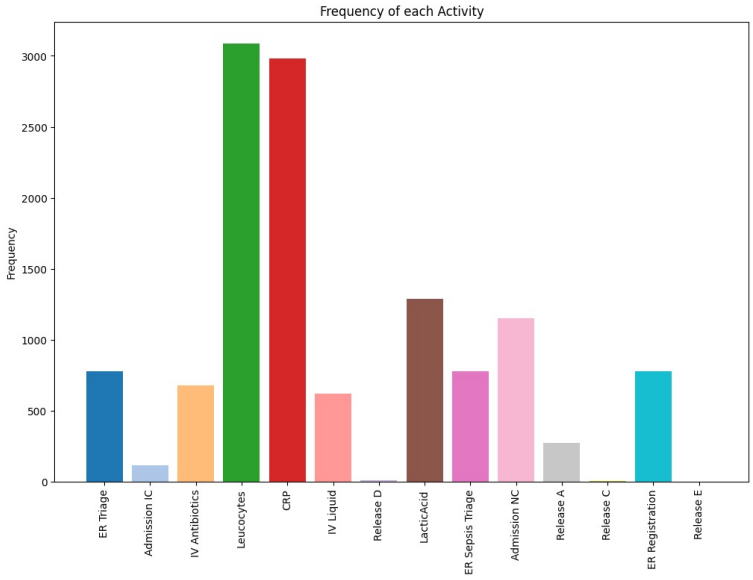


Figure 1: Activity Frequency

An import aspect of the data is the frequency of each activity in the data, as seen in figure 1. This is important as it gives a good overview how the activities are distributed, and the common ones need more focus during the pre-processing step, to make sure their data is consistent. Later on we will look at feature importances and then we see which features are very common in comparison to their importance. As can be seen, the CRP and Leucocytes are by far the most frequent activities, while there are obviously almost no release activities.

To get a good understanding how to pre-process the data and how to use it for training, it is important to understand what the datatypes of the features are. The datatype of each feature can be seen in table 1. We have to make sure all the data is converted into numerical data before we feed it into the Regressor.

Feature	**Type**	**Feature**	**Type**
Case ID	string	Age	int
Activity	string	DiagnosticIC	bool
Complete Timestamp	datetime	DiagnosticSputum	bool
Variant	string	DiagnosticLiquor	bool
Variant index	int	DiagnosticOther	bool
lifecycle:transition	string	SIRSCriteria2OrMore	bool
org:group	string	DiagnosticXthorax	bool
InfectionSuspected	bool	SIRSCritTemperature	bool
DiagnosticBlood	bool	DiagnosticUrinaryCulture	bool
DisfuncOrg	bool	SIRSCritLeucos	bool
SIRSCritTachypnea	bool	Oligurie	bool
Hypotensie	bool	DiagnosticLacticAcid	bool
SIRSCritHeartRate	bool	Diagnose	string
Infusion	bool	Hypoxie	bool
DiagnosticArtAstrup	bool	DiagnosticUrinarySediment	bool
Leucocytes	float	DiagnosticECG	bool
CRP	int	LacticAcid	float
remaining_time(days)	int		

Table 1: Feature Types

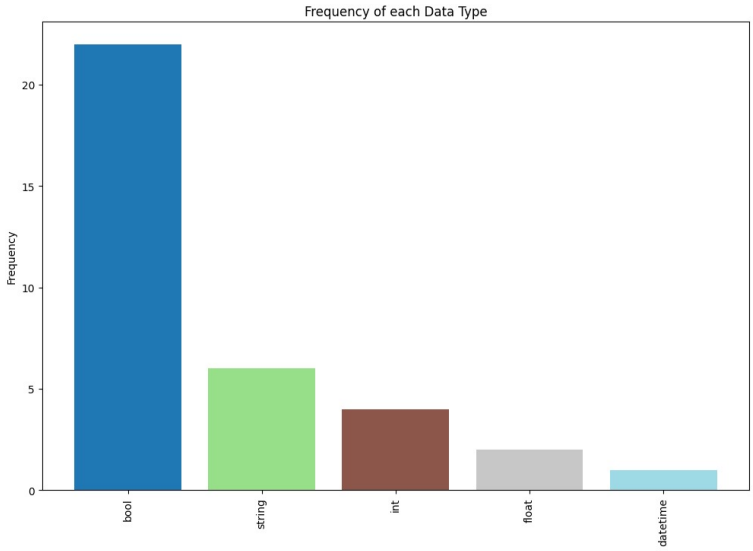


Figure 2: Data Types in the dataset

As shown in figure 2, most of the data consists out of bools. This data is easy to use, as it can directly be used in the regression model as 0 or 1. However, each row that has no value in it gets the value NaN. As a lot of rows are missing these values, it is important to handle this correctly in the pre-processing.

Another used datatype is string. This type can not directly be used in the regression model and should be handled during pre-processing. A good way of doing this is one-hot encoding, where each occuring string gets its own feature. Even as by the bools, the empty cells must be handled correctly during pre-processing.
Floats and ints are quite easy, as they can directly be used in the regression model, like the bools, but as by the bools, the empty cells will be NaN. These values must be pre-processed correctly to be able to be used in the regression model.

The last datatype used is the datetime. This type cannot directly be used in the regression model, but it is not neccesary for the Regressor. All rows have the Complete Timestamp feature filled in, and this is the only feature with datetime as datatype, so there are no empty datetime values.

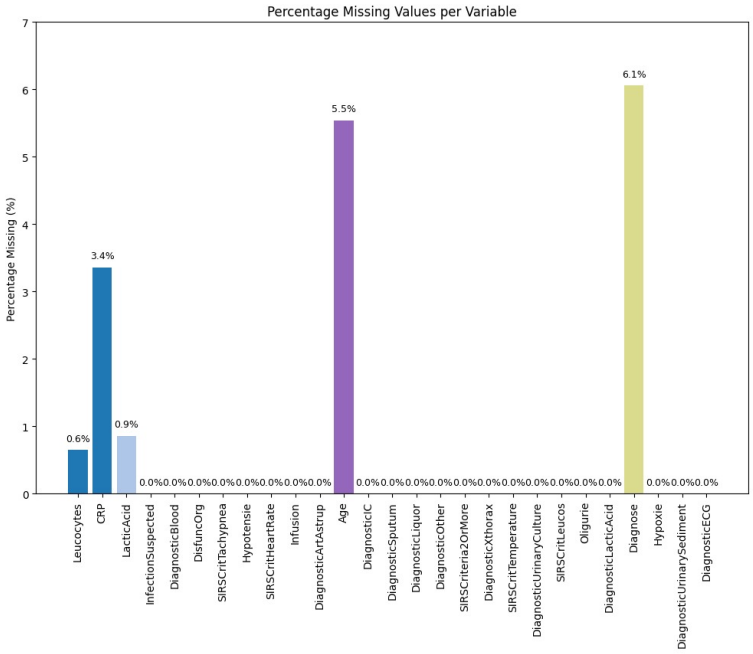


Figure 3: Missing data per feature

Besides the empty cells that are correlated to the activity of a row, there are also some missing cells that should be in the row, as we can see in Figure 3. To handle this correctly, it is important to know how often this occurs for each feature. In this graph, we can see how often a value is missing for each feature, while it should be present. As can be seen, most of the features are not missing at all, but the features "Leucocytes", "CRP", "LacticAcid", "Age" and "Diagnose" are missing values. Except from the Diagnose, this are all numeric values. These missing values should be handled correctly during the pre-processing. By eye-balling the data-sheet, we also noticed that these values are sometimes missing, but sometimes also misplaced and entered with a wrong activity. We will talk further about this in the pre-processing section.

3 Methods and Experimental Setups

3.1 Experimental Setup / Model Selection

We were given the choice between the 4 Regressors, as shown in Table 2, when implementing this task:

Regression Tree	
Pros	Cons
Explainable	Overfitting
Fast	

Random Forest	
Pros	Cons
Less Overfitting	Slow
	Less Explainable

kNN	
Pros	Cons
Explainable	Very slow on large datasets
	Bad in high dimensions

MLP	
Pros	Cons
Scalable to large datasets	Not Explainable
Good in high dimensions	Hyperparameters hard to tune
	Need lots of data to avoid overfitting

Table 2: Comparison of Regressors

For our purposes, we chose the Random Forest model. Compared to the other models, this model performs relatively well out of the box with default hyper-parameters. It captures complex relationships better than a single Regression Tree or kNN, but is also not too complicated that it requires very careful tinkering with the hyper-paramaters, like in MLP. Due to it being multiple regression trees, it'll also be less prone to overfitting. Unfortunately, in the discussion section we found out that Decision Trees are actually not a great fit for this domain.

For splitting the data into test data and train data we used temporal split. This method makes sure that the training data consists of cases from earlier periods, while the test data contains cases from later periods. In this way there is no data leakage, where future information might inadvertently influence the training process. If this happens the model performance would be overestimated. The data is split by a time threshold and is split so the test data is 22% and the training data 78% of the total data. Which is a proven solid distribution [1] [2].

3.2 Pre-processing

As concluded during data exploration, our raw dataset is highly unstructured and incomplete, requiring pre-processing before it can be utilized effectively. This involves dropping redundant features, converting non-numeric values to numeric ones, handling missing data and ensuring that the data is consistent.

Data Cleaning
As discussed before, before we event start making the labels, we first remove the "ER Return" activity, this will improve performance by a lot, as we want to predict the time a patient will stay in the hospital, not how long he will stay at home, then return to the hospital. This will remove most outliers from the data.

We drop the features "Variant", "Variant index", and "lifecycle:transition". The features "Variant" and "Variant index" are removed because they represent the sequence of events for a particular patient, with each sequence assigned a unique Variant index. Since predicting instances based solely on known variants is not useful for our model, as it is only relevant within this dataset—our goal is to also predict previously unseen sequences—these features are excluded during pre-processing. Similarly, the feature "lifecycle:transition" is dropped because it consistently holds the same value, "complete," and thus adds no value to the dataset.

During the data exploration, we concluded that certain values are missing in activities where they are expected to be present. For instance, the activity "ER registration" should include values for "Age," "Diagnosis," and all features with a boolean value, such as "Infusion" and "InfectionSuspected." Similarly, the activity "leucocytes/CRP/lactic acid" should contain their corresponding test scores. However, in our original dataset, this is not always the case. For example, it is possible for a row with the activity "CRP" to lack a value for "CRP." These instances are deemed unreliable for our analysis; therefore, when such a value is missing, we remove the entire case from our dataset.

Besides making sure these activities are not missing data, we also have to ensure that this data is not filled in at a different activity by accident, which during data exploration, we found out was the case. So we make sure that all the cases that have activities with extra data get removed.

We filter outliers from the dataset using a threshold-based approach. If the "remaining time(days)" feature exceeds this threshold, the instance is classified as an outlier and excluded from the preprocessed dataset, we kept this step in, but it becamse less relevant after removing "ER Return".

Data Transformation
The features "Diagnose", "org:group", and "Case ID" are retained but converted into numerical values. Additionally, all boolean values (true/false) are converted to numeric values of 0 and 1. For missing values in "Leucocytes", "CRP", and "LacticAcid", we compute the mean value. The available values for these features are also normalized, using a MinMaxScaler

We are not only dropping features; we are also adding one. During pre-processing, we create a feature called "prev activity", which identifies the activity a patient engaged in prior to the current one. This feature is relevant as it provides a better indication of the patient's progression through the hospital process. By incorporating this feature, our model can make more accurate predictions.

As a big part of our data up until this point are strings, we apply a method called aggregation encoding, so we can use this for regression. Aggregation encoding first uses one-hot-encoding to make every activity a column in our dataset, using 0/1 to determine if it is the activity or not, then we take a cumulative sum over these values a row also contains more information about previous activities.

We also experimented with and without bucketing. With bucketing, we make subsets of our data, looking at specific cases with an activity count that is =< than the bucket length.

3.3 Evaluation Metrics

In regression we have the following error measures to optimize our model:

- Mean Absolute Error(MAE): Gives equal weight to all errors. Useful when large outliers are less important and far-away datapoints aren't as important.
- Mean Squared Error(MSE): Takes the square of the distance, further datapoints' error are counted quadratically more important. Useful when far-away outliers are very important and when far-away errors are important(i.e. in high stake environments).

- Root Mean Squared Error(RMSE): Same as MSE, but the square root is taken, so it's in the same units as our original data. Good for interpretability, but loses some nice mathematical properties.
- R-Squared(R^2): More a metric on how well the model fits, than to calculate error. Usually used to compare models or to compare datasets within the same model. Uses MSE to compare. Score explanation for R^2 are explained in table 3:

x<0	Model performed worse than comparison.
x = 0	Model performed the same as comparison.
0<x<1	Model performed better than comparison.
x=1	Model is perfect.

Table 3: Score Table for R^2

For the purposes of our dataset and model, R^2 is probably most useful, because it does not only measure error, it also compares it to our baseline, which depending on the different encodings we will use, will change along with our error measures. MAE will be more of a general error measure for moderate mistakes and RMSE can show use how well we removed our outliers during the Pre-processing step and if we will be making very larger mistakes in our predictions. In our domain I'd say RMSE is a little more important than MAE, as telling a patient they will be spending 4 weeks in the hospital when they will only spend 2 days is a little more problematic than telling them they will be spending 1 week in the hospital and they will actually spend around 3 days. MSE is important for certain mathematical properties, but we won't be looking at those properties much in our research.

3.4 Implementation Details

Various Python libraries were used for the implementation of the algorithms. We used pandas for pre-processing and managing the data. For plotting graphs we used the matplotlib library. Also we used sklearn quite a lot. We used the MinMaxScaler from sklearn.preprocessing to normalize the data where needed. And the LabelEncoder to encode the data where needed. In the pre-processing we used numpy for some calculations.

To train the models we again used some packages from sklearn to train the different regressors. So for each regressor we imported the right regressor package. And we also imported the needed error packages from sklean.metrics for the measurement on the models.

We trained all the four models with the default parameters with the train data, and evaluated on the test data. We set seeds for the models, so we can see the changes in errors very clearly when experimenting with different pre-processing methods.

4 Results and Discussion

Random Forest Regressor

Encoding	Model	Training MAE	Test MAE	Training MSE	Test MSE	Training R^2	Test R^2	Training RMSE	Test RMSE
Agg-state and no bucketing	RF regressor	2.30	4.82	12.7	54.4	0.65	-0.20	3.56	7.37
Agg-state and prefix length 5	RF regressor	2.14	4.42	10.80	46.20	0.63	-0.20	3.29	6.79
Agg-state and prefix length 10	RF regressor	2.02	4.65	9.41	48.77	0.70	-0.20	3.07	6.98
Agg-state and prefix length 15	RF regressor	1.97	5.93	7.54	78.89	0.84	-0.16	2.75	8.89
Only-activity	RF regressor	3.54	4.83	26.30	54.91	0.28	-0.23	5.13	7.41
Elapsed time	RF regressor	1.49	4.80	4.84	53.41	0.87	-0.19	2.20	7.30

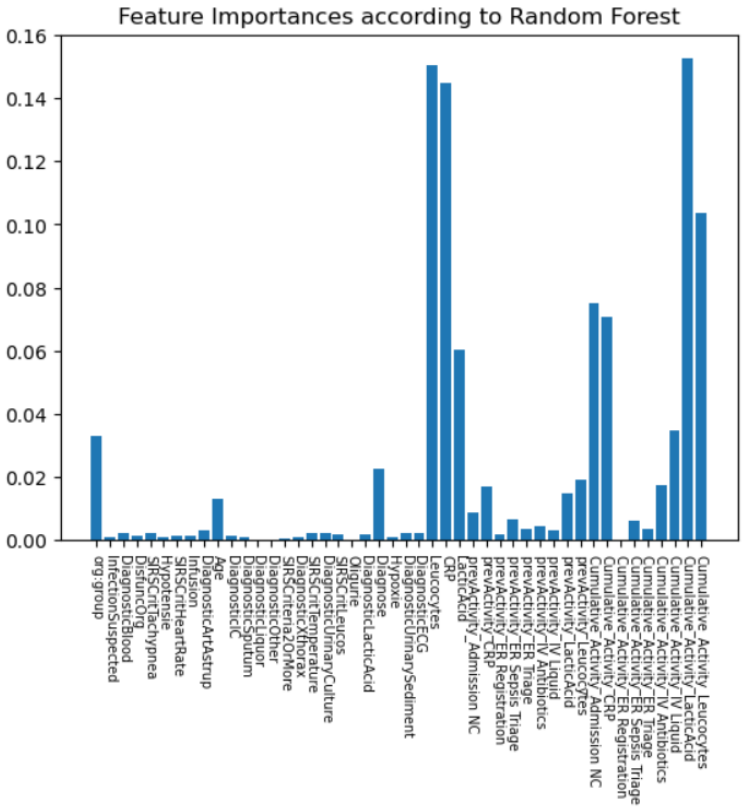
Table 4: Random Forest Regressor Performance

Looking at the results from Random Forest in Table 4, we can see from the test R^2 score that for the Random Forest model the performance is quite awful. It doesn't matter which encoding we used, we always got a negative score, meaning we always scored worse than our baseline, which was to just take the mean, we found out that this is because Decision Trees are not a great model for this dataset, we are working with continuous labels and Decision Trees can make an incredible amount of splits on continuous variables. Because of all these splits, our Training scores all look quite good, but the moments we have to

generealize to our Testing scores, the performance goes down a lot.

Looking at Test MAE and Test RMSE, we can see that RMSE is always a bit higher than our MAE, this is natural because RMSE will measure data-points further away with a higher error. Due to the removal of the "Er Return" activity and the outlier removal at the pre-processing step, we got RMSE closer to our MAE. We also tried to add some bucketing, with prefix length 5, we saw our Test MAE and RMSE go down a bit, but our Test R^2 stayed the same, meaning that even though we got a lower error, the baseline model also expected us to to better in this subset, so there's not much improvement here. In prefix length 10 we see the same thing, but the baseline model expected us to do a little worse here, which we also did and this kept our Test R^2 the same again. When we look at prefix length 15, this changes. Our Test MAE and RMSE seem to be higher than any other prefix length, but our R^2 seems to lower, meaning that the baseline expects us to do relatively bad, but our model actually got relatively closer predictions.

Finally, we added 2 more encodings: "only-activity" and "elapsed time". With the only-activity encoding we decided to remove all the features and only look at the order of activities, it turned out that we now deleted too much relevant information and our performance went slightly down. Adding the elapsed time activity helped the performance of the model, albeit very sligthly, we saw very slight improvements across the board compared to the model without it.



In figure 4 we can clearly see that unsurprisingly the lab results of the Leucocytes, CRP and LactidAcid were very important in predicting a patient's remaining time. Looking at this graph, we can also see that the different kind of IV's, diagnose, age and the org:group played a significant role in predictions, albeit it less important than the lab results. Most other features didn't add much, so for the ER Registration activity, only Age and Diagnose were really important. prevActivity also didn't add much, it only added a bit of importance when the prevActivity was a lab test.

Figure 4: Feature Importances according to Random Forest

MLP Regressor

Encoding	Model	Training MAE	Test MAE	Training MSE	Test MSE	Training R^2	Test R^2	Training RMSE	Test RMSE
Agg-state and no bucketing	MLP regressor	4.17	4.91	31.10	49.07	0.15	-0.10	5.58	7.01
Agg-state and prefix length 5	MLP regressor	3.83	4.17	28.45	39.14	0.022	-0.02	5.33	6.25
Agg-state and prefix length 10	MLP regressor	3.59	4.03	29.97	40.06	0.03	-0.01	5.47	6.40
Agg-state and prefix length 15	MLP regressor	4.77	5.51	40.71	67.26	0.12	0.01	6.38	8.20
Only-activity	MLP regressor	3.85	4.56	30.13	49.27	0.18	-0.10	5.49	7.02
Elapsed time	MLP regressor	4.03	4.61	31.88	48.05	0.13	-0.07	5.65	6.92

Table 5: MLP Regressor Performance

For the bonus assignment, we decided to include another Regressor in our research: MLP. I'm starting off this section with mentioning that MLP's are very complicated and usually require a lot hyper-parameter tuning to get the most optimal performance. Because the focus of this research was not on MLP, we only used default hyper-parameters. Originally we thought that MLP's would not work great out of the box, but we got proven wrong with the results in Table 5.

First of all, we're encountering a bit of a rare phenomenon when comparing these results to Table 4. When looking at the "No bucketing" encoding in both models we can see that for MLP, the MAE goes up(4.91 vs 4.82), the RMSE goes down(7.01 vs 7.4) and the R^2 goes up(-0.095 vs -0.020). This means that the model makes more smaller mistakes, less larger mistakes and overall, performs slightly better, especially considering in our domain, where smaller error are a lot less important than making big error where you tell a patient they're going to spend weeks longer in the hospital than they actually will.

But for bucketing, only-activity and elapsed-time we actually see better values across the board for all errors(MAE, RMSE, MSE, R^2) compared to Random Forest, especially RMSE. Meaning MLP is just plain better at dealing with these complex relationships and (partly) continuous data, even when it's not tuned. But all this is relatively speaking, as we mostly of course still see very bad performance and in all cases but 1(MLP, with prefix length 15), just taking the mean as a prediction is better than using this model.

A lot of what was said about the relationships between MAE, RMSE and R^2 and the different encodings in the Random Forest section are also relevant here.

5 Conclusions

In conclusion, the best model(from the 2 we have tested) for this task will be MLP, due to its ability to capture complex relationships well and deal with continuous variables well, but even MLP struggled to even get a positive R^2 score. For encoding, using our pre-processing steps as described in the pre-processing section, in combination with aggregation encoding and including an elapsed time feature seems to get the best results. For bucketing we saw some mixed results, if the goal is to get the lowest MAE and RMSE, a bucket length of 5 was optimal. If the goal was getting the lowest R^2 then a bucket length of 15 seemed optimal. The most important features were the lab results and lab activites.

For the future direction of Predictive Process Monitoring, we can wholeheartedly recommend to not use Decision Trees for this purpose. We were starting to go in the right direction with MLP, but even MLP's will fall short for this purpose. You could explore further with different Neural Network models, as the domain does seem to be quite complex that we might want to go into the direction of Neural Network models. You could also use some other encoding methods, like Auto-Encoder, to figure out which are important features.

References

[1] Nikolaj Buhl. Training, validation, test split for machine learning datasets, 2023.

[2] A. Aylin Tokuç. Splitting a dataset into train and test sets, 2024.