CMRM Homework Assignment No. 1 (HWA1)

November 7, 2024

1 Description

The main topic of this homework is electrocardiogram (ECG) classification based on rhythmic features. The goal is to implement an algorithm that is able to classify ECG waveforms into five classes of Arrhythmia by taking into account only rhythmic features. In order to solve such a task, you will have to extract a precise feature vector and train a classifier on the MIT-BIH dataset. Use the Jupyter Notebook named `Homework1.ipynb` to implement the code, and explain in a report, step by step, what you have implemented.

1.1 Question 1

When we deal with machine learning/deep learning algorithms, the first action is to load, analyze, and preprocess the dataset. Typically, such models are built within python scripts, and datasets are downloaded locally. Often, datasets are organized in `.csv` files freeing us from the burden of downloading GBs of `.wav` files. For our ECG classification process, we will consider the MIT-BIH dataset, which you can find in the folder `dataset`. In particular:

1. Add to your conda environment `pandas` by running `pip install pandas`.
2. Add to your conda environment `sklearn` by running `pip install sklearn` (or you can also try with `pip install scikit-learn`).
3. Load the train dataset `mitbih_train.csv` using the `read_csv` method of `pandas` and store it into a DataFrame named `df`. You find further information online.
4. Load the test dataset `mitbih_test.csv` using the `read_csv` method of `pandas` and store it into a DataFrame named `df2`.
5. Analyze the DataFrame exploiting the methods of `pandas`. You can also gather information from the internet. What is the dataset for? How many items are present? Which is their data shape? How many classes? Write everything in your report.
6. Define a dictionary named `label_names` associating the keys `[0, 1, 2, 3, 4]` to the values (labels) `['N', 'S', 'V', 'F', 'Q']`. Just for your knowledge, these labels are the short for the five different Arrhythmia conditions taken into account in this dataset. In particular, the labels are associated to `['Normal', 'Fusion of paced and normal', 'Premature ventricular contraction', 'Artial premature', 'Fusion of ventricular and normal']`, respectively.
7. Take a look at the distribution of the classes inside the train and test sets. Use the method `value_counts()` to print the number of items of each class (*Hint*: note that the column containing such a class – actually, the associated key – is the last one, i.e., with index `187`). Then, create a bar plot having on the `xaxis` the labels and on the `yaxis` the number of items with that label in the dataset. Do this for both the train and the test datasets.

8. Define the sampling frequency of the curves `Fs = 360` and plot some of them. In particular, plot the waveforms in the `range(first, last, 10000)`, where `first` and `last` are associated to the indices of the first and last items, respectively. The plots should have time in seconds on the `xaxis`, and the associated label as `title`.

Typically, the 80% of the dataset is used for training, while the 20% for testing the model. In our case, data are already divided into two subsets as we have two distinct files. We will not use the whole dataset for training our classifier because this can easily occupy the whole RAM at disposal to Jupyter Notebook. Due to this reason, the performance of our classifier will probably not be the best possible but still sufficient for the scope of this assignment. Then:

- In order to subsample the datasets, import the function `train_test_split` from `sklearn.model_selection`. Such a function returns two sets drawn from the one in input according to the options provided as argument. We are interested only into the first returned value, thus the second one is to be discarded.
- Apply the function to split the train DataFrame `df` using the following options: `train_size=0.1` and `random_state=28`. Name the first returned value `train_df` (it is a DataFrame itself) and discard the second one.
- Apply the function once again to split this time the test DataFrame `df2` using the following options: `train_size=0.1` and `random_state=5`. Name the first returned value `test_df` and discard the second one.
- Print the shape of the new DataFrames: you must obtain `(8755, 188)` for `train_df` and `(2189, 188)` for `test_df`.
- Repeat the last two points of the previous list (i.e., point 7 and point 8) in order to print and plot the distributions of classes inside the newly defined DataFrames.
- Finally, remove the column related to the labels (i.e., the last column) from `train_df` and store it inside a new variable named `labels_train`. Such a new variable must be a list: you can use the `tolist()` method. Do the same for `test_df` and store said column into a variable named `labels_test` (and convert it into a list). Check and print the new shapes: you must obtain `(8755, 187)` for `train_df`, `(2189, 187)` for `test_df`, `(8755, )` for `labels_train`, and `(2189, )` for `labels_test`.

You will find guidelines in `Homework1.ipynb`. Remember to provide comments in the report.

1.2 Question 2

Now that you have loaded the dataset, you are able to process it. In particular, different curves will be characterized by different dynamic ranges; in addition, for the purpose of this task, we want the curves to cross the zero multiple times. Thus, in order to improve the generalization capability of the algorithms, we typically perform normalization on the data such that the model learns how to deal with waveforms characterized by same codomain. Perform the following steps:

- Define the variables `train_list` and `test_list` by converting the values of `train_df` and `test_df` into lists. You can use the `tolist()` method.
- Apply preprocessing on both the `train_list` and `test_list`. In particular, define a `MinMaxScaler` with `feature_range=(-1, 1)`. Then, apply it to `train_list` and store the result into `train_set`; do the same for `test_list` but store the result into `test_set`. Pay attention and reason about the difference between the `fit_transform` and the `transform` methods. What should we employ for the train set? And for the test set? Explain why in your report. Then, verify the processing by plotting the first curve of `train_list` on top of the first curve of `train_set`. Please, add labels to the axes and the legend.

In order to accomplish classification, we would like to extract a feature vector containing different rhythmic features. Hence, define the function `compute_feature_vector` which takes as inputs the ECG waveform `x`, the sampling frequency `Fs`, the length of the window `N`, the hop size `H`, and returns the feature vector. You find a prototype in the notebook. Perform then the following steps:

1. compute the standard deviation and the mean of the ECG waveform;
2. compute the zero-crossing rate using the `librosa.feature.zero_crossing_rate` function. Write in the report the definition of zero-crossing rate and why this is useful from the rhythmic perspective;
3. compute the standard deviation and the mean of the zero-crossing rate;
4. compute the STFT of the waveform using the `librosa.stft` function with `n_fft=win_length=N` and `hop_length=H`. Then take the absolute value and store the result into the variable `C`.
5. compute the spectral flux using the `librosa.onset.onset_strength` function with `S=librosa.amplitude_to_db(C, ref=np.max)`. What is the definition of spectral flux? How can it be computed if we do not want to exploit `librosa`?
6. compute the standard deviation and the mean of the spectral flux;
7. define the feature vector `f_vector` by concatenating all the features you have computed (together with standard deviations and means);

Once the function is defined:

- Test it by computing the `f_vector` on the first waveform of the `train_set`. Print it and plot it.
- Use it for computing the `f_vector` of both the training set and the test set by collecting the results into two lists: `train_fvector` and `test_fvector`. In order to have a clue on the processing time, instantiate a progress bar using `tqdm`. In this phase, you must stick with the feature vector as is.
- Check the shapes of the two lists.

Provide comments in the report on the performed steps. In particular, address the questions mentioned in step 2 and step 5. What is the aim of a novelty function? Why do you think it is worth to add different rhythmic features? Can you spot any trade-off?

1.3 Question 3

We have extracted a feature vector containing different rhythmic features. It is now time to select a model for accomplishing classification. For this task, we are going to train a Support Vector Machine (SVM). The main idea of SVMs is finding a frontier which separates observations into classes. In particular, the objective of an SVM classifier is to find the best $P-1$ dimensional hyperplane – also called the decision boundary – which can separate a P-dimensional space into the classes of interest. Notably, a hyperplane is a subspace whose dimension is one less than that of its ambient space. SVM identifies the endpoints or end vectors that support this hyperplane while also maximizing the distance between them. Address the following points:

- SVMs take advantage of kernels for accomplishing clusterization. Study and learn how SVMs work and explain how different kernels can be considered for improving the performance of the model.
- Train an SVM using the `sklearn.svm.SVC` function. Start by setting `C=10` and `kernel='rbf'`.
- Save the model using the string format `f'my_model/svc_{kernel}_C_{C}_N_{N}_H_{H}'`. You can use, for example, `joblib` or `pickle`. In this way, you can load it without the need for training from scratch.
- Give a definition for hyperparameter. Typically, in order to tune hyperparameters, we try different combinations of model settings. In this point, however, we do not perform a grid search as we leave it for Question 1.5.
- In order to assess the quality of your training, compute the accuracy on the training set. Explain the concept of overfitting.

It is time to perform classification. By solving these points, you will find out if the model is able to accomplish the desired task. In particular:

- Test the trained classifier on the test set.

- Print the accuracy. What can you tell just by looking at this mere number?
- What is a confusion matrix? What does it represent? Plot the confusion matrix for your prediction using `sklearn`. Does the classifier perform in the same fashion for all the classes? Is there a class for which it performs better? If yes, why?
- At this point, you should have understood that the mere accuracy is not enough to verify the performance of a classifier. In particular, it is always better to take a look at other metrics. To this aim, use the function `classification_report` of the library `sklearn.metrics` to compute recall and f1-score. Please, provide the definitions for this two metrics and comment on the results. How should we interpret the values? Do we have to minimize or maximize them?
- The confusion matrix is built upon the information provided by the so-called *false positive*, *false negative*, *true positive*, and *true negative*. This is valid for two-class classification problems. In multi-class classification problems, we cannot formally talk about these metrics, but we can improperly (for example, if we collapse everything into a "normal-ill" classification problem). Write the definition of each of them and how can we obtain such values. When we solve tasks in a data-driven fashion, we should always take into account the characteristics of the particular task that we are solving. We do not have just to look at numbers. In particular, our task is to detect an anomaly in the heart rate. In the light of this consideration, apart from the true positive, which of the other three values is the one that we care the most? Once you have chosen it, compute it and print it.

Print and plot everything, and provide comments in the report. How does the model perform on the test set? Why does it perform differently from the training set? Comment on the confusion matrix. Can you explain why the relative accuracy of certain classes is higher than others?

1.4 Question 4

If we take a look at the items for each class (the bar plots), we immediately see that the dataset is really unbalanced. What we typically do (or at least wish to do) is to let the model learn from a balanced representation of the input space, otherwise some biases can arise. To this aim, we may think to redefine the dataset in such a way to have the same amount of items per class. Thus:

- Initialize an empty DataFrame named `train_df`. Starting from `df`, extract a new dataset by using for each class the `sample` method as `sample(641, random_state=42)` and assign it to `train_df` (*Hint*: use `loc` and `iloc` for finding all the items related to a certain class, and then apply the `sample` method to draw only from that restricted amount of items). Print and check the shape of `train_df`: it must be equal to `(3205,188)`.
- Do the same for the test set. Initialize an empty DataFrame named `test_df`. Starting from `df2`, extract a new dataset by using for each class the `sample` method as `sample(162, random_state=42)` and assign it to `test_df`. Print and check the shape of `test_df`: it must be equal to `(810,188)`.
- Finally, convert the values of `train_df` and `test_df` to lists.

In this process, however, we are limited by the smallest number of items in the classes, i.e., `641` for the train set and `162` for the test set. We know that the performance of machine learning models is strictly dependent on data, and, typically, the more the data the better the performance. Thus, in order to remove the constraint imposed by the dataset, we can rely on *data augmentation* techniques. The purpose of such techniques is on the one hand to increase the number of tracks in the train dataset, on the other hand to improve the robustness and generalizability of the model. The augmentations are typically applied with a certain probability in order to prevent biases in the datasets. You are going to implement two simple methods: the first will stretch the waveform, the second will modify the amplitude. Address the following points:

- Define a class name `augment`. The constructor, in this case, is not responsible for any particular setting.
- Define a class method named `stretch`, which has as arguments `self` and the signal `x`, and performs the operations:

1. define the new number of samples l as

$$l = 187 \cdot \left(1 + \frac{\beta - 0.5}{3}\right), \tag{1}$$

   where β is a random number in $[0, 1]$ and can be computed with the `random.random()` function. Cast then l as an `int`;
2. resample the signal `x` to have l samples using the `scipy.signal.resample` function and assign the result to the variable `y`;
3. then, implement the following conditional structure: if $l < 187$ create an np.array `y1` full of zeros of shape `(187,)` and assign the slice `y1[:l] = y`. Else, assign the slice `y1[:187] = y`;
4. return `y1`.

- Define a class method named `amplify`, which has as arguments `self` and the signal `x`, and performs the operations:
  1. generate a random number $\alpha \in [-0.5, 0.5]$;
  2. return the value `x * (`α `+ 1)`.

Now, we need a method that applies such augmentations. Define a class method named `perform`, which has as arguments `self`, `x`. As mentioned above, we want the augmentations to be randomly applied. In other words, we want the method to apply `stretch`, `amplify`, both, or neither of them in a random fashion. To this aim, perform the following operations:

1. initialize an empty variable named `performed_augmentations`. In this variable, you will store the augmentations that are performed such that you can keep track of them (since you do not know which ones are applied as they are random);
2. implement the conditional structure: if the `np.random.binomial(1, 0.5) == 1`, apply the `stretch` to `x` and add the string `'stretch'` to `performed_augmentations`;
3. implement the conditional structure: if the `np.random.binomial(1, 0.5) == 1`, apply the `amplify` to `x` and add the string `'amplify'` to `performed_augmentations`;
4. return the values `x, performed_augmentations`.

At this point, you have your class `augment` and you are ready to test it. Thus:

- Instantiate the class `augment`.
- Perform the augmentation on the first item of `train_list` (remember to remove the last column) and print the list of augmentations.
- Plot the augmented version on top of the original one. Add labels and a legend.
- Repeat the previous points multiple times in order to test the randomness of the augmentation.

It is now time to apply such augmentations to our train dataset. In particular:

- Define the variable containing the number of waveforms per class that we want to add to our dataset `n_aug=100`.
- Instantiate the class. Then, for each class, start directly from `df` and extract `n_aug` items using `sample(n_aug, random_state=16)`. You can exploit `loc` and `iloc` to obtain the subset of `df` related to a certain class.
- For each of the `n_aug` items perform the augmentation (remember to remove the last column) and concatenate the augmented waveform to `train_df`.
- Print the counts for each class and the shape of `train_df` like you did in some of the previous points. You should obtain a shape of `(3705, 188)` and an equal count per class.
- Convert both `train_df` and `test_df` to lists with names `train_list` and `test_list`, respectively.

- Fit and apply a new `MinMaxScaler` for normalizing both `train_df` and `test_df` and repeat all the remaining operations that are required to have your data ready for training your classifier.

Print and plot data when required and add labels to the axis, as well as legends. Write comments in the report. Which other augmentations could you implement to serve the scope (you do not have to actually implement them)?

1.5 Question 5

It is time to train our new classifier on the augmented dataset and perform the classification in order to verify if the new dataset is actually able to provide us with a better classification. In particular:

- Compute the feature vectors of `train_set` and `test_set`. Then, train the classifier starting with a kernel `rbf` and `C=10`.
- Test the trained classifier on the test set.
- Print the accuracy. How is it different with respect to the previous case? Does the model overfit?
- Plot the confusion matrix for your prediction and use the function `classification_report` to obtain accuracy, recall, and f1-score values. Print also the metric (among the improper true negative, false positive, and false negative) that you selected in Question 1.3. How are the results? Does the classifier perform generally better? If yes, can you guess why? How are the recall and f1-score values with respect to the previous case?
- Test different parameter configurations. By keeping fixed the kernel, try to compare the performance (by plotting and printing all that you considered in the previous points) of the classifier using all the values in the set `C={0.1, 1, 100, 1000, 10000, 100000}`. Does the performance improve if we increase `C` or not? Why? What does `C` controls in the training of the SVM? Which conclusions can you draw then? Which is the best configuration?
- Now, you can vary all the hyperparameters (kernel, `C`, `N`, `H`, etc.). Which is the best-performing kernel? Can you explain why?

Print and plot everything, and provide comments in the report. How does your best model perform on the test set? Why do we have different performance with this new dataset with respect to the previous one? Comment on the confusion matrix. Can you identify biases of the classifier toward precise classes? Can you think of methods for improving the classification and, in particular, the metrics you selected?

2 General Rules

There are two Homework Assignments (HWAs) for Computer Music Representations and Models (CMRM), each worth 20% of the final grade, and this is HWA1. HWAs can be done individually, or in groups of two students. Please, keep in mind that students of the same group will be given the same grade, irrespective of their contribution, therefore, please make sure that all members equally contribute to the assignment.

The maximum grade attainable is 30/30, but if the HWA is turned in by **Wednesday Dec 4** (before midnight) you will get a bonus of 3 points. If you turn it in after that date but before the first *appello* (exam), you will get the full grade without bonus. If you turn it in after the first *appello* (i.e., end of Jan 2025/beginning of Feb 2025), you will get a penalty of 5 points. It is highly recommended turning both HWAs by the assigned deadlines in order to optimize the effort, keep the pace, and maximize the grade.

3 Files to be Delivered

You are required to deliver the following files:

1. a report, containing all answers to the questions and comments to the code. Include your surname/surnames in the title of the report (e.g., `Rossi.docx` or `Bianchi_Rossi.docx`). You can use whatever editor, even LaTeX, and, in general, you can provide a pdf file (rather than a doc file);

2. the filled `Homework1.ipynb` file. This is already divided into different sections and cells according to the questions that you are required to solve. In order to ease the solution, the notebook is provided with some guidelines in the form of comments. Rename the notebook with your surname/surnames (e.g., `Rossi.ipynb` or `Bianchi_Rossi.ipynb`). Please, add comments to the code, and plot or print all intermediate results. It is suggested to add titles, axis labels, and/or legends to the plots.

Zip the report and the notebook. Name the zip file using your surname/surnames (e.g., `Rossi.zip` or `Bianchi_Rossi`). The zip must be turned in by Wednesday Dec 4 (before midnight). **Only one student for group must load the zip file on WeBeep**.