
SAPIENZA
UNIVERSITÀ DI ROMA

# Know A Person By Their Heart

## Biometric System Project Report

*Deborah Dore*

1994616

dore.1994616@studenti.uniroma1.it

*Giada Mauri*

1806590

mauri.1806590@studenti.uniroma1.it

Academic Year 2021-2022

# Contents

# 1 Introduction

Electrocardiography is the process of producing an electrocardiogram (ECG or EKG)[4]. An electrocardiogram expresses unique cardiac features of an individual in a way that is painless, noninvasive and fast. An electrocardiography is used to determine abnormal heart rhythm, if there are blocked/narrowed arteries in the heart, to detect heart attack and to determine if a particular treatment (es. pacemaker) is working. During a canonical ECG, 12 sensors are placed in the chest and in the limbs. The electrodes are connected through wires to a monitor that records the electrical signals of the heart.

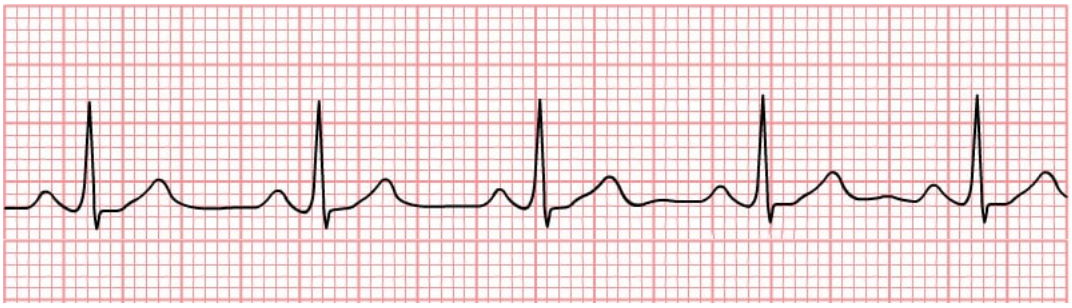


Figure 1: Regular Sinus Rhythm

As we can see from the figure, in every curve we can distinguish 5 waves. This waves are identified with the letters **P,Q,R,S and T**.

- *P wave*: represents atrial depolarization, the contraction of the heart. It lasts about 0.08 seconds.
- *R, S, Q waves*: together form the so-called **QRS Complex**. It represents the ventricular depolarization. Usually, it lasts 0.12 seconds.
- *T wave*: happens when we have the relaxation of the ventricles. This wave represents ventricular repolarization.

Together they form the **PQRST Complex** as shown in the figure below:

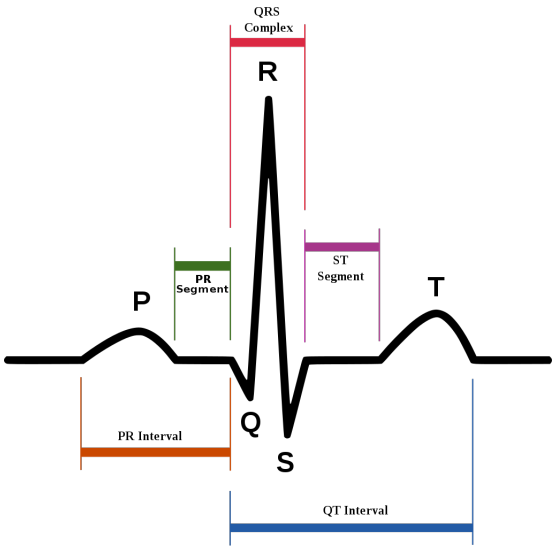



Figure 2: ECG Waves

An Electrocardiogram has extremely discriminative characteristics in the biometric field. An ECG has the advantage of being *universal* since we can trace an ECG for every person, *unique* because is sufficiently distinguishable across individuals, *acceptable* meaning that the process of recording an ECG is not invasive and so the patient shouldn't be against it, *collectable* because we can acquire it with modern sensors, *permanent* since it's stable and durable. Furthermore, it is particularly resistant to spoofing. However, it changes frequently based on a person emotional and health state. However, ECG are susceptible to several type of noises and therefore signal denoising is an important part in the prepossessing step.

Our goal is to build a performing and usable system that recognises a person based on an Electrocardiography taken using a modern smartwatch. We started from the analysis of the ECG waves described before. These waves have a unique pattern, and from the analysis of these waves, it is possible to extract five fiducial points **P, Q, R, S** and **T**. We can extract various information from these fiducial points, such as the distance between the points, the amplitude and the angle, the slope between the points, and height.

The project is composed of a Data Processing component, a Classification component, an Enrollment component, a Prediction component, an Evaluation component and a central component.

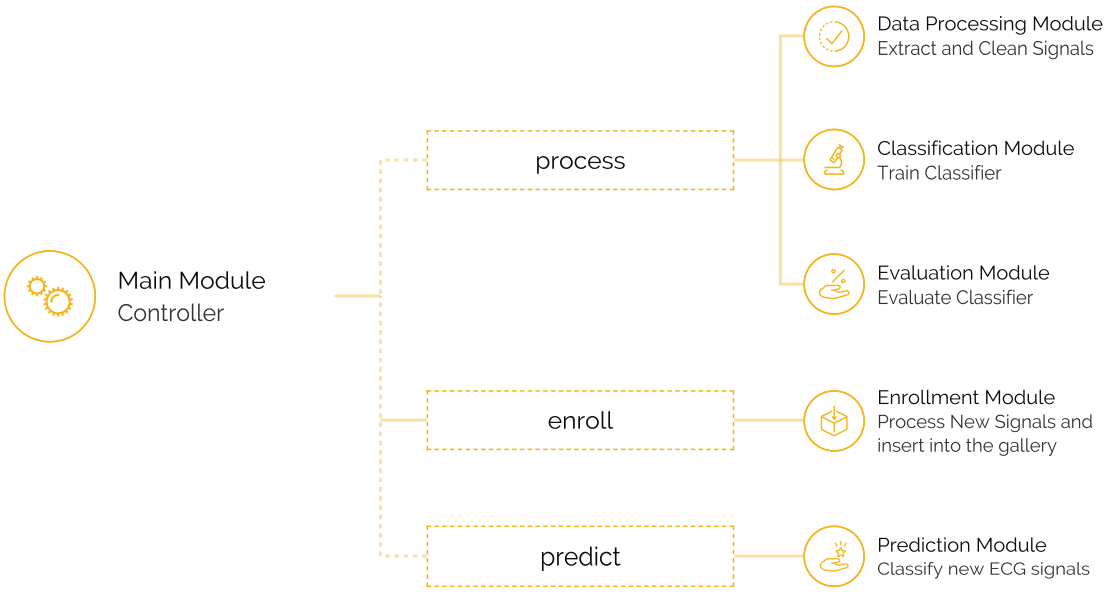



Figure 3: Project Structure

Raw ECG signals are denoised and transformed in the Data Processing component to perform peak segmentation and extract relevant features. The Classification component is where a chosen model is trained and tested with the dataset, producing predictions. The Enrollment component processes new signals and inserts them into the dataset, while the Evaluation component evaluates the overall process using different biometric evaluation techniques. The Prediction component uses the previously trained model to make predictions about new given ECG signals. From the central module we can decide which action to carry on using custom keywords:

*process* that processes the signals, train and evaluates the model, *enroll* that enrolls new ECG and inserts them into the gallery and *predict* that classifies given ECG signals.

The following chapters explain how to extract ECG signals, modify them, analyze them, and extract features. In order to do so, different libraries have been used: the Scikit-Learn library along the Neurokit2, Scipy, Pandas and Numpy for signal detection and peak extraction. A model for classifying instances has been presented in the third chapter. The fourth chapter contains the relative observations and evaluations. Evaluating the classifier has required the use of the Cumulative Match Characteristic and the Cumulative Match Score. As will be explained, these techniques are helpful when discussing identification closed-set systems. Furthermore, in the fifth chapter, it will be presented how an Apple Watch Series 7 has been used to extract real-time ECG.

# 2 Data Processing

## 2.1 Dataset

The dataset used is "PTB Diagnostic ECG Database" form PhysioNet[1].
The ECGs were obtained using a non-commercial, PTB prototype recorder with the following specifications:

- 16 input channels, (14 for ECGs, 1 for respiration, 1 for line voltage)
- Input voltage: ±16 mV, compensated offset voltage up to ± 300 mV
- Input resistance: 100 Ω (DC)
- Resolution: 16 bit with 0.5 $\mu$V/LSB (2000 A/D units per mV)
- Bandwidth: 0 - 1 kHz (synchronous sampling of all channels)
- Noise voltage: max. 10 $\mu$V (pp), respectively 3 $\mu$V (RMS) with input short circuit
- Online recording of skin resistance
- Noise level recording during signal collection

The database contains 549 records from 290 subjects, with age between 17 to 87 years, 209 men and 81 women, and 1 to 5 records for each subjects.
Each record includes 15 simultaneously measured signals:

- The conventional 12 leads *(i, ii, iii, avr, avl, avf, v1, v2, v3, v4, v5, v6)*
- The 3 Frank lead ECGs *(vx, vy, vz)*

The conventional 12 leads are stored in a .dat file for each record and the 3 Frank lead are stored in .xyz file for each record. Signals are digitalized with a frequency of 1000 samples per second, a resolution of 16 bit over a range of ± 16.384 mV.
For each record there is an header file (.hea) with information about the subject such as age, gender, diagnosis and, if available, other clinical data including: medical history, medication and interventions, coronary artery pathology, ventriculography, echocardiography, and hemodynamics. We chose this dataset because it contains a suitable number of records collected from subjects with a wide range of characteristics and health diagnosis, making the evaluation more realistic. The other main reason that led to this choice is the presence of an identifier for each subject in the records, since we need it as a ground truth in the evaluation process.

## 2.2 Feature Extraction

The dataset provided the records of 294 patients. The ECG waveform came under the *.dat* and *.xyz* extensions that are not straightforward to manipulate. To extract the ECG data points from these files is convenient to use the **WFDB software package**. A collection of native python scripts for reading and writing WFDB signals.
Every record was processed, cleaned and denoised, and the result was saved into a

*.csv* file. The ECG lead extracted was the **V4** because it is the most useful since it is obtained by placing the V4 electrode in the fifth right intercostal space in the mid-clavicular line.
First, each record is saved into a WFDB record object, and then the signal is extracted from the record, as shown below.

```
record = wfdb.rdrecord(path_to_patient_dir, channel_names=['v4'])
signal = record.p_signal.ravel()
```

The wfdb record can be plotted to show the ECG:

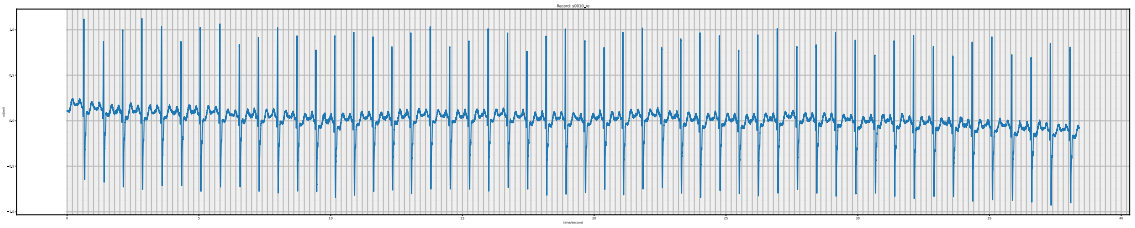


Figure 4: WFDB Record

When recording an ECG, unwanted signals are merged with the ECG signal causing obstacles and noise that must be removed using proper signal processing methods. There are four principal artifacts in an ECG signal: baseline wander, powerline interference, EMG noise and electrode motion. Each of them can be removed using appropriate filters: High Pass Filter, Band Stop Filter, Low Pass Filter and Smoothing Filter.

### 2.2.1 High-pass filter

*Baseline wander* is the effect that makes the signal go up and down instead of straight. This phenomenon is due to the patient's movement and breathing and can be removed using a High Pass Filter. Figure 5 shows an ECG signal where we can clearly distinguish areas where the signal went up and down repeatedly.

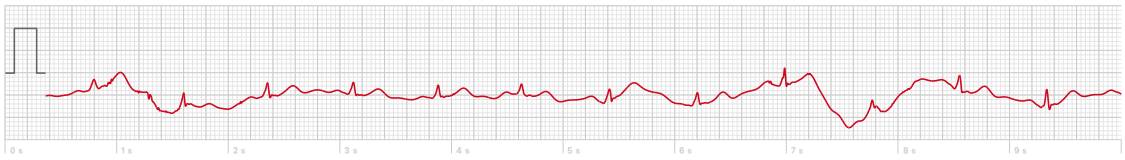


Figure 5: ECG signal with baseline wander noise

Since the frequency of the Baseline Wander is in the range of 0.5 Hz, using a Finite Impulse Response high-pass with a cut-off frequency of 0.5 Hz will remove the noise.
The High Pass Filter is implemented using the python library Scipy, as shown below:

```
nyq_rate = 1000 / 2
width = 0.2 / nyq_rate
```

```
ripple_db = 12
num_of_taps, beta = kaiserord(ripple_db, width)
if num_of_taps % 2 == 0:
    num_of_taps = num_of_taps + 1
cutoff_hz = 0.5
filter_hf = firwin(num_of_taps, cutoff_hz / nyq_rate, window=('kaiser',
↪  beta), pass_zero='highpass')
```

The Nyquist rate, also known as the minimum frequency rate required to sample analogical signals without losing information, is set to 500 due to the sampling rate of the signals (which is 1000 Hz). The filter's maximum ripple is set to 12 db. The width of the filter is the region where it transitions from non-attenuation to attenuation. The Kaiser window technique is used to determine the filter window parameters. Figure 6 displays a graphical representation of what we discussed before.

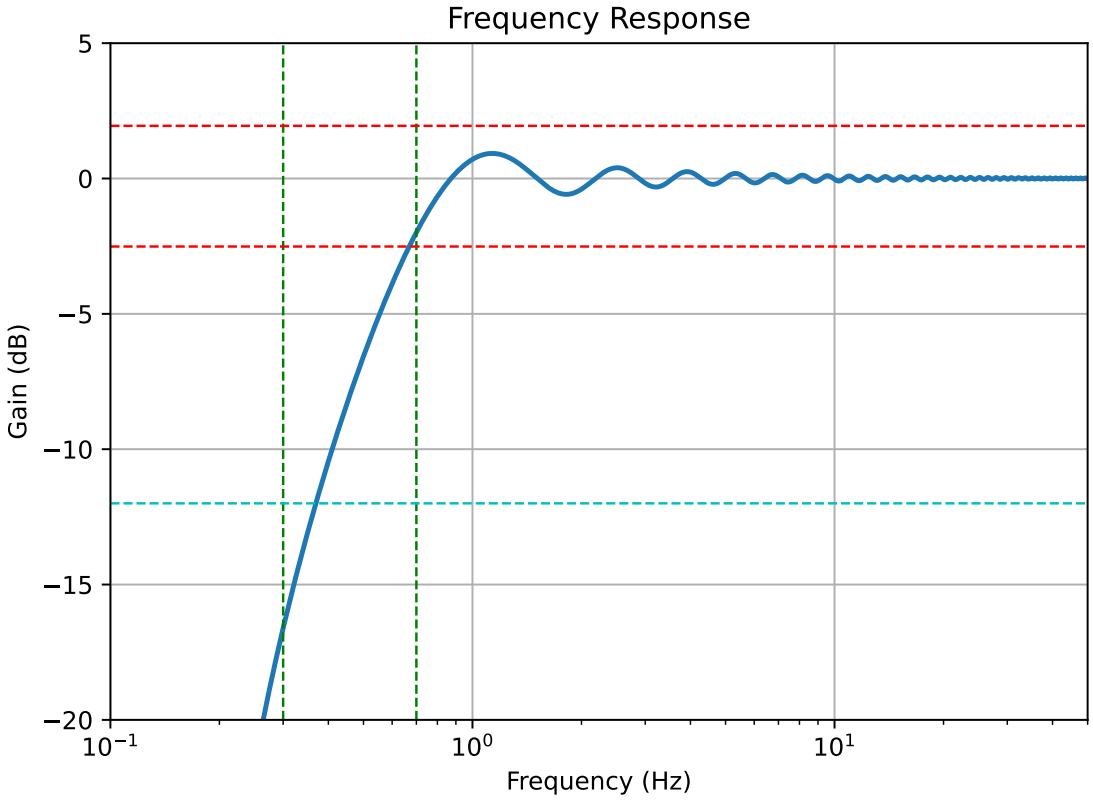



Figure 6: High Pass Filter

### 2.2.2 Band Stop filter

*Notch Filters*, which are *Band Stop Filters* with narrow stopband, are helpful to suppress the *powerline interference* and *muscle contraction noise*. *Powerline interference* caused by the electromagnetic fields is a typical ECG noise source. This noise is characterized by 50/60 Hz sinusoidal interference and makes the ECG unreliable since it can modify the delineation of low amplitude waveforms. In particular, it affects the low-frequency ECG waves like P wave and T wave. One can use a highly

selective notch filter designed at a particular frequency to suppress these unwanted distortions from the ECG signal.

```
nyq_rate = 1000 / 2
width = 0.2 / nyq_rate
ripple_db = 12
num_of_taps, beta = kaiserord(ripple_db, width)
if num_of_taps % 2 == 0:
    num_of_taps = num_of_taps + 1
cutoff_hz = np.array([59.5, 60.5])
filter_bs = firwin(num_of_taps, cutoff_hz / nyq_rate, window=('kaiser',
↪  beta), pass_zero='bandstop')
```

In our case, the Band Stop cut-off frequencies are 59.5 and 60.5 Hz. The transition region, filter ripple, and Nyquist rate are identical to the High Pass Filter. The filter is graphically represented in Figure 7.

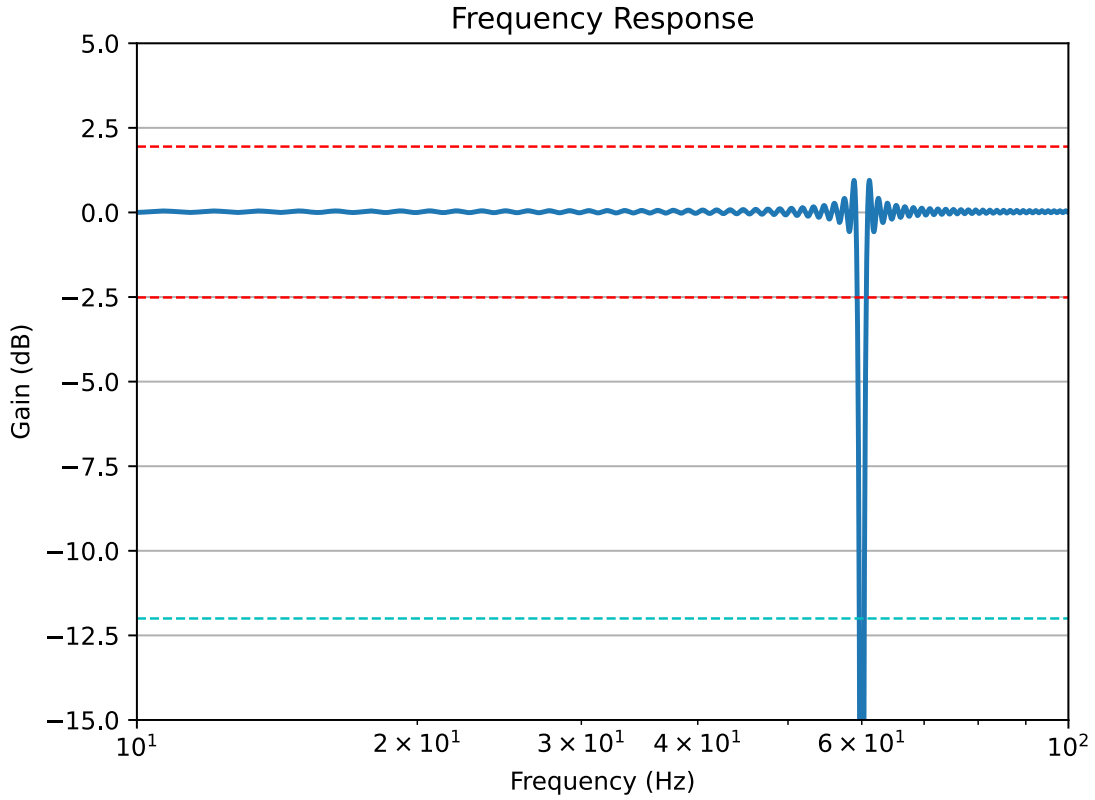



Figure 7: Band Stop Filter

### 2.2.3 Low-pass filter

Low pass data filtering has an impact on the waveform's harmonic content since it eliminates higher order harmonics by definition. The implementation of this filter is displayed below.

```
nyq_rate = 1000 / 2
width = 1 / nyq_rate
```

```
ripple_db = 12
num_of_taps, beta = kaiserord(ripple_db, width)
if num_of_taps % 2 == 0:
    num_of_taps = num_of_taps + 1
cutoff_hz = 100
filter_lp = firwin(num_of_taps, cutoff_hz / nyq_rate, window=('kaiser',
↪  beta), pass_zero='lowpass')
```

The Nyquist rate is identical to that of the High Pass Filter. However, the transition region is smaller than the one of the High Pass Filter. The filter's maximum ripple is set to 12 db. The Kaiser window technique is used to determine the filter window parameters. The attenuation frequency for the filter is set at 100 Hz. The filter is illustrated visually in Figure 8.

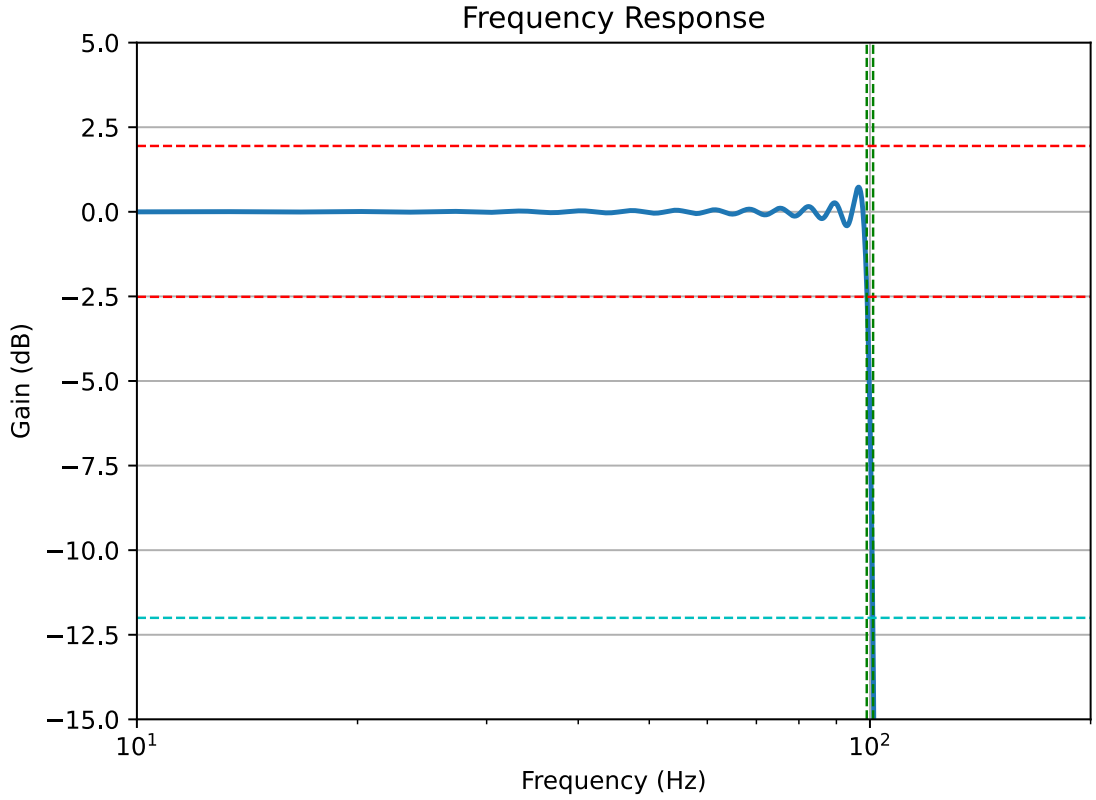



Figure 8: Low Pass Filter

### 2.2.4 Smoothing filter

After noise filtering, the denoised ECG signal is then handed to a Smoothing filter. Smoothing a signal results in an approximation function that tries to capture essential patterns in the data while ignoring the noise. A smoothing filter thereby eliminates the signal's roughness. The smoothed signal may be used to detect peaks in the ECG signal with higher accuracy.
Because the Scipy library includes a ready-to-use Savitzky-Golay filter to apply to an array, this is a reasonably straightforward procedure. The filter window's length

(the number of coefficients) is set to 29. The polynomial used to fit the samples has a third order.

```
smoothed_signal = savgol_filter(signal, 29, 3)
```

The result of the filter can be illustrated graphically (Figure 9).

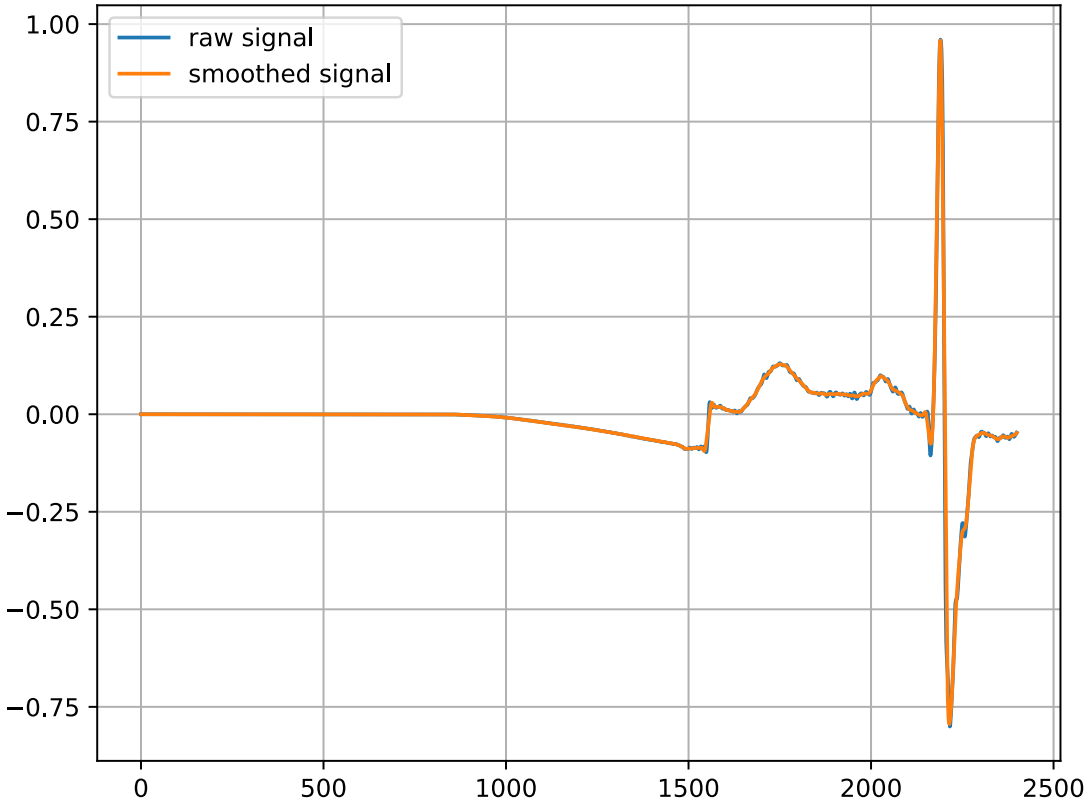
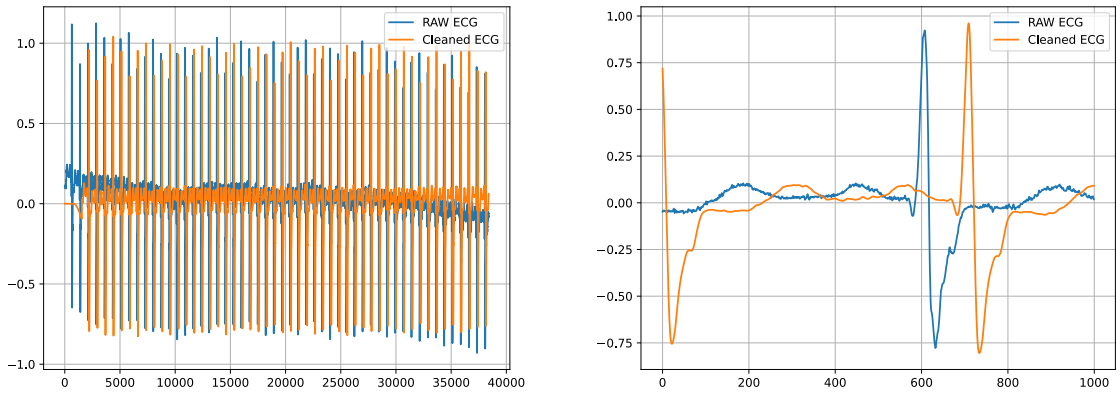



Figure 9: Smoothing Filter

### 2.2.5 Results

The outcome of applying all of those filters is a signal that has been denoised, cleaned, smoothed, and is ready for the peak extraction step. The signal produced by applying the four filters is shown below in Figure 10a and in detail in Figure 10b.




(a) Signal Denoised (b) Detail Signal Denoised

Figure 10: Signal cleaned and denoised

As can be seen, the signal is smoother and follows a straight line than the unprocessed signal. The translation is caused by the Kaiser window, but it is not an issue.

### 2.2.6 Peak Extraction

Now that the signals are cleaned and denoised, we can extract the peaks. For this purpose, we have utilized two main libraries: *Scipy* and *Neurokit2*.
The Neurokit2 library provides a straightforward function to extract the T, Q, P and S peaks given the R peak. We have used Scipy's function *find_peaks* to extract the R Peaks.

```
find_peaks(cleaned_signal, prominence=1, distance=100)
```

We can notice two parameters specified in the function: prominence and distance. The prominence of a peak refers to how noticeable it is due to its height and location with other peaks. The distance is the needed minimum horizontal distance between adjacent peaks in samples.
Looking at Figure 2, we observe that the R peak is the more prominent, and there is a minimum distance between two R peaks. Due to this, the prominence parameter was set to 1 in the find peak's function and the distance to 100. The prominence and the distance parameters will allow better recognition of the peaks. ECG signals with fewer than 15 R peaks will not be considered.
To extract the other peaks, we have used the *ecg_delineate* function of the Neurokit2 library that delineates the QRS complex that uses the discrete wavelet transform method. A discrete wavelet transform decomposes a signal into several sets, each including a time series of coefficients describing the signal's time evolution in the appropriate frequency band.

```
nk.ecg_delineate(cleaned_signal, rpeaks=r_peak, sampling_rate=1000,
    ↪ method="dwt", show=False, show_type='peaks')
```

The result is shown in Figure 11.

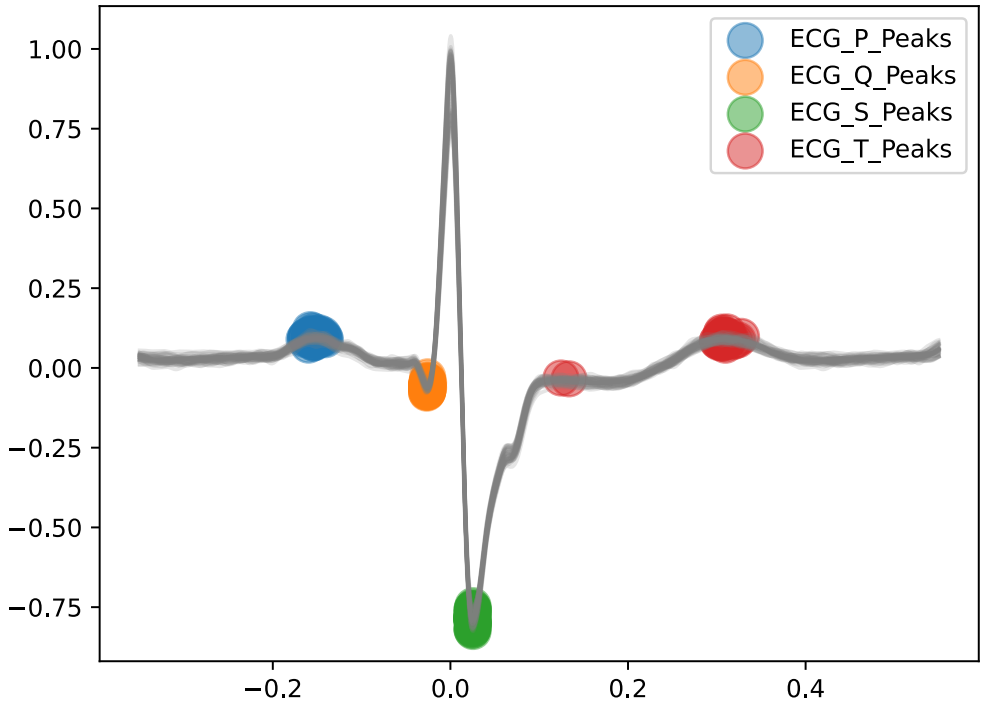



Figure 11: Peak Extraction

The K Nearest Neighbour approach was performed to each list of peaks to eliminate the nan values (peaks that were not accurately recognized) based on the values of their neighbors. The mean value from $n$ nearest neighbors discovered in the training set is used to impute each sample's missing data. If the traits that neither sample lacks are similar, the samples are near. After that, the average of each peak list was extracted to start constructing the dataset.
The dataset is composed of 58 features extracted from the ECG signal:

1. Time features: $T_x$, $P_x$, $Q_x$, $S_x$, $PQ_{time}$, $PT_{time}$, $QS_{time}$, $QT_{time}$, $ST_{time}$, $PS_{time}$, $PQ - QS_{time}$, $PQ - QS_{time}$, $QT - QS_{time}$

2. Amplitude features: $T_y$, $P_y$, $Q_y$, $S_y$, $PQ_{ampl}$, $QR_{ampl}$, $RS_{ampl}$, $QS_{ampl}$, $ST_{ampl}$, $PS_{ampl}$, $PT_{ampl}$, $QT_{ampl}$, $ST - QS_{ampl}$, $RS - QR_{ampl}$, $PQ - QS_{ampl}$, $PQ - QT_{ampl}$, $PQ - PS_{ampl}$, $PQ - QR_{ampl}$, $PQ - RS_{ampl}$, $PQ - RS_{ampl}$, $RS - QS_{ampl}$, $RS - QT_{ampl}$, $ST - PQ_{ampl}$, $ST - QT_{ampl}$

3. Distance features: $PQ_{dist}$, $QR_{dist}$, $RS_{dist}$, $ST_{dist}$, $QS_{dist}$, $PR_{dist}$, $ST - QS_{dist}$, $RS - QR_{dist}$

4. Slope features: $PQ_{slope}$, $QR_{slope}$, $RS_{slope}$, $ST_{slope}$, $QS_{slope}$, $PT_{slope}$, $PS_{slope}$, $QT_{slope}$, $PR_{slope}$

5. Angle features: $PQR_{angle}$, $QRS_{angle}$, $RST_{angle}$, $RQS_{angle}$, $RSQ_{angle}$, $RTS_{angle}$

All of the characteristics were estimated using the position of R as the origin in mind. The amplitude (y), time duration (x), the difference between amplitudes

(ampl), distance (dist), the difference between times (time), slope, angle, area, and the ratio of several characteristics were recovered from the PQRST fragment.

## 2.3 Feature Analysis

The high number of features requires a feature analysis to remove the features with low importance that will only slow the training phase. This part must be carried out before carrying on with feature engineering. It is essential to visualize the correlation and treat highly correlated features. In highly correlated variables, the changes in one variable would cause changes in another. As a result, the model will be unstable and vary significantly given a slight change in the data or model. We will perform the feature importance analysis using a Random Forest Regressor.
First, all the categorical features must be encoded: in our case, the only categorical feature is the patient name of the class we want to predict. Secondly, the dataset must be split to fit the model. The fitted model will then be used to perform the permutation importance and extract the feature importance values shown in the plot below.

```
df = pd.read_csv(dataset)
X = df.copy()
y = X.pop('PATIENT_NAME')

feature_names = X.columns

enc = LabelEncoder()
y = enc.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=True,
    ↪ test_size=0.3)

forest = RandomForestRegressor()
forest.fit(X_train, y_train)

result = permutation_importance(forest, X_test, y_test, n_repeats=10,
    ↪ random_state=42, n_jobs=2)
forest_importances = pd.Series(result.importances_mean,
    ↪ index=feature_names)
```

From the plot in Figure 12, displayed using the logarithmic scale to better distinguish between classes, we can notice many features whose importance is very low. Every feature below 0.001 will be dropped; since all these features are not helpful to the model.

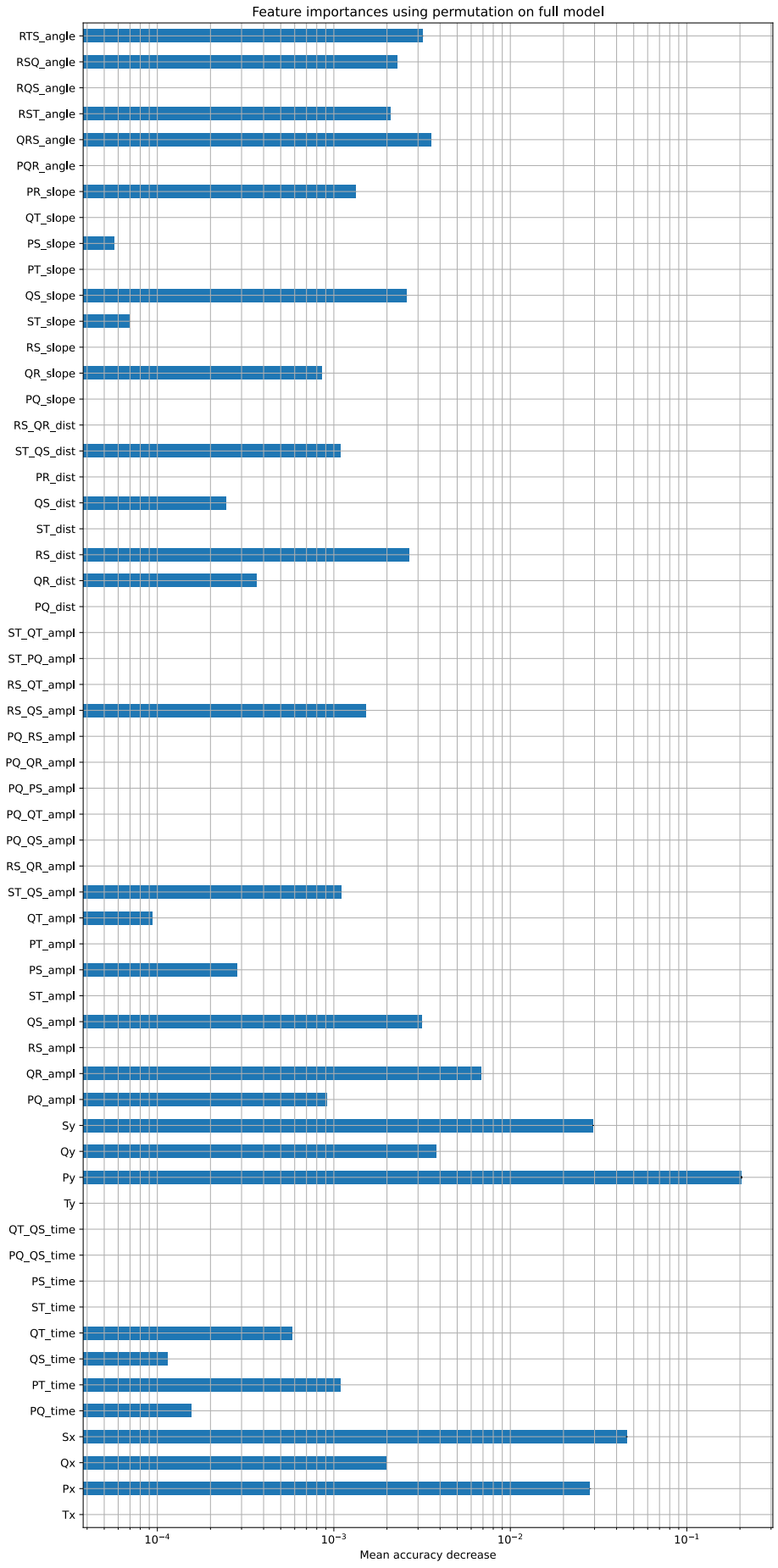



Figure 12: Feature Importance

## 2.4 Feature Transformation

### 2.4.1 Data Balancing & Missing Values Removal

Our dataset did not provide many signals, and after the signal cleaning and denoising, the sufficiently strong signals were even less. Due to this, the dataset, the number of instances for each class, was unbalanced.

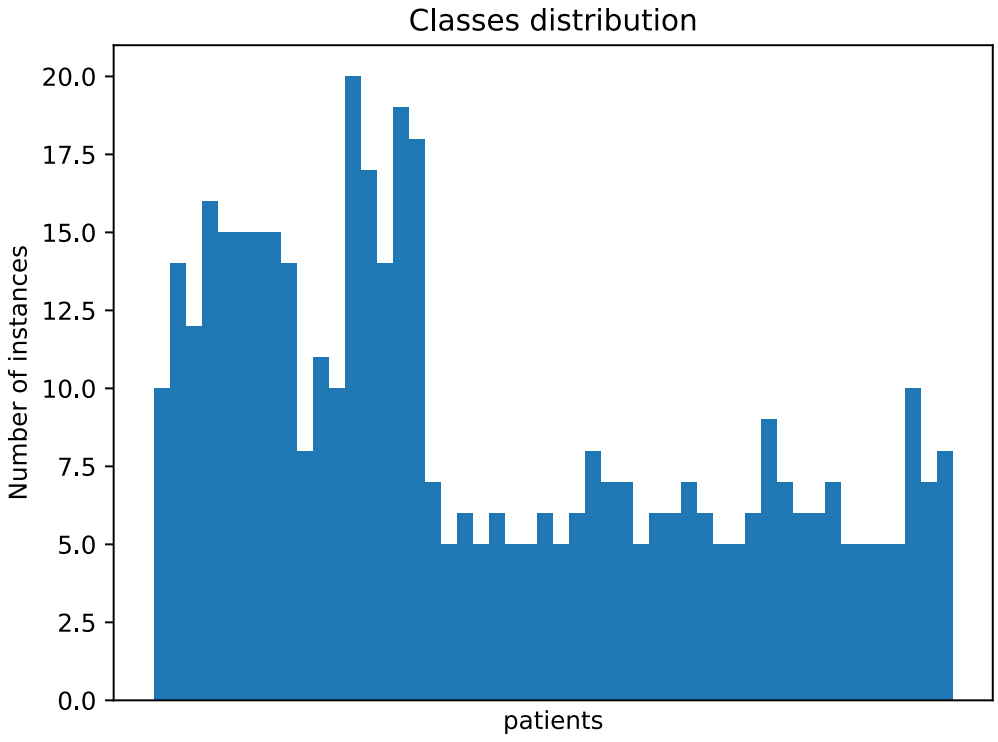



Figure 13: Class distribution

An unbalanced dataset can lead to a biased model that cannot classify the patients correctly. We have used the RandomOversampler algorithm from the imbalanced learn library to address this problem. The RandomOversampler is used to oversample the minority classes by randomly selecting samples and replacing them. It is possible to produce the bootstrap more smoothly. This method increased the number of instances in the minority class, balancing the dataset.

```
1 imputer = KNNImputer()
2 X = imputer.fit_transform(X)
```

Also, the K Nearest Neighbour Imputer, previously discussed, was applied to the features to remove missing values since the models cannot process those values.

```
1 oversample = RandomOverSampler(random_state=42)
2 X, y = oversample.fit_resample(X, y)
```

# 3 Classification

Now that the dataset has been balanced, we can start training a model.

## 3.1 Training and Testing

As we said before, the model cannot understand categorical values; we must encode them before starting the training. In order to accomplish this, we have used the Label Encoder by the scikit-learn library.

```
enc = LabelEncoder()
y = enc.fit_transform(X.pop('PATIENT_NAME'))
```

The second step was to scale all training sets to a value in the [0,1] range. For this, we have used the Min-Max Scaler.

```
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```

The dataset was split into training (70%) and testing (30%) with the shuffling option activated. This option shuffles the dataset to reduce volatility and guarantee that the model effectively generalizes new, previously unknown data points.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=True,
    ↪ random_state=42)
```

## 3.2 Model Selection

We trained many categories of models in order to determine which was the best model to use:

- **Support Vector Classifier**: a supervised machine learning algorithm. Each data point in an n-dimensional space is plotted as a point in an SVM, with the value of each feature being the value of a certain coordinate. The classification is then completed by locating the hyper-plane that clearly distinguishes the two classes.
- **Decision Tree Classifier**: supervised machine learning algorithm. Internal nodes contain dataset attributes, branches represent decision rules, and each leaf node provides the conclusion in a tree-structured classifier.
- **Multi-Layer Perceptron Classifier**: a class of feedforward artificial neural network. There are at least three nodes in an MLP: an input layer, a hidden layer, and an output layer. Except for the input nodes, each node is a neuron with a nonlinear activation function. It uses Backpropagation to update each node weight and perform a better classification.
- **Random Forest Classifier**: ensemble learning approach that works by training time by generating a large number of decision trees. The class chosen by most trees will be the output.

- **AdaBoost Classifier**: is a particular method of training a boosted classifier by turning several poor learners into strong learners. Boosting algorithms work on the idea of first building a model on the training dataset, then building a second model to correct the faults in the first. This technique is repeated until the mistakes have been reduced to a minimum and the dataset has been accurately forecasted.

Every model was trained in parallel using a custom function that, after the training, evaluates the model using large-known classification metrics such as: classification report and cross-validation.
The *classification report* function provided by the scikit-learn library displays **Precision**, **Recall**, **F1-Score** and **Support** for each class trained.
**Cross-validation** is carried out by partitioning the data into several subsets, leaving one out, from which the model is trained and testing the model on the hold-out set. Cross-Validation was used along with the Repeated Stratified K Fold methodology. Using this strategy, data is split into folds, ensuring that each fold has the same proportion of observations with a given categorical value, such as the class outcome value. Cross-validation considerably decreases bias since we are utilizing the majority of the data for fitting, and it also significantly reduces variance because the majority of the data is also used in the test dataset.
The function used to carry out the training and the evaluation is reported below:

```
def work(name, model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    score = model.score(X_test, y_test)
    report = pd.DataFrame(metrics.classification_report(y_test, y_pred,
    ↪  zero_division=0, output_dict=True)).T
    #  cross val to avoid overfitting
    cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=2,
    ↪  random_state=1)
    n_scores = cross_val_score(model, X_train, y_train,
    ↪  scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
    return name, score, report, n_scores
```

As previously stated, each model was trained in parallel using the joblib's *Parallel* function.

```
models = {
    'svm': svm.SVC(),
    'dtree': tree.DecisionTreeClassifier(),
    'mlpn': neural_network.MLPClassifier(),
    'randforest': ensemble.RandomForestClassifier(),
    'adaboost': ensemble.AdaBoostClassifier(),
}

res = joblib.Parallel(n_jobs=len(models),
↪  verbose=2)(joblib.delayed(work)(name, model, X_train, y_train,
↪  X_test, y_test) for name, model in models.items())
```

The best model with higher accuracy was proved to be the **Random Forest Classifier**. Random Forest Classifiers, as said before, are generated by combining several unpruned trees and adding *source of randomness* to create robust and independent learners. In fact, each tree is trained on a subset of the dataset created using the **Boostrap technique**: each tree extracts $m$ samples from a dataset of $n$ instances with replacement. The samples that are not extracted are used as *testing set*. At each node of the tree, the best split is also chosen on a subset of features randomly selected. This adds robustness and independence between the trees so that we don't have to worry about overfitting.

## 3.3 Hyper-parameter auto tuning

Not yet satisfied with the result, we decided to use the Randomized Search CV that applies a Randomized search on hyperparameters. Random search is a strategy for finding the optimum solution for a created model by using random combinations of hyperparameters. The parameters used to tune the Random Forest Classifier were the following:

- The number of trees in the forest
- The number of features to consider when looking for the best split
- The maximum depth of the tree
- The minimum number of samples required to split an internal node
- The minimum number of samples required to be at a leaf node
- Whether bootstrap samples are used when building trees

```
parameters = {
             'n_estimators': [int(x) for x in np.linspace(start=200,
             ↪  stop=2000, num=10)],
             'max_features': ['auto', 'sqrt'],
             'max_depth': [int(x) for x in np.linspace(10, 110,
             ↪  num=11)],
             'min_samples_split': [2, 5, 10],
             'min_samples_leaf': [1, 2, 4],
             'bootstrap': [True, False]
}
```

The Randomized Search CV fits 3 folds for each of 100 candidates for a total of 300 fits.

```
clf = RandomizedSearchCV(estimator=model,
↪  param_distributions=parameters, n_iter=100, cv=3, verbose=2,
↪  random_state=42, n_jobs=-1)
clf.fit(X_train, y_train)
```

The search took about one hour. It provided an accuracy of 0.8535031847133758 on the testing dataset and the best parameters where:

- **Number of estimators**: 1400
- **Minimum samples split**: 2
- **Minimum samplles leaf**: 1
- **Maximum number of features**: auto
- **Maximum depth of the tree**: 100
- **Bootstrap**: true

We could not enhance our model's accuracy considerably; the model has improved slightly.
This model was used to predict the class for the testing set, and the results were saved in a CSV file, ready for the evaluation phase.

# 4 Evaluation

Biometric systems are susceptible to a variety of faults. In this part, we provide an overview of the most often used performance measures in the literature.
Before starting to evaluate the system, let us define some concepts. There are two types of recognition operation:

- **Verification**: The user claims an identity and the system perform a 1:1 matching to verify the identity
- **Identification**: There is no identity claimed and the system perform a 1:N matching to determine the correspondence with one of the subjects in the gallery

Our system falls under the identification case. We don't initially claim the identity, the classifier predicts a label for each probe submitted performing a 1:N matching and the predicted class is compared to ground truth.

Identification can be subdivided into another two categories:

- **Open set**: where some probes may not belong to any subject in the gallery and the system has a reject option.
- **Closed set**: where all probes belong to enrolled subject, the system doesn't have a reject option but may return the wrong identity.

In this case we are in an closed set identification.

## 4.1 Metrics

Cumulative Match Score and Cumulative Match Characteristic are the metrics used in literature to evaluate a Closed-Set Identification system. In the following subsections, we have used them in order to better understand what our system is capable of doing.

### 4.1.1 CMS

The **Cumulative Match Score** (CMS) at a certain rank $k$, can be defined as the probability of identification at that rank $k$. It judges the ranking capabilities of an identification system. The Cumulative Match Score at rank one is also defined as Recognition Rate.
It is calculated computing the ratio between the number of individuals correctly predicted among the number of ranks and the total number of individuals in the system.

We initially imported the ground truth labels and classification scores from the previously computed predictions using the test set to compute the CMS. We labelled the scores with the corresponding class. The Scikit-learn library provides the *predict_proba* function that computes estimates for all classes and orders them by the classes' label.

```
# import data
x = pd.read_csv(dataset)
real = x.REAL
scores = x.SCORES

# label score
for i in range(len(scores)):
for j in range(len(scores[0])):
    scores[i][j] = (scores[i][j], j)
```

Subsequently we calculated the CMS for a number of ranks that equals the number of classes in the gallery.

```
# compute CMS
ranks = len(scores[0])
CMS = dict()
c = 0
for k in range(ranks):
    CMS[k + 1] = c
    for i in range(len(real)):
        s_scores = sorted(scores[i], reverse=True)
        if s_scores[k][1] == real[i]:
            CMS[k + 1] += 1
            c += 1
    CMS[k + 1] = CMS[k + 1] / len(real)
```

Applying this evaluation metric we found out that:

- The system has a Recognition Rate (CMS at rank 1) of **0.86**. This means that 86% of users where correctly recognized while 14% of users where misclassified
- The Cumulative Match Score at rank 5 is **0.88**, showing a slow rise

In general, the better the algorithm, the higher the rank k CMC percentage. At rank 1, our models has a very good accuracy. We can be satisfied with the results, although improvement is always possible.

### 4.1.2 CMC

The Cumulative Match Characteristics (CMC) curve visualize the Cumulative Match Score for a certain number k of ranks. It reports the probability that the classifier predicts the correct class in the first k places.

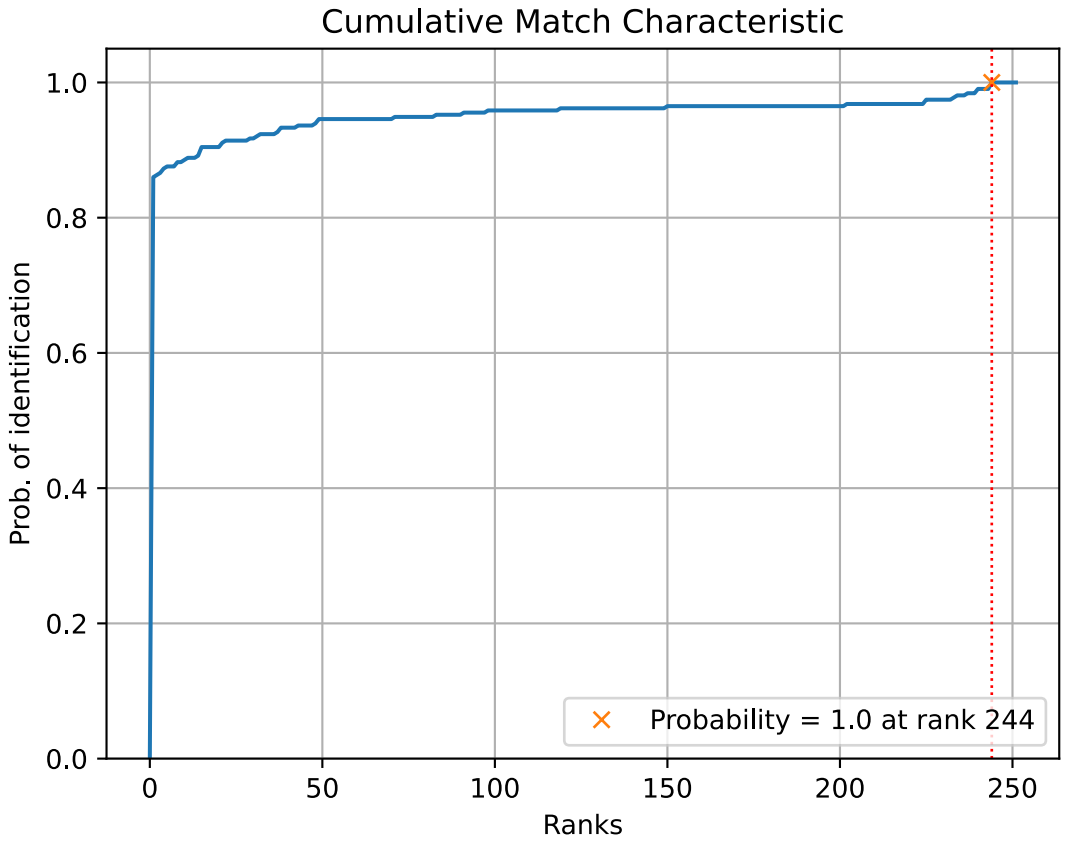



Figure 14: CMC

The curves above show the results of the evaluation for our system. In Figure 14, we can see the curve where the number of ranks equals the number of classes in the system, reaching the higher rank, the score goes to the maximum value of 1.0 at rank 244. As expected, the curve rises rapidly as the rank rises. The faster it grows, the better the model can classify correct examples in the first k ranks.

We can analyze in detail the result for the first fifth ranks.

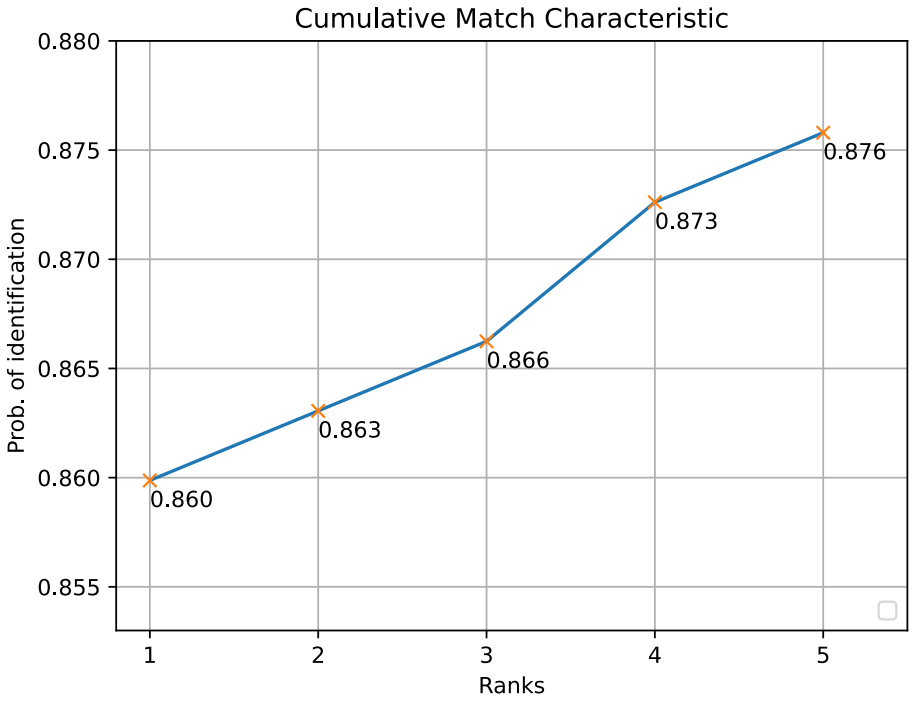



Figure 15: CMC at rank 5

As we can see from the Figure 15 there is nearly no improvement from the recognition between rank 1 and 2. While rank 3 has an improvement of 0.6 %, rank 4 of 1.5 % and rank 5 of 1.8 % with respect to the first rank.
From the Cumulative Match Characteristic curve above, we can say that the probability of correctly classifying users is very high, 86 %, since the model always choose the user with the highest chance. From the analysis of the scores computed on the train set, we can observe that if the user is correctly classified, the probability of the real class of the user is the highest between all classes, that probability is very high, with peaks of 90 %. However, if the user is not instantly recognized, the rank at which the real class of the user is placed is very high. This could signify that templates of the same user in the gallery are very different between themselves. The model cannot extract all the various features, and probes submitted to the system contain features that the system could not extract during the training phase. This problem could be quickly resolved by enrolling more templates of the same user. We have already tried to solve this issue by balancing the dataset. However, the Random Over Sampler only duplicates existing instances without adding anything new. We could use other approaches like SMOTE that creates convex combinations of neighbouring instances, but the SMOTE algorithm requires more examples than the one we can provide. Therefore, the better solution will be to search for other instances of the same users in relative datasets.

As we said before, the Recognition Rate of our system is 86 %. Given the data we had, and the state of the art of our problem (ECG as a biometric trait is still under investigation), we can say that it is a good result.

# 5 Identify users using an Apple Watch

The last Apple Watch 7 contains an electrical heath sensor that can record a heartbeat and its rhythm. Using the ECG application built-in in the iPhone, we can record an ECG anywhere and export it in a PDF or CSV format. The Apple Watch Series 7's ECG app creates an ECG that is identical to a single-lead (or Lead I) ECG. In tests comparing the ECG app on Apple Watch to a regular 12-lead ECG obtained at the same time, there was agreement between the ECG app categorization of the rhythm as sinus or AFib compared to the standard 12-lead ECG. The ability of the ECG app to accurately classify an ECG recording into AFib and sinus rhythm was tested in a clinical trial of approximately 600 subjects, and demonstrated 99.6% specificity for sinus rhythm classification and 98.3% sensitivity for AFib classification for the classifiable results[6]. We chose this smartwatch because of its ease of use and the speed at which we can export an ECG.

## 5.1 Implementation

In order to create a closed set identification system, we must define an Enrollment Module where the extracted ECG are processed and inserted into the gallery.

### 5.1.1 Extract ECG from an Apple Watch 7

As we said before, recording an ECG using an apple watch is a straightforward mechanism. We can place the watch on our wrists, open the ECG app and hold our finger in the so-called *crown* for 30 seconds. This mechanism is displayed in Figure 16 below.


Figure 16: Recording an ECG using an Apple Watch

Each ECG, if the quality is sufficient, is saved into the Health App on the iPhone, from where we can export the ECG in both PDF and CSV format. We are interested in the CSV format, which contains the raw signal.
The file created from the ECG contains other information besides the signal:

- Name
- Date of birth
- Date of recording
- Classification (i.e. sinuosidal rithm)
- Symptoms
- Software version
- Device
- Sample y (512 Hz)
- Electrodote (Electrodote I)
- Unit (µV)

### 5.1.2 Enrolling new classes

We changed our code to process the ECG file taken from the Apple Watch to adapt it to the new file.
First, we must capture several ECGs with the Apple Watch. Some of them can be used for the *enrollment* phase, while others can be used for the *prediction phase*. The ECG we want to use for the enrollment phase must be placed in the directory *enrollment* for the script to find it and process it. Then, we can start the program in the *enrollment mode*. The signal will be processed and inserted into the gallery. The classifier will then be trained again with the new instances. We must re-train the model to insert new templates in the gallery because we cannot add a new class to an already trained model.
The training process requires some minutes for the model to re-fit all the datasets. A snippet of the code, with explanatory comments, is shown below.

```
1  # load old model
2  model = joblib.load(best_model)
3  X = pd.read_csv(dataset)
4
5  # encode categorical value
6  enc = LabelEncoder()
7  y = enc.fit_transform(X.pop('PATIENT_NAME'))
8
9  # save new label class encoding
10 np.save('classes.npy', enc.classes_)
11
12 # scale the dataset
```

```
scaler = MinMaxScaler()
X = scaler.fit_transform(X)

# split into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=True,
    ↪  random_state=42)

# fit the new model
model.fit(X_train, y_train)

# save the new trained mode
joblib.dump(model, 'model.joblib', compress=3)
```

### 5.1.3 Identify users

Now that the model has been trained we the new class, we can test it using the ECG extracted with the apple watch that we have put aside before. This ECGs must be placed into the *classify/to_be_predicted* directory. When the script is started in *prediction mode*, these signals are analyzed, and the previously saved model is used to forecast the new classes. The predicted value will then be compared to the true value and shown in the standard output. The code to identify individuals is shown below.

```
# load previusly trained model
model = joblib.load(best_model)

# process signals
process_new_data()

# load and prepare gallery
data_to_predict = pd.read_csv('classify/to_predict.csv')
y = data_to_predict.pop('PATIENT_NAME')
X = data_to_predict.copy()

# scale dataset
scaler = MinMaxScaler()
X = scaler.fit_transform(X)

# encode categorical label
enc = LabelEncoder()
enc.classes_ = np.load('classes.npy', allow_pickle=True)

# make predictions
y_pred = model.predict(X)

# save predictions
y_pred = enc.inverse_transform(y_pred)

# load into standard output
for y_predicted, y_true in y_pred, y:
```

```
28      if y_predicted == y_true:
29          print("CORRECTLY RECOGNIZED:", y_predicted)
30      else:
31          print("NOT RECOGNIZED:", "PREDICTED", y_predicted, "INSTEAD
            ↪  OF", y_true)
```

This project provides a method for implementing a recognition system based on a person's ECG. Nowadays, most smartwatches incorporate authentication mechanisms that require a password. Rather than the password, we might use the model we created. So instead of having a password, the smartwatch could recognize the wearer by monitoring its heartbeat.
Users will not need to enter the password anymore due to this. Furthermore, if the smartwatch is stolen and the thief discovers the password, he can easily access the watch's information and credit card. If the smartwatch is lost while using an ECG as an authentication method, no one will ever access it because an ECG is unique, and the smartwatch must be worn to perform authentication. Any attack would be futile.

# 6 Conclusion

In conclusion, we studied and developed a biometric system using the heartbeat, extracted from an ECG, as a trait. We operate with a multi-class classifier, testing different models to find the most accurate one, which turned out to be the Random Forest Classifier.

Regarding the evaluation, we choose the Cumulative Match Score and Cumulative Match Characteristic curve method because is the more suitable for the identification closed set operation we used for our system. The results show that the recognition rate equals **0.86**, this mean that the classifier predicts the correct individual with a precision of 86%

The system proved to be operationally sound, but can be improved in a multimodal environment combining the heartbeat with additional biometric traits, like fingerprints for example, in order to make it more resistant to attacks and spoofing.

# 7 References


[1] *Dataset: PTB Diagnostic ECG Database*. URL: https://physionet.org/content/ptbdb/1.0.0/.

[2] *ECG interpretation: Characteristics of the normal ECG (P-wave-QRS complex, ST segment, T-wave)*. URL: https://ecgwaves.com/topic/ecg-normal-p-wave-qrs-complex-st-segment-t-wave-j-point/.

[3] Mohamad El-Abed, Christophe Charrier, and Christophe Rosenberger. *Evaluation of Biometric Systems*. 2022.

[4] *Electrocardiography*. URL: https://en.wikipedia.org/wiki/Electrocardiography.

[5] "Signal processing techniques for removing noise from ecg signals". In: *jber* (2019). DOI: 10.17303/jber.2019.3.101.

[6] *Take an ecg with the ecg app on apple watch*. 2022. URL: https://support.apple.com/en-ie/HT208955.

[7] Anthony Ngozichukwuka Uwaechia and Dzati Athiar Ramli. "A Comprehensive Survey on ECG Signals as New Biometric Modality for Human Authentication: Recent Advances and Future Challenges". In: *IEEE Access* 9 (2021), pp. 97760–97802. DOI: 10.1109/ACCESS.2021.3095248.

[8] Steven A. Israela John M. Irvineb Andrew Chengb Mark D. Wiederholdc Brenda K. Wiederholdd. "ECG to identify individuals". In: *Patter Recognition* 38 (2005), pp. 133–142.

# List of Figures