Univ.-Prof. Dr.-Ing. Matthias Boehm
Technische Universität Berlin
Faculty IV - Electrical Engineering and Computer Science
Berlin Institute for the Foundations of Learning and Data (BIFOLD)
Big Data Engineering (DAMS Lab) Group

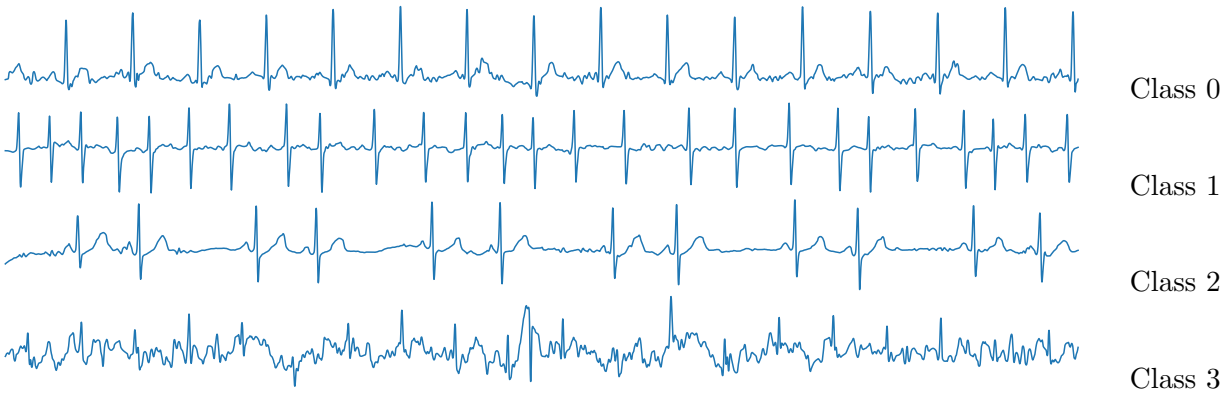
1 AMLS SoSe 2025: Exercise – ECG Time Series Classification

Published: Apr 12, 2025 (last update: Apr 22)
Deadline: Jul 15, 2025; 11.59pm

This exercise is an alternative to the AMLS programming projects and aims to provide practical experience in the exploratory development of machine learning (ML) pipelines. The task is to create a classifier for univariate echocardiogram (ECG) time series data. You may choose any programming language(s) and utilize existing open-source ML systems and libraries. The expected result is a zip archive named `AMLS_Exercise_<student_ID>.zip` (replace `<student_ID>` by your student ID) of max 20 MB, containing:

- The source code used to solve the individual sub-tasks (in a sub-folder each).
- A PDF report of up to 8 pages (10pt), including the names of all team members, a summary of how to run your code, and an explanation of the solutions to the individual sub-tasks.
- Three CSV files with your test predictions, located in the root of your .zip archive, named **base.csv**, **augment.csv**, and **reduced.csv**. These files should use the same format as `y_train.csv`.

Data: This exercise uses ECG time series. Training and test data can be downloaded from the TU-Cloud. We only provide training labels, and in the different subtasks you should use the test data to materialize your predictions (which will be included as part of your submission). Each ECG signal is classified into four classes: (0) normal, (1) AF (tachyarrhythmia, which is uncoordinated atrial activation), (2) other (rhythms that do not fall into either normal or AF), and (3) noisy (too noisy to classify) heart rhythms. The signals are sampled at 300 Hz. The training data is given in a binary format of a 32-bit integer for the length of the time series, followed by the values as 16-bit signed integers (see the data parser for help).



Grading: This exercise is conducted in teams of 1 to 3 persons (one submission). The grading is a *pass/fail* for the entire team. Exercises with $\geq 50/100$ points are a pass, and the quality expectations increase with the team size. Exercises with ≥ 90 points receive **5** extra points in the exam.

```
def forward(self, x, lengths):
  x, _ = pad_packed_sequence(x, batch_first=True)
  x, lengths = self.stft(x, lengths)
  x = log2(x.unsqueeze(1))
  x, lengths = self.conv1(x, lengths)
  x, lengths = self.conv2(x, lengths)
  x = x.view(x.size(0), -1, x.size(3))
  x = x.permute(0, 2, 1)
  x = pack_padded_sequence(x, lengths,
    batch_first=True, enforce_sorted=False)
  _, (ht, ct) = self.rnn(x)
  x = self.fc(ht[-1])
  return x
```

Figure 1: A baseline you can include or start from is a model using a short-time Fourier transform on the univariate input, followed by two 2D convolutions with pooling and ReLU activation, then a recurrent neural network layer, of which you feed the last hidden state of the time series into a linear layer, to produce an output prediction. Note that we intentionally do not provide the model size because the input time series are of variable length.

1.1 Dataset Exploration (15/100 points)

As a starting point, analyze the provided data. Summarize the class distribution, the lengths of time series, and basic descriptive statistics. In particular, report any characteristics that could be used to deduce which class a time series belongs to. After analyzing the data characteristics, construct a validation split from your training data that reflects the characteristics of the full training data. In the report, justify your selection.

Expected Results: Summarize the data characteristics, visualize ECG time series of the individual classes, and explain the differences between classes by utilizing the collected statistics. Finally, provide the code to select a validation subset of the training data.

1.2 Modeling and Tuning (40/100 points)

Construct an ML pipeline—using the prepared training and validation sets—for classifying the ECG time series into the four classes. Figure 1 shows an optional model architecture. Choose an appropriate loss function and evaluation metric and evaluate this metric on both the train and validation data. Train models for at least two different model architectures, and tune the hyper-parameters of your model (e.g., regularization parameters, channels, learning rate, optimizers, early stopping). Finally, report the evaluation metric on the test data.

Expected Results: Runnable code for training and inference, as well as descriptions of the used model architectures, ML pipelines, and their evaluation. The report must include statistics or plots highlighting the quality of the trained models. You should further justify the selected architectures and solutions, as well as reason why they are applicable to ECG data. Use your final model to produce the **base.csv** predictions of the test dataset for your submission.

1.3 Data Augmentation and Feature Engineering (30/100 points)

Improve your model quality via data augmentation techniques and feature engineering. Suggestions for augmentations include time stretching/compression, time shifting, adding noise, random cropping, resampling, amplitude scaling, and frequency domain augmentations via inverse Fourier transformations. You can use specialized libraries of your choice to extract additional features or perform data

augmentation. Suggested libraries include: BioSPPy, HeartPy, WFDB and pyECG. Depending on the level of engineered features, it might be beneficial for you to change the model into an ensemble of various techniques (e.g., random forest, SVM, and NN).

All selected techniques must be implemented as a part of your ML pipeline to operate as preprocessing steps on the entire dataset, or on individual mini-batch elements during training.

Expected Results: Code for applying the data augmentation techniques and a summary of their impact on model quality. For full points, the report must include justifications for the selected augmentation techniques and their placement in your ML pipeline (on batches or the entire dataset). Use the model trained on augmented data to produce the **augment.csv** predictions of the test dataset.

1.4 Data Reduction (15/100 points)

Finally, please try to reduce the input data. The provided training data in the zip archive is 62MB = 64,131,756 Bytes, while the uncompressed binary file is 116 MB = 120,641,086 Bytes. Using the same pipeline from your augmentation pipeline, construct a new parser that directly reads your own input data format. This task can take many directions, and you are encouraged to analyze the compound effect of multiple techniques:

- **Data Sampling:** One direction is to select a representative subset of the original time series. A naïve approach (and thus, only baseline) is random sampling. More advanced techniques include finding a coreset (i.e., a subset of the original data achieving high accuracy) [1, 2] and generating a smaller synthetic training dataset that produces similar or better accuracy [3].

- **Lossy Compression:** Another strategy is to approximate the time series via lossy compression. Examples include Piecewise Constant Approximation (PCA), Piecewise Linear Approximation (PLA), Quantization, Line Simplification, and Fourier Transform with thresholding. If you use any lossy approximation of data, include the means absolute error (MAE) and the mean square error (MSE) of the original and the lossy approximation.

- **Lossless Compression:** While Zip does provide good compression ratios, other lossless compression techniques could produce better compression and runtime tradeoffs. Furthermore, the provided binary files use 16-bit for each value. However, the ECG signal rarely uses the entire value range of 16-bit. Therefore, an option for better lossless serialized data is to treat the outliers as exceptions and store the normal data using fewer bits (e.g., PDICT, PDELTA, PFOR).

- **Embeddings:** Alternatively, one could reduce the data size by creating fixed-length embeddings of the individual ECG time series.

Note that the code of your solution is not allowed to exceed 1MB (including external libraries). Zip in Ubuntu is 199KB = 203,768 Bytes, making it valid together with the data parser. We do not include the programming language, compiler, or runtime memory/disk usage in this limit.

Expected Results: Code to reduce the training data, and code to read the custom training data you constructed. The report should contain results from reducing the dataset to (at least) 10%, 25%, and 50% of the original dataset size (the zipped version). Additionally, make a line plot that shows your metrics as we vary the datasize, including the training results of the original and augmented model on 100% of the data. For full points, your solution must beat (or at least be equal to) a stratified random sample on all selected size ratios. Use the 25% reduced dataset to produce the **reduced.csv** predictions of the test dataset for your submission.

Data parser

The following code is an example of reading binary training data in Python. There are two examples: one reads the unzipped binary, and one directly reads the zipped binary fused with decompression.

```
import struct
import zipfile
def read_zip_binary(path):
  ragged_array = []
  with zipfile.ZipFile(path, 'r') as zf:
    inner_path = path.split("/")[-1].split(".")[0]
    with zf.open(f'{inner_path}.bin', 'r') as r:
      read_binary_from(ragged_array, r)
  return ragged_array

def read_binary(path):
  ragged_array = []
  with open(path, "rb")as r:
    read_binary_from(ragged_array, r)
  return ragged_array

def read_binary_from(ragged_array, r):
  while(True):
    size_bytes = r.read(4)
    if not size_bytes:
      break
    sub_array_size = struct.unpack('i', size_bytes)[0]
    sub_array = list(struct.unpack(f'{sub_array_size}h', r.read(
   sub_array_size * 2)))
    ragged_array.append(sub_array)
```

References

[1] Chengcheng Guo, Bo Zhao, and Yanbing Bai. 2022. DeepCore: A Comprehensive Library for Coreset Selection in Deep Learning. arXiv:2204.08499 [cs.LG] https://arxiv.org/abs/2204.08499

[2] Yeachan Kim and Bonggun Shin. 2022. In Defense of Core-set: A Density-aware Core-set Selection for Active Learning. arXiv:2206.04838 [cs.LG] https://arxiv.org/abs/2206.04838

[3] Shiye Lei and Dacheng Tao. 2024. A Comprehensive Survey of Dataset Distillation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46, 1 (Jan. 2024), 17–32. https://doi.org/10.1109/tpami.2023.3322540