AMLS Exercise Report

Mohammed Zain Maqsood (458387)
Ahmed Helal Taha Elsetiha (464792)

July 2025

1 How to run the code

The complete code is well structured and contained in one Jupyter Notebook `main.ipynb`. Navigating in the notebook is quite easy. To execute the project, one needs to run the first section (Task 1.1) in order to initialize all relevant variables. After that, the other sections can be executed.

2 ECG Time-Series Classification

2.1 Dataset Exploration

2.1.1 Visualizing Signals

In the first step, we analyzed the provided training dataset. To get an idea of how the samples looked like, we visualized two from each class. Figure 1 shows the corresponding plot.

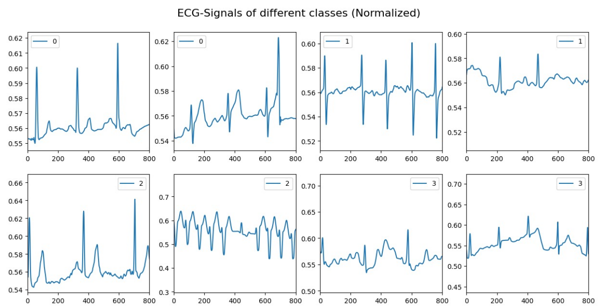


Figure 1: Two ECG-Signals from each class

2.1.2 Class Distribution

The complete dataset consists of 6179 samples. Investigating the class distribution, we found out that it is strongly imbalanced. Class 0 (normal) with ca.

59% and Class 2 (Noisy) with 28.9% are the largest. Quite smaller is Class 1 (AF) with ca. 9% and Class 0 with negligible 3.7%.

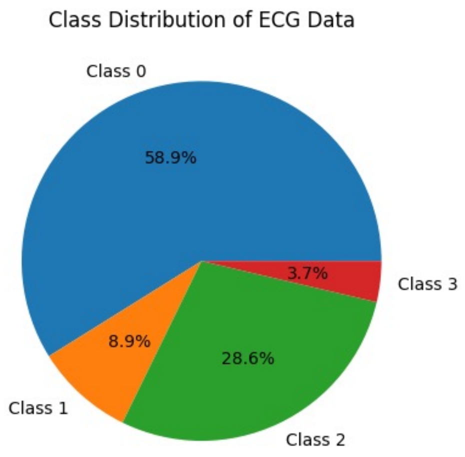


Figure 2: Class distribution of the dataset

In order to get a model that doesn't learn the majority class because of the significant class imbalance and is able to generalize well, we have to consider some common techniques in the next task to explicitly address this problem.

2.1.3 Signal Lengths

As common for Time-Series data, the lengths of the signals strongly vary, ranging from 2714 to 18286.

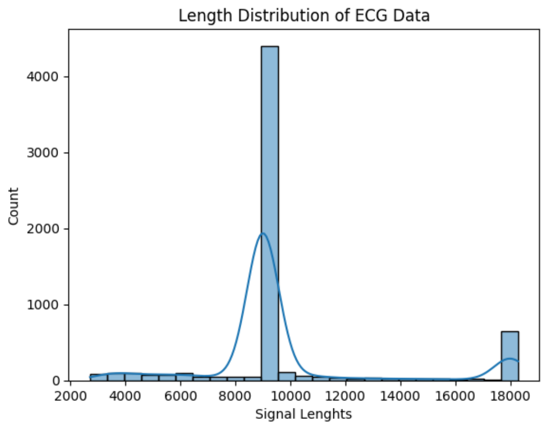
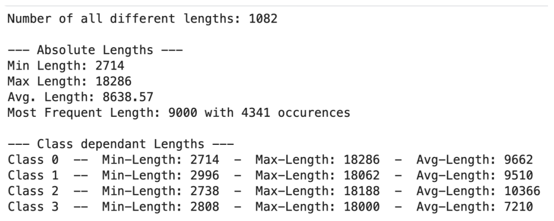


Figure 3: Length Distribution of Training Data

About 70% (4341 samples) have length 9000. Deciding based on the lengths to which class a signal belongs is in general not possible. However, Class 3 tends to have samples that are shorter on average than the rest. This inhomogeneity in length has also to be addressed, especially when working with Mini-Batches and Dataloaders.



```
Number of all different lengths: 1082

--- Absolute Lengths ---
Min Length: 2714
Max Length: 18286
Avg. Length: 8638.57
Most Frequent Length: 9000 with 4341 occurences

--- Class dependant Lengths ---
Class 0  --  Min-Length: 2714  -  Max-Length: 18286  -  Avg-Length: 9662
Class 1  --  Min-Length: 2996  -  Max-Length: 18062  -  Avg-Length: 9510
Class 2  --  Min-Length: 2738  -  Max-Length: 18188  -  Avg-Length: 10366
Class 3  --  Min-Length: 2808  -  Max-Length: 18000  -  Avg-Length: 7210
```

Figure 4: Lengths-Distribution per class

There are several techniques to tackle this problem, e.g. zero-padding or interpolation to a fixed length, depending on the actual model architecture, e.g. RNN's can handle data with inhomogeneous length, but more on that in the next task.

2.1.4 Descriptive Statistics

Figure 5 shows a box-plot diagram for the averaged descriptive statistics over each class. The medians of all classes are relatively close to each other, where class 0 has the lowest and class 3 the highest. The values of the first three classes are similarly distributed, with greater variance for class 3 due to it being the noisy class. The same applies to the min and max values for the first three classes, whereas class 3 has a greater value range.

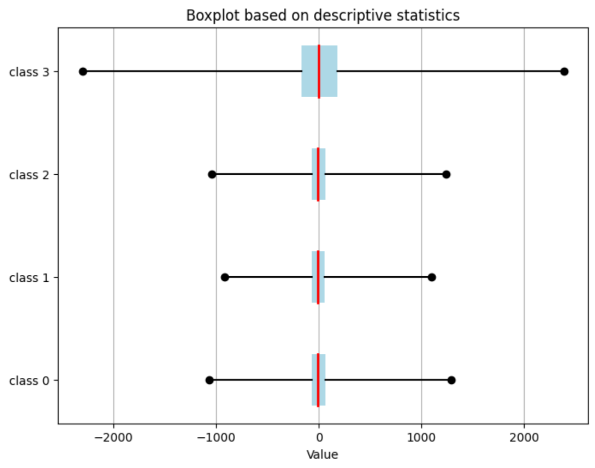
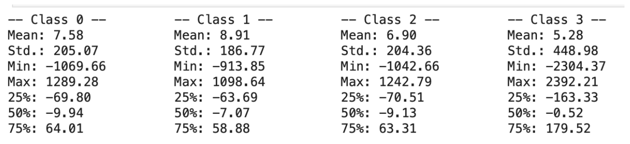


Figure 5: Descriptive statistics of the full dataset per class

Taking a closer look, it appears that class 0 & 2 have almost identical statistics, which can be a problem, because the model can have a hard time discriminating between both classes. And since class 0 is the majority class, the risk of misclassifying class 2 samples with class 0 is high, leading to a high False-Negative Rate for class 2.



```
-- Class 0 --          -- Class 1 --          -- Class 2 --          -- Class 3 --
Mean: 7.58             Mean: 8.91             Mean: 6.90             Mean: 5.28
Std.: 205.07           Std.: 186.77           Std.: 204.36           Std.: 448.98
Min: −1069.66          Min: −913.85           Min: −1042.66          Min: −2304.37
Max: 1289.28           Max: 1098.64           Max: 1242.79           Max: 2392.21
25%: −69.80            25%: −63.69            25%: −70.51            25%: −163.33
50%: −9.94             50%: −7.07             50%: −9.13             50%: −0.52
75%: 64.01             75%: 58.88             75%: 63.31             75%: 179.52
```

Figure 6: Detailed statistics of the dataset per class

The analyzed statistics substantiate the averaged signals per class as shown in figure 7. The avg. signals for Class 0 & 2 are showing a similar structure, whereas class 1 and especially class 3 deviate stronger.

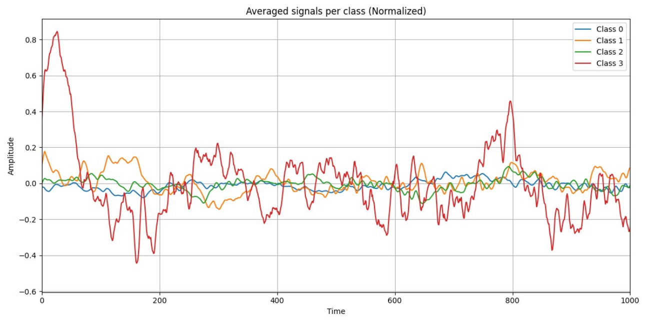


Figure 7: Signals averaged per class

2.1.5 Training- and Validation-Set

As the dataset is highly imbalanced, a random split into train and validation data would be undesirable, because it might happen that the train split gets a minor or no share of one class. This could lead to bad performance during validation/inference, because predicting labels for this class is hard, since the model could not sufficiently learn patterns from this class. Therefore, one could use *stratification* to guarantee that the train and validation split has the same class distribution as the original dataset. In addition to that, we want the validation split to reflect the characteristics of the full training data in order to make validation as effective as possible. For that, we analyzed the descriptive statistics of the validation split (see figure 8) and it shows similar characteristics as the original dataset. Thus, we think that this approach is sufficient and the validation split is representative of the whole dataset. We decided to make an 80/20 split.

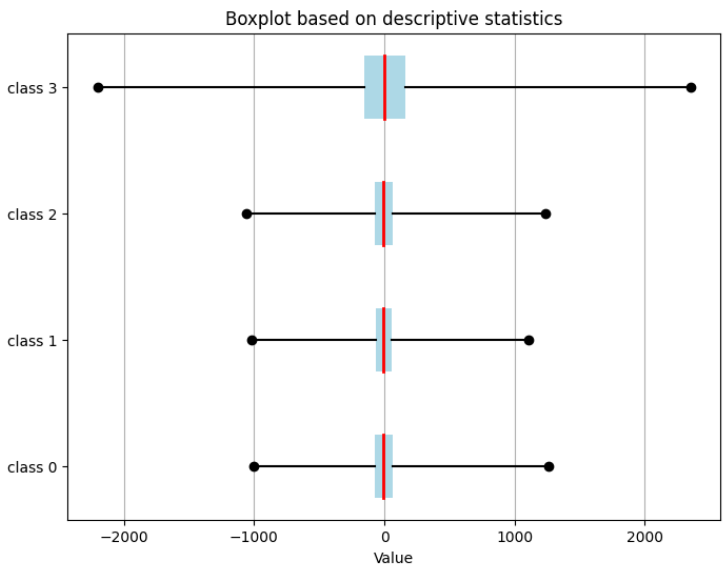


Figure 8: Descriptive statistics of the validation split per class

2.2 Modeling and Tuning

We decided to select a Neural Network as our main model, and a Random Forest as a classical approach for comparison.

2.2.1 Neural Network

For feature extraction, we used Short-Time Fourier Transform (STFT), because it dismantles each ECG-Signal into short sections of time and applies to each section the Fourier Transform, which leads to a 2D-Representation of each signal. Since convolutional layers are great at capturing local patterns

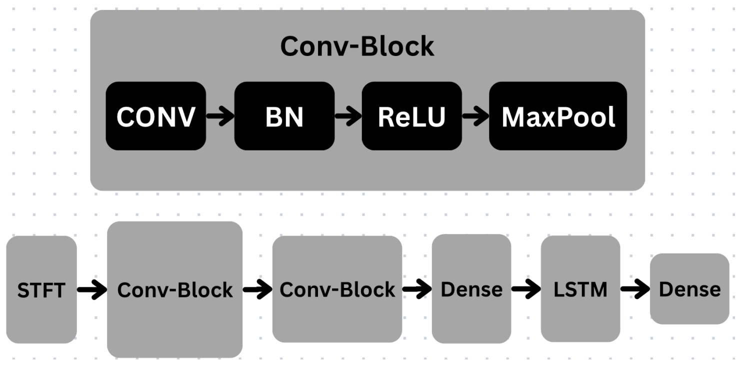


Figure 9: Model Architecture of our model

from 2D-Data e.g. images, and ECG-Signals have a lot of them e.g. QRS-Complex, we thought it is well suited for this scenario. batch normalization is added for stable training and maxpooling for reducing the signal dimensions and filtering important features. The other main component of our model is a Long Short-Term Memory (LSTM) that is capable of learning long-term dependencies over time.

Signal Resampling Since we have inhomogeneous signal lengths, we decided to resample the signals to a fixed length of 9000, but only those that were shorter than that. Signals which were too long were simply sliced at a random start position to length 9000. We chose 9000 as the fixed length because ca. 70% of the signals do have this length as explored in the previous task. We intentionally preferred this method over techniques like zero-padding because we wanted to avoid rigorously recalculating the length of each batch after each applied layer in the forward pass, which can easily lead to miscalculations and therefore bad performance of the model.

Loss-Function As our loss function, we'll use cross-entropy because it has shown great success for classification problems and it penalizes false positives with high certainty quite hard, which ensures effective learning of minority classes. Since we have imbalanced training data as mentioned earlier and we don't want the model to predominantly predict the majority class, we decided to add the following weights for each class to the loss function to mitigate this issue:

$$w_{\text{c}} = \frac{n_{\text{samples}}}{n_{\text{classes}} \cdot n_{\text{c}}}$$

- n_{samples}: Number of all samples
- n_{classes}: Number of classes
- n_{c}: Number of samples in class c

Evaluation Metric Due to imbalanced data, it is misleading to use metrics like accuracy for evaluation because it ignores the imbalance. Therefore, other metrics like precision, recall, F1-Score, or AUC-ROC are more suitable for our scenario. We decided to choose the F1-Score as our main metric for model evaluation, which is a harmonic mean between precision and recall. Moreover, we'll use the average F1-Score, which measures the balance between precision and recall across all classes.

Hyperparameters The following hyperparameters retrieved the best results: `epochs=35`, `batch_size=128`, `lr=0.0015`, `step_size=15` and `gamma=0.05` (the learning rate is halved every 15th epoch), `n_fft=256` (STFT), `hop_length=64` (STFT), `channels=[32, 64]` and `kernel_size=6` for the respective conv. layers and finally `input_size=128`, `hidden_size=192` and `num_layers=1` (LSTM).

Results Figure 10 shows the training and validation loss development during model training. In the first half of the training, the validation loss strongly fluctuates, but after that, it is able to stabilize. Finally, both losses decrease over time, meaning the model was able to learn some relevant patterns from the training data without over- or underfitting the data.

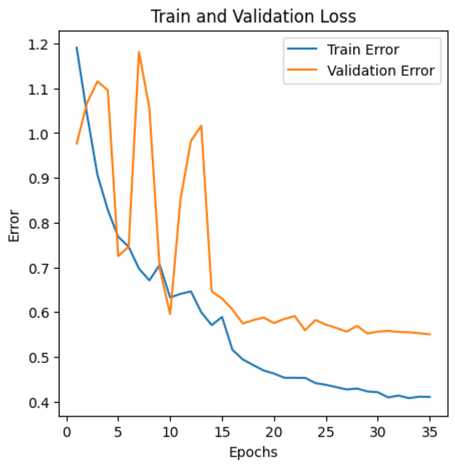


Figure 10: Training and validation loss during model training

Looking at the overall performance on the training set, our model was able to reach solid F1-Scores across all classes. In particular, almost all class 0 samples were predicted correctly with high precision, but with major confusion to class 2. At class 1, almost every class sample was found, but many class 0 and class 2 samples were mistakenly confused with class 1. Class 2 shows the lowest performance among all classes, with many misclassifications to class 0, most likely due to the similar structural appearance of both classes, as found out in the previous task.

```
              precision   recall  f1-score   support
0                  0.92     0.89      0.90   2910.00
1                  0.71     0.94      0.81    439.00
2                  0.81     0.74      0.77   1412.00
3                  0.66     1.00      0.80    182.00
accuracy           0.85     0.85      0.85      0.85
macro avg          0.78     0.89      0.82   4943.00
weighted avg       0.86     0.85      0.85   4943.00
```

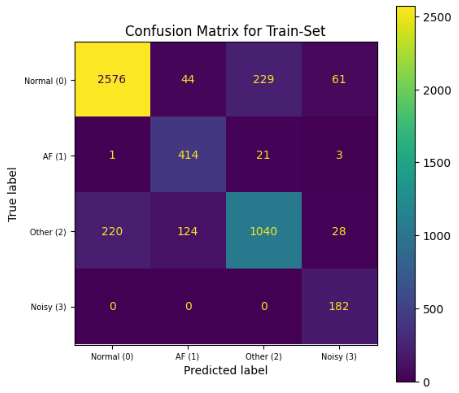


Figure 11: Classification report and confusion matrix for training set

Finally, all Noisy (class 3) samples were found by our model, but with moderate precision, presumably because of specific samples from the other classes with some greater noise.

As assumed, the model shows weaker performance on the validation set. F1-Score for class 0 & 1 is to a small extent lower than on the training set, with significant decline in performance on class 2 & 3. In particular, the model is unsure about many samples from class 2 whether they belong to it or not. The avg. F1-Score with 0.74 seems solid overall, and the model has learned something, but since the F1-scores of class 2 & 3 are lower, it shows that our model tends to prefer class 0, which is most likely because class 0 is the majority class with a greater homogeneous structure across the class.

```
              precision   recall  f1-score   support
0                  0.88     0.83      0.85    728.00
1                  0.72     0.85      0.78    110.00
2                  0.70     0.67      0.68    353.00
3                  0.51     0.84      0.64     45.00
accuracy           0.79     0.79      0.79      0.79
macro avg          0.70     0.80      0.74   1236.00
weighted avg       0.80     0.79      0.79   1236.00
```

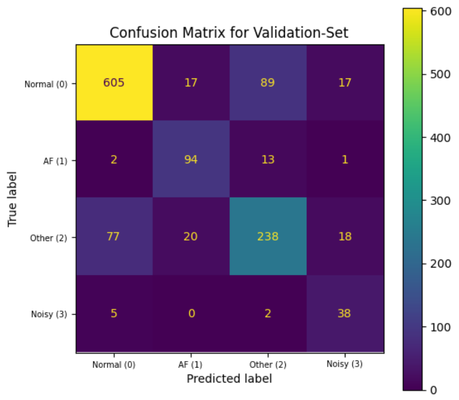


Figure 12: Classification report and confusion matrix for validation set

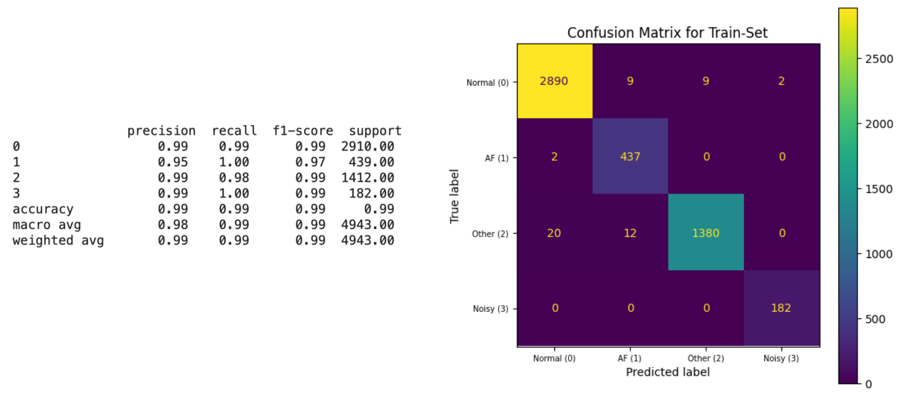
We used this model to generate the **base.csv**, because it shows an overall better performance than the Random Forest, as we will see in the next subsection. Furthermore, we will use this model as a baseline for the next two tasks.

2.2.2 Random Forest

As the second model, we chose a Random Forest, which is an ensemble technique. It combines several smaller decision trees into one large tree, making it mostly robust against overfitting, and it is known to be a good method for small to medium size datasets. Furthermore, we chose this as our second model because we wanted to investigate how well a simpler and more classical machine learning algorithm works with the highly imbalanced ECG data.

Our pipeline consists of feature extraction, model training, and validation. For the features, we resampled the signal to a length of 9000, applied STFT, and in order to get vectors for every sample, we averaged across the frequency axis, getting the averaged amplitude for all frequencies at all time bins.
Hyperparameters which lead to the best results were: `n_features=100` (100 trees), `max_depth=12` (max. depth of a tree) and the same class weights as for the neural network.

Results With a F1-Score of 0.99, the model learns every class almost perfectly, which immediately creates the impression that instead of learning relevant patterns from the training data, the model rather memorized it, leading to overfitting. This assumption arises because we saw that the neural network had a confusion problem between class 0 & class 2, which is also underpinned by almost identical statistics. That's why we now take a look at the evaluation with the validation set in order to see whether the model generalized on unseen data.



```
              precision  recall  f1-score  support
0                  0.99    0.99      0.99  2910.00
1                  0.95    1.00      0.97   439.00
2                  0.99    0.98      0.99  1412.00
3                  0.99    1.00      0.99   182.00
accuracy           0.99    0.99      0.99     0.99
macro avg          0.98    0.99      0.99  4943.00
weighted avg       0.99    0.99      0.99  4943.00
```

Figure 13: Classification report and confusion matrix for training set

Looking at the classification report confirms the supposition of overfitting. The model's avg. F1-Score is 0.51 and has very poor performance on

```
              precision   recall  f1-score   support
0                  0.66     0.92      0.77    728.00
1                  0.53     0.28      0.37    110.00
2                  0.59     0.23      0.33    353.00
3                  0.69     0.49      0.57     45.00
accuracy           0.65     0.65      0.65      0.65
macro avg          0.62     0.48      0.51   1236.00
weighted avg       0.63     0.65      0.60   1236.00
```

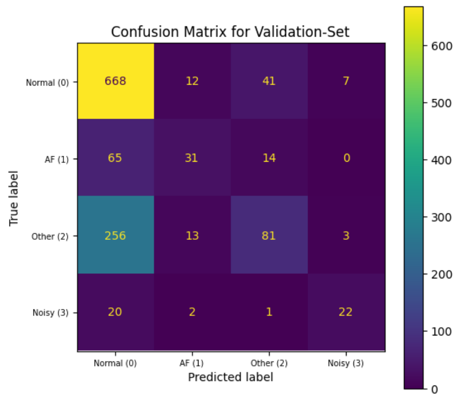


Figure 14: Classification report and confusion matrix for validation set

class 1 & 2 with F1-Scores of 0.37 and 0.33 respectively. Thus, the model didn't generalize on unseen data. Therefore, our neural network approach seems to be the better method for our specific problem setting and data, because it is able to discriminate the classes much better.

2.3 Data Augmentations and Feature Engineering

Techniques like data augmentation can be used to enlarge datasets by applying transformations to existing data in order to generate new samples. Especially when dealing with imbalanced data or models having a hard time distinguishing between specific classes, one can use augmentation to mitigate those problems.

Since the model has a confusion between class 0 & 2, we decided to augment approx. 20% of each of these classes. For that, we added moderately gaussian noise and amplitude scaling. Since both classes have almost identical statistics, we think this approach can mitigate the confusion during training, because it adds variance to these classes and increases the data size, so the model can learn from more data.
We added the augmentation step in the constructor of the Dataset class (in `dataset.py`), where all the preprocessing happens, because we wanted to avoid doing that step on batches to ensure faster training.

Results Figure 15 shows a similar development of both losses compared to the original model, with validation loss stabilizing after around 18 epochs. At the end of the model training, the validation loss reaches ca. 0.53, whereas the original model reached ca. 0.55, leading to a slight performance improvement on unseen data.

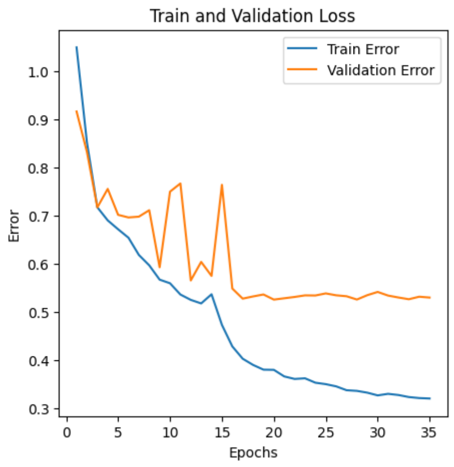


Figure 15: Training and validation loss during model training

The model was able to improve throughout all classes on the training set. Especially on class 0 & 2 the F1-Scores increased from 0.9 to 0.93 and 0.77 to 0.81 respectively, meaning the model was able to distinguish better between both classes. The avg. F1-Score increased from 0.82 to 0.85.

```
              precision    recall  f1-score   support
0                  0.95      0.91      0.93   4379.00
1                  0.72      0.94      0.81    439.00
2                  0.86      0.84      0.85   2103.00
3                  0.68      1.00      0.81    182.00
accuracy           0.89      0.89      0.89      0.89
macro avg          0.80      0.92      0.85   7103.00
weighted avg       0.90      0.89      0.89   7103.00
```

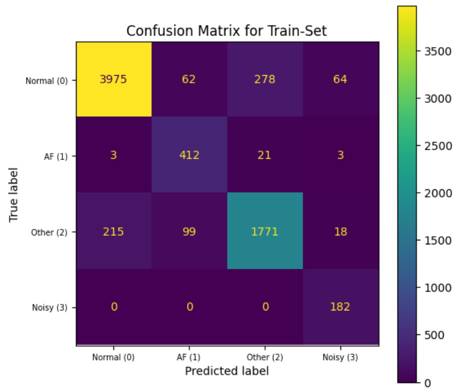


Figure 16: Classification report and confusion matrix for training set

Also on the validation set, the model showed better performance, leading to 0.85 (original) to 0.86 on class 0 and 0.68 (original) to 0.72 on class 2 and avg. F1-Score improved from 0.74 (original) to 0.76.
If we take a closer look at the confusion matrix, it is noticeable that this improvement primarily stems from the fact that class 2 samples were slightly more often correctly classified. The confusion caused by class 0 & 2 still exists, leading to the supposition that both classes hold too similar properties, such that the model could distinguish them like class 1 & 3.

```
              precision    recall  f1-score   support
0                  0.89      0.83      0.86     728.0
1                  0.71      0.85      0.78     110.0
2                  0.71      0.72      0.72     353.0
3                  0.57      0.82      0.67      45.0
accuracy           0.80      0.80      0.80       0.8
macro avg          0.72      0.81      0.76    1236.0
weighted avg       0.81      0.80      0.80    1236.0
```

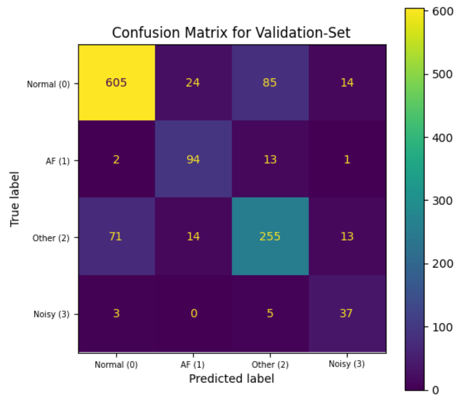


Figure 17: Classification report and confusion matrix for validation set

2.4 Data Reduction

Compression of the data was done on the original training split (80% of the dataset). The validation split (20% of the dataset) stays the same for all used compressions/stratifications to guarantee a fair comparison.

Compression Schema We used Piecewise Constant Approximation (PCA) for lossy compression. In concrete, we divided a given signal into k windows and reported the average for each window. We think that the mean is a good statistic as an approximation because it is able to still preserve a good amount of information of the original signal, as we will see in the following.

Compression Error The trend clearly shows that the more data is compressed, the higher the loss of information, which was expected to happen, since we replace one value with several per window.

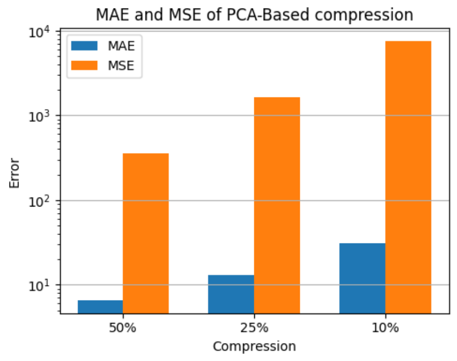


Figure 18: Mean Absolute Error and Mean Squared Error for compressed data

MAE for the 50%-compression is 6.57, which is a low error. 25%-compression has a moderate error of 13.04, and 10%-compression has a high information loss of 30.9, which means we lose a significant amount of information in this compression version. As expected, MSE is always higher compared to MAE due to the square of errors, and thus larger deviations get penalized much harder.

Compression Sizes Figure 19 shows that our approach is able to, in fact, reduce the data to the specified ratios because we used an efficient binary format to save the compression results. With the same data parser constructed in the first task, we can read our custom data format.

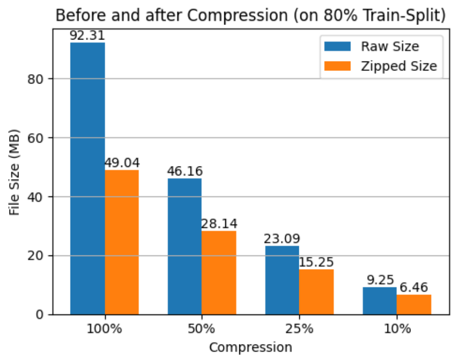
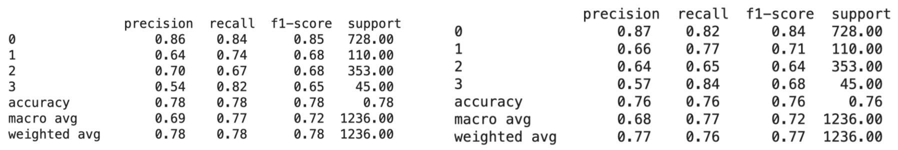


Figure 19: Compression results in MB

Training Results In the following, we will compare the training results with the compressed data to the original model from task 1.2, the augmented model from task 1.3, and with the models trained on the stratified ratios (50%, 25%, 10%).

50% compressed/stratified data

The results show that our model trained on the 50% compressed data has almost the same performance as the corresponding model trained on a 50% stratified split. The latter shows minor better performance at classes 1 & 3, whereas the compressed is stronger on class 0 & 2 (see F1-Scores). Compared to the original and augmented models, they are weaker as expected.



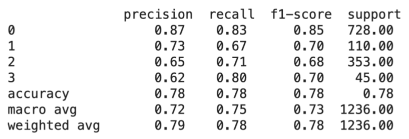
```
              precision  recall  f1-score  support
0                  0.86    0.84      0.85   728.00
1                  0.64    0.74      0.68   110.00
2                  0.70    0.67      0.68   353.00
3                  0.54    0.82      0.65    45.00
accuracy           0.78    0.78      0.78     0.78
macro avg          0.69    0.77      0.72  1236.00
weighted avg       0.78    0.78      0.78  1236.00
```

```
              precision  recall  f1-score  support
0                  0.87    0.82      0.84   728.00
1                  0.66    0.77      0.71   110.00
2                  0.64    0.65      0.64   353.00
3                  0.57    0.84      0.68    45.00
accuracy           0.76    0.76      0.76     0.76
macro avg          0.68    0.77      0.72  1236.00
weighted avg       0.77    0.76      0.77  1236.00
```

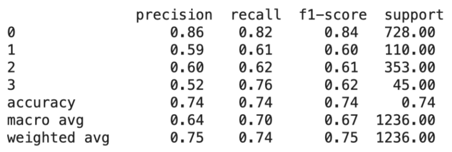
Figure 20: Classification reports of models trained on compressed (left) and stratified (right) data

25% compressed/stratified data

The model with the compressed data clearly outperforms the model with stratified data, reflected by the avg. F1-Score of 0.73 and 0.67, respectively. Interestingly, the 25% compressed data model shows slightly better performance than the 50% model, which had effectively double the data. That means 25% of the original information is still enough for the model to learn meaningful patterns from the data.



```
              precision  recall  f1-score  support
0                  0.87    0.83      0.85   728.00
1                  0.73    0.67      0.70   110.00
2                  0.65    0.71      0.68   353.00
3                  0.62    0.80      0.70    45.00
accuracy           0.78    0.78      0.78     0.78
macro avg          0.72    0.75      0.73  1236.00
weighted avg       0.79    0.78      0.78  1236.00
```

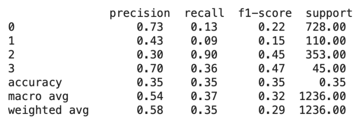


```
              precision  recall  f1-score  support
0                  0.86    0.82      0.84   728.00
1                  0.59    0.61      0.60   110.00
2                  0.60    0.62      0.61   353.00
3                  0.52    0.76      0.62    45.00
accuracy           0.74    0.74      0.74     0.74
macro avg          0.64    0.70      0.67  1236.00
weighted avg       0.75    0.74      0.75  1236.00
```

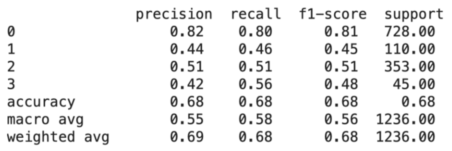
Figure 21: Classification reports of models trained on compressed (left) and stratified (right) data

10% compressed/stratified data

Here the performance of the two models deviates significantly. The stratified model still reached an avg. F1-Score of 0.56, whereas our 10%-compression achieves an insufficient score of 0.32. That means the information loss is too great for the model to learn reasonable patterns from the data, and our compression is therefore not able to beat the stratified versions in all compression sizes.



```
              precision  recall  f1-score  support
0                  0.73    0.13      0.22   728.00
1                  0.43    0.09      0.15   110.00
2                  0.30    0.90      0.45   353.00
3                  0.70    0.36      0.47    45.00
accuracy           0.35    0.35      0.35     0.35
macro avg          0.54    0.37      0.32  1236.00
weighted avg       0.58    0.35      0.29  1236.00
```



```
              precision  recall  f1-score  support
0                  0.82    0.80      0.81   728.00
1                  0.44    0.46      0.45   110.00
2                  0.51    0.51      0.51   353.00
3                  0.42    0.56      0.48    45.00
accuracy           0.68    0.68      0.68     0.68
macro avg          0.55    0.58      0.56  1236.00
weighted avg       0.69    0.68      0.68  1236.00
```

Figure 22: Classification reports of compressed (left) and stratified (right) data

Final Results

Finally, figure 23 shows the discussed results in a line plot, presenting that the model from task 1.3 is the best model, followed by the original model from task 1.2 and the models from this task.
In conclusion, these results clearly show that compression, to a certain extent, can be useful while still yielding reasonable performance, especially when dealing with storage or computation limitations.

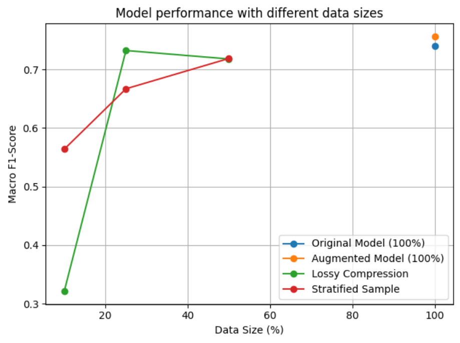


Figure 23: Final results of data reduction