ML1: Project report

Group B

Jose Manuel García Bello
Sara Gómez Feás
Ovidio Mantegia Moar
Daniel Vila de la Cruz
Carlos Villar Martínez

Master in Artificial Intelligence 2022/23

November 2022

Contents

1 Introduction

Every investigation, work, or project arises from an inquietude. In the current world, heart diseases are the principal cause of death and, even if they aren't lethal, they are still one of the most common diseases people suffer from. Given this, a system capable of detecting them in order to prevent them could be extremely useful.

There are multiple ways to detect heart disease, being the analysis of an electrocardiogram (ECG) one of the most reliable. This test is an everyday thing in hospitals and what it does is measure the heart's rhythm and electrical activity.

Here is where the question arises, can the last advances in technology - by analyzing electrocardiograms - help us to detect and classify these affections in an easier, more accurate, and faster way?

1.1 Problem explanation

Throughout this project, we are going to focus on three different diseases: premature ventricular contraction, supraventricular premature beat and fusion of paced and normal beat. Although none of them alone use to lead to a chaotic and sudden cardiac death, they can cause us to develop irregular heart rhythms and weaken the heart muscle.

Detecting these diseases in time can prevent them from evolving and the patient from developing serious consequences. That's why we are going to create a program that, through machine learning, may detect and classify these conditions. The main idea is to train a model capable of doing this classification with the highest possible accuracy.

To do so, we need to create five different classes that our program can choose from when classifying the data. Three classes corresponding to the three diseases mentioned would not be enough to carry out the classification; we also need two more to account for normal heartbeats and unclassifiable heartbeats (they aren't normal heartbeats but they don't belong to any of the other three classes).

For this, we also need an according dataset (to be described later). This must contain data on the heartbeat of different patients as well as the diseases of the same, otherwise, the model would not be able to learn to relate the electrocardiogram data with the corresponding disease or absence of it.

Although the possible diagnosis of heart disease using ECG is made by reviewing a whole ECG with many heartbeats (Education, Oxford Medical. (2014)) interpretation of a single heartbeat could give information about possible heart abnormalities.

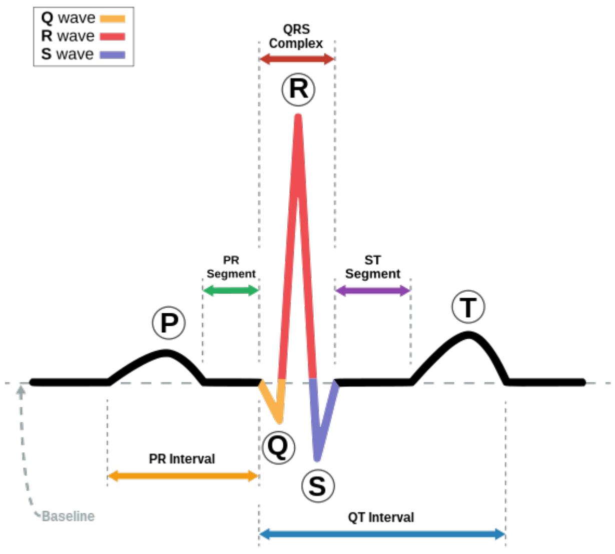

Figure 1: ECG model of a heartbeat in normal sinus rhythm

Therefore, the main objective of this project is to obtain a model that, when we introduce a new electrocardiogram with heartbeat data, can correctly identify if our patient is healthy or if he has some type of arrhythmia.

1.2 Data description

An important part of creating an accurate model is having a good dataset to train it on. Before saying anything else, we may mention that the Oxford Dictionary defines a dataset as a data collection treated as a single unit by a computer.

The dataset we are going to use is the ECG Heartbeat Categorization Dataset (Fazeli, S), which is composed of collections of heartbeat signals. Each row in the dataset represents a heartbeat (Figure 1), which is categorized into its possible abnormalities.

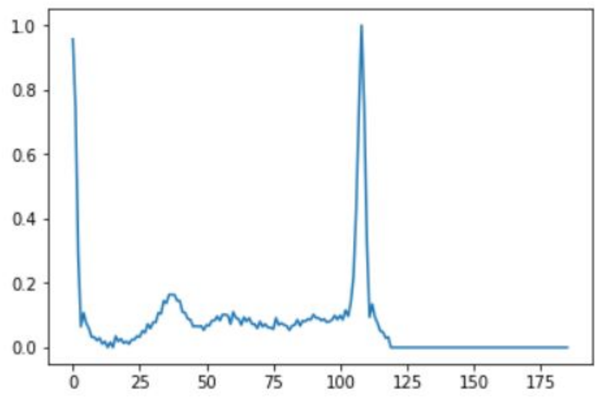

Figure 2: Sample hearbeat from the dataset.

This dataset contains information from two other datasets, the MIT-BIH Arrhythmia Database (Moody, G.B., Mark, R.G. (2001)) and The PTB Diagnostic ECG Database (Fu, Xigwen. (2021)). We are going to focus our work on using the first one, which contains a total number of 109446 samples, divided into train and test. The second dataset contains 14552 samples, that can be used to expand the project if needed.

As mentioned, for this project, we present a pretty big one. It consists of a CSV file that contains 109446 samples, each one a heartbeat of a patient.

The data is ordered as follows.

We have 109446 rows (where each one corresponds to a patient) and 188 columns. Within these 188 columns, the first 187 contain the data of the electrocardiogram signal values for the duration of a heartbeat sampled at a frequency of 125Hz (so they correspond to around 1.5 seconds of ECG); and the last one contains information about the disease that the patient is suffering from.

Each of the columns corresponding to the electrocardiogram data represents its value at a moment 't'. These values range from 0 to 1, with 0 being the minimum and 1 being the maximum.

The diseases are shown in the last row, in which each letter represents belonging to one of the possible classes:

- **N** – [**class 0 / NEGATIVE**] : Normal beat (Non-ectopic beats).
- **S** – [**class 1 / POSITIVE**] : Supraventricular premature/ectopic beat.
- **V** – [**class 2 / POSITIVE**] : Premature/ectopic ventricular contraction.
- **F** – [**class 3 / POSITIVE**] : Fusion of paced and normal beat.
- **Q** – [**class 4 / POSITIVE**] : Unclassifiable beat.

The dataset is freely available at: train set and test set.

By analyzing the dataset by column frequencies and correlation between features, we can check that there are features at the end of the dataset with almost all values being zero (see figure 3).

Also, by analyzing the correlation matrix of the dataset (see figure 4), we can verify that the first features appear to be related to each other and we can see a similar relationship at the end of the dataset features, related to the highest occurrence of zero values.

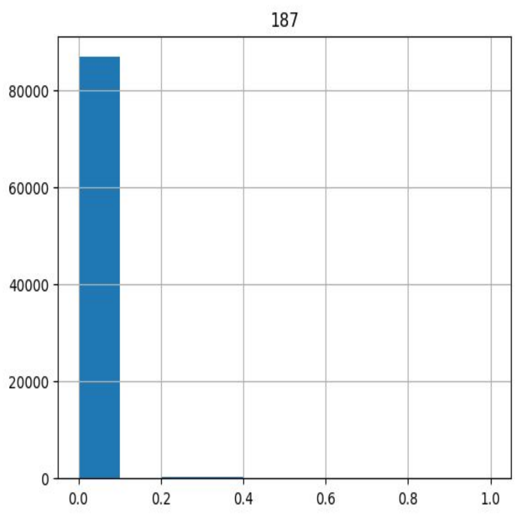

Figure 3: Frequency for feature number 187

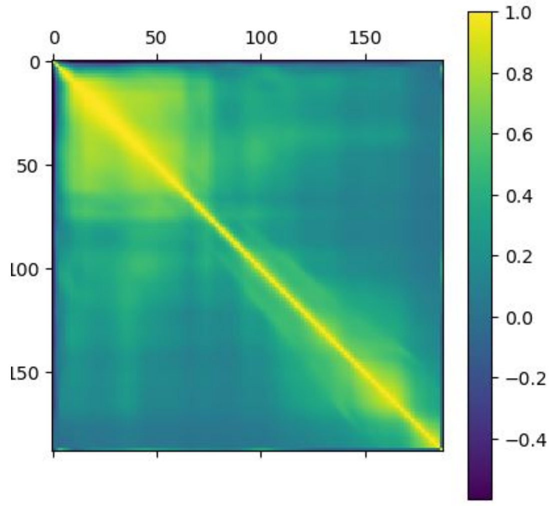

Figure 4: Dataset correlation matrix

It is important to note that, as expected, we will have many more samples corresponding to the class normal (N), so if we don't do some kind of preprocessing, we may get a model with high precision that doesn't really work. For example, imagine our dataset has 95% of normal condition samples, if we use our data this way, maybe our model will only correctly detect normal beats, but it will still be accurate to 0.95.

In figure 5, this disproportionate distribution of classes can be seen.

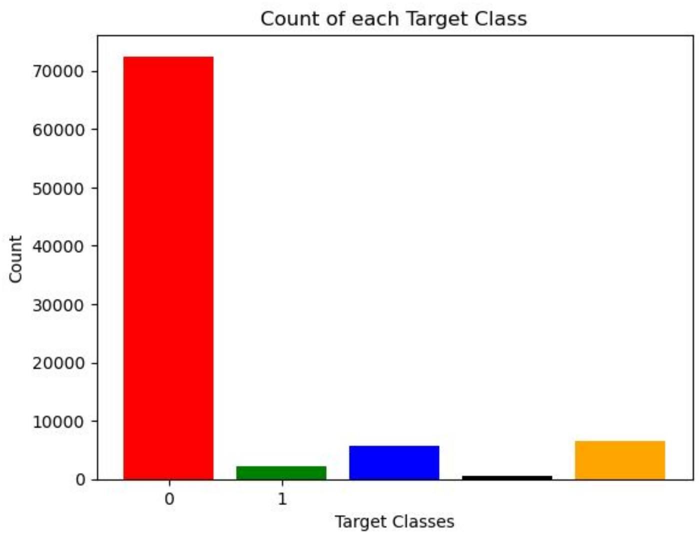

Figure 5: Histogram of classes

To avoid this, what we can do is multiply the data of the remaining classes to balance, as much as possible, the number of features of each one. We can even reduce the amount of data from the normal class (N) by a certain percentage.

1.3 Metrics

Due to the unbalanced distribution of the data, the accuracy metric is not enough to properly assess the performance of the classification models.

In the dataset used, around 90% of the samples correspond to the normal (N) class, so simply by predicting always normal (N), a model could achieve some 90% accuracy. Hence, other metrics that assess the classification performance in a more detailed way are necessary.

For the purpose of this project the main metrics to be used are:

Accuracy. The total amount of correctly predicted classes divided by the number of samples. As mentioned, this is just another indicator, but cannot be the main one for this problem.

Precision (Positive Predictive Value or PPV). The proportion of true positives out of all predicted positives. The higher, the fewer false positives. In the context of this problem, it is important to minimize the false positives, since they mean some patients might be diagnosed (and maybe eventually treated) with a heart condition that they do not have.

Recall (Sensitivity). The ratio of true positives from all the real positives. The higher it is, the fewer false negatives. In the context of this problem, it is fundamental to minimize the false negatives, since they mean the patients might have a heart condition that goes unnoticed.

F1-score. This is the main metric that better assesses the overall performance of the model by combining the precision and recall metrics, which we want to maximize. It is the harmonic mean of the precision and recall values.

All of the metrics described above are to be measured for training, validation and testing. The most important ones to evaluate the overall performance of the model are the testing metrics, which are calculated using the testing dataset, whose samples are not used during training or validation.

For training and validation metrics, two alternatives were used:

Holdout validation. Splitting the training dataset to generate a validation set with a percentage of the samples and measuring the validation metrics with those samples, which were not used for training the models.

k-fold Cross-validation. Generating k disjoint subsets of the training dataset and splitting each of them into training and validation sub-subsets. Then, each model is trained and the validation metrics are calculated for each subset with each of the sub-subsets. Finally, the metrics for each subset are averaged to provide an estimation of the validation metrics. After obtaining the validation metrics, the selected models are trained with the full training set.

With this technique, the use of all training data for training is guaranteed, while a better estimation of the overall validation metrics can be calculated.

So long the problem is a multi-class classification problem, for each of the classes the precision, recall, and f-score metrics are calculated. Then, in order to provide a unique metric for the model, the resulting per-class metrics are averaged.

Seeing the unbalanced nature of the classes for this problem, and that all of the classes are equally important, the *macro* averaging was chosen.

1.4 Comparing models

Considering the description of the metrics used and how to calculate them, the main way of comparing the performance of the classifier models used for this problem is the F1-score value calculated on the test set of the dataset.

In the case of similar f1-scores, the model with higher recall (sensitivity) could be considered better for the problem because it will yield fewer false negatives, which, as aforementioned, could imply that a patient with a heart condition is not identified.

Then, if both the f-score and the recall are similar, the models must have a similar precision (by definition) and also a similar accuracy. In such cases, the simpler and more efficient models will be considered better.

Another way of comparing classifier models is by using ROC curves for the model.

The ROC curves are plotted on a graph where the vertical axis is the TPR (True Positive Rate, sensitivity or recall) and the horizontal axes are the FPR (False Positive Rate, the complement of the specificity). Each curve represents how the model performs in both metrics with different thresholds to distinguish the classes.

The better the model performs, the closer the curve will pass to the (0,1) corner, where the TPR is 1 and the FPR is 0; in other words, where the sensitivity and specificity are maximum and, hence, the precision and f-score are also maximum.

Here is a sample graph with the ROC curves for some models evaluated for the problem (figure 6).

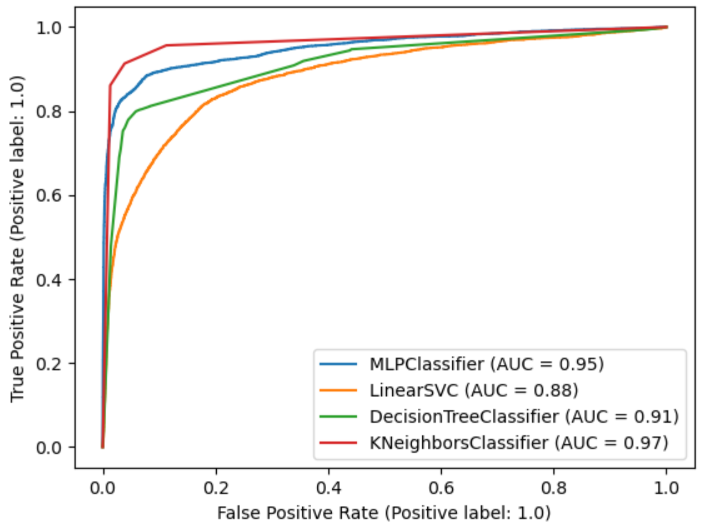

Figure 6: ROC curves for 4 sample classifiers.

1.5 Code structure

The main objective of this section is to describe the developed code structure and the files which are part of this structure. We have grouped the code developed during the course and the new code developed in several Julia files and developed a main Julia script which calls other modules to execute the whole process.

Although we have included code developed during the course, we have decided to generally use scikit libraries instead of our developed set of functions. Therefore, the learning, testing, validation and evaluation process is developed using the scikit package.

The developed code uses Julia Structs, abstracting the main objects that will be used during the code and creating an abstract layer between scikit and project to obtain a more readable code and easier to integrate into all the developed approaches. We have also divided the developed code into several Julia files whose descriptions can be seen in the subsections below.

As an overview of the developed code and its structure, we have included the following folders in the project root that are already defined in the project guideline:

- A folder named 'dataset' with the base dataset used in the project approaches.
- A folder with the name 'utils' that contains the Julia libraries developed in previous notebooks and libraries developed for this project.

We have also defined the following folders in the project root, in order to maintain our root folder as clean as possible:

- A folder named ’result’ that stores the ML metrics for each approach executed.
- A folder named ’approaches’ with the definition of the approaches developed and tested in this project.

1.5.1 General architecture of the code developed

The developed code abstracts the concepts of architecture, program and process. A machine learning classification problem has a defined architecture, such as an ANN or a kNN, a program defined by the original dataset to be used, and a process that represents the steps to be done in the learning process. Those structures have been developed independently, therefore different architectures can be executed using the same data set without any major code changes.

For defining a machine learning architecture we have defined the Julia struct ’approach’, which contains attributes related to the architecture to use. The main attributes defined in this struct are the array of machine learning architectures (ANN, SVM, kNN or decision trees), the dimension reduction to be applied to the original data and the number of folds in cross-validation if required.

Secondly, we have defined the Julia structure ’problem’, which defines the dataset to be used and establishes if the problem would be a multiclassification problem or if it will be transformed into a binary classification problem.

Finally, we have defined a starting point that represents the training, validation and evaluation of the specified machine learning architecture and corresponds to the run function that receives as input parameters a ’problem’ and an ’approach’ and returns a list of ’evaluation’ structs that correspond to the specific trained model and its most relevant statistics to be analyzed once the process is finished.

Below appears an example of how the execution of a problem would be coded following this architecture, where it is possible to check the concepts explained above.

The ’approach’ and ’problem’ structs are created using helper function and start the machine learning process by calling the ’run’ function, which returns a structure containing the models and the metrics associated with those models. In this case, we are creating an approach with a single kNN. We are preprocessing the data by multiplying the class samples and reducing their dimensionality using PCA, using 10-fold cross-validations and creating two ensembles.

```
approach = createApproach(
    name="dummieApproach",
    multiplyClassSamples=[1,2,1,3,1],
    knns=[buildKNN(neighbors=3)],
    useKFoldCrossValidation=true,
    kFolds=10,
    ensembles=[SoftVoting, RandomForest],
    numberBestModels=3,
    reduceDimensionality=true,
    dimensionalityReduction=DR_PCA)
```

```
multiClassProblem = buildProblem(
    trainSetPath="dataset/mitbih_train.csv",
    testSetPath="dataset/mitbih_test.csv",
)

result = run(multiClassProblem, approach)
```

In order to run a complete approach, we have coded two different ways to perform the execution of an approach.

The first option is to run the training process on all models, evaluate those models, calculate the defined metrics, and store all metric results in a CSV file for further analysis. In this case, we first train all the models and then calculate their metrics (see figure 7).

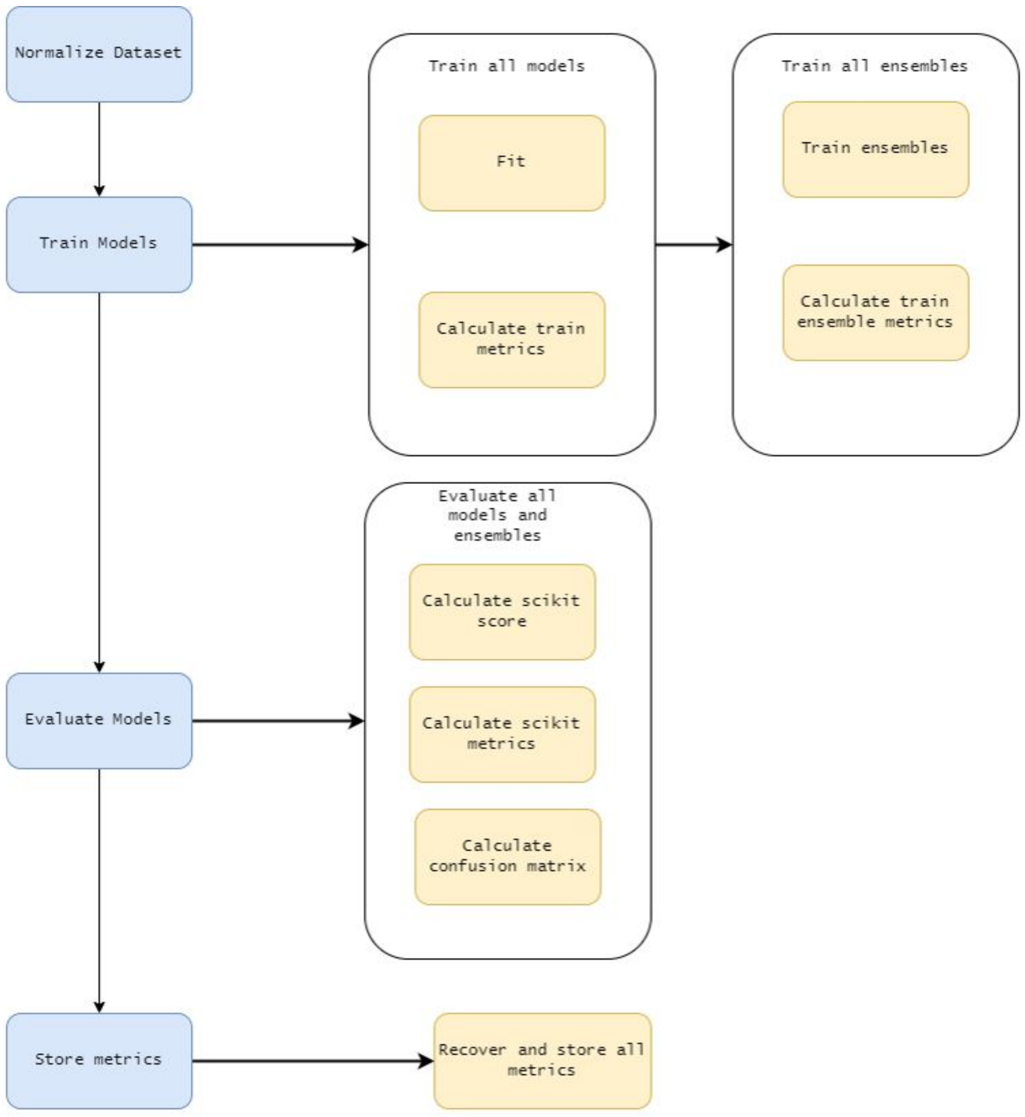

Figure 7: Run approach

The second option for running an approach is to run the entire process (train, evaluate, and store metrics) for each one of the models defined in the approach. In this case, we start by training, evaluating, and storing metrics for each model and end up creating the ensembles with the best models, training those ensembles, and calculating their metrics.

See Figure 8 for a schematic of this process included for clarification.

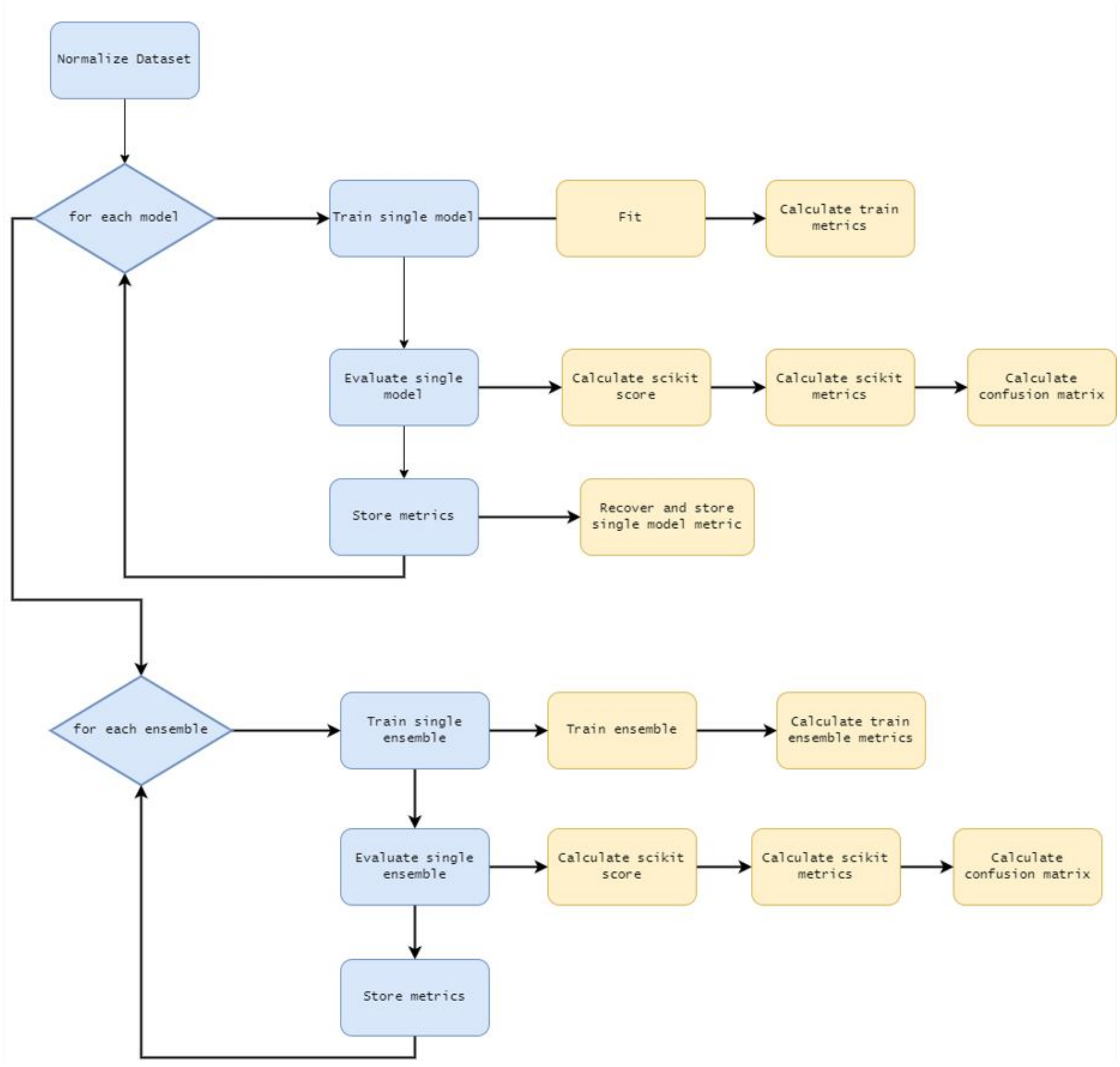

Figure 8: Run approach per model

1.5.2 Utils folder

Once we have explained the overall process, we are going to introduce each folder included in the project and its contents.

First, we are going to explain the Julia libraries developed in the utils folder, in which belongs the following Julia files:

- **models.jl**. In this file, we have codified the structs and type definitions designed for this project and the helper functions associated with creating the most important structs, such as Metrics, ANN, SVM, KNN or DT.
- **preprocess.jl**. Library with functions related to preprocessing the original dataset, and modifying the number of class instances required to balance the number of examples per instance.
- **libraries.jl**. Library with functions developed during this course.
- **approach.jl**. Library with functions used for executing the 'approach' with the 'problem' defined. It contains functions to normalize the dataset, train, validate, evaluate the models defined and calculate metrics for a specific model
- **approachNew.jl**. Library with extra functions required to execute process per model instead of all models at the same time.
- **dimensionality.jl**. Library that contains functions to reduce the dataset dimensionality.
- **ensembles.jl**. Library with helper functions for creating ensembles.
- **postProcess.jl**. Library with helper functions used to convert the metrics result of a specific process execution in a CSV file for further analysis
- **postProcessNew.jl**. Library with extra functions required to post-process data per model instead of all models at the same time.

1.5.3 Approaches folder

Approaches folder contains one Julia file per approach defined in the project. Each approach contains the architectures chosen for the specific approach (models, data reduction dimensionality, cross-fold configuration and ensemble configuration) and defines a function which returns the configured 'approach'.

1.5.4 Result folder

Result folder contains the CSV files with the model metrics, which are the result of the approaches executed, including all models evaluated and metrics related to the testing, validation and evaluation of the trained models.

1.5.5 Considerations about development

We had decided to mainly use the scikit libraries instead of the developed ones, so a general review of the scikit library was necessary to find the essential methods to complete the development. We also faced some issues related to SVMs. Using scikit LinearSVM did not work with ensembles because it does not provide method predict_proba so we had to wrap them into a CalibratedClassifierCV.

The random seed was set to ensure the repeatability of the results. This can be found at the beginning of the library.jl, which is imported as the first Julia library in the whole process.

1.5.6 Considerations for executing process

File Main.jl in the root folder corresponds with the file to be executed using Julia to launch the overall process. This process points at datasets stored under the 'dataset' folder and accesses them using relative URI's. So Main process can find and read properly those files it is required to execute it from the 'code' folder.

1.6 Bibliographic analysis

The classification of ECG heartbeats is a well-known problem in the area of machine learning. Due to the great diversity of data management practices, multiple techniques are still being explored, and there is not a preferred one (Gupta et al 2022)[1].

- The first study (Mian Qaisar and Hussain 2021)[3] is focused on detecting five different arrhythmia types: Right Bundle Branch Block (RBBB), Left Bundle Branch Block (LBBB), Normal signals (N), Atrial Premature Contraction (APC) and Premature Ventricular Contraction (PVC). Since their model had problems classifying the 'LBBB' class, they made a new 4-classes dataset, combining 'LBBB' with 'RBBB'.

  Their first step is to subsample each signal, and then the noise is removed using a band-pass FIR filter. For the classification, they used kNN, ANN, SVM, Random Forest and Bagging.
  The results show that the Random Forest achieves the highest classification accuracy of 97% in the case of the 5-class dataset, while in the 4-class dataset the better results are achieved by the SVM, with a classification accuracy of 99%.

- The second technique we will talk about is using Convolutional Neural Networks (Xu and Liu 2020)[5], a technique which is growing in popularity nowadays. This study is focused on detecting 5 types of arrhythmia heartbeat: non-ectopic beat (N), supra ventricular ectopic beat (SVEB, S), ventricular ectopic beat (VEB, V), fusion (F), and unknown (Q).

  They also used a raw signal, corrupted by noise, so their first step is to denoise it using wavelet filters. The network used consists of nine layers, including four convolutional layers, two subsampling layers, two fully connected layers, and one softmax layer.

  The results of the proposed model are a sensitivity of 99.2% and positive predictivity of 99.4% in VEB detection; a sensitivity of 97.5% and positive predictivity of 99.1% in SVEB detection; and an overall accuracy of 99.43%.

- Lastly, a less common approach for this problem is applying RNN (LSTM) and density-based clustering (Zhang et al 2017)[6]. The focus of this study is to classify a personal signal into the same classes as the previous study.
  Their first step is to obtain common training data using a clustering technique and train a common model. After that, when a new record appears, they use again clustering to find representative patterns in that record as patient-specific data. A patient-specific model is trained based on a common model with patient-specific data.

  Three datasets were used for evaluation, and the average results of the proposed model were: a sensitivity of 97.4%, positive predictivity of 97.9%, and an accuracy of 99.6% in VEB detection; a sensitivity of 86.7%, a positive predictivity of 89.0%, and an accuracy of 98.9% in SVEB detection.

2 Development

This section includes a brief description of the characteristics of each approach, the execution results and some comments about it.

2.1 Approach 1

2.1.1 Approach description

In this approach, we address the binary problem. To do so, the dataset used is the one described in Section 1.2. As mentioned in said section, this dataset contains 109446 samples, each of which is a heartbeat from a patient. Each of these features has 187 columns of data corresponding to an electrocardiogram beat and 1 column that contains the label of the class to which it belongs.

In order to convert the problem into a binary one, the 4 classes corresponding to the different heart diseases are grouped in the same class: *not being healthy*. Therefore, we have a model that classifies heartbeats as negative, i.e., belonging to the normal class (N), or positive. By grouping the data this way, the binary model has a more balanced distribution of negative/positive samples.

All dataset features has values between 0 and 1 and we have no detected any outsider value in the features, so we have decided to normalize the dataset using a min-max approach.
This approach has the following characteristics:

1. Deals with the binary problem.
2. Multiply by 0.5 the class with the largest number of instances (the normal class) and by 2 the remaining classes.
3. Applies a dimensionality reduction using the PCA technique, halving the number of features.

4. Applies 5-fold cross-validation in order to select the best hyperparameters for the models.

5. Fits 8 ANN models with 1 or 2 layers,

6. Fits 8 SVM models, all of them linear, due to the big size of the dataset.

7. Fits 6 Decision Tree models with different maximum depths.

8. Fits 6 kNN models with a different number of neighbors.

9. Fits 3 ensemble models:

   (a) A soft-voting classifier using the best 5 models from cross-validation.

   (b) A hard-voting classifier using the best 5 models from cross-validation.

   (c) A random forest classifier.

2.1.2 Results obtained

Firstly, we trained the ANN models, see Table 1. All of them present quite good results, being somewhat worse perhaps those with simpler architectures (fewer layers and less number of neurons in each one). The one having one layer with 16 neurons or the one having two layers of 8 and 16 neurons could be considered the best models among the ANNs.

	Accuracy			**F-Score**			**Precision**			**Recall**		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
ANN_[32]	94.6	93.8	91.6	94.5	93.7	86.9	95.0	94.4	83.8	94.2	93.4	91.6
ANN_[16, 32]	94.4	93.2	92.2	94.3	93.1	87.6	94.6	93.6	84.6	94.1	92.9	92.1
ANN_[16, 8]	94.6	93.7	95.8	94.6	93.6	92.5	94.7	93.9	93.5	94.5	93.5	91.6
ANN_[4]	91.4	90.3	90.3	91.3	90.1	84.7	91.9	91.1	82.0	91.0	89.8	88.8
ANN_[8]	92.9	92.2	94.2	92.8	92.1	90.4	93.2	92.7	88.7	92.6	91.9	92.4
ANN_[16]	94.3	93.1	95.9	94.2	93.0	**92.9**	94.5	93.5	92.9	94.0	92.8	92.8
ANN_[8, 8]	93.1	92.1	90.7	93.0	92.0	84.8	93.6	92.8	82.7	92.7	91.6	87.6
ANN_[16, 16]	93.9	93.2	91.1	93.8	93.1	86.1	94.2	93.6	83.1	93.6	92.9	90.8

Table 1: Metrics of the ANN models.

Secondly, we trained the SVM models, see Table 2. The first three models present a similar performance, with very little difference between the results obtained when evaluating the models with $C = 1.5$, $C = 3.5$ and $C = 15$.

From there, we can see how an increase in the value of C does not imply a performance improvement, quite the opposite.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
SVM_lin_1.5	81.8	80.0	66.8	81.4	79.3	61.9	82.0	80.7	64.9	81.2	79.3	76.0
SVM_lin_3.5	81.8	80.0	66.8	81.4	79.3	61.9	82.0	80.7	64.9	81.2	79.3	76.0
SVM_lin_15	81.6	79.3	69.3	81.4	78.7	63.9	81.7	79.8	65.7	81.4	78.8	77.1
SVM_lin_20	79.6	77.3	75.2	79.4	76.6	68.7	80.7	79.0	68.0	79.8	77.3	79.5
SVM_lin_35	77.7	76.1	20.2	76.9	74.7	18.5	80.7	79.4	57.1	77.5	75.7	51.5
SVM_lin_50	73.8	70.6	80.1	72.2	68.1	**72.6**	79.3	78.1	70.4	74.2	70.7	80.2
SVM_lin_75	73.0	71.3	19.8	69.8	67.0	18.1	79.0	77.1	57.0	73.4	71.5	51.3
SVM_lin_100	73.8	71.6	75.7	71.5	67.7	68.9	80.0	79.1	68.0	73.0	70.6	79.1

Table 2: Metrics of the SVM models.

Thirdly, we trained the DT models (see Table 3), being the model which brings the best results the one having a depth = 10. As we can see, there is a correlation between complexity and performance.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
DT_5	86.6	84.2	86.3	86.2	83.6	78.1	87.5	85.7	76.1	85.8	83.3	81.0
DT_6	88.5	86.3	84.2	88.3	85.8	76.1	89.2	87.4	73.7	87.9	85.5	80.7
DT_7	90.1	88.1	87.9	89.9	87.7	81.1	90.9	89.3	78.5	89.5	87.3	85.2
DT_8	91.5	89.7	88.1	91.3	89.5	81.1	91.9	90.2	78.8	91.1	89.2	84.3
DT_9	92.7	91.1	88.8	92.6	90.9	**81.9**	93.2	91.5	79.8	92.3	90.7	84.7
DT_10	93.8	91.8	88.6	93.8	91.6	81.6	94.1	92.0	79.5	93.6	91.5	84.6

Table 3: Metrics of the DT models.

Fourthly, we trained the kNN models (see Table 4), being the model which brings the best results the one having a $k = 4$. However, it is worth mentioning that all the trained kNN show quite good results, being, as expected, the type of model that presents the best metrics.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
kNN_3_unif	98.4	95.9	95.4	98.4	95.9	92.4	98.4	96.0	90.4	98.5	95.8	94.9
kNN_4_unif	96.9	95.6	96.9	96.9	95.5	**94.5**	97.1	95.9	94.8	96.7	95.3	94.2
kNN_5_unif	97.0	95.7	96.6	96.9	95.7	94.0	97.1	95.9	93.5	96.8	95.5	94.6
kNN_6_unif	96.6	94.8	96.6	96.6	94.7	94.0	96.8	95.2	93.5	96.4	94.5	94.6
kNN_7_unif	96.6	94.9	96.0	96.6	94.8	93.2	96.7	95.1	92.0	96.5	94.7	94.6
kNN_8_unif	95.8	94.6	96.7	95.8	94.5	94.1	96.1	95.0	94.3	95.6	94.2	93.9

Table 4: Metrics of the kNN models.

Finally, we used 3 ensembles (see Table 5).
As it is mentioned in the approach characteristics, the Hard Voting and Soft Voting classifiers use the 5 best models, which in this case are 5 kNN models with values $k = 3, 5, 4, 7, 6$.

Regarding the results obtained, Hard and Soft voting performance is similar, while Random Forest has much poorer metrics.

	Accuracy			**F-Score**			**Precision**			**Recall**		
ENSEMBLE	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
SoftVoting	97.1	95.7	96.4	97.1	95.6	93.8	97.2	95.8	93.0	97.0	95.5	94.8
HardVoting	97.0	95.6	96.6	96.9	95.5	**94.0**	97.1	95.8	93.5	96.8	95.4	94.6
RandomForest	86.6	83.6	91.6	86.1	82.4	85.3	88.9	87.2	85.4	85.5	82.2	85.3

Table 5: Metrics of the ensembles.

Comparing the best model of each technique (Table 9), the first noticeable thing is the low performance of the SVM model. Regarding the rest of the models, the ANN and the kNN are quite balanced, while the DT presents a worse performance.

The model that brings the best results is the kNN, followed closely by the ANN. Both models perform over 90% in all metrics and the kNN even presents metrics close to 95% or higher in some cases. The third model based on performance is the DT, which obtains good training and validation results but lags in test metrics.

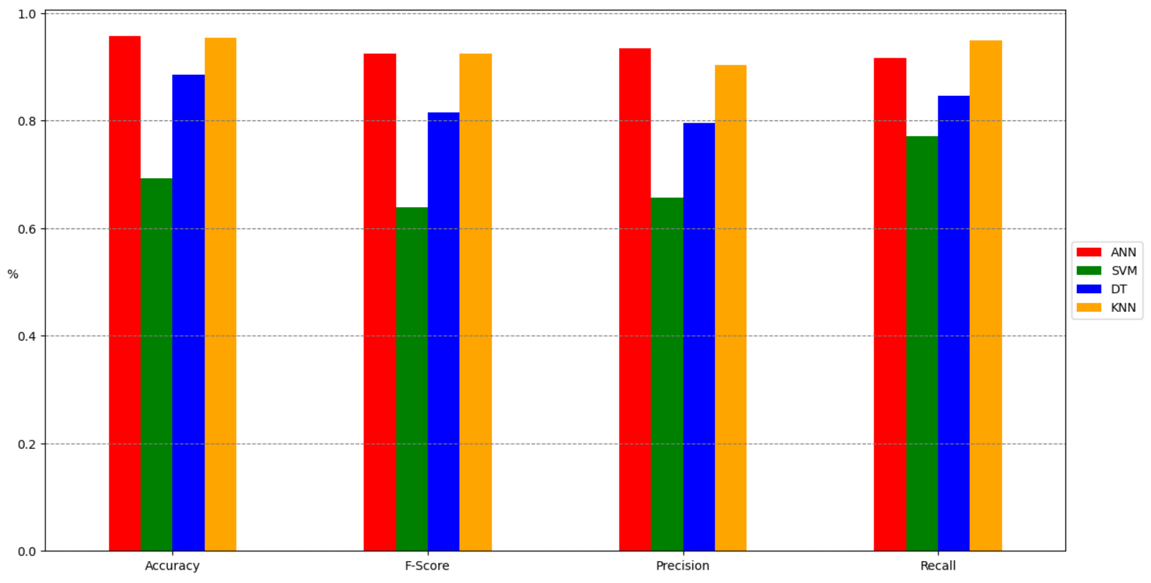

Figure 9: Comparison of the best models

In Table 10, the best model (kNN with $k = 3$) is compared to the ensembles. Once again, the Random Forest ensemble resembles the worst option, although it presents metrics that in most cases exceed 85%.

The other ensembles obtain pretty similar results to the best model.

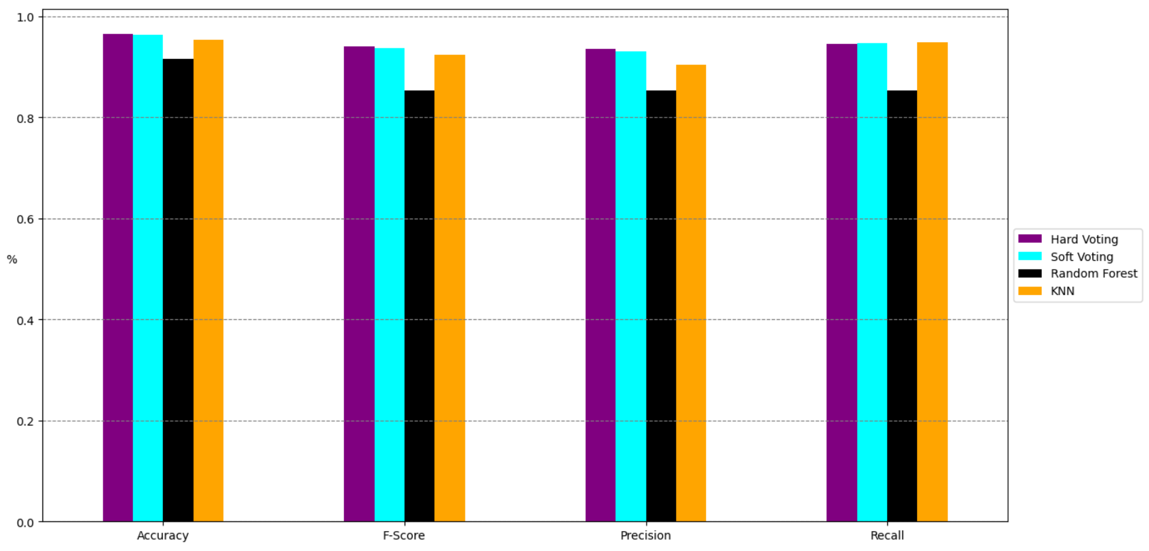

Figure 10: Comparison of ensembles and the best model (kNN)

2.2 Approach 2

2.2.1 Approach description

The main goal of this approach is to solve a 5-class classification problem. The dataset that is going to be used is the one described in Section 1.2, containing 109446 samples. We have 187 columns corresponding to the information offered by the electrocardiogram and the last column containing the class of each sample.

In this approach, we are going to proceed without using dimensionality reduction or k-fold cross-validation.

This approach has the following characteristics:

1. Deals with the multi-class problem.
2. This approach doesn't use all the normal samples, it just uses a 30% of them, the rest of the classes are multiplied by 2.5, 1.5, 5, 1.5 respectively.
3. Doesn't apply dimensionality reduction
4. Doesn't apply fold cross-validation.
5. Fits 8 ANN models with 1 or 2 layers.
6. Fits 8 SVM models, all of them linear, due to the big size of the dataset.
7. Fits 6 Decision Tree models with different maximum depths.
8. Fits 6 kNN models with different number of neighbors.
9. Fits 3 ensemble models:
    (a) A hard-voting classifier.

    (b) A soft-voting classifier.
    (c) A XGBoost classifier.

10. Also, the dataset has features with values between 0 and 1 with no outsider identified, so we have decided to normalize the dataset using a min-max approach instead of a zero-mean approach.

2.2.2 Results obtained

First of all, we trained the ANN models (Table: 6). As we mentioned before, there are eight different models with one or two layers, we can clearly see is that the one that gives us the best results is the one with 64 neurons.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
ANN_[4, 4]	85.2	85.0	90.6	81.3	81.3	69.4	86.8	86.7	69.8	78.5	78.5	76.3
ANN_[2, 2]	76.5	76.5	85.1	68.2	68.0	58.7	76.0	75.6	55.4	65.0	64.8	65.9
ANN_[16, 8]	92.8	91.9	94.9	90.6	89.4	79.3	93.7	92.2	76.1	88.2	87.0	84.2
ANN_[4]	85.8	85.7	91.7	81.5	81.7	71.2	87.0	86.9	68.3	77.9	78.2	76.7
ANN_[8]	88.5	87.6	91.3	85.3	83.8	70.3	88.1	86.5	65.3	83.3	81.9	80.9
ANN_[16]	92.7	90.8	93.9	90.5	88.2	75.8	91.8	89.6	71.6	89.4	87.1	85.7
ANN_[32]	95.3	93.1	94.8	93.9	91.4	78.9	94.7	92.4	74.1	93.1	90.6	87.9
ANN_[64]	96.8	94.7	95.0	96.0	93.4	**80.8**	96.3	93.5	76.1	95.7	93.4	89.0

Table 6: Metrics of ANN

After training the ANN models, we did the same with the SVM models (Table: 7), we can see that all of them show similar performance, being the model with C = 4.0 the one which brings the best results.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
SVM_lin_0.5	79.6	78.8	80.6	76.0	75.0	**56.9**	75.5	74.5	52.2	77.5	76.4	75.9
SVM_lin_1.0	79.5	78.9	80.5	76.0	75.0	56.6	75.5	74.2	51.9	77.4	76.6	75.8
SVM_lin_1.5	79.5	78.9	80.2	75.9	75.2	56.5	75.3	74.6	51.9	77.4	77.0	75.6
SVM_lin_2.0	79.4	78.9	80.4	75.7	75.5	56.5	75.1	74.9	51.8	77.4	76.7	75.6
SVM_lin_2.5	79.5	78.9	80.5	75.9	75.4	56.8	75.4	74.9	52.1	77.4	76.9	76.0
SVM_lin_3.0	79.4	79.3	80.0	75.7	75.6	56.1	75.2	75.1	51.4	77.3	77.3	75.5
SVM_lin_3.5	79.5	78.8	80.5	75.9	74.8	56.6	75.5	74.4	52.1	77.5	77.4	75.5
SVM_lin_4.0	79.8	79.0	80.4	76.2	75.2	56.5	75.7	74.8	51.9	77.7	76.5	75.7

Table 7: Metrics of the SVM models.

The next step was training the DT models (Table: 8), in this case, we can observe that they offer quite different results being the one that uses depth = 8 the best of all of them. It's not hard to conclude that exists a correlation between complexity and performance.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
DT_3	72.1	71.9	87.0	54.9	54.5	52.5	60.1	60.0	52.7	53.0	52.8	53.0
DT_4	77.3	76.8	84.9	69.7	69.1	58.2	76.0	75.0	57.6	67.3	67.1	66.2
DT_5	83.3	83.2	89.5	77.8	77.7	65.3	82.7	82.8	64.2	75.3	75.3	74.2
DT_6	86.7	86.2	91.7	82.7	82.1	69.8	87.1	86.6	67.7	79.6	79.0	77.6
DT_7	88.5	87.6	92.3	85.2	84.1	72.8	89.7	88.1	69.7	82.3	81.3	79.9
DT_8	90.1	88.5	93.5	87.5	85.4	**75.1**	92.3	89.9	73.2	84.1	82.2	80.1

Table 8: Metrics of the DT models.

In Table 9, the results obtained for the kNN models can be seen.
The one that gives the best results is the one with k=4.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
kNN_3	97.4	94.5	95.3	97.0	93.3	80.9	96.8	93.5	76.2	97.3	93.2	89.0
kNN_4	96.3	94.0	96.0	95.6	**92.7**	81.9	96.5	93.5	78.6	94.8	92.0	87.8
kNN_5	95.8	93.6	95.3	94.8	92.2	79.9	95.3	92.6	75.6	94.5	92.0	88.7
kNN_6	95.0	92.9	95.6	93.9	91.0	80.5	95.0	92.4	77.0	92.9	89.9	87.2
kNN_7	94.6	93.0	95.4	93.3	91.3	80.1	94.1	92.7	76.1	92.7	90.1	88.2
kNN_8	94.0	92.8	95.6	92.5	91.0	80.8	94.2	92.9	77.4	91.0	89.4	87.2

Table 9: Metrics of the KNN models.

Lastly, we used 3 ensembles (Table: 10). The performance of Hard and Soft voting is not exactly the same but we can say that they are similar, while XGBoost is a lot better in every aspect.

	Accuracy			F-Score			Precision			Recall		
ENSEMBLE	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
XGBoost	100	97.8	97.6	100	97.5	**89.2**	100	97.7	88.7	100	97.2	89.7
HardVoting	96.4	94.2	96.0	95.7	92.7	82.4	96.4	93.9	78.9	95.1	91.7	88.6
SoftVoting	97.2	94.5	96.0	96.6	93.3	82.2	96.7	93.6	78.0	96.5	93.1	89.7

Table 10: Metrics of the ensembles.

Comparing the best model of each technique (Figure: 11), the first noticeable thing is the low performance of the SVM model. In the other models, there was a relation between complexity and performance, so, as we were unable to use another SVM than linear, the results aren't as good as other models.

The model that brings overall better results is the kNN4, followed closely by the ANN. Both models perform over 80% in three of the four metrics, and over 95% in accuracy. The third model based on performance is the DT, which obtains good accuracy results, but stands a bit behind in the other metrics.

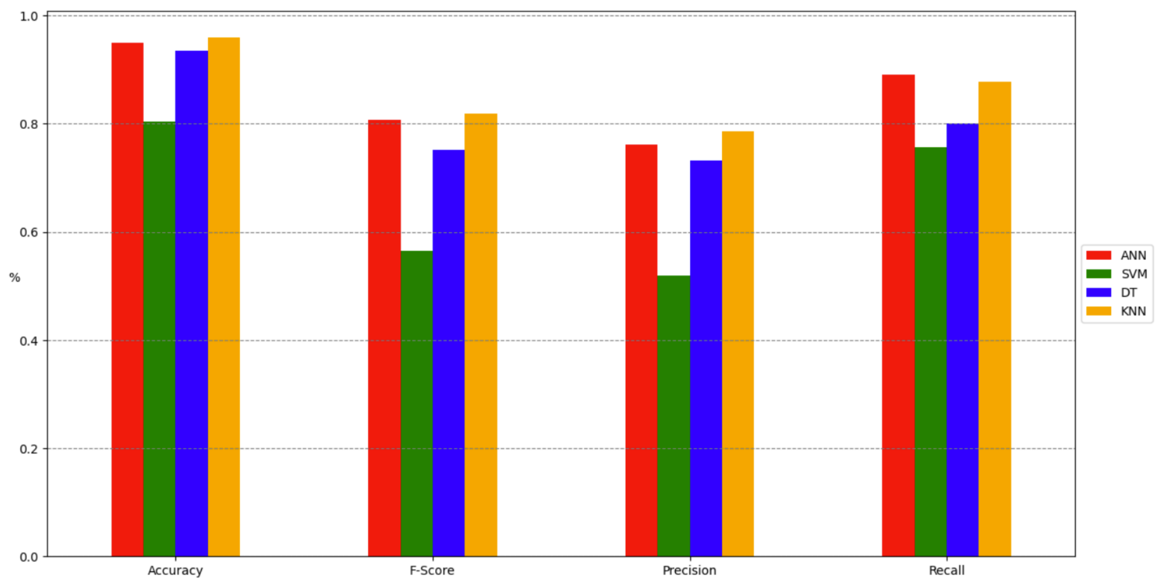

Figure 11: Comparison of the best models

Comparing the best model, kNN4, with the ensembles (Figure: 12), we can see that the hard voting ensemble is the one with poorer behaviour but not far from the rest, while the XGBoost presents the best results.

Regarding the recall metrics we can see that all of them perform almost the same. Also, if we look into the accuracy, we can see that all of them perform over 95%.

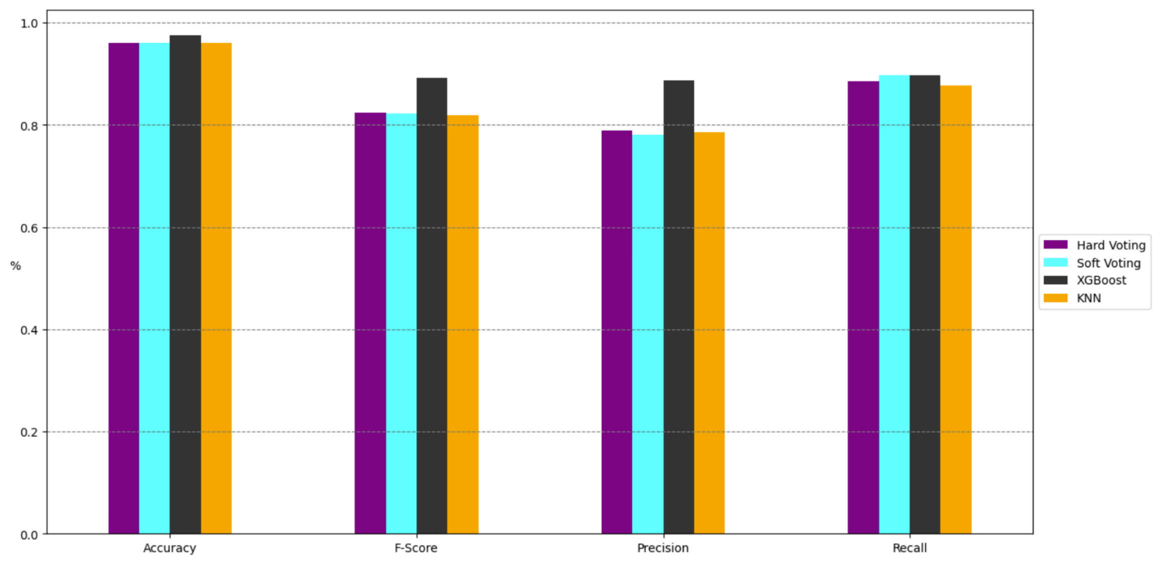

Figure 12: Comparison of ensembles and the best model (kNN)

2.3 Approach 2b

2.3.1 Approach description

In this approach, the problem to be solved is a 5-class classification problem. To do so, the dataset used is the one described in Section 1.2. As discussed in said section, this dataset contains 109446 samples, each of which is a heartbeat from a patient. Each of these features has 187 columns of data corresponding to the electrocardiogram beat and 1 column that contains the label of the class to which it belongs.

The main goal of this approach is to test how well the models would perform using PCA as dimensionality reduction, so for this comparison, we will use the same models as in the previous approach.

All dataset features has values between 0 and 1 and we have no detected any outsider value in the features, so we have decided to normalize the dataset using a min-max approach.
This approach has the following characteristics:

1. Deals with the multi-class problem.
2. This approach doesn't use all the normal samples, it just uses a 30% of them, the rest of the classes are multiplied by 2.5, 1.5, 5, 1.5 respectively.
3. Applies a dimensionality reduction using the PCA technique, halving the number of features.
4. Doesn't apply fold cross-validation.
5. Fits 8 ANN models with 1 or 2 layers.
6. Fits 8 SVM models, all of them linear, due to the big size of the dataset.
7. Fits 6 Decision Tree models with different maximum depths.
8. Fits 6 kNN models with different number of neighbors.
9. Fits 3 ensemble models:
   (a) A hard-voting classifier.
   (b) A soft-voting classifier.
   (c) A XGBoost classifier.

2.3.2 Results obtained

Firstly, we trained the ANN models (Table: 11). In this case, there is not a clear winner, as several models perform better on different metrics: on accuracy, the model with two layers, 16 and 8 neurons on each one; on f-score, the model with one layer and 64 neurons; on precision, the model with one layer and 16 neurons; and on recall, the model with one layer and 64 neurons. So taking this into account, we rank as the best model this last one, because our main target is the f-score metric.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
ANN_[4, 4]	82.5	81.9	86.6	77.5	76.4	61.3	82.6	81.5	56.7	74.0	72.9	71.8
ANN_[2, 2]	71.9	71.6	80.5	50.0	50.2	41.1	61.3	60.5	48.1	52.0	52.1	49.7
ANN_[16, 8]	87.5	87.2	92.7	83.8	83.0	73.2	90.3	89.9	71.9	79.4	78.6	75.5
ANN_[4]	80.8	81.2	86.8	75.5	75.4	63.0	82.1	81.5	59.9	71.5	71.7	67.4
ANN_[8]	85.9	85.5	89.2	81.7	81.3	66.4	86.4	85.8	62.0	78.8	78.4	75.8
ANN_[16]	87.9	87.9	91.2	84.1	83.8	71.8	90.6	89.8	72.9	79.6	79.6	72.9
ANN_[32]	89.3	88.1	92.1	86.0	84.4	73.6	90.1	88.5	70.7	83.0	81.4	78.2
ANN_[64]	90.0	89.7	90.5	87.0	86.7	**73.9**	90.3	89.9	70.1	84.8	84.6	79.9

Table 11: Metrics of ANN

Secondly, we trained the SVM models (Table: 12). All models present a similar performance, being the model with C = 0.5 the one which brings the best results on each metric.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
SVM_lin_0.5	78.0	77.5	66.7	74.3	74.1	**44.7**	73.3	72.6	39.7	76.2	76.2	73.9
SVM_lin_1.0	78.1	77.8	65.2	74.4	74.0	43.5	73.3	73.0	39.0	76.3	75.9	72.4
SVM_lin_1.5	78.2	77.8	64.0	74.5	74.2	43.0	73.4	73.0	38.7	76.2	76.3	73.0
SVM_lin_2.0	78.1	77.2	61.0	74.4	73.3	41.3	73.3	72.0	37.7	76.3	75.4	71.3
SVM_lin_2.5	78.0	77.6	65.1	74.3	73.9	43.6	73.3	72.7	39.1	76.2	75.9	72.4
SVM_lin_3.0	78.0	78.0	64.7	74.4	74.3	43.1	73.2	73.2	38.8	76.2	76.2	72.4
SVM_lin_3.5	78.2	77.8	63.7	74.5	74.3	43.0	73.4	73.1	38.7	76.3	76.1	72.4
SVM_lin_4.0	78.0	77.6	62.2	74.3	73.7	41.9	73.2	72.6	38.0	76.3	75.4	72.2

Table 12: Metrics of the SVM models.

Thirdly, we trained the DT models (Table: 13), being the model which brings the best results the one having a depth = 8.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
DT_3	66.9	66.6	80.9	50.1	50.0	43.1	53.4	53.2	46.9	52.4	52.5	52.6
DT_4	73.2	72.7	77.8	63.7	62.6	47.0	71.6	69.7	48.1	61.2	60.5	60.0
DT_5	80.4	79.8	79.5	75.2	74.4	51.9	80.3	79.5	49.2	71.5	70.7	67.2
DT_6	83.9	83.2	86.6	79.2	78.1	59.6	85.8	84.6	59.2	74.8	73.7	69.1
DT_7	86.4	85.1	85.9	82.9	81.1	60.1	88.2	86.3	58.4	79.0	77.4	69.6
DT_8	88.2	85.8	87.5	85.1	82.2	**62.1**	90.8	87.4	61.5	81.1	78.4	70.9

Table 13: Metrics of the DT models.

Fourthly, we trained the kNN models (Table: 19), being the model which brings the best results the one having a k = 4, in all but precision and recall, where it is outperformed by k = 6 and k = 7 respectively.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
kNN_3	96.8	93.3	93.7	96.3	92.0	76.6	96.2	92.6	72.0	96.4	91.7	86.4
kNN_4	95.2	92.2	94.6	94.3	90.3	**77.9**	95.6	91.5	74.8	93.2	89.5	85.0
kNN_5	94.9	92.0	94.0	93.8	90.1	76.6	94.5	91.0	72.8	93.3	89.4	85.6
kNN_6	93.9	91.9	94.5	92.5	90.0	77.8	94.1	92.0	75.1	91.2	88.3	84.9
kNN_7	93.7	91.8	93.7	92.4	89.5	75.6	93.4	91.1	72.0	91.6	88.4	85.5
kNN_8	92.7	90.8	94.3	90.8	88.6	77.1	93.1	91.0	74.6	89.0	86.6	84.4

Table 14: Metrics of the kNN models.

Lastly, we used 3 ensembles (Table: 15). The performance of Hard and Soft voting is similar, while XGBoost is better in everything but recall.

	Accuracy			F-Score			Precision			Recall		
ENSEMBLE	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
XGBoost	100	96.4	95.3	100	96.0	**81.8**	100	96.7	81.7	100	95.4	82.2
HardVoting	94.8	92.8	94.4	93.8	91.4	77.6	94.7	92.4	74.0	93.1	90.6	86.0
SoftVoting	95.3	92.6	94.2	94.3	90.9	77.3	94.8	91.6	73.3	93.9	90.7	86.5

Table 15: Metrics of the ensembles.

Comparing the best model of each technique (Figure: **??**), the first noticeable thing is the low performance of the SVM model.

The model that brings overall better results is the kNN, followed closely by the ANN. The third model based on performance is the DT, which obtains good results on accuracy, but stands behind in the other metrics.

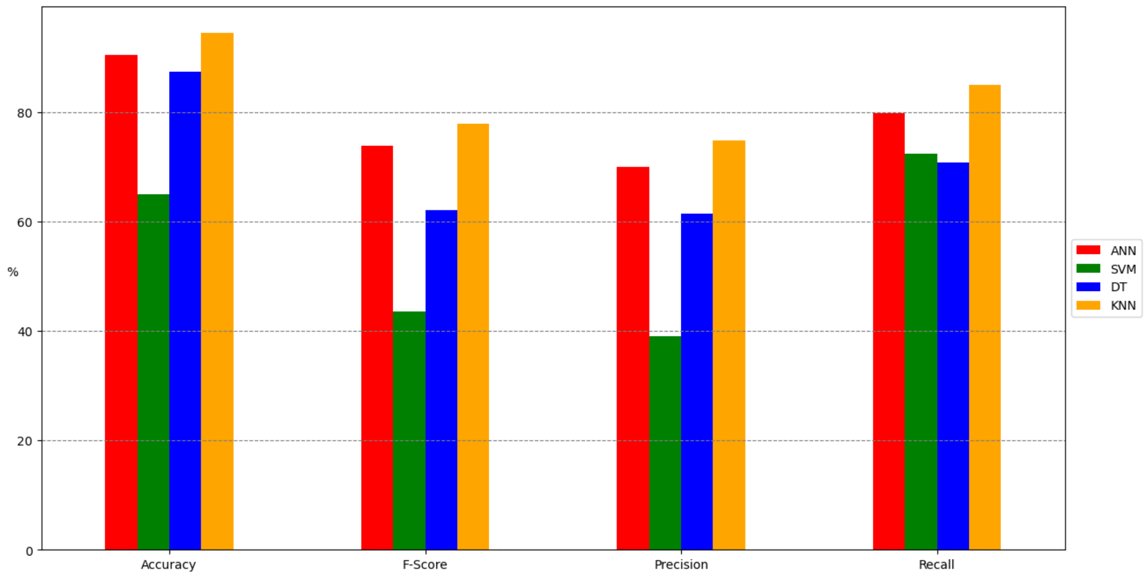

Figure 13: Comparison of the best models

Comparing the best model, kNN with k = 4, with the ensembles (Figure: 14), they all present similar accuracy. The XGBoost stands out from the others in f-score and precision but falls behind at recall.

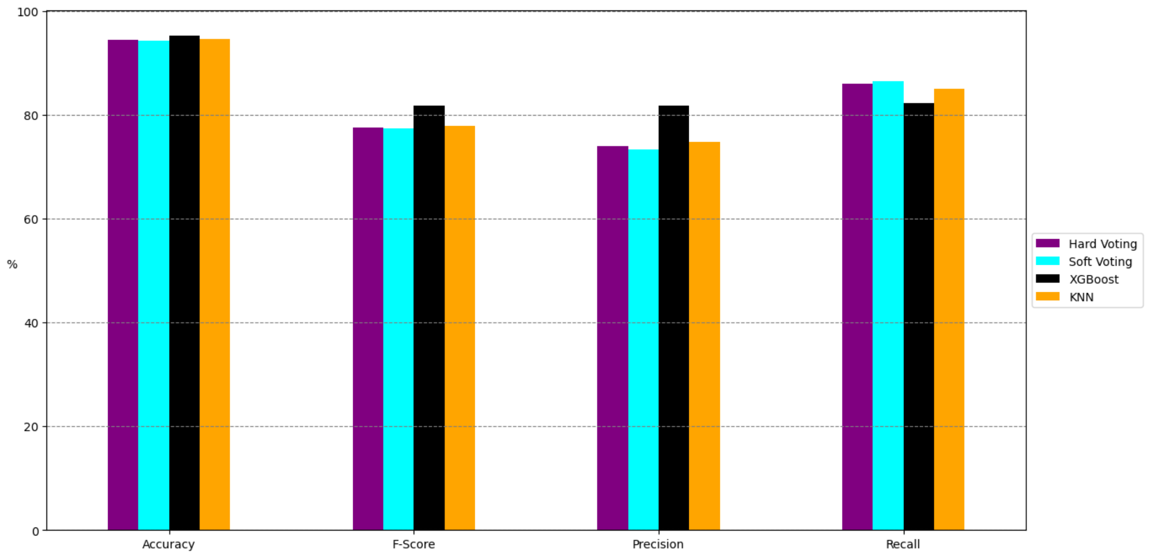

Figure 14: Comparison of ensembles and the best model (kNN)

2.4 Approach 3

2.4.1 Approach description

In this approach, the problem to be solved is a 5-class classification problem. To do so, the dataset used is the one described in Section 1.2. As mentioned in said section, this dataset contains 109446 samples, each of which is a heartbeat from a patient. Each of these features has 187 columns of data corresponding to the electrocardiogram beat and 1 column that contains the label of the class to which it belongs.

After running approach 2b (Section 2.3) and seeing the results, dimensionality reduction is ignored in this new approach, as it seems to worsen the performance of the models. The main goal of this approach is to test how well the models will perform using cross-validation.

Since all the feature values are between 0 and 1 and we have not detected any outsider values, we performed a min-max normalization.

This approach has the following characteristics:

1. Deals with the multi-class problem.
2. Uses all the normal samples and multiplies five times the samples of all the other classes.
3. Doesn't apply dimensionality reduction
4. Applies 5-fold cross-validation to select the best hyperparameters for the models.

5. Fits 8 ANN models with 1 or 2 layers.
6. Fits 8 SVM models, all of them linear, due to the big size of the dataset.
7. Fits 6 Decision Tree models with different maximum depths.
8. Fits 6 kNN models with different number of neighbors.
9. Fits 3 ensemble models:
   (a) A hard-voting classifier using the best 5 models from cross-validation.
   (b) A soft-voting classifier using the best 5 models from cross-validation.
   (c) A random forest classifier

2.4.2 Results obtained

Firstly, we trained the ANN models (Table: 16), being the model which brings the best results the one having one layer with 32 neurons. As we can see in the graph, there is a correlation between complexity and performance.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
ANN_[4,2]	86.0	85.6	90.8	73.5	72.8	67.8	78.8	78.0	65.8	70.9	70.4	72.0
ANN_[4,4]	88.1	87.8	92.8	80.5	79.9	72.9	86.1	85.2	74.5	76.6	76.3	73.6
ANN_[6,6]	89.7	89.4	94.3	83.5	82.9	76.8	87.8	87.1	77.4	80.5	80.0	79.4
ANN_[8,8]	91.9	91.2	94.0	86.5	85.5	76.6	90.9	89.6	74.6	83.2	82.5	80.5
ANN_[16,16]	94.6	93.7	96.1	91.1	89.8	82.9	92.9	91.4	82.3	89.5	88.4	84.8
ANN_[8]	91.0	90.6	94.8	85.2	84.7	77.2	89.0	88.3	77.7	82.4	81.9	79.5
ANN_[12]	92.9	92.2	95.4	88.2	87.2	80.4	91.3	90.1	79.4	85.6	84.7	83.3
ANN_[32]	95.7	94.5	96.5	93.0	91.2	**84.0**	94.7	92.7	82.3	91.5	89.9	86.6

Table 16: Metrics of the ANN models.

Secondly, we trained the SVM models (Table: 17). All models present a similar performance, being the model with C = 20 the one which brings the best results in all but recall, where the best model has C = 10.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
SVM_lin_5 .	81.6	81.3	82.8	73.8	73.4	58.5	72.6	72.2	54.4	76.3	75.9	74.9
SVM_lin_0 .	81.6	81.3	82.8	73.8	73.4	58.6	72.7	72.2	54.4	76.4	76.0	75.0
SVM_lin_5 .	81.6	81.3	83.0	73.9	73.4	58.8	72.8	72.4	54.7	76.4	76.0	74.9
SVM_lin_0 .	81.5	81.3	82.7	73.9	73.5	58.5	72.8	72.4	54.3	76.5	76.1	75.0
SVM_lin_10 .	81.0	80.7	83.4	72.1	71.6	59.4	71.6	71.0	55.0	74.3	74.0	75.1
SVM_lin_20 .	82.2	82.0	83.8	73.9	73.6	**59.6**	74.0	73.7	56.3	74.7	74.6	74.2
SVM_lin_50 .	80.3	80.0	76.0	70.2	69.6	50.9	73.1	72.3	52.0	70.6	70.1	69.3
SVM_lin_100 .	80.4	80.3	83.1	71.0	70.7	58.0	73.4	73.1	52.9	71.0	70.7	69.8

Table 17: Metrics of the SVM models.

Thirdly, we trained the DT models (Table: 18), being the model which brings the best results the one having a depth = 8. Again, there is a correlation between complexity and performance.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
DT_3	77.2	77.0	87.0	53.9	53.6	51.8	58.9	58.5	52.7	51.9	51.5	52.0
DT_4	80.6	80.3	84.6	68.5	68.2	58.3	73.9	73.6	60.2	67.3	66.8	67.6
DT_5	85.3	85.1	89.5	76.0	75.5	64.5	80.8	80.3	63.7	73.3	72.9	71.0
DT_6	88.6	88.0	93.1	80.8	79.9	71.9	88.3	87.5	73.0	75.9	74.9	74.6
DT_7	90.6	89.7	94.2	84.3	83.1	76.3	91.2	89.9	77.2	79.4	78.3	76.8
DT_8	91.9	90.8	94.6	86.7	85.1	**77.5**	92.9	91.0	78.1	82.1	80.7	78.6

Table 18: Metrics of the DT models.

Fourthly, we trained the kNN models (Table: 19), being the model which brings the best results the one having a k = 4, in all but recall, where it is surpassed by all models but the one with k = 8.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
kNN_3	98.0	95.8	96.6	97.2	93.8	84.7	97.0	94.1	81.5	97.4	93.6	89.2
kNN_4	97.1	95.8	97.1	95.6	93.7	**86.0**	97.0	95.0	85.1	94.5	92.7	87.7
kNN_5	96.8	95.6	96.9	95.1	93.4	84.8	95.8	94.0	82.9	94.5	93.0	88.2
kNN_6	96.3	95.3	96.8	94.4	92.8	84.6	95.8	94.1	83.1	93.4	91.8	87.8
kNN_7	96.0	95.0	96.4	93.9	92.3	83.0	94.7	93.2	80.4	93.5	91.9	88.4
kNN_8	95.3	94.6	96.5	92.5	91.3	83.1	95.0	93.9	81.8	90.3	89.2	87.2

Table 19: Metrics of the kNN models.

Lastly, we used 3 ensembles (Table: 20). The performance of Hard and Soft voting is similar, while RandomForest is worse in everything but precision. This ensemble also stands out for having a very low recall.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
HardVoting	96.8	95.6	96.9	95.1	93.3	**84.9**	96.0	94.2	83.2	94.4	92.8	88.3
SoftVoting	96.9	95.6	96.8	95.4	93.4	84.7	95.9	94.0	82.4	94.9	93.1	88.6
RandomForest	86.4	86.3	94.2	75.3	74.9	74.7	93.1	92.9	86.4	66.8	66.4	67.2

Table 20: Metrics of the ensembles.

Comparing the best model of each technique (Figure: 15), the first noticeable thing is the low performance of the SVM model. As we can see, in the other models there were a relation between complexity and performance, so, as we were unable to use another SVM than linear, the results aren't as good as other models.

The model that brings overall better results is the kNN, followed closely by the ANN. Both models perform over 80% in all metrics, and over 95% in accuracy. The third model based on performance is the DT, which obtains good results on accuracy, but stands behind in the other metrics.

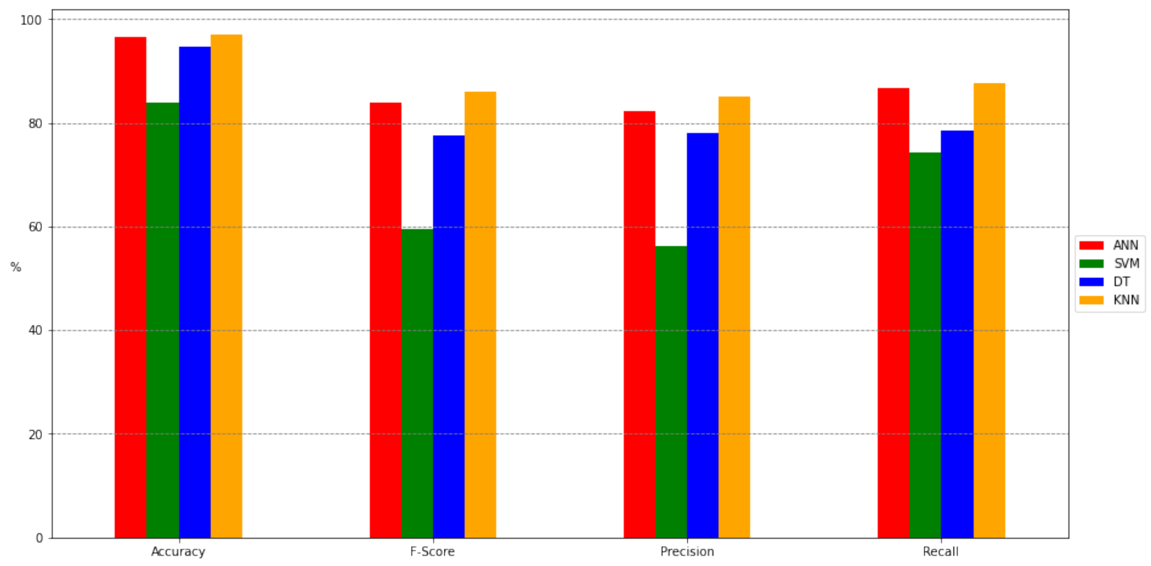

Figure 15: Comparison of the best models

Comparing the best model, kNN with k = 4, with the ensembles (Figure: 16), again, the Random Forest ensemble resembles the worst in everything but precision, where it obtains the best results.
The other ensembles obtain pretty similar results as the best model.

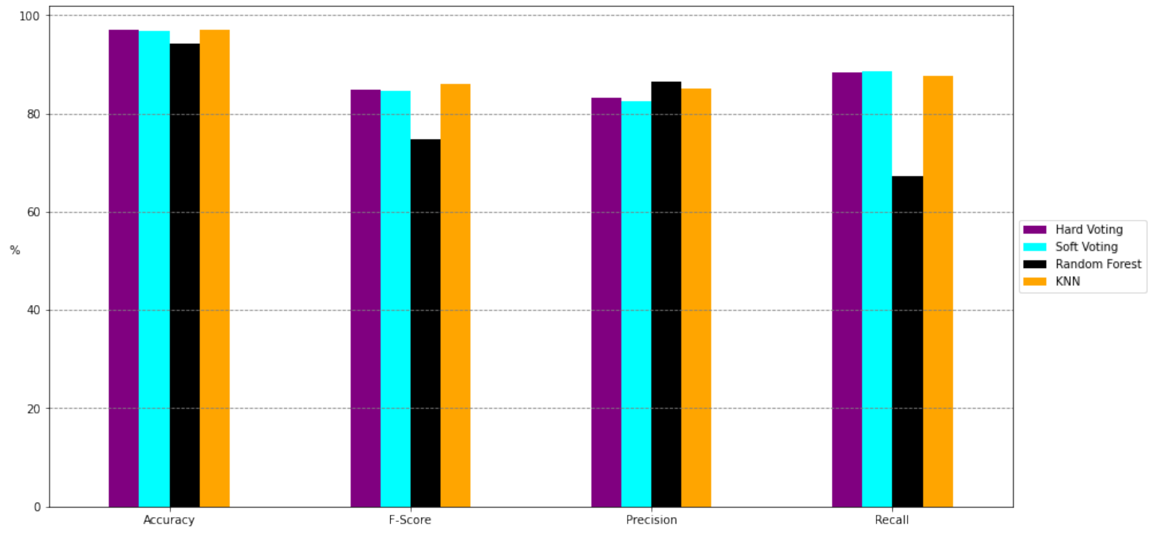

Figure 16: Comparison of ensembles and the best model (KNN)

2.5 Approach 4

2.5.1 Approach description

In this approach, the problem to be solved is the same 5-class classification and the dataset used is the one described in Section 1.2: each row represents an electrocardiogram beat with 187 features and one target feature.

As we mentioned in Section 1.2, we have detected that some columns - mainly at the end of the dataset- have values close to zero or zero. This suggests that a dimension reduction could be performed on the dataset before starting the process.

In this case, we apply a custom dimensional reduction by selecting the first 87 columns. During the dataset analysis, we saw that most of the columns at the end of the dataset have zero values, so it could be possible to reduce the number of features in the dataset without reducing the accuracy of the trained model.

We have trained a kNN with the original data set selecting only the first i columns, from 2 to 187 (excluding function 188 that corresponds to the target) and we have detected that the best accuracy and f1-score metrics correspond to the model trained with the first 87 features. Also, the dataset has features with values between 0 and 1 with no outsider identified, so we have decided to normalize the dataset using a min-max approach instead of a zero-mean approach.

This approach has the following characteristics:

1. Deals with the multi-class problem.
2. Multiplies for 2 the classes with less number of instances and multiplies by 0.5 the class with the higher number of instances.
3. Applies custom dimensional reduction, in this case selecting the first 87 columns.
4. Applies 10-fold cross-validation to select the best hyperparameters for the models.
5. Fits 8 ANN models with 1 or 2 layers.
6. Fits 8 SVM models, with sigmoid and rbf kernel and with iteration limit due to the big size of the dataset.
7. Fits 6 Decision Tree models with different maximum depths.
8. Fits 6 kNN models with different number of neighbors and weights.

   Scikit kNN model has two kinds of values for hyperparameter 'weight': uniform (all points in each neighborhood are weighted equally) or distance (weight points by the inverse of their distance). In this approach, we have chosen the same neighbors with different weights for comparing metrics between them.
9. Fits 2 ensemble models:
   (a) A SoftVoting ensemble using the best 4 models from cross-validation.
   (b) A XGBoost ensemble.

2.5.2 Results obtained

We are going to analyze the results obtained separately, all of them expressed in per hundred.

First, we are going to talk about the ANNS (see Table 21). We can check that the best f-score belongs to the ANN with one layer and 16 neurons, but the best accuracy belongs to ANN with two layers (16 and 8 neurons per layer). There is a correlation between the number of neurons and high accuracy / f-score.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
ANN_[16]	94.3	93.7	95.7	87.8	86.8	**81.7**	92.0	90.7	81.9	84.7	83.9	81.6
ANN_[16, 8]	95.1	94.4	95.8	89.4	88.3	81.5	92.8	91.5	79.6	86.9	85.9	84.8
ANN_[8, 8]	92.9	92.5	96.0	85.2	84.5	80.1	89.8	89.1	83.5	82.0	81.3	77.4
ANN_[8]	91.6	91.2	94.5	82.9	82.3	76.9	88.5	87.8	77.2	79.2	78.7	76.8
ANN_[6]	90.6	90.3	93.5	80.9	80.4	74.0	87.0	86.6	75.4	76.9	76.5	73.9
ANN_[4, 4]	88.6	88.3	92.0	75.4	74.8	71.0	83.2	82.7	69.7	71.9	71.3	73.3
ANN_[4]	86.8	86.6	92.0	74.9	74.5	67.3	82.5	82.0	71.7	71.1	70.8	67.6
ANN_[2, 2]	84.3	84.1	91.6	63.8	63.7	59.4	68.9	68.8	60.4	61.6	61.5	58.5

Table 21: Metrics of ANN

Secondly, we trained the kNN models (see table 22). We got the best metrics results in all models (without ensembles) with the kNN-4 with uniform weight, so selecting the weight as 'distance' does not improve the metrics. In this case, the overall metrics seem better for a lower value of neighbors.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
kNN_4_uni	97.6	96.8	97.7	94.4	92.7	**88.1**	97.5	95.1	89.4	91.9	90.7	87.0
kNN_4_dist	100.0	98.8	97.0	100.0	98.1	86.6	100.0	96.8	84.0	100.0	99.6	89.7
kNN_5_uni	97.4	96.8	97.3	94.3	93.3	87.4	96.0	94.8	86.6	92.7	91.9	88.2
kNN_5_dist	100.0	99.0	97.3	100.0	98.4	87.2	100.0	97.3	86.1	100.0	99.6	88.4
kNN_6_uni	97.2	96.3	97.4	93.9	92.0	87.6	96.3	94.9	87.0	91.8	89.5	88.1
kNN_6_dist	100.0	99.0	97.3	100.0	98.4	87.2	100.0	97.3	85.9	100.0	99.6	88.5

Table 22: Metrics of kNN

The Decision Tree models do not have better performances than kNN or ANN models. The best metric result, f-score and average (see table 23), belongs to a Decision Tree with a depth of 8. There seems to be also a correlation between the hyperparameter depth and high accuracy / f-score.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
DT_3	75.4	75.3	86.7	54.4	54.3	51.3	61.4	61.1	53.4	50.9	50.9	50.1
DT_4	80.3	80.2	86.5	61.1	60.8	54.1	76.8	74.0	58.0	58.7	58.6	57.8
DT_5	85.1	84.8	89.5	66.5	66.2	64.0	83.9	78.7	63.9	63.8	63.6	65.6
DT_6	88.3	88.0	93.9	70.0	69.4	70.0	87.9	83.4	75.8	65.8	65.4	66.6
DT_7	89.7	89.1	93.4	80.0	79.1	73.2	84.7	83.7	73.4	77.5	76.7	76.5
DT_8	91.6	90.9	94.2	84.1	83.0	**77.1**	89.2	87.9	78.2	80.5	79.5	77.5

Table 23: Metrics of DT

It should be noted that SVM performance, reflected in Table 24, is very bad because we are dealing with a more unbalanced dataset than in any other approach and, instead of using linear SVM we are using sigmoid and rbf SVM.

In addition, it would be needed to set a max number of epochs for the SVM models due to the dataset size. Something worth noting is that performance is much worse in sigmoid SVM than in rbf.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
SVM_sigm_1.5	23.8	23.8	41.2	10.6	10.6	13.1	11.4	11.6	16.8	20.8	20.8	16.5
SVM_sigm_2.0	22.9	22.9	43.5	9.8	9.9	13.5	10.3	10.7	16.8	19.5	19.6	16.4
SVM_sigm_1.5	6.6	6.5	2.6	5.4	5.2	2.0	6.5	5.2	1.5	23.4	23.1	23.5
SVM_sigm_2.0	6.6	6.4	2.5	5.5	5.3	1.8	6.2	6.4	2.7	23.4	23.2	20.9
SVM_rbf_1.5	34.6	34.5	20.5	36.5	36.5	25.8	48.2	48.2	38.6	66.8	66.7	66.2
SVM_rbf_2.0	38.4	38.5	20.6	39.5	39.7	25.7	50.4	50.7	33.3	65.9	66.0	62.1
SVM_rbf_1.5	42.9	42.8	17.4	46.0	45.9	29.0	57.6	57.6	41.2	77.2	77.1	75.3
SVM_rbf_2.0	45.9	45.9	17.3	49.1	49.1	**29.6**	59.1	59.1	42.2	79.6	79.5	75.1

Table 24: Metrics of SVM

Finally, comparing the different ensemble metrics, we deduce that the best ensemble classifier is XBoost with an accuracy of 97.9 and an 89.9 f-score (see table 25). The Softvoting ensemble was created with the four better models (kNN with 6 neighbors, kNN with 5 neighbors, kNN with 4 neighbors and kNN with 5 neighbors) and presents a somewhat worse performance.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
SoftVoting	100.0	99.0	97.3	100.0	98.4	87.2	100.0	97.3	86.2	100.0	99.6	88.4
XGBoost0	100.0	99.6	97.9	100.0	99.4	**89.9**	100.0	99.0	92.8	100.0	99.8	87.5

Table 25: Metrics of ensembles

Several models got an accuracy of 100 during training but this performance decreases on validation and test metrics.

The graph 17 represents the accuracy, f-score, precision and recall for best kNN, ANN, DT and SVM models. Graph 18 shows a comparison between the ensembles tested in this approach and the best model. Accuracy and recall values are very close but the XBoostD ensemble metrics are better on precision and f-score.

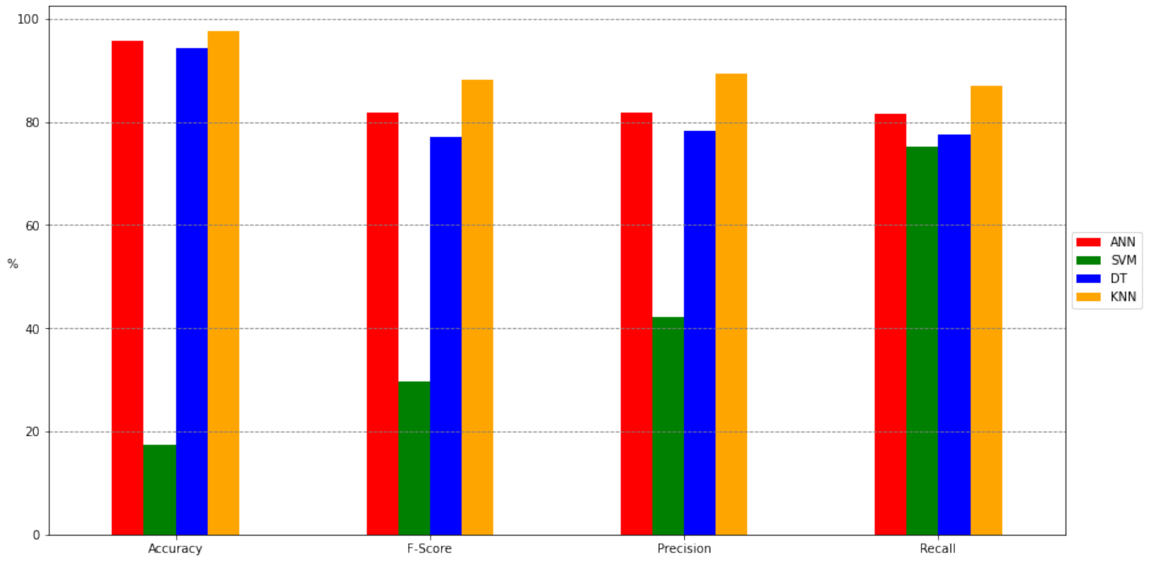

Figure 17: Comparison of the best models

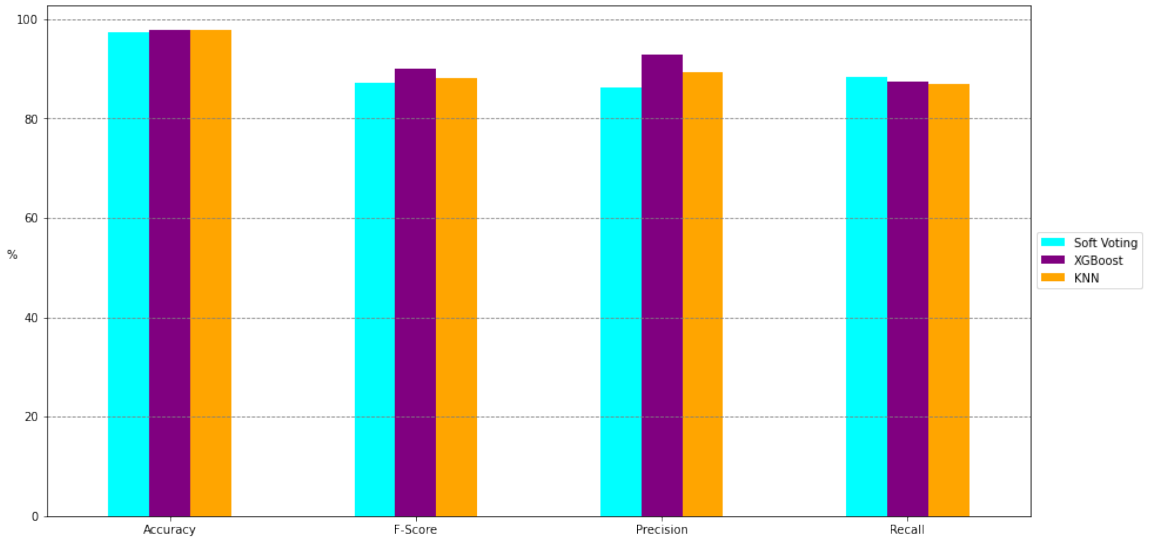

Figure 18: Comparison of ensembles and the best model (KNN)

2.6 Approach 5

2.6.1 Approach description

This approach has the following characteristics:

1. Deals with the multi-class problem.
2. Uses all the normal samples and multiplies five times the samples of all the other classes.
3. Applies a dimensionality reduction using the PCA technique, almost halving the number of components. By analysing the dataset we found that almost half of the features have a very small variance.
4. Applies 5-fold cross-validation to select the best hyper-parameters for the models.
5. Fits 8 ANN models with 1 or 2 layers.
6. The SVM models were trained and evaluated in approach 5b due to performance issues handling the entire dataset.
7. Fits 6 Decision Tree models with different maximum depths.
8. Fits 6 kNN models with different number of neighbors.
9. Fits 3 ensemble models:
   (a) A custom stacking model using binary and multi-class classifiers.
   (b) A hard-voting classifier using the best 5 models from cross-validation.
   (c) A Gradient Boosting ensemble model.
10. Dataset has features with values between 0 and 1 with no outsider identified, so we have decided to normalize the dataset using a min-max approach instead of a zero-mean approach.

The custom stacking model contains 2 classifiers: 1 binary classifier that determines if the sample is normal or not, and a multi-class classifier that, in case the sample is not normal, determines which class of condition it has. It is not a conventional stacking model since it does not have a classifier model to provide the output. The binary classifier is trained with the converted targets where all the non-normal are grouped together as the *positive* class. The multi-class classifier is trained only with the samples that are not normal.

The intention within this model is to split the responsibilities so that no single model has to deal with the imbalanced dataset issue. The binary model will have a more balanced distribution of negative/positive samples, while the multi-class model will have a much more balanced distribution of the different subtypes of positive classes. This is similar to performing a one-vs-all strategy with the binary model, and then using the multi-class classifier to resolve which of the "all" class is the appropriate one.

2.6.2 Results obtained

The best-performing models overall were the kNN models (Table: 27), with f-scores around 0.85, followed by the ensembles (Table: 29). The kNN models might perform a bit worse in terms of time complexity than the ensembles for the prediction of inputs, but the training and validation of the ensembles and their complexity, apart from the poorer classification metrics, would suggest not using the ensembles.

The best ANN (Table: 26) is the one with a single layer of 32 neurons, but it falls considerably behind the kNNs with an f-score of 0.73.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
ANN_[4,4]	84.0	84.0	48.6	74.4	74.4	30.7	78.8	78.9	31.4	72.4	72.4	55.0
ANN_[8,8]	90.4	90.3	90.8	85.0	85.0	69.9	88.1	88.0	65.0	83.1	83.0	81.7
ANN_[2,2]	77.7	77.7	75.6	56.8	56.8	47.0	60.7	60.7	43.3	57.3	57.3	59.6
ANN_[16,8]	93.1	93.0	87.8	88.6	88.4	66.1	90.3	90.1	63.2	87.5	87.4	74.8
ANN_[4]	84.0	84.0	64.3	75.7	75.7	46.2	80.8	80.7	41.2	73.6	73.6	71.0
ANN_[8]	89.4	89.3	88.4	83.8	83.8	68.1	86.3	86.3	62.7	82.4	82.4	81.5
ANN_[16]	92.1	92.1	85.7	87.3	87.3	65.8	89.8	89.7	59.3	85.7	85.6	78.8
ANN_[32]	94.6	94.5	92.2	90.5	90.4	**73.6**	92.9	92.8	67.8	88.8	88.8	85.1

Table 26: Approach 5: Metrics of ANN

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
kNN_2_uni	100.0	99.6	96.6	100.0	99.5	**85.1**	100.0	99.2	86.1	100.0	99.7	84.1
kNN_4_uni	99.6	99.2	96.0	99.4	99.0	83.3	99.2	98.5	81.2	99.7	99.5	85.7
kNN_6_uni	99.2	98.9	95.4	99.0	98.6	81.7	98.5	97.9	78.0	99.5	99.3	86.5
kNN_8_uni"	98.9	98.6	94.7	98.6	98.2	80.3	97.9	97.3	75.5	99.3	99.1	86.9
kNN_10_uni	98.6	98.3	94.2	98.2	97.8	79.3	97.3	96.8	73.8	99.1	99.0	87.3
kNN_12_uni	98.3	98.1	93.6	97.8	97.5	78.1	96.8	96.3	72.0	99.0	98.8	87.7

Table 27: Approach 5: Metrics of kNN

The decision tree models (Table: 28) perform very badly, with f-scores below 0.5, and also very low metrics of precision, recall and accuracy.

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
DT_2	57.7	57.7	36.0	37.5	37.5	20.9	36.5	36.5	25.7	39.6	39.6	36.1
DT_4	69.3	69.3	55.3	45.2	45.2	29.4	48.5	48.4	29.9	47.5	47.5	41.1
DT_6	78.6	78.5	54.8	71.6	71.6	36.1	73.0	72.9	35.5	71.9	71.8	59.3
DT_8	86.3	86.2	72.4	81.2	81.1	**47.2**	84.0	83.9	42.2	79.3	79.2	65.5
DT_10	90.4	90.2	72.8	86.6	86.4	44.5	89.5	89.1	40.3	84.5	84.4	57.4
DT_12	93.7	93.2	74.4	91.3	90.7	44.1	93.5	92.6	39.7	89.5	89.2	55.9

Table 28: Approach 5: Metrics of DT

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
Custom	99.2	98.9	95.4	99.0	98.6	**81.7**	98.5	97.9	78.0	99.5	99.3	86.5
HardVoting	99.2	98.9	95.4	99.0	98.6	81.7	98.5	97.9	78.0	99.5	99.3	86.5
GradientBoosting	80.6	80.5	68.8	75.3	75.2	44.7	84.4	84.2	48.8	71.0	71.0	56.8

Table 29: Approach 5: Metrics of ensembles

The HardVoting model used the 5 best cross-validation models: `kNN_2_uni_0`, `kNN_4_uni_1`, `kNN_6_uni_2`, `Custom` and `kNN_8_uni_4`.

Here is a comparison of the best-performing base models (Figure: 19), led by far by the kNN models, which show an overall good f-score, precision and recall, for relatively low complexity.

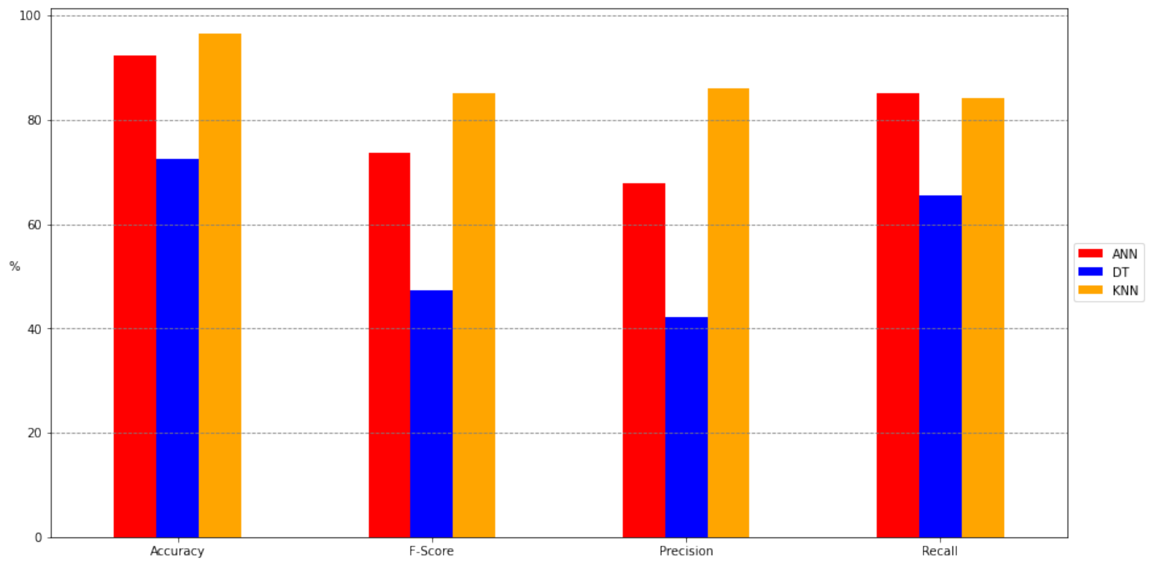

Figure 19: Comparison of the best models

Here is a comparison of the best-performing base model, the kNN, with the ensembles (Figure: 20).

It seems clear that the kNN outperforms all the ensembles - except for a little bit smaller recall - while also being a much simple classifier. According to these metrics, it would be the chosen model. The cross-validation of the kNNs with such a large dataset takes a long time, but some of the ensembles took even longer for training and validation.

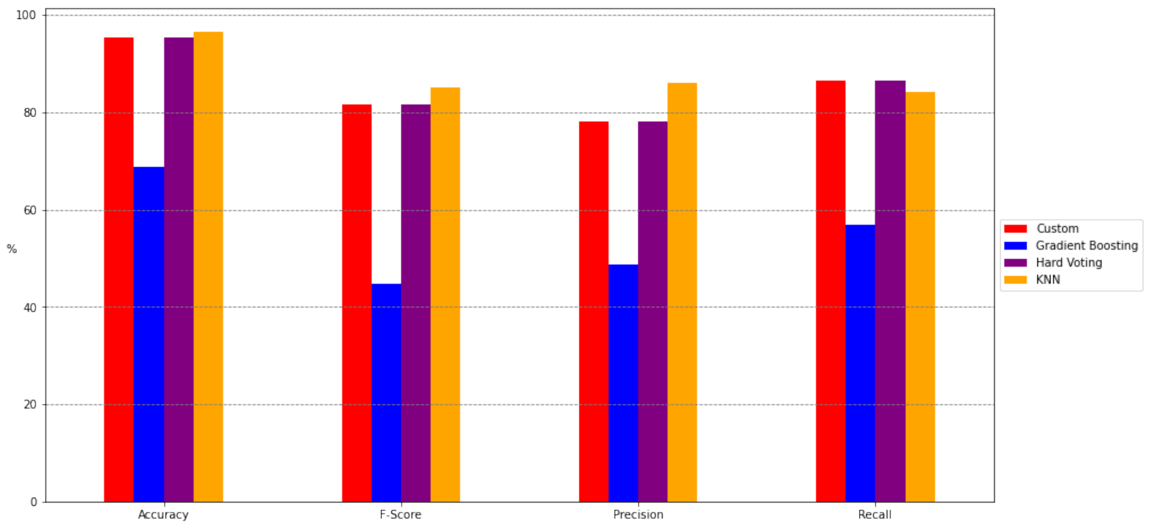

Figure 20: Comparison of ensembles and the best model (KNN)

2.7 Approach 5b

This approach variant has the following characteristics:

1. Deals with the multi-class problem.
2. Uses half of the normal samples and multiplies five times the samples of all the other classes.
3. Applies a dimensionality reduction using the PCA technique, almost halving the number of components. By analyzing the dataset, we found that almost half of the features have a very small variance.
4. Applies holdout validation with 20% of the training set, to improve performance.
5. Fits 8 SVM models, some of them linear and the rest wrapped in a bagging classifier to improve efficiency. Also, the maximum number of iterations was limited in all cases.
    - 2 linear SVMs with different regularization parameters.
    - 2 bagged SVMs with a sigmoid kernel and different C and gamma parameters.
    - 4 bagged SVMs with RBF kernel with different C and gamma parameters.
6. The dataset has features with values between 0 and 1, with no outsider identified, so we have decided to normalize the dataset using a min-max approach instead of a zero-mean approach.

The SVM training complexity is quadratic on the number of samples, that's why, considering the large dataset we were using, we had to explore different techniques to improve the training efficiency. In this case, we tried to use a linear kernel, which is much more efficient but does not perform as well as other non-linear kernels.

Another technique we applied here was creating an ensemble model wrapping the SVM in a bagging classifier. This way, each internal SVM model is trained with a subset of the training set, with 10% of the number of samples. Due to the quadratic complexity of the number of samples, it performs much better than 1 single classifier with all the training set.

Finally, we also limited the number of iterations for the SVM. But, in any case, the metrics obtained for these models are the worst of all (Table: 30).

	Accuracy			F-Score			Precision			Recall		
MODEL	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**	**fit**	**val**	**test**
SVM_lin_1.5	32.4	32.8	82.8	9.8	9.9	**18.1**	6.5	6.6	16.6	20.0	20.0	20.0
SVM_lin_2.0	32.5	32.4	82.8	9.8	9.8	18.1	6.5	6.5	16.6	20.0	20.0	20.0
*SVM_sig_1.5	2.9	2.8	0.7	1.1	1.1	0.3	0.6	0.6	0.1	20.0	20.0	20.0
*SVM_sig_2.0	2.8	3.0	0.7	1.1	1.2	0.3	0.6	0.6	0.1	20.0	20.0	20.0
*SVM_rbf_1.5	12.7	12.3	3.2	10.4	10.3	3.5	22.6	21.0	2.2	38.9	38.7	38.5
*SVM_rbf_2.0	13.9	14.3	6.8	13.4	13.6	5.6	41.5	41.3	17.9	38.8	38.9	36.6
*SVM_rbf_1.5	13.4	13.5	3.2	13.6	13.5	4.5	36.4	36.2	3.1	39.4	39.2	38.1
*SVM_rbf_2.0	46.5	46.6	6.9	45.6	45.1	18.0	67.2	66.6	20.7	62.9	62.7	47.4

Table 30: Approach 5b: Metrics of SVM

The SVM with an * are the ones that were wrapped using a bagging classifier.

3 Final discussion

We come here to the last part of this report. After everything we have seen previously, the work done and the results obtained, we can elicit some thoughts and some conclusions.

The first thing to take into account is that having five different approaches and using five different ensembles allows us to make some comparisons, not just about their results but also about the time they take to run or the difficulty they imply.

Regarding this project's results, we can clearly see that in all of our approaches the models that get the worst results are by far the SVM. The main reason for this is that, due to the data, we were unable to use other than linear SVM in our computers, which is proved to give pretty bad results compared with other SVMs when dealing with ECGs (Martis et al) [2]. The exception is approach 4, where the results are even worse because we had to limit the number of epochs and reduce the class multiplier configuration.

In the same way that we can deduce that the worst model has been the SVM, we can also see that, in general, the best model has been the kNN. In all the approaches, the best model has been a kNN, except in approach 2, where the best one is an ANN - still closely followed by the kNN. This leads us to think that it is the best model to deal with this type of problem using the resources we have.

As far as ensembles are concerned, we tried five different ones: HardVoting, SoftVoting, XGBoost, RandomForest and GradientBoosting. Even though the HardVoting and SoftVoting ensembles made a really good job, it was the XGBoost that showed the best performance in the approaches in which it was used. Also, the RandomForest showed the worst results.
The GradientBoosting ensemble was a bit of a failure because it offered even worse results than the RandomForest.

We also decided to do two very similar approaches (approach 2 and approach 2b) to see the difference obtained by applying dimensionality reduction using PCA and without applying it. The conclusion is that, by using PCA, the results obtained are slightly worse.

Comparing the best model of each approach (Figure: 21), the best results are obtained in approach 1. The main reason for this is that this approach deals with a simplification of the problem, focusing on classifying only into two classes.

Aside from this exception, the best results are obtained by the fourth approach, the one which only takes into account the first 87 columns. We can then conclude that the most valuable information is stored there. This information could be very relevant to projects that try to develop a real-time system for evaluating an ECG from a real patient, such as Rodriguez et al. 2005 [4], where it is mentioned that only anomalous signals are processed in real-time and not whole ECG signals.

Lastly, approach 2b brings the worse results for the reasons mentioned before.

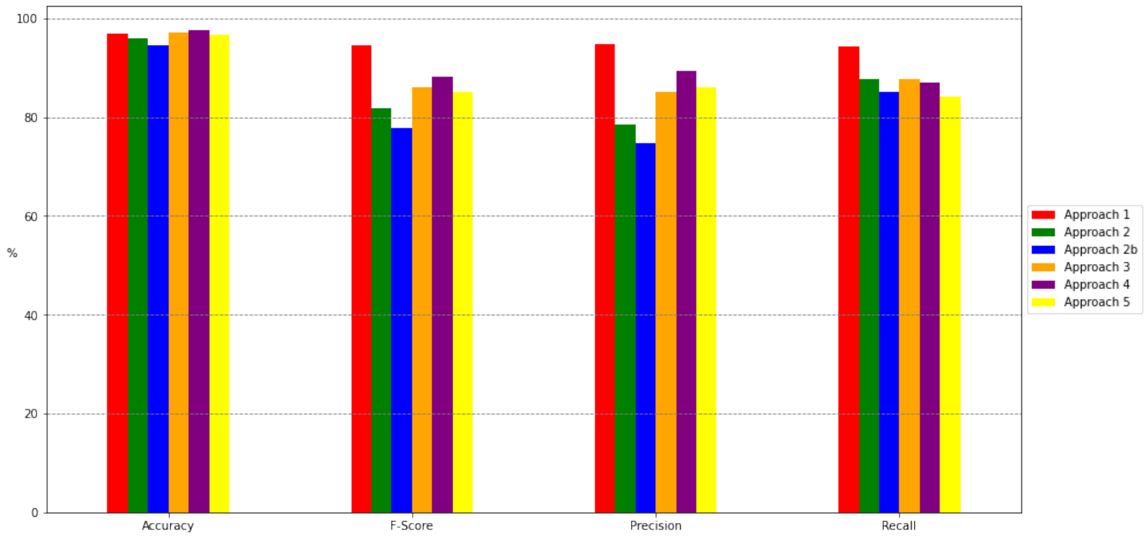

Figure 21: Comparison of the best model of each approach

We can also compare our results with those of the literature mentioned earlier in this paper. These are a bit worse, but they don't stray too far, which makes us think that - taking into account the resources that we have - our program is quite good.

When it comes to execution time, there are huge differences among approaches. For example, approach 2 takes about fifteen minutes to end, while approaches 4 and 5 can last hours running. This is due to the fold cross-validation, so depending on what we are looking for, one is gonna be better for our purpose than the others.

3.1 Other work

Jupyter notebook to work with the Julia models in Google Collab. Seeing the long runtimes, we tried to make the program work in Google by cloning and adapting this Jupyter notebook `https://github.com/Dsantra92/Julia-on-Colab`, but, after managing to make it run, a runtime error happened. After that, we finally ceased our efforts due to the upcoming deadline. See the notebook in the code attached to the submission.

3.2 Future work

1. Try other techniques and hyperparameters to improve generalization.
2. Obtain other similar datasets or more samples for the positive classes.
3. Improve the stacking model based on a one-vs-all approach for the normal vs. non-normal and other models to detect each of the positive classes.
4. Explore other transformations of the input data, such as transforming the ECG wave to the domain of frequency using the Discrete Fourier Transform.
5. Use ECG data with more than one heartbeat.
6. Find a dataset with a patient identifier to perform cross-validation with data from multiple patients.

References

[1] Varun Gupta et al. "A review of different ECG classification/detection techniques for improved medical applications". In: *International Journal of System Assurance Engineering and Management* (2022), pp. 1–15.

[2] Roshan Joy Martis, U Rajendra Acharya, and Lim Choo Min. "ECG beat classification using PCA, LDA, ICA and discrete wavelet transform". In: *Biomedical Signal Processing and Control* 8.5 (2013), pp. 437–448.

[3] Saeed Mian Qaisar and Syed Fawad Hussain. "An effective arrhythmia classification via ECG signal subsampling and mutual information based subbands statistical features selection". In: *Journal of Ambient Intelligence and Humanized Computing* (2021), pp. 1–15.

[4] J. Rodriguez, A. Goni, and A. Illarramendi. "Real-time classification of ECGs on a PDA". In: *IEEE Transactions on Information Technology in Biomedicine* 9.1 (2005), pp. 23–34. DOI: 10.1109/TITB.2004.838369.

[5] Xuexiang Xu and Hongxing Liu. "ECG heartbeat classification using convolutional neural networks". In: *IEEE Access* 8 (2020), pp. 8614–8619.

[6] Chenshuang Zhang et al. "Patient-specific ECG classification based on recurrent neural networks and clustering technique". In: *2017 13th IASTED International Conference on Biomedical Engineering (BioMed)*. 2017, pp. 63–67. DOI: 10.2316/P.2017.852-029.