Project 1: Time-Series

Intensive Care Time Series Modeling for Mortality Predictions on the Physionet 2012 Challenge Dataset

Teaching Assistants: Manuel Burger and Robin Geyer

Abstract

Intensive care patients, usually being severely ill, are closely monitored by state of the art medical devices and sensors to measure vital signs at regular intervals. Additionally, doctors take frequent laboratory tests and any type of treatment administered via infusion, injection, or oxygen supplied through various ventilation equipment, are all meticulously recorded.

These large amounts of data can be overwhelming for intensive care physicians to analyze properly while under time pressure. Machine learning algorithms are prime candidates to leverage these large amounts of data and distill them into actionable outputs for clinical decision support.

The goal of this project is to get familiar with such a promising type of healthcare data, while at the same time learning how to tackle the challenges of working on noisy, multi-variate, irregularly-sampled and sparse, and confounded real world data.

Resources

The core resource for this project is the Physionet 2012 Challenge Dataset. Please download the data, which is publicly available from the Physionet platform (~1.8GB in total):

- Physionet Challenge Overview Page https://physionet.org/content/challenge-2012/1.0.0/ (the download is at the bottom of the page with various options on how to retrieve the data). The page has a detailed description of the data and how it was prepared and used for the Challenge.
- Download the ZIP file directly from Physionet containing all data
- The data is also already available on the cluster under `~/mlh4_data/p1/*`

Important Note!

While we are using the data published and prepared for the Physionet 2012 Challenge, the goal of this project is **not** to follow the exact same steps of the challenge. We are using the same data and we are solving one of the tasks, but the steps and metrics might differ. Please carefully read the project instructions in this document.

Data: The data provided by the challenge splits patients into three sets: A, B, and C. Generally, if not stated otherwise by the specific project instruction step, use set A for

training, set B for validation, and set C exclusively for testing and reporting the final performance of your method after training on A and selecting your model using B.

Deliverables

- Submit the report and code on Moodle: **07.04.25**
- There is no restriction on the number of plots etc.. We encourage you to aim for a compact report e.g. a 4 (at most 5 page and with an Appendix) workshop paper style structure. You can use the NeurIPS paper template available here.
- The report must be handed in as a PDF.
- Please make sure to state all group member names and Legi numbers on the report
- The report has to be self-contained, i.e. no references to code.
- Underlined sections within questions specify how many points can be achieved by solving that specific subquestion.
- You will also need to hand in your code. Please include a requirements.txt or similar for your Python environment and a README.md explaining how to run your code or specify the used existing environment on the student cluster.
- Use the train split for training and the validation split for tuning hyperparameters only. Report results on the test set. Note that the performance of the different methods can vary a lot.
- Using publicly available code is okay, but properly reference repositories when you use them. Of course, you are not allowed to use the code of other teams from the course.
- If not noted otherwise, report performances on the binary classification task using area under the curves (receiver-operator and precision-recall i.e. AuROC and AuPRC)

Outline (Total Points: 51pts)

1. Data Processing and Exploration (5pts)
2. Supervised Training (16pts)
3. Representation Learning (12pts)
4. Foundation Models (12pts)
5. General Questions (6pts)

1 Data Processing and Exploration (5 Pts)

Our goal is to predict whether a patient will be discharged alive from the intensive care unit (ICU) or die inside the ICU after we observe 48 hours of stay. Patients with shorter stays have been removed from the dataset for the challenge. As such we aim to solve a binary classification problem. We measure the performance of our predictor using the area under the receiver operator curve (AuROC, sklearn function `roc_auc_score`) and the area under the precision recall curve (AuPRC, sklearn function `average_precision_score`).

As input data we consider the full set of 37 variables measured irregularly across the 48 hours and additionally we consider the following 4 pieces of static information for each patient: age, height, weight, and gender. **Please do not use *ICUType* as data to train your model as we might make use of it otherwise.**

Q1.1: Data Transformation (1 Pts)

In the downloaded data we are given an archive file for each patient set: `set-{a,b,c}.tar.gz`. Each set contains a `{PatientID}.txt` file where the time series data of a single patient is stored (including the static information at timestamp 00:00). Process such that you obtain a regular time-grid of 48 hours (49 hourly steps including the 00:00 step). Your final output for each patient should be a table with 41 columns for data (37 time-series variables and 4 static variables), one column for the timestamp from 00:00 to 48:00 and one column for the PatientID. This format will allow you to store each of the three sets {a,b,c} in a single .parquet file, which will help you to run queries on top for the following analysis.

Extract the labels for each set from the Outcomes-{a,b,c}.txt files and load the *In-hospital_death* column.

Note: The above format will introduce a lot of missing data, since we do not observe each measurement at every point in time. For now, fill those with Nan/Null values, we'll deal with them later. Careful, when rounding timestamps, should you round up or down to preserve temporal causality? We must not make information from the future available at an earlier time!

Q1.2 Exploratory Data Analysis (2 Pts)

Get familiar with the data by looking at the distributions of the individual variables. This is real world patient data, some variables will show very beautiful normally distributed bell shapes, however, others not. Take note of these shapes and let it guide your future data transformation decisions! Is all the data numerical or are there some codes or categories? State two interesting observations. (1 pt)

Consider also patients by demographic information such as gender, age (you can build bins of e.g. 5 or 10 years), or the *ICUType*. Do you observe differences in the distributions of some of the values across the groups? (1 pt)

Q1.3 Preprocess data for Machine Learning (2 Pts)

The data prepared in step Q1.1 contains large fractions of missing values. Most machine learning methods do not like missing values and prefer to get dense data representation inputs. When working with time series data, we have to be careful to preserve causality in our imputation schemes. We must not use future information to impute our current value estimates.

Use a simple forward filling imputation scheme, which always forwards the last known value to fill a missing value. What can you do when there's no previous value? Feel free to change this step later if you feel it improves performance, but this should give you a baseline way to deal with the problem. (1 pt)

Furthermore, while some machine learning algorithms such as tree-based methods (e.g. Random Forest) can deal with native value distributions such as full integers and some even categorical inputs, other methods need further processing. Linear regression methods, SVMs, and especially any type of neural network need the data to be scaled to a range around 0 to be numerically stable during their optimization procedure. Maybe your insights from Q1.2 can help you choose appropriate scaling methods? Choose, apply, and reason about your scaling method. (1 pt)

Careful: scaling methods and the like should always only be fitted on the training set!

2 Supervised Learning (16 Pts)

Q2.1 Classic Machine Learning Methods (5 Pts)

Classic ML methods, such as Logistic Regression or Random Forests, take a fixed set of features as input. These models learn linear or non-linear combinations of the features to make a prediction (regression or classification). They often have a notion of feature importance but also need careful feature design. Feature design may involve adding hand-designed features that are based on a priori knowledge about the dataset or the application (e.g. signal processing knowledge in case of time series).

1. Choose (at least) two different classic (non-deep) ML classifiers and train them on a very simple set of "features". We suggest for each variable you choose a simple function such as mean, max, last-measured, to obtain a vector of size 41 (37 dynamic, 4 static variables) for each patient. Report test set C performance (AuROC and AuPRC) (1pt)
2. Feature design and engineering is important for classic ML methods, especially when applied to time-series, since many classical methods do not explicitly consider a time dimension. Repeat (1) with additional features of your choice, try to improve the predictive performance of your method, and report again test set performance (3pts). See e.g. https://tsfresh.readthedocs.io/en/latest/ for signal processing-based features, which can help static models such as linear regression gain temporal insights through smart feature extraction. Report test set C performance (AuROC and AuPRC) (3pts).
3. Evaluate the pros and cons of the different classifiers used (1pt), how do your features contribute to the performance? Can some methods take better advantage of the extracted features?

Q2.2 Recurrent Neural Networks (4 Pts)

Recurrent Neural Networks (RNNs) are designed to handle sequential data of variable lengths by applying the same network to every time point, preserving a notion of memory through a hidden state vector. As RNNs suffer from vanishing or exploding gradient issues, Long Short-Term Memory (LSTM) networks use gating mechanisms to control the flow of information in and out of the memory cell[1]. Implement an LSTM network, train it on set A and report test set C performance (2 pts). Neural networks can learn to extract relevant features from the data, as such, you can more or less pass your output from Q1.3 directly to the network. While a recurrent neural network will produce an output at every time-step we can only leverage one to make the final prediction, which one should you use? Maybe you can also aggregate them?

Vanilla implementations of RNNs and LSTM are unidirectional in that they process time series in a sequential manner. For some time-series applications, we must make sure our model can always only access and learn from past data during training and testing. However, here we provide only data from the first 48 hours and predict a target further into

[1] Hochreiter and Schmidhuber, "Long short-term memory", 1997.

the future. Why might a bidirectional model be also useful here[2] (1 pt)? Implement and train a bidirectional model (1 pt). Report test performance and briefly discuss your findings?

Hint: make use of existing model implementations in neural network libraries (keras, pytorch, etc).

Q2.3a: Transformers (3 Pts)

The attention mechanism[3] can help capture longer-range dependencies in the data. Implement, train, and test a simple transformer model (2 pts). What differences and advantages do transformers show over recurrent neural networks (1 pt)?

Q2.3b: Tokenizing Time-Series Data and Transformers (4 Pts)

In Q1.1 we created a data representation, which allows us to feed multi-variate, irregularly-sampled time series data into deep neural network sequence architectures (Q2.2, Q2.3a). However, you might have noticed that we filled a lot of missing values using imputation methods. Based on the work by Horn et al.[4] we might be able to avoid these imputation steps.

Instead of modeling a sequence of time steps, we can model a sequence of measurements! Horn et al. proposed to encode each measurement into three components: time, variable, and value. Reprocess the data to obtain for each patient and each measurement a triplet (t, z, v) where t is a scaled representation of time (e.g. to the range [0,1]), z is a categorical encoding of the variable (you will have 41 categories and you could for example one-hot encode them), and v is the scaled value observed (you can reuse the same scaling methods already fitted in Q1.3 when preparing the time grid data. (2 pts)

Feed those sequences through a Transformer architecture, train, validate, and report test set performance. Can you achieve the same performance or even better as with the imputed time grid? (2 pts)

[2] Schuster and Paliwal, “Bidirectional recurrent neural networks”, 1997.
[3] Vaswani et al., “Attention is all you need”, 2017.
[4] Horn et al., “Set Functions for Time Series”, 2020.

3 Representation Learning (12 Pts)

Not all datasets are labeled as collecting labels, especially for medical data, is expensive. *Representation Learning* tries to overcome the lack of labels by first solving surrogate tasks. The training results in low-dimensional representations that are transferable to similar datasets and useful for different downstream tasks.
There are different approaches to self-supervised representation learning, such as autoencoder-based approaches or contrastive learning[4,5]. You could familiarize yourself with contrastive learning approaches for time series (see e.g. Awesome Self-Supervised Learning for Time Series (SSL4TS)) and, especially, the InfoNCE approach[6]. (Remark: you are free to use any other representation learning approach too.)

Q3.1 Pretraining and Linear Probes (4 pts)

1. Pretrain an encoder model of your choice (2 pts), including your choice of data representation (time grid from Q1.3 or tokenized measurements from Q2.3b). How do you monitor this pre-training step? Note that the architecture of your encoder should be as similar as possible to that trained in a prior question, for a fair comparison in the next stage.
2. Freeze/fix the weights of your pretrained network and compute a single embedding vector for each patient. Train a logistic regression (i.e. a linear probe) on the training set to predict the target only from your pretrained embeddings. Compare your results to the supervised performances obtained in prior tasks. (2 pts)

Q3.2 Simulate label scarcity (4 pts)

Keep your pretrained network from Q3.1 and the computed embedding vector for each patient.

- Train three different **supervised** (as in Q2.x) models with the same (or as similar as possible) architecture as your pretrained network, but only use 100, 500, and 1000 patients from the training set and report your full test set performance (2 pts).
- Train three linear probes (as in Q3.1 step 2) using only 100, 500, 1000 labelled patients and report the full test set C performance. (2 pts).

Do you observe an advantage of your pretrained network under label scarcity? How long is it useful to invest in pretraining versus directly training the supervised model?

Q3.3: Visualising Learned Representations (4 Pts)

In the prior questions you have pretrained an encoder and obtained an embedding vector for each patient.

[4] Bengio, Courville, and Vincent, “Representation Learning: A Review and New Perspectives.”
[5] Ericsson et al., “Self-Supervised Representation Learning.”
[6] Oord, Li, and Vinyals, “Representation Learning with Contrastive Predictive Coding.”

Visualize your learned representations through a dimensionality reduction technique such as t-SNE[7] or UMAP[8] (2 Pt). Are data points with different labels distributed identically? (1 pt) Use a quantitative clustering metric to assess the quality of your dimensionality reduction w.r.t. target class labels (1 pt).

4 Foundation Models (12 Pts)

Large Language Models (LLMs) are unquestionably a magical piece of machine learning development. However, can they be used for time-series forecasting? And can they be used for the challenging multi-variate, irregularly-sampled, and noisy type of data coming from intensive care units?

We would like to investigate whether these massive pretraining efforts can be leveraged for our purposes!

Q4.1 Prompting an LLM to solve a time-series problem (4 Pts)

In this step the goal is to perform some form of feature engineering to transform the data observed in the first 48 hours of stay into a single piece of text and then prompting the language model to classify whether the patient could die or get discharged alive. The goal is to have the LLM give you the predicted answer in its text output space!

- Transform each patient's time series data to a simple aggregated piece of text. To save valuable text and token space and run models faster, try to come up with a compact summary. You could leverage your feature engineering work from Q2.1 and instead of passing all values, pass something like *"max. heart rate 180 bpm"*. (2 pts)
- Choose an LLM, you can use OpenAIs API (but this will cost) or we suggest you run a small LLM locally or on the cluster (e.g. choose a model in the 1B, 2B, maybe at most 7B parameter range) using https://ollama.com and its matching python package: https://github.com/ollama/ollama-python. You can choose to either just feed the text description and prompt for a prediction or you might find that performing few-shot predictions where you previously populate the context with a few examples of descriptions and predictions from the training set, might give you much better answers. Be mindful with your context space to keep the model running fast! Report the test set performance of your predictions. Can you still use AuROC/AuPRC if getting binary responses? Maybe you can ask the model for a score in a range of 1-10? (2 pts)

Q4.2 Using LLMs to retrieve embeddings (3 Pts)

Use again the ollama-python package to not retrieve a text response directly from your model, but retrieve a single vector embedding for each piece of text description you have

[7] Van der Maaten and Hinton. "Visualizing data using t-SNE", 2008.
[8] McInnes et al., "Umap: Uniform manifold approximation and projection for dimension reduction", 2018.

created for each patient in Q4.1. (1 pt) This should give you a dataset of embeddings for each patient very similar to the output of your own representation learning model in section 3 of the project.

Train a linear probe on the full training set of LLM embeddings and report test set C performance. How do the results compare to your supervised in section 2 and your own representation learning model, which is domain specific?, trained in section 3? (1 pt)

Finally, also visualize your LLM embeddings same as in Q3.3 and compare the clustering of the LLM embeddings with the domain specific pretrained embeddings. (1 pt).

Q4.3 Using time-series foundation models (5 Pts)

While LLMs are an exciting development, they have not been built specifically to work with time-series data. However, naturally researchers are also scaling time-series specific models, such as the Chronos[9] family of models developed by researchers at Amazon. We suggest that you consider one of the Chronos models for the following tasks:

- Compute a single embedding for each patient using a pretrained time-series foundation model. Note that the Chronos family of models has been trained for univariate (single variable) time-series data, but we have multiple variables. Think about a simple aggregation scheme to end up with a single embedding vector for each patient. You could take the average embedding across all channels. (2 pts)
- Train a linear probe on the embeddings and report test set performance. (2 pts)
- Given that you might end up with individual embedding vectors for each variable, can you think of a very simple neural network architecture, which can learn to aggregate the different channels embedded by the pretrained model in a smarter way than just a non-parametric pooling (such as averaging or summing)? Propose and train such a light-weight architecture, which can learn to aggregate channels and report test set performance. (1 pt).

[9] https://github.com/amazon-science/chronos-forecasting

5 General Questions (6 Pts)

To conclude, we ask you to answer the following questions to recap and reason about the project.

Q5.1: There are many machine learning settings where classic methods are still competitive with deep learning architectures. Have you observed this in this project? Why is this (not) the case? (2 pts)

Q5.2: Can you think of an attention-related bottleneck regarding very (very) long time series? Conceptually, which deep methods from above are more suitable for such long time series? (2 pt)

Q5.3: What are some challenges in using self-supervised representation learning? What difficulties have you observed in your approach? Can you think of additional ones? (2 pt)