Software prototype (Interface) for cardiac arrhythmias detection from electrocardiograms

Authors:

OUKHOUYA Kawtar

CHIKHI Rania

AGHNAJ Ithri

CHAFIQ Mohamed

JABBAR Samir

OUBRAHIM Hassan

Advisor:

Diego Hernán Peluffo-Ordoñez

Ben Guerir, Morocco - April, 2022

Acknowledgement:

We would like to express our gratitude to our supervisor, Mr. Deigo Peluffo, who guided us throughout this project.

Abstract

Electrocardiogram (ECG) signals are used to analyze the cardiovascular activity in the human body and have a primary role in the diagnosis of several heart diseases. The QRS complex is the most important and distinguishable component in the ECG because of its spiked nature and high amplitude. Automatic detection and delineation of the QRS complex in ECG is of extreme importance for computer aided diagnosis of cardiac disorder. Therefore, the accurate detection of this component is crucial to the performance of subsequent machine learning algorithms for cardiac disease classification. The aim of the present work is to detect the QRS wave from electrocardiogram (ECG) signals. Initially the noise was removed from the signal followed by Rpeaks detection and segmentation, characterization and we have applied Knn and SVM (Support Vector Machines) techniques for classification the analysis has been done on MIT-BIH Arrhythmia database.

Keywords: Pre-processing, Segmentation, Signal Processing, Cardiac arrhythmia, Classification.

Contents:

Introduction:

Heart is a muscular organ that acts like a pump to continuously send blood throughout your body. If disease or injury weakens the heart and body's organs did not receive enough blood to work normally. Electrocardiography is a device that records the electrical activity of the heart over time. It is the gold standard for diagnosis of cardiac arrhythmias.

Nowadays cardiac related diseases are increasing so rapidly in all age groups across the world. Based on previous studies it has already proven & concluded that most of the deaths are not happening because of heart attacks but it is happening because of the improper heartbeat rhythms that is because of Cardiac Arrhythmia hence causing this as a Saviour problem commonly observed now a days. Arrhythmia refers to any type of change in heart rhythm pattern from the normal rhythm of electrical impulses. In the case of Arrhythmia electrical pulse can be very fast, very slow or irregular in nature than the normal pulses. When our heart does not beat properly then it is unable to pump out blood effectively. This abnormal behavior causes many uncertain deaths. Cardiac arrhythmia does not show any specific symptoms hence it is very difficult to track Cardiac Arrhythmia in its early stage. Hence our proposed methodology helps in early detection & classification of Cardiac Arrhythmia which ultimately can became a superior solution in detection of Arrhythmia in its early stage. The person having Arrhythmia may show following types of heartbeat rhythms:

a. Too slowly (Bradycardia)

b. Too quickly (Tachycardia)

c. Irregular or skipping beat rhythms

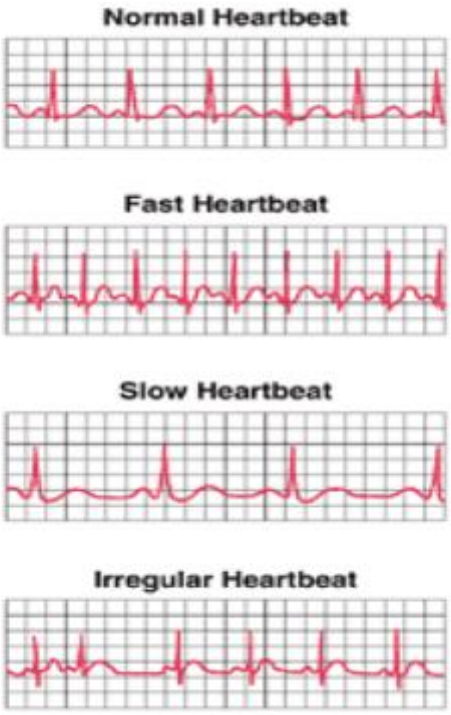


Fig 1: Various Heart beats

As shown in Figure 1, various heartbeat sequences, these are various types of signals captured from ECG sensors. As we know the heart of a healthy & fit person beats with a heartbeat rate of 60 to 100 beats per minute, while quicker heartbeat beats with heartbeat rate greater than around 100 heart beats per minute with minimal R-R interval. Slow heartbeat beats with heartbeat rate less than 60 heart beats per minute with maximum R-R interval. With the help of this heartbeat per minute & it beating sequences, we are going to detect & diagnose Cardiac Arrhythmia & its further classification.

Background:

ECG signal:

The Electrocardiogram (ECG) is an established technique in cardiology for the analysis of cardiac condition of patients. In its basic definition, ECG is the electrical representation of the contractile activity of the heart and can be recorded easily by using surface electrodes on the limbs or chest of the patient. The ECG is one of the most recognized and used biomedical signals in the field of medicine. The rhythm of the heart in terms of beats per minute (bpm) can be easily calculated by counting the R peaks of the ECG wave during one minute of recording. More importantly, rhythm and the morphology of the ECG waveform is altered by cardiovascular diseases and abnormalities such as cardiac arrhythmias, which their automatic detection and classification is the main focus of this study. ECG signal contains essential information about the cardiac pathologies affecting the heart, characterized by five peaks known as fiducial points, which are represented by the letters P, Q, R, S, and T. The QRS complex is the depolarization of the right and left heart ventricles, which is used as a reference point for signal analysis. The P wave is the result of the depolarization of the atrium, while the ventricle causes the rest of the peaks. The diagnosis of the signal relies on the morphology of the waves, as well as the duration of each peak and the segments that make up it. Therefore, detection of each section of the ECG signal is essential for health professionals in screening, diagnosis, and monitoring of several heart conditions.

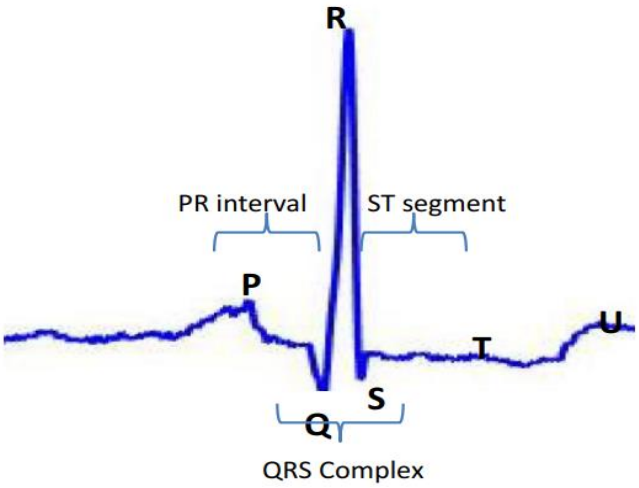


Fig 2: Signal Morphology.

Machine learning approach for detection of ECG beats:

Many methods and approaches for computer-aided ECG arrhythmia classification over the last decades, fostering productive cross-disciplinary efforts that engineers, physicists or non-linear dynamics researchers are no strangers to. Almost every computer-aided ECG classification approach involves four main steps, namely, the preprocessing of the ECG signal, the heartbeat detection, the feature extraction, and selection and finally the classifier construction. The preprocessing of the ECG signal and the heartbeat detection are out of the scope of this work, both widely studied, and the heartbeat detection is close to optimal results.

In order to accomplish arrhythmia classification, minor preprocessing needs to be applied to the source ECG records. In our system, the processing of the ECG recordings includes the following steps:

- → **ECG re-sampling**: ECG signals are processed with a common sampling rate of 250 Hz. The AHA database (250 Hz) keeps its original sampling rate, and the MIT-BIH AR database (360 Hz) is resampled to 250 Hz using the PhysioToolkit software package.
- → **ECG filtering**: All ECG recordings are filtered in a bandwidth ν (Hz) $\in$ [0.5, 35], to correct the baseline and remove unwanted high frequency noise. A Butterworth high-pass filter (with a cutoff frequency $\nu c = 0.5$ Hz) and a finite impulse response filter of 12th order (35 Hz, at 3-dB point) are used, following standard procedure.
- → **Heartbeat detection**: To determine the position of the heartbeats, the annotated positions provided by the databases are used. In the MIT-BIH AR database the annotation position occurs at the largest of the local extrema of the QRS complex. Beat detection is beyond the scope of this study. Highly accurate automated beat detection methods have already been reported.
- → **RR calculation**: The RR interval is defined as the time interval between successive heartbeats. The RR interval associated to a heartbeat i, RR(i), corresponds to the time difference between the heartbeat i and the previous heartbeat $(i - 1)$.
- → **Heartbeat segmentation**: The ECG signal is segmented around the annotated position given by each database. The size of the segmented heartbeat is 240 ms (60 samples at 250 Hz) and it is centered around the annotation position.
- → **Heartbeat normalization**: Each segmented heartbeat is normalized between $[-1, 1]$. This scaling operation results in a signal that is independent of the original ECG recording amplitude.

The classification of the ECG signal is a very important and challenging task. It can provide substantial information about the CVDs of a patient without the involvement of a

cardiologist. Only a technician is required to attach the probes, and the machine learning based solution can automatically diagnose the CVDs of the patient. This technique can immediately prioritize the patients that need urgent medical attention. Many supervised learning algorithms were used for classification such as Support vector machine classifier, Artificial-neural-network, K-nearest neighbors.

ECG database:

The MIT-BIH Arrhythmia Database contains 48 half-hour excerpts of two-channel ambulatory ECG recordings, obtained from 47 subjects studied by the BIH Arrhythmia Laboratory between 1975 and 1979. Twenty-three recordings were chosen at random from a set of 4000 24-hour ambulatory ECG recordings collected from a mixed population of inpatients (about 60%) and outpatients (about 40%) at Boston's Beth Israel Hospital; the remaining 25 recordings were selected from the same set to include less common but clinically significant arrhythmias that would not be well-represented in a small random sample.

The recordings were digitized at 360 samples per second per channel with 11-bit resolution over a 10-mV range. Two or more cardiologists independently annotated each record; disagreements were resolved to obtain the computer-readable reference annotations for each beat (approximately 110,000 annotations in all) included with the database.

The data: https://www.physionet.org/physiobank/database/mitdb/

Materials and methods:

In Fig. 3, the architectural diagram of our proposed model is depicted. The diagram illustrates the steps involved for ECG signal classification. After Pre-processing we proceeded to normalization of the database on the raw ECG signal, Filtering, heartbeat segmentation then characterization is performed. The database is then divided into training and testing for classification. Then the output from the output is validated, and other metrics are calculated.

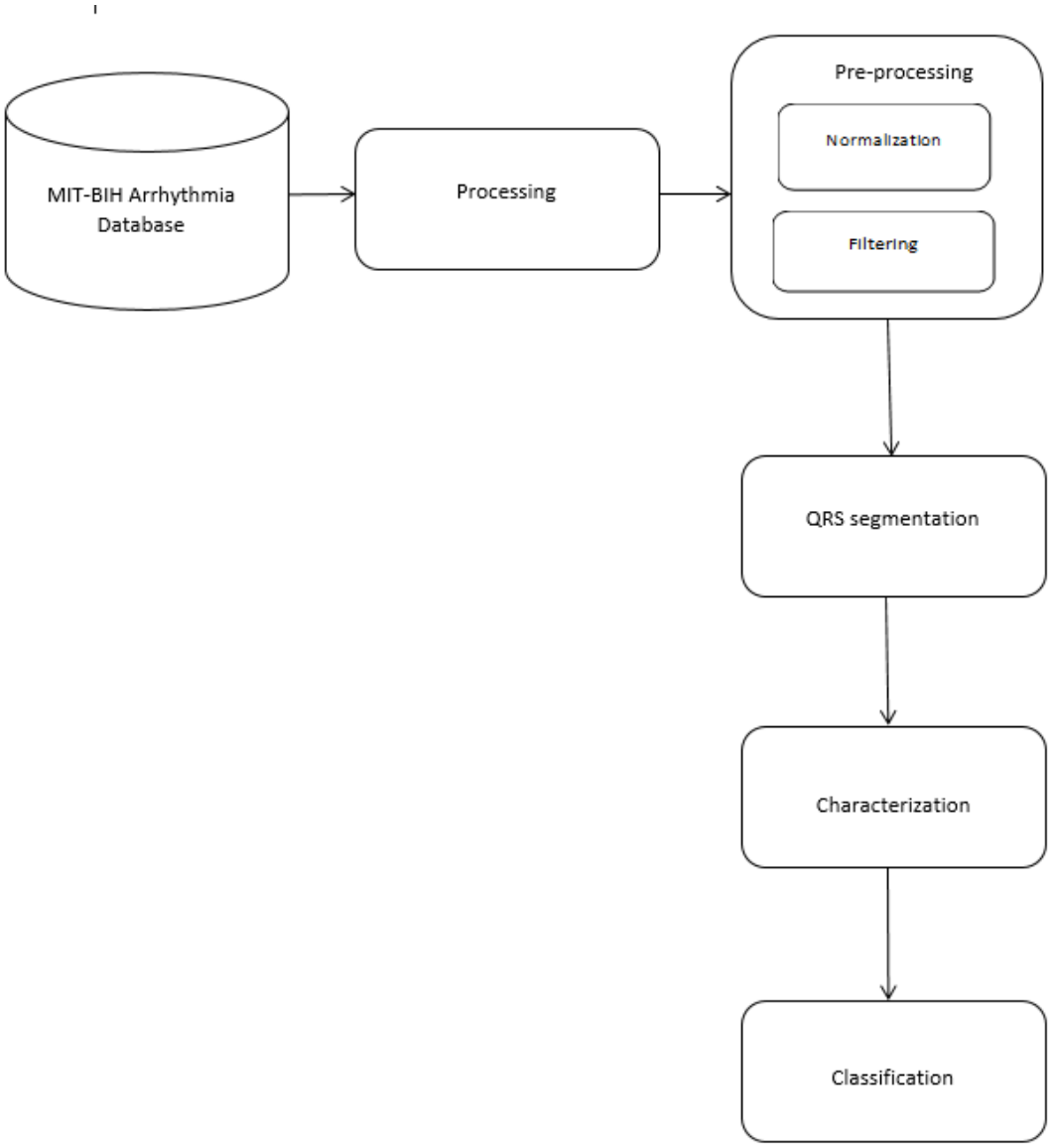


Fig 3: Workflow Diagram

Data in its raw form is not useful to any organization. Data processing is the method of collecting raw data and translating it into usable information. It is usually performed in a step-by-step process by a team of data scientists and data engineers in an organization. The raw data is collected, filtered, sorted, processed, analyzed, stored, and then presented in a readable format.

I. Processing:

1. What is Data Processing?

Data in its raw form is not useful to any organization. Data processing is the method of collecting raw data and translating it into usable information. It is usually performed in a step-by-step process by a team of data scientists and data engineers in an organization. The raw data is collected, filtered, sorted, processed, analyzed, stored, and then presented in a readable format.

Here the data that we will use “MIT-BIH Arrhythmia Database” download from physionet is not in CSV format that can be processed in python but It’s available in 3 format files:

Hea file extension text file describes the content of a signal.

```
header = wfdb.rdheader('data/mit-bih-arrhythmia-database-1.0.0/101')
display(header.__dict__)
```

```
{'record_name': '101',
 'n_sig': 2,
 'fs': 360,
 'counter_freq': None,
 'base_counter': None,
 'sig_len': 650000,
 'base_time': None,
 'base_date': None,
 'comments': ['75 F 1011 654 x1', 'Diapres'],
 'sig_name': ['MLII', 'V1'],
 'p_signal': None,
 'd_signal': None,
 'e_p_signal': None,
 'e_d_signal': None,
 'file_name': ['101.dat', '101.dat'],
 'fmt': ['212', '212'],
 'samps_per_frame': [1, 1],
 'skew': [None, None],
 'byte_offset': [None, None],
 'adc_gain': [200.0, 200.0],
 'baseline': [1024, 1024],
 'units': ['mV', 'mV'],
 'adc_res': [11, 11],
 'adc_zero': [1024, 1024],
 'init_value': [955, 992],
 'checksum': [29832, 19589],
 'block_size': [0, 0]}
```

DAT file extension is usually a generic data file that stores information specific to the application it refers in our case it stores ecg signal.

Atr file content information about Rpeaks and signal types (labels).

For that we will use the WFDB package to transform our data to CSV files.

The WFDB Python Package:

A Python-native package for reading, writing, processing, and plotting physiologic signal and annotation data. The core I/O functionality is based on the Waveform Database (WFDB)

Physiological waveforms - such as electrocardiograms (ECG), electroencephalograms (EEG), and electromyograms (EMG) - are generated during routine care. These signals contain information that can be used to understand underlying conditions of health. Effective processing and analysis of physiological data require specialized software. The waveform database (WFDB) package for Python is a library of tools for reading, writing, and processing physiological signals and annotations, written in the Python programming language. Core components of the Python package are loosely based on the specifications of the original C-language WFDB software. We aim to implement as many of the core WFDB features as possible, with user-friendly APIs. Additional features are added over time.

The original Waveform Database (WFDB) Software Package, written in C, provides a large collection of software for processing and analyzing physiological waveforms. Python is an increasingly popular language for data processing and analysis, particularly for tasks such as machine learning.

The software can be installed directly from Python using the following command:

```
$ pip install wfdb
```

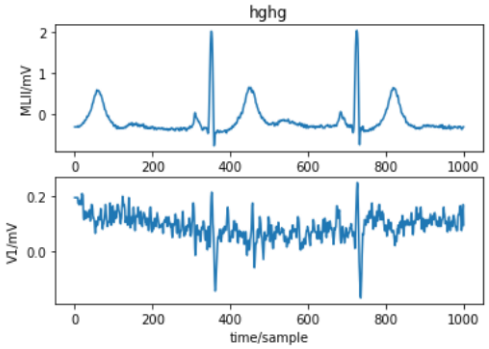
After that, we use several Signal Processing Applications like:

‘rdsamp’ prints samples from the specified record; ‘-f’ and ‘-t’ options may be used to specify a range of sample numbers, and a subset of signal numbers may be selected using the ‘-s’ option. The output of ‘rdsamp’, or any similar text, can be converted into a WFDB record using ‘wrsamp’.

rdann - read a WFDB annotation file:

rdann reads the annotation file specified by record and annotator, and writes a text-format translation of it on the standard output, one annotation per line. The output contains (from left to right) the time of the annotation in hours, minutes, seconds, and milliseconds; the time of the annotation in samples; a mnemonic for the annotation type; the annotation subtyp, chan, and num fields; and the auxiliary information string, if any (assumed to be a null-terminated ASCII string).

```
record = wfdb.rdrecord('mit-bih-arrhythmia-database-1.0.0/106',sampfrom=0,sampto=1000, channels=[0,1])
#print(record.p_signal)
wfdb.plot_wfdb(record=record,title='hghg')
```




```
data_files = ["data/" + file for file in os.listdir("data") if ".dat" in file]
```

```
def read_file(file, participant):
    """Utility function
    """
    # Get signal
    data = pd.DataFrame({wfdb.rdsamp(file[:-4])[1]['sig_name'][0]: wfdb.rdsamp(file[:-4])[0][:, 0],wfdb.rdsamp(file[:-4])[1]
    data["Sample"] = range(len(data))
    anno = wfdb.rdann(file[:-4], 'atr')
    anno = pd.DataFrame({"Rpeaks": anno.sample,"type": anno.symbol})
    return data, anno
```

II. Results and discussion:

1. Loading data:

In [5]:
```
name=100
for participant, file in enumerate(data_files):
    ecg = []
    rpeaks = []

    print("Participant: " + str(participant + 1) + "/" + str(len(data_files)))

    data, anno = read_file(file, participant)

    # Store with the rest
    ecg.append(data)
    rpeaks.append(anno)

    # Store additional recording if available
    if "x_" + file.replace("data/", "") in os.listdir("data/x_mitdb/"):
        print("  - Additional recording detected.")
        data, anno = read_file("data/x_mitdb/" + "x_" + file.replace("data/", ""), participant)
        # Store with the rest
        ecg.append(data)
        rpeaks.append(anno)

    ecg = pd.concat(ecg).to_csv(str(name)+".csv", index=False,header=["MLII", "V5", "Sample"])
    rpeaks = pd.concat(rpeaks).to_csv(str(name)+"Rpeaks.csv", index=False)

    name=name+1

# Save
#df_ecg = pd.concat(dfs_ecg).to_csv("ECGs.csv", index=False)
#df_rpeaks = pd.concat(dfs_rpeaks).to_csv("Rpeaks.csv", index=False)
print('wslat')
```

```
Participant: 1/48
Participant: 2/48
Participant: 3/48
Participant: 4/48
Participant: 5/48
Participant: 6/48
Participant: 7/48
Participant: 8/48
Participant: 9/48
  - Additional recording detected.
Participant: 10/48
```

In [6]:
```
record = pd.read_csv('100.csv')
```

In [7]:
```
record.head(20)
```

Out[7]:

	MLII	V5	Sample
0	-0.145	-0.065	0
1	-0.145	-0.065	1
2	-0.145	-0.065	2
3	-0.145	-0.065	3
4	-0.145	-0.065	4
5	-0.145	-0.065	5
6	-0.145	-0.065	6
7	-0.145	-0.065	7
8	-0.120	-0.080	8
9	-0.135	-0.080	9
10	-0.145	-0.085	10
11	-0.150	-0.085	11
12	-0.160	-0.075	12
13	-0.155	-0.070	13
14	-0.160	-0.070	14
15	-0.175	-0.065	15
16	-0.180	-0.055	16
17	-0.185	-0.050	17
18	-0.170	-0.050	18
19	-0.155	-0.040	19

In [8]:
```
round(100*(record.isnull().sum()/len(record.index)), 2)
```

Out[8]:
```
MLII      0.0
V5        0.0
Sample    0.0
dtype: float64
```

In [9]:
```
Rpeaks=pd.read_csv('100Rpeaks.csv')
```

In [10]:
```
#data analys
Rpeaks['type'].describe()
print(Rpeaks['type'].value_counts())
```

```
N    2239
A      33
+       1
V       1
Name: type, dtype: int64
```

In [13]:
```
Rpeaks.head()
```

Out[13]:

	Rpeaks	type
0	18	+
1	77	N
2	370	N
3	662	N
4	946	N

N: Normal

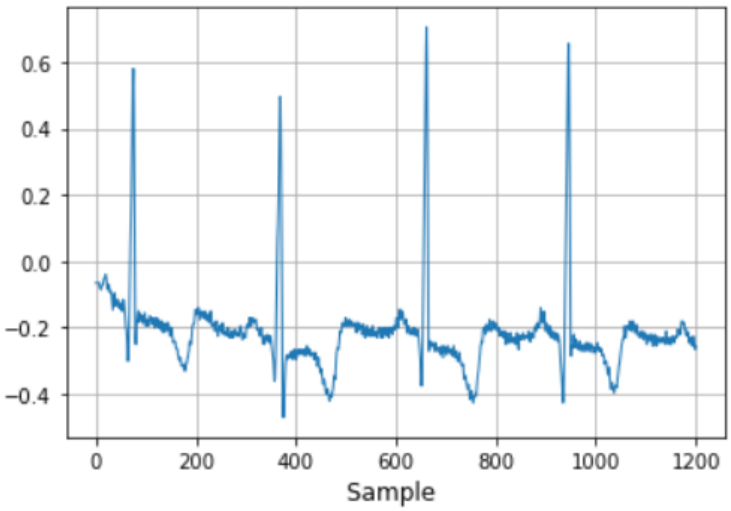
A: An atrial premature beat is an extra heartbeat caused by electrical activation of the atria (upper chambers of the heart) from an abnormal site before a normal heartbeat would occur.

V: Ventricular ectopic beat

+: Unknown beat

2. Date visualization:

In [13]:
```
%matplotlib inline
x=record['V5']
nhb=4 #number of heartbeats
lhb=300 #average length of a heartbeat
start=0
end=nhb*lhb

X=x[start:end].values
ecg=(X)
ecg=np.transpose(ecg)
fs=360 #1 sec of signal
ts=1/fs
t=np.linspace(0, np.size(ecg), np.size(ecg))
times=t*ts

where = Rpeaks['Rpeaks'].values <end
samp = Rpeaks['Rpeaks'].values[where]

types = np.array(Rpeaks['type'].values)
types = types[where]

plt.plot(t, ecg)


# Get the annotation position

    # Use just the first letter


plt.xlabel('Sample ', fontsize=12)
plt.show()
```



III. Pre-processing:

Although ECG data obtained from MIT-BIH database is not expected to contain as much disruptive noise as ECG data obtained directly from a patient; it still contains some noise which requires attention to improve the subsequent steps of the system. Therefore, signal pre-processing step is focused on removing noise from ECG recordings.

As a first step, Feature scaling is one of the most important data preprocessing steps in machine learning. Algorithms that compute the distance between the features are biased towards numerically larger values if the data is not scaled. Tree-based algorithms are insensitive to the scale of the features. Also, feature scaling helps machine learning, and deep learning algorithms train and converge faster.

There are some feature scaling techniques such as Normalization and Standardization that are the most popular. In this study Min-Max scaling is used to remove the noise present in the ECG signals.

Almost all the ECG recordings also contain high and low frequency noise that are imposed by several factors like, the contraction of the muscles, respiratory movements, poor electrode contact and presence of other external devices. To remove high frequency noise, mostly caused by patients' muscle contractions during recording, third order low-pass filter is used to pass a signal that has a frequency below the cutoff frequency, which retains some user-designated value. All signals with frequencies above the edged cutoff frequency. and the signals are filtered.

All the above steps are applied to all training and testing ECG records and filtered ECG signals are obtained and ready for the next steps.

1. Heartbeat normalization:

Heartbeat Normalization is an essential step to preprocessing as it helps us to remove unnecessary information in the form of noise and get the signal with the maximum essential information. Usually, the min-max method or technique is employed for the above purpose. Normalization or Min-Max Scaling is used to transform features to be on a similar scale. The new point is calculated as Equation described below is employed to normalize all amplitude values.

$$X\_new= (X - X\_min)/(X\_max - X\_min)$$

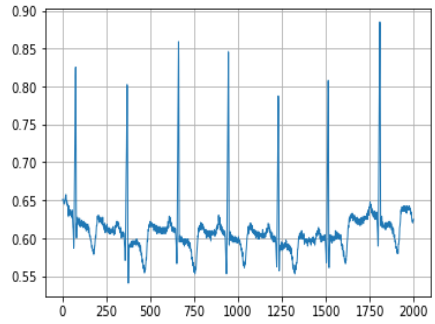
This scales the range to [0, 1] or sometimes [-1, 1]. Geometrically speaking, transformation squishes the n-dimensional data into an n-dimensional unit hypercube. Normalization is useful

when there are no outliers as it cannot cope with them. Usually, we would scale age and not income because only a few people have high incomes, but the age is close to uniform.

```
In [16]: #Normalize_all_data()
         def Normalize_all_data(file):
             df = pd.read_csv(str(file) + '.csv')
             df['MLII'] = (df['MLII'] - df.describe()['MLII']['min'])/(df.describe()['MLII']['max']-df.describe()['MLII']['min'])
             df['V5'] = (df['V5'] - df.describe()['V5']['min'])/(df.describe()['V5']['max']-df.describe()['V5']['min'])
             file = file + 1
             return df
```

Normalization

```
In [17]: df=Normalize_all_data(100)

In [18]: type(df)
Out[18]: pandas.core.frame.DataFrame

In [19]: df["V5"][0:2000].plot()
Out[19]: <AxesSubplot:>
```



2. Filtering :

Several adaptive filter structures are proposed for noise cancellation and arrhythmia detection. In this study, we are using low-pass filter that passes signals with a frequency lower than a certain cutoff frequency and attenuates signals with frequencies higher than the cutoff frequency.
The transfer function of a 3rd order low-pass filter is:

$$H(s) = \frac{A}{(1 + s/\omega_1)(1 + s/(Q\,\omega_2) + s^2/\omega_2^2)}$$

where:
A – DC gain;
ω_1 – radial frequency of the first stage, K1 × ω;
ω_2 – radial frequency of the second stage, K2 × ω;
Q – quality factor of the second stage;
ω – pass-band radial frequency of the filter;
s – complex frequency.

The adaptive filter essentially minimizes the mean-squared error between a primary input, which is the noisy ECG, and a reference input, which is either noise that is correlated in some way with the noise in the primary input or a signal that is correlated only with ECG in the primary input. Different filter structures are presented to eliminate the diverse forms of noise: baseline wander, 60 Hz power line interference, muscle noise, and motion artifact. An adaptive recurrent filter structure is proposed for acquiring the impulse response of the normal QRS complex. The primary input of the filter is the ECG signal to be analyzed, while the reference input is an impulse train coincident with the QRS complexes.

```
import scipy.io.wavfile
import scipy.signal
import numpy as np
import matplotlib.pyplot as plt

# read ECG data from the WAV file
#sampleRate, data = scipy.io.wavfile.read('ecg.wav')
sampleRate=360
times = np.arange(len(df["V5"]))/sampleRate

# apply a 3-pole lowpass filter at 0.1x Nyquist frequency
b, a = scipy.signal.butter(3, 0.1)
#filtered = scipy.signal.filtfilt(b, a, df["V5"])
df["V5"]=scipy.signal.filtfilt(b, a, df["V5"])
df["MLII"]=scipy.signal.filtfilt(b, a, df["MLII"])
```

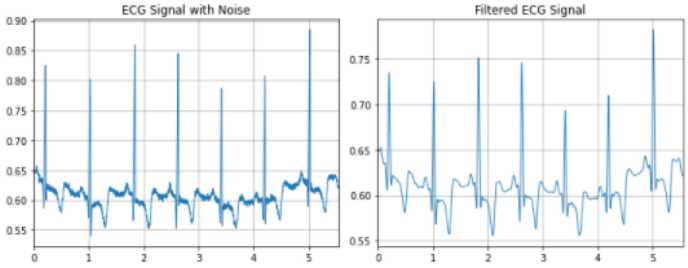
```
df["V5"]
```

```
0         0.650083
1         0.649726
2         0.649384
3         0.649076
4         0.648819
            ...
649995    0.673418
649996    0.667284
649997    0.662170
649998    0.658072
649999    0.654871
Name: V5, Length: 650000, dtype: float64
```

```
filtered[0:100]
```

```
array([0.65019323, 0.6498799 , 0.64956221, 0.6492427 , 0.64893037,
       0.64864141, 0.64839886, 0.64823076, 0.64816657, 0.64823215,
       0.64844401, 0.64880421, 0.64929734, 0.64989014, 0.65053394,
       0.65116913, 0.65173114, 0.65215732, 0.65239387, 0.65240196,
       0.65216172, 0.65167342, 0.65095503, 0.65003688, 0.64895545,
       0.64774838, 0.64645217, 0.64510186, 0.64373166, 0.64237532,
       0.64106624, 0.63983685, 0.63871731, 0.63773272, 0.63689958,
       0.63622288, 0.63569609, 0.63530495, 0.63503325, 0.63486723,
       0.63479642, 0.6348112 , 0.63489935, 0.63504304, 0.63521679,
       0.63538548, 0.63550196, 0.63550467, 0.6353169 , 0.63484946,
       0.63400715, 0.63269841, 0.63084782, 0.62841152, 0.62539584,
       0.62187753, 0.61802231, 0.61409622, 0.61046506, 0.60757891,
       0.60594162, 0.60606725, 0.60842653, 0.6133861 , 0.62114528,
       0.6316782 , 0.64469289, 0.65961895, 0.67563045, 0.69170325,
       0.70669992, 0.71947387, 0.72898478, 0.73441672, 0.73528653,
       0.73152399, 0.7235017 , 0.71199666, 0.6980809 , 0.68296178,
       0.6678118 , 0.65362901, 0.64115252, 0.63083513, 0.62286035,
       0.61718708, 0.61360716, 0.61180438, 0.61140731, 0.61203275,
       0.61331926, 0.61495112, 0.61667221, 0.61829075, 0.61967655,
       0.6207537 , 0.62149175, 0.62189725, 0.62200631, 0.6218767 ])
```

```
plt.figure(figsize=(10, 4))

plt.subplot(121)
plt.plot(times[0:2000], df["V5"][0:2000])
plt.title("ECG Signal with Noise")
plt.margins(0, .05)

plt.subplot(122)
plt.plot(times[0:2000], filtered[0:2000])
plt.title("Filtered ECG Signal")
plt.margins(0, .05)

plt.tight_layout()
plt.show()
```



3. Heartbeat segmentation:

Segmentation is one of the most important stages in ECG signal processing because the morphological pattern analysis is the starting procedure for filtration, heart rate variability studies (HRV), and beat classification for beat grouping. Due to its relevance, there is a wide range of literature regarding segmentation of ECG signals, considering that those studies are focused mainly on the fiducial point estimation analyzing the R peak, since this is the starting point for finding the P and T waves.

What is a segment?

A segment in an ECG is the region between two waves. PR segment starts at the end of the P wave and ends at the start of the QRS complex. The ST segment starts at the end of the QRS wave and ends at the start of the T wave. The TP segment is found between the end of the T wave and the beginning of the next P wave; It is the true isoelectric segment in the ECG. With segments, you talk about morphology: elevation or depression or progression of segments.

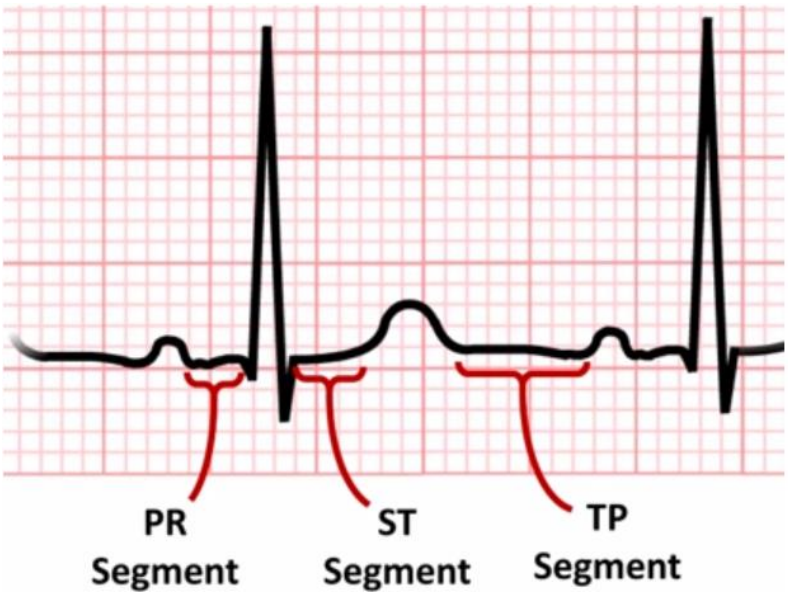
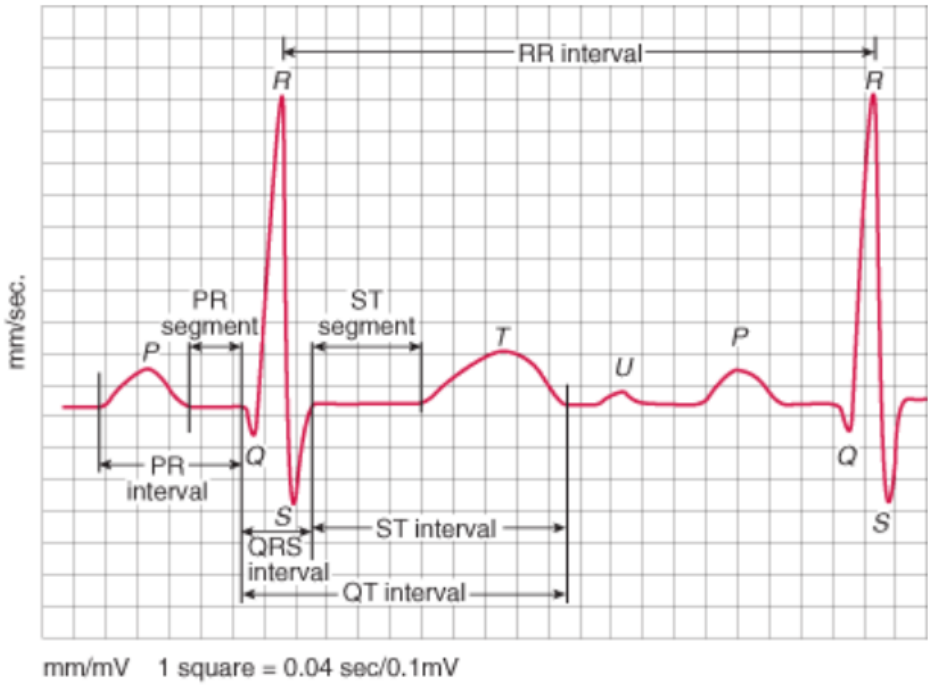


Fig 4: Segment

What are intervals?

An interval in an ECG is a duration of time that includes one segment and one or more waves. The PR (or PQ) interval starts at the start of the P wave and ends at the start of the QRS. It denotes the conduction of the impulse from the upper part of the atrium to the ventricle. The QRS interval covers the QRS complex from beginning to end. [The QRS complex also covers an interval]. The QT interval starts at the start of the QRS and ends at the end of the T wave. It denotes the electrical systole of the heart. Intervals are only described based on their duration of time. You speak of it as a duration and so cannot talk about the morphology or depression or elevation of an interval.



P wave = Atrial depolarization. The positive wave of depolarization spreads from the SA node and is conducted throughout the cells of the atria through gap junctions in that connect these cells.

PR segment = depolarization of the AV node. When current is passing through the AV node. It is a flat line because the wave is not strong enough to be recorded on the voltmeter.

PR interval = Wave goes over the atrium and through the AV node and ends just before it activates the ventricles to depolarize.

Q wave = Ventricular Septal Depolarization

R wave = Resultant or major ventricular muscle depolarization. The resultant vector is directed downward and leftward.

S Wave = Basal Ventricular depolarization; depolarization of the base of the ventricles. Note the apex of the heart is the L. pointed end. The base of the ventricles connects to the atria.

ST segment = During the ST segment, all the ventricular myocardium is depolarized. All have positive charges. there is nothing potential difference to be recorded by the voltmeter (ECG machine).

T wave= represents ventricular repolarization.

QT interval = Important because it captures the beginning of ventricular depolarization through the plateau phase to the ventricular repolarization. It covers the entire ventricular activity. During this time, the action potential was generated and terminated in the ventricular tissue. The beginning of the QRS complex is the start of ventricular systole and that goes until the end of the T wave. Ventricular diastole starts when the T wave ends.

U wave. Sometimes the electrical activity of the ventricular papillary muscle is out of phase with the rest of the ventricles and will record as a “U” wave that shows after the T wave

In [50]:
```
Rpeaks101["inter_min"] = np.nan
Rpeaks101["inter_max"] = np.nan
for i in range(2,len(Rpeaks101)-2):
    b= Rpeaks101['Rpeaks'][i]
    a= Rpeaks101['Rpeaks'][i-1]
    #[c1,c2] represent the heartbeat interval
    c1=b-int(abs(0.3*(b-a)))
    c2=int(abs(0.6*(a-b)))+b
    Rpeaks101['inter_min'][i]=c1
    Rpeaks101['inter_max'][i]=c2
```

In [51]:
```
Rpeaks101
```

Out[51]:

	Rpeaks	type	inter_min	inter_max
0	18	+	NaN	NaN
1	77	N	NaN	NaN
2	370	N	283.0	545.0
3	662	N	575.0	837.0
4	946	N	861.0	1116.0
...	...	...	...	...
2269	648978	N	648905.0	649125.0
2270	649232	N	649156.0	649384.0
2271	649484	N	649409.0	649635.0
2272	649734	N	NaN	NaN
2273	649991	N	NaN	NaN

2274 rows × 4 columns

4. Segmentation of QRS complexes

In ECG signal processing, a remarkable stage for identifying cardiac pathologies or analyzing HRV series corresponds to the detection of main waves and complexes of the ECG signals, such as P wave, T wave and QRS complex. some methods to estimate the fiducial point of a ECG signal for the posterior wave detection are described. QRS complex is calculated with a symmetric window around the detected R peak and the HRV is calculated considering every distance between consecutive R peaks.

ci = y(pi – α*Fs : pi + β*Fs)

pi : is the location of the R-peak of the i-nth beat of the recording.

Ci : is the QRS complex of the i-th beat i.

Fs: is the sampling rate

5. Characterization:

The main objective of this stage is to extract the information with greater discriminating power from an ECG signal, eliminating the information that is irrelevant in its recognition. The removal
of characteristics is one of the stages in which the good performance of the cardiac arrhythmia recognition system. Next, it is explained in detail several techniques that have shown good performance in previous research.

To avoid analysis over QRS complexes of different length, their segmentation is carried out for a fixed window length, that is, each j–th complex dj is accomplished as follows,

$$\begin{aligned} x_1 &= l_j - l_{j-1}, && (RR \text{ interval}) \\ x_2 &= l_{j-1} - l_{j-2}, && (pre-RR \text{ interval}) \\ x_3 &= l_{j+1} - l_j, && (post-RR \text{ interval}) \end{aligned}$$

where lj is the R–peak time location of the j-th heartbeat and Fs = 1/Ts is the sampling frequency. Nonetheless, it must be quoted that some morphologies might exhibit S– waves lasting exceptionally more than usual, and therefore they can be missed if using such a short processing window, then, as usually recommend, QRS width is fixed to be of 200 ms length.

5.1. **HRV :**

Heart rate variability (HRV) provides indirect insight into autonomic nervous system tone and has a well-established role as a marker of cardiovascular risk. Recent decades brought an increasing interest in HRV assessment as a diagnostic tool in detection of autonomic impairment, and prediction of prognosis in several neurological disorders. Both bedside analysis of simple markers of HRV, as well as more sophisticated HRV analyses including

time, frequency domain and nonlinear analysis have been proven to detect early autonomic involvement in several neurological disorders. Furthermore, altered HRV parameters were shown to be related with cardiovascular risk, including sudden cardiac risk, in patients with neurological diseases.

– HRV–derived features (x1, x2, x3): Interval parameters providing information about sequences of heartbeats with unusual timing, namely:

$$
\begin{aligned}
x_1 &= l_j - l_{j-1}, && (RR \text{ interval}) \\
x_2 &= l_{j-1} - l_{j-2}, && (pre-RR \text{ interval}) \\
x_3 &= l_{j+1} - l_j, && (post-RR \text{ interval})
\end{aligned}
$$

It should be remarked that atrial (S) and ventricular (V) ectopic beats manifest abrupt changes on fiducial point intervals, which in turn, affect the respective values of heartbeat interval features.

In [60]:
```
##RR interval
Rpeaks101["RR_interval"] = np.nan
for i in range(1,len(Rpeaks101)):
    start= Rpeaks101['Rpeaks'][i]
    end= Rpeaks101['Rpeaks'][i-1]
    Rpeaks101['RR_interval'][i]=start-end

print(len(Rpeaks101['RR_interval']))
```

```
2274
```

In [61]:
```
Rpeaks101
```

Out[61]:

	Rpeaks	type	inter_min	inter_max	pre_RR	post_RR	RR_interval
0	18	+	NaN	NaN	NaN	59.0	NaN
1	77	N	NaN	NaN	NaN	293.0	59.0
2	370	N	283.0	545.0	59.0	292.0	293.0
3	662	N	575.0	837.0	293.0	284.0	292.0
4	946	N	861.0	1116.0	292.0	285.0	284.0
...	...	...	...	...	...	...	...
2269	648978	N	648905.0	649125.0	256.0	254.0	245.0
2270	649232	N	649156.0	649384.0	245.0	252.0	254.0
2271	649484	N	649409.0	649635.0	254.0	250.0	252.0
2272	649734	N	NaN	NaN	252.0	257.0	250.0
2273	649991	N	NaN	NaN	250.0	NaN	257.0

2274 rows × 7 columns

In [62]:
```
#pre-RR
Rpeaks101["pre_RR"] = np.nan
for i in range(2,len(Rpeaks101)):
    start= Rpeaks101['Rpeaks'][i-1]
    end= Rpeaks101['Rpeaks'][i-2]
    Rpeaks101['pre_RR'][i]=start-end

print(len(Rpeaks101['pre_RR']))
```

```
2274
```

In [63]:
```
Rpeaks101
```

Out[63]:

	Rpeaks	type	inter_min	inter_max	pre_RR	post_RR	RR_interval
0	18	+	NaN	NaN	NaN	59.0	NaN
1	77	N	NaN	NaN	NaN	293.0	59.0
2	370	N	283.0	545.0	59.0	292.0	293.0
3	662	N	575.0	837.0	293.0	284.0	292.0
4	946	N	861.0	1116.0	292.0	285.0	284.0
...	...	...	...	...	...	...	...
2269	648978	N	648905.0	649125.0	256.0	254.0	245.0
2270	649232	N	649156.0	649384.0	245.0	252.0	254.0
2271	649484	N	649409.0	649635.0	254.0	250.0	252.0
2272	649734	N	NaN	NaN	252.0	257.0	250.0
2273	649991	N	NaN	NaN	250.0	NaN	257.0

2274 rows × 7 columns

In [64]:
```
#post-RR
Rpeaks101["post_RR"] = np.nan
for i in range(0,len(Rpeaks101)-1):
    start= Rpeaks101['Rpeaks'][i+1]
    end= Rpeaks101['Rpeaks'][i]
    Rpeaks101['post_RR'][i]=start-end

print(len(Rpeaks101['post_RR']))
```

```
2274
```

In [65]:
```
Rpeaks101
```

Out[65]:

	Rpeaks	type	inter_min	inter_max	pre_RR	post_RR	RR_interval
0	18	+	NaN	NaN	NaN	59.0	NaN
1	77	N	NaN	NaN	NaN	293.0	59.0
2	370	N	283.0	545.0	59.0	292.0	293.0
3	662	N	575.0	837.0	293.0	284.0	292.0
4	946	N	861.0	1116.0	292.0	285.0	284.0
...	...	...	...	...	...	...	...
2269	648978	N	648905.0	649125.0	256.0	254.0	245.0
2270	649232	N	649156.0	649384.0	245.0	252.0	254.0
2271	649484	N	649409.0	649635.0	254.0	250.0	252.0
2272	649734	N	NaN	NaN	252.0	257.0	250.0
2273	649991	N	NaN	NaN	250.0	NaN	257.0

2274 rows × 7 columns

5.2. Prematurity

Prematurity features (x4, x5, x6): Defined parameters, x4 = x1 - x2 and

x5 =x3 - x1, are assumed to be relevant since they make possible the identification of S arrhythmia type, when reflecting the increase or decrease of the heart rate. Besides, if any heartbeat occurring after another S–labeled event is regarded as normal, the above couple of features will change of sign. Feature x6 accounts for

the number of consecutive S that is also sensitive to an increase of the heart rate, exceeding the normal range set for x4. The parameter x6 is expressed as follows:

$$x_6 = \left(\frac{x_3}{x_1}\right)^2 + \left(\frac{x_2}{x_1}\right)^2 - \left(\frac{1}{3}\sum_{i=1}^{3} x_i^2 \log(x_i)^2\right).$$

Besides, the first and second squared terms are sensitive to abrupt changes of heart rate, whereas the last addend is inferred as unnormalized Shannon entropy, which increases the value of x6 whenever heart rate is steadily increasing.

```
In [66]: #Diference between RR and pre-RR intervals
         Rpeaks101['RR_preRR']=Rpeaks101['RR_interval']-Rpeaks101['pre_RR']

In [67]: #Diference between post-RR and RR intervals
         Rpeaks101['postRR_RR']=Rpeaks101['post_RR']-Rpeaks101['RR_interval']

In [68]: Rpeaks101
```

Out[68]:

	Rpeaks	type	inter_min	inter_max	pre_RR	post_RR	RR_interval	RR_preRR	postRR_RR
0	18	+	NaN	NaN	NaN	59.0	NaN	NaN	NaN
1	77	N	NaN	NaN	NaN	293.0	59.0	NaN	234.0
2	370	N	283.0	545.0	59.0	292.0	293.0	234.0	-1.0
3	662	N	575.0	837.0	293.0	284.0	292.0	-1.0	-8.0
4	946	N	861.0	1116.0	292.0	285.0	284.0	-8.0	1.0
...	...	...	...	...	...	...	...	...	...
2269	648978	N	648905.0	649125.0	256.0	254.0	245.0	-11.0	9.0
2270	649232	N	649156.0	649384.0	245.0	252.0	254.0	9.0	-2.0
2271	649484	N	649409.0	649635.0	254.0	250.0	252.0	-2.0	-2.0
2272	649734	N	NaN	NaN	252.0	257.0	250.0	-2.0	7.0
2273	649991	N	NaN	NaN	250.0	NaN	257.0	7.0	NaN

2274 rows × 9 columns

5.3. Morphology

Since most analyzed arrhythmias change the shape of the QRS complex, their characterization can be attained by commonly used time and spectral based techniques. Therefore, regarding the former techniques, the following features are worth to be considered: A couple of features that are sensitive to abnormal QRS complexes: x7, which computes a morphological dissimilarity by means of Dynamic Time Warping (DTW) approach between the current QRS complex, and a linearly averaged QRS complex of the last n heartbeats. Next, a parameter, which is sensitive to ventricular arrhythmias exhibiting abnormal QRS complexes such as ventricular extrasystoles (V) or branch blocks (N), is defined as x8 = |max{dj}/min{dj}|, being dj the current QRS complex. In addition, since the morphological notoriety of branch block heartbeats, the QRS energy, which

is a straightforward feature to detect the previously described types of heartbeats are estimated as x9 = PL i=1 d dj[i]2 , where Ld is the processing length of the j-th QRS complex.

On the other hand, spectral–based representation features that have been used in the field of signal compression are also considered, because only few coefficients are needed to reconstruct the signal. In this line of analysis, the Hermite coefficient are used and can be computed. The elements of Hermite base is ranged in intervals (-t0, t0) with t0 = 100 ms, in order to set the length of window to be 200 ms.

Morphology

In [128]:
```
x6=[]
x7=[]
x8=[]
for i in range(len(QRS)-1):
    x6.append(pow(QRS[i],2).sum())  # QRS complex energy
    x7.append(abs(QRS[i].max()/QRS[i].where(lambda x: x != 0).min())) #QRS complex polarity
    x8.append(QRS[i].var())  #QRS Variance

x6 = np.array(x6)
x7 = np.array(x7)
x8 = np.array(x8)
```

In [ ]:
```
```

In [129]:
```
x6[0]
```

Out[129]:
```
27.149942052097508
```

In [130]:
```
#Ts=360
#QRS=Pj -(Ts/10)
a=36
b=36
Rpeaks101["QRS_min"] = np.nan
Rpeaks101["QRS_max"] = np.nan

for i in range(2, len(Rpeaks101)-1):
    Rpeaks101["QRS_min"][i]=Rpeaks101['Rpeaks'][i]-a
    Rpeaks101["QRS_max"][i]=Rpeaks101['Rpeaks'][i]+b
```

In [131]:
```
Rpeaks101
```

Out[131]:

	Rpeaks	type	RR_interval	pre_RR	post_RR	RR_preRR	postRR_RR	inter_min	inter_max	QRS_min	QRS_max
0	18	+	NaN	NaN	59.0	NaN	NaN	NaN	NaN	NaN	NaN
1	77	N	59.0	NaN	293.0	NaN	234.0	NaN	NaN	NaN	NaN
2	370	N	293.0	59.0	292.0	234.0	-1.0	283.0	545.0	334.0	406.0
3	662	N	292.0	293.0	284.0	-1.0	-8.0	575.0	837.0	626.0	698.0
4	946	N	284.0	292.0	285.0	-8.0	1.0	861.0	1116.0	910.0	982.0
...	...	...	...	...	...	...	...	...	...	...	...
2269	648978	N	245.0	256.0	254.0	-11.0	9.0	648905.0	649125.0	648942.0	649014.0
2270	649232	N	254.0	245.0	252.0	9.0	-2.0	649156.0	649384.0	649196.0	649268.0
2271	649484	N	252.0	254.0	250.0	-2.0	-2.0	649409.0	649635.0	649448.0	649520.0
2272	649734	N	250.0	252.0	257.0	-2.0	7.0	NaN	NaN	649698.0	649770.0
2273	649991	N	257.0	250.0	NaN	7.0	NaN	NaN	NaN	NaN	NaN

2274 rows × 11 columns

In [132]:
```
Rpeaks101["polarity"] = np.nan
Rpeaks101["energy"] = np.nan
Rpeaks101["variance"] = np.nan

for i in range(2, len(Rpeaks101)-2):
    Rpeaks101["energy"][i]=x6[i-2]
    Rpeaks101["polarity"][i]=x7[i-2]
    Rpeaks101["variance"][i]=x8[i-2]
```

In [133]:
```
Rpeaks101
```

Out[133]:

	Rpeaks	type	RR_interval	pre_RR	post_RR	RR_preRR	postRR_RR	inter_min	inter_max	QRS_min	QRS_max	polarity	energy	variance
0	18	+	NaN	NaN	59.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	77	N	59.0	NaN	293.0	NaN	234.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	370	N	293.0	59.0	292.0	234.0	-1.0	283.0	545.0	334.0	406.0	1.274258	27.149942	0.001726
3	662	N	292.0	293.0	284.0	-1.0	-8.0	575.0	837.0	626.0	698.0	1.282246	28.147244	0.002275
4	946	N	284.0	292.0	285.0	-8.0	1.0	861.0	1116.0	910.0	982.0	1.304503	27.858090	0.002185
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2269	648978	N	245.0	256.0	254.0	-11.0	9.0	648905.0	649125.0	648942.0	649014.0	1.219892	29.736719	0.001398
2270	649232	N	254.0	245.0	252.0	9.0	-2.0	649156.0	649384.0	649196.0	649268.0	1.216435	28.628887	0.001244
2271	649484	N	252.0	254.0	250.0	-2.0	-2.0	649409.0	649635.0	649448.0	649520.0	1.260504	28.708300	0.001910
2272	649734	N	250.0	252.0	257.0	-2.0	7.0	NaN	NaN	649698.0	649770.0	NaN	NaN	NaN
2273	649991	N	257.0	250.0	NaN	7.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

2274 rows × 14 columns

IV. ECG classification

This section details the algorithms necessary for the correct processing, reduction, and classification of ECG data. This stage is of vital importance in the detection process of cardiac arrhythmias.

1. Data cleaning :

```
dfEcg=df = pd.DataFrame(Rpeaks101)
dfEcg.drop(['inter_min','inter_max','QRS_min','QRS_max'], axis = 1, inplace = True)
dfEcg = dfEcg.drop(dfEcg.index[[0,1]])
dfEcg = dfEcg.drop(dfEcg.index[[len(dfEcg)-2,len(dfEcg)-1]])
dfEcg = dfEcg.reset_index(level=0)
dfEcg.drop(['index'], axis = 1, inplace = True)
```

```
round(100*(dfEcg.isnull().sum()/len(Rpeaks.index)), 2)
```

```
Rpeaks          0.0
type            0.0
sampling_rate   0.0
RR_interval     0.0
pre_RR          0.0
post_RR         0.0
RR_preRR        0.0
postRR_RR       0.0
polarity        0.0
energy          0.0
variance        0.0
dtype: float64
```

```
dfEcg['polarity'] = dfEcg['polarity'].fillna(dfEcg['polarity'].mean())#replace nan value with mean
dfEcg['variance'] = dfEcg['variance'].fillna(dfEcg['variance'].mean())#replace nan value with mode
```

```
round(100*(dfEcg.isnull().sum()/len(Rpeaks.index)), 2)
```

```
Rpeaks          0.0
type            0.0
sampling_rate   0.0
RR_interval     0.0
pre_RR          0.0
post_RR         0.0
RR_preRR        0.0
postRR_RR       0.0
polarity        0.0
energy          0.0
variance        0.0
dtype: float64
```

```
dfEcg
```

	Rpeaks	type	sampling_rate	RR_interval	pre_RR	post_RR	RR_preRR	postRR_RR	polarity	energy	variance
0	370	N	293	293.0	59.0	292.0	234.0	-1.0	1.483709	27.232405	0.003063
1	662	N	292	292.0	293.0	284.0	-1.0	-8.0	1.516746	28.260137	0.003952
2	946	N	284	284.0	292.0	285.0	-8.0	1.0	1.529412	27.961235	0.003753
3	1231	N	285	285.0	284.0	284.0	1.0	-1.0	1.413625	26.589805	0.002152
4	1515	N	284	284.0	285.0	294.0	-1.0	10.0	1.439614	27.261020	0.002554
...	...	...	...	...	...	...	...	...	...	...	...
2265	648477	N	274	274.0	269.0	256.0	5.0	-18.0	1.482843	28.834560	0.003045
2266	648733	N	256	256.0	274.0	245.0	-18.0	-11.0	1.517730	29.253700	0.003976
2267	648978	N	245	245.0	256.0	254.0	-11.0	9.0	1.443678	29.843747	0.002887
2268	649232	N	254	254.0	245.0	252.0	9.0	-2.0	1.413712	28.715284	0.002496
2269	649484	N	252	252.0	254.0	250.0	-2.0	-2.0	1.450472	28.808115	0.003370

2270 rows × 11 columns

```
def min_max_scaling(df_norm):
    listdf=['Rpeaks','RR_interval','pre_RR','post_RR','RR_preRR','postRR_RR','polarity','energy','variance']
    # apply min-max scaling
    for column in listdf:
        df_norm[column] = (df_norm[column] - df_norm[column].min()) / (df_norm[column].max() - df_norm[column].min())

    return df_norm

# call the min_max_scaling function
dfEcg_normalized = min_max_scaling(dfEcg)
dfEcg_normalized
```

	Rpeaks	type	sampling_rate	RR_interval	pre_RR	post_RR	RR_preRR	postRR_RR	polarity	energy	variance
0	0.000000	N	293	0.479452	0.000000	0.474886	1.000000	0.363905	0.006631	0.490143	0.058245
1	0.000450	N	292	0.474886	0.672414	0.438356	0.343575	0.343195	0.007118	0.527039	0.075241
2	0.000887	N	284	0.438356	0.669540	0.442922	0.324022	0.369822	0.007305	0.516308	0.071437
3	0.001326	N	285	0.442922	0.646552	0.438356	0.349162	0.363905	0.005598	0.467073	0.040815
4	0.001764	N	284	0.438356	0.649425	0.484018	0.343575	0.396450	0.005981	0.491170	0.048506
...	...	...	...	...	...	...	...	...	...	...	...
2265	0.998449	N	274	0.392694	0.603448	0.310502	0.360335	0.313609	0.006618	0.547661	0.057900
2266	0.998843	N	256	0.310502	0.617816	0.260274	0.296089	0.334320	0.007132	0.562709	0.075700
2267	0.999220	N	245	0.260274	0.566092	0.301370	0.315642	0.393491	0.006041	0.583892	0.054880
2268	0.999612	N	254	0.301370	0.534483	0.292237	0.371508	0.360947	0.005599	0.543379	0.047389
2269	1.000000	N	252	0.292237	0.560345	0.283105	0.340782	0.360947	0.006141	0.546712	0.064116

2270 rows × 11 columns

2. KNN CLASSIFIER :

kNN algorithm: k- Nearest Neighbour might be the most simplest of all the classification algorithms. This method is usually used to carry out classifications. The classification is performed by finding the minimum distance from a data set which contains the input or training data and data set which contains the reference values.

K-nearest-neighbour classification was developed from the need to perform discriminant analysis. For the classification to occur:

- Assume the availability of training data
- Size of the training data set (k)

For each training data item, distance is calculated using distance equation and then arranged in ascending order. x and y are the true and measured value of ECG parameters and N is the number of values taken:

$$d(x_i,\ x_j) \equiv \sqrt{\sum_{r=1}^{n}(a_r(x_i) - a_r(x_j))^2}.$$

As this method is intuitive, simple to implement and understandable, it is used for biomedical research purposes.

Training and testing data:

```
X = np.array(dfEcg_normalized.loc[:, dfEcg_normalized.columns != 'type'])
y = np.array(dfEcg_normalized.loc[:, dfEcg_normalized.columns == 'type']).reshape(-1, 1)

# split into training and testing datasets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 2, shuffle = True)
```

```
#Grid search
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
#creat a new KNN model
Knn2 = KNeighborsClassifier()
grid_param={'n_neighbors': range(1,31),
 'weights' : ['uniform', 'distance'],
 'metric' : ['euclidean', 'manhattan', 'minkowski']}
grid = GridSearchCV(Knn2, grid_param, cv = 10, scoring = 'accuracy')
grid.fit(X_train,y_train)
print(grid.best_score_)
print(grid.best_params_)
print(grid.best_estimator_)
```

```
0.9760739912403317
{'metric': 'manhattan', 'n_neighbors': 1, 'weights': 'uniform'}
KNeighborsClassifier(metric='manhattan', n_neighbors=1)
```

```
from sklearn.metrics import accuracy_score
y_grid = grid.predict(X_test)
accuracy_score(y_test,y_grid)
```

```
0.9681020733652312
```

```
y_grid
```

```
array(['N', 'N', 'V', 'N', 'N', 'N', 'N', 'N', 'V', 'N', 'N', 'N', 'N',
       'N', 'N', 'N', 'N', 'V', 'V', 'V', 'V', 'V', 'N', 'N', 'N', 'N',
       'N', 'N', 'N', 'V', 'N', 'N', 'N', 'N', 'V', 'V', 'N', 'N', 'N',
       'N', 'N', 'N', 'N', 'V', 'N', 'N', 'N', 'N', 'V', 'N', 'N', 'V', 'V',
       'N', 'V', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'V', 'V', 'V',
       '+', 'N', 'N', 'N', 'V', 'V', 'N', '+', 'V', 'N', 'N', 'V', 'N',
       'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'V', 'N', 'N', 'N', 'N',
       'N', '+', 'N', 'N', 'N', 'N', 'N', 'V', 'V', 'N', 'N', 'N', 'N',
       'N', 'N', 'V', 'N', 'V', 'N', 'N', 'V', 'N', 'N', 'N', 'N', 'N',
       'N', 'N', 'N', 'N', 'N', 'N', 'V', 'N', 'N', 'V', 'N', 'N', 'N',
       'N', 'N', 'V', 'N', 'N', 'N', 'V', 'N', 'N', 'N', 'N', 'N', 'N',
       'N', 'N', 'V', 'V', 'V', 'N', 'N', 'N', 'V', 'V', 'N', 'V', 'N',
       '+', 'N', 'N', '+', 'V', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
       'N', 'N', 'V', 'N', 'N', 'N', 'N', 'V', 'V', 'N', 'N', 'V', 'N',
       'V', 'N', 'N', 'N', 'N', 'N', 'N', '+', 'N', 'N', 'N', 'V', 'N',
       'V', 'N', 'N', 'N', 'V', 'N', 'N', '+', 'N', 'N', '+', 'V', 'N',
       'N', '+', 'N', 'N', 'N', 'N', 'N', 'V', 'V', 'N', 'N', 'N', 'N',
       'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'V', 'N', 'V', 'N', 'N',
       'N', 'V', 'V', 'V', 'N', 'N', 'N', 'N', 'N', 'N', 'V', 'N', 'N',
       'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', '+',
       'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', '+', 'N', 'N', '+',
       'N', 'N', 'N', 'N', 'N', '+', 'N', 'N', '+', 'N', 'N', 'N', 'N',
       'V', 'V', 'N', 'N', 'N', 'N', 'N', 'V', 'V', 'N', 'N', 'V', 'N',
       'N', 'V', 'N', 'N', 'N', 'N', 'N', 'V', 'N', 'N', 'N', 'V', 'V',
       'N', 'N', 'V', 'N', 'N', 'V', 'N', 'V', 'V', 'N', 'N', 'N', 'V',
       'N', 'N', 'N', 'N', '+', 'V', 'N', 'N', 'V', 'V', 'N', 'V', 'N',
       'V', 'N', 'N', 'V', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
       'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', '+', 'N', 'N', 'N',
       'V', 'N', 'V', 'N', 'N', 'N', 'N', 'V', 'N', 'N', 'V', 'V', 'N',
       '+', 'N', 'N', 'V', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
       'N', 'N', 'N', 'N', 'N', 'N', 'N', 'V', 'N', 'N', 'V', 'N', 'N',
       'V', 'N', 'N', 'V', 'N', 'N', 'N', 'N', 'V', 'N', 'N', 'N', 'N',
       'N', 'N', 'V', 'V', 'N', 'N', 'N', 'V', 'N', 'V', 'V', 'N', '+',
       'N', 'V', 'N', 'V', 'V', 'N', 'V', 'N', 'V', 'V', 'N', 'N', '+',
       'N', 'N', 'N', 'N', 'V', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'V',
       'V', 'N', '+', 'V', 'N', 'N', 'N', 'N', '+', 'N', 'N', '+', 'N',
       'V', 'N', 'N', '+', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
       'N', 'N', 'V', 'V', 'N', 'N', 'N', 'N', 'V', 'V', 'N', 'N', 'N',
       'V', 'N', 'N', 'V', 'N', 'N', 'N', 'V', 'N', 'V', 'N', '+', 'N',
       'N', 'N', 'N', 'V', 'N', 'V', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
       'N', '+', 'N', 'N', 'N', 'V', 'N', 'N', 'N', '+', 'N', 'N', 'N',
       '+', 'N', 'N', 'N', 'V', 'N', 'N', 'V', 'N', 'N', 'N', 'N', 'N',
       'N', 'N', 'N', 'N', '+', 'N', 'N', 'N', 'V', 'N', 'N', 'V', 'V',
       'N', 'V', 'V', '+', 'N', 'V', 'N', 'N', 'N', 'V', '+', 'N', 'N',
       'N', 'V', '+', 'N', 'N', 'N', 'N', 'N', 'V', 'N', 'N', 'N', 'N',
       'N', 'V', 'N', 'V', 'N', 'N', 'N', 'N', 'V', 'N', 'N', '+', 'N',
       'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
       'N', 'V', 'N', 'V', 'N', 'N', 'N', 'V', 'N', 'N', 'V', 'N', 'N',
       'N', 'V', 'V'], dtype=object)
```

```
from sklearn.metrics import accuracy_score ,classification_report
print(classification_report(y_test,y_grid))
```

```
              precision    recall  f1-score   support

           +       0.81      0.68      0.74        38
           N       0.98      0.99      0.98       455
           V       0.98      0.98      0.98       133
           ~       0.00      0.00      0.00         1

    accuracy                           0.97       627
   macro avg       0.69      0.66      0.68       627
weighted avg       0.96      0.97      0.97       627
```

Results discussion:

The kNN algorithm offers good classification performance in most recordings for the detection 4 types of cardiac arrhythmias.

3. SVM:

A Support Vector Machine (SVM) is a supervised learning algorithm that has been shown to have good performance as a classifier. The SVM Algorithm iterates over a set of labeled training samples to find a hyperplane that produces an optimal decision boundary by finding data points, known as support vectors, that maximizes the separation between classes.

To gauge the performance of the classifier an F1 score is computed, which is a useful measure of the level of precision and recall in a machine learning system. This can easily be extended to multiclass problems by calculating the averages of scores for the classes in question. Precision is the portion of instances among the classified instances that are relevant, while recall or sensitivity is the fraction of correctly classified relevant instances that have been retrieved over the total amount of relevant instances. An algorithm with high precision over a data set will return more relevant results than irrelevant ones. For cardiac diagnosis this is critical as false positive and false negative errors should be avoided. Precision is the ratio of correctly classified true positives tp, over the sum of true positives tp and falsely classified positives fp:

$$Precision = \frac{t_p}{t_p + f_p}$$

An algorithm with high recall will classify most of the relevant data correctly and can be thought of as the ratio of correctly classified true positives tp, over the sum of true positives tp and false negatives fn (the number of instances falsely classified as negative instances):

$$Recall = \frac{t_p}{t_p + f_n}$$

There is usually a trade-off between precision and recall as it is possible to have an algorithm with high precision but low recall and vice versa. For example, the algorithm may be precise by correctly classifying a subset of arrythmia cases, however if it could achieve this by being stringent in its classification and could exclude many other cases, which would give it a low recall. The balance between precision and recall can be captured using an F1 score which is the harmonic mean of the precision and recall scores, where a score of 1 indicates perfect precision and recall:

$$F_1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}}$$

```
from sklearn import svm
import pandas as pd
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')


support = svm.LinearSVC(random_state=20)

# Train the model using the training sets and check score on test dataset
support.fit(X_train, y_train)
predicted= support.predict(X_test)
score=accuracy_score(y_test,predicted)
print("Your Model Accuracy is", score)
```

```
Your Model Accuracy is 0.9665071770334929
```

```
from sklearn.metrics import accuracy_score ,classification_report
print(classification_report(y_test,predicted))
```

```
              precision    recall  f1-score   support

           +       0.95      0.47      0.63        38
           N       0.96      1.00      0.98       455
           V       1.00      1.00      1.00       133
           ~       0.00      0.00      0.00         1

    accuracy                           0.97       627
   macro avg       0.73      0.62      0.65       627
weighted avg       0.96      0.97      0.96       627
```

Support vector machine (SVM) method was used for classification in this work.

for this transfer. We used linear Support Vector Classification which is similar to SVC with parameter kernel=’linear’, but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples.

The aim is to find a group separator when the groups distance from the separator is the largest possible. SVM use to

search for separator support vectors, which are the points near the separator.

The basic version of the SVM is applicable for the separation of only five groups of cardiac arrhythmias types (+ N V ~) using one vs one method (There are several modified methods for applying SVM to separate multiple groups).

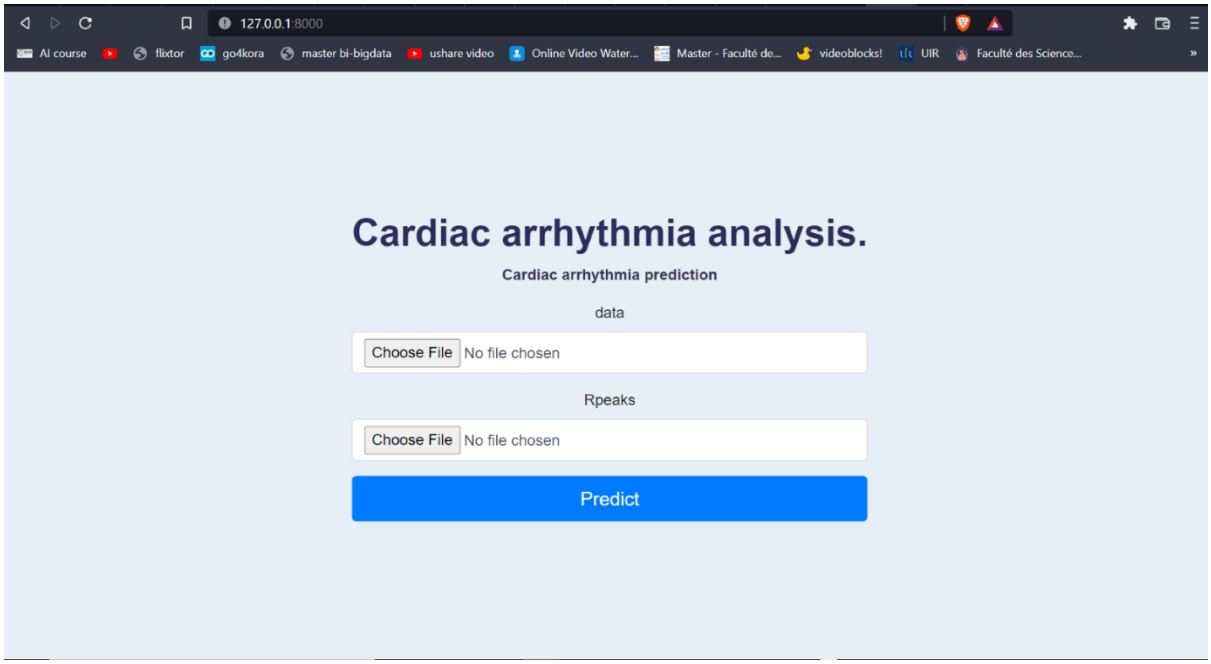
Results and discussion:

Classification accuracy is described by variables: accuracy (Acc), sensitivity (Se), specificity (Sp).

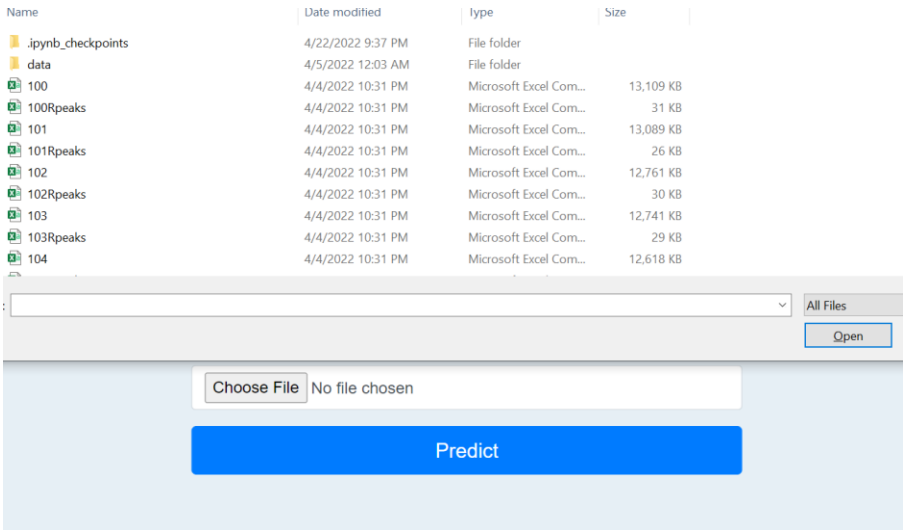
In this paper, SVM has shown a great performance with an accuracy score of 0.96 on the test set. meaning that our model made only 4 misclassifications out of 100 correct ones.

V. Graphical interface:

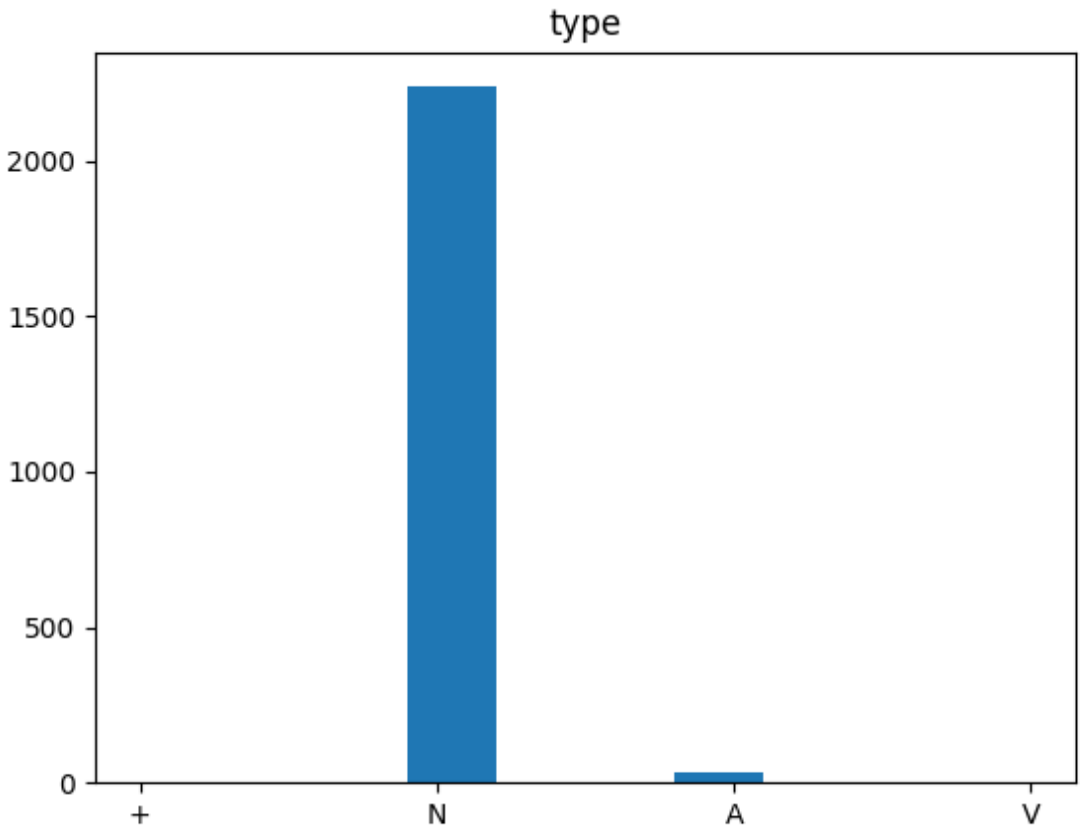
Website homepage:

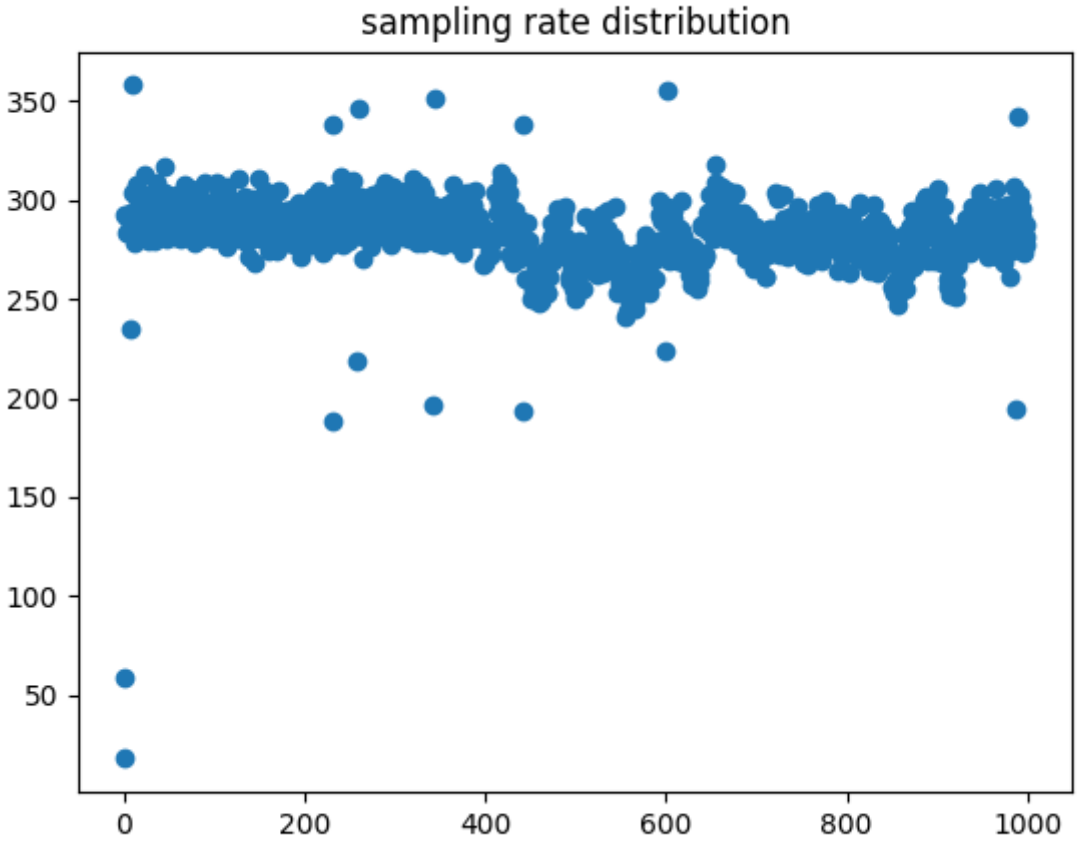


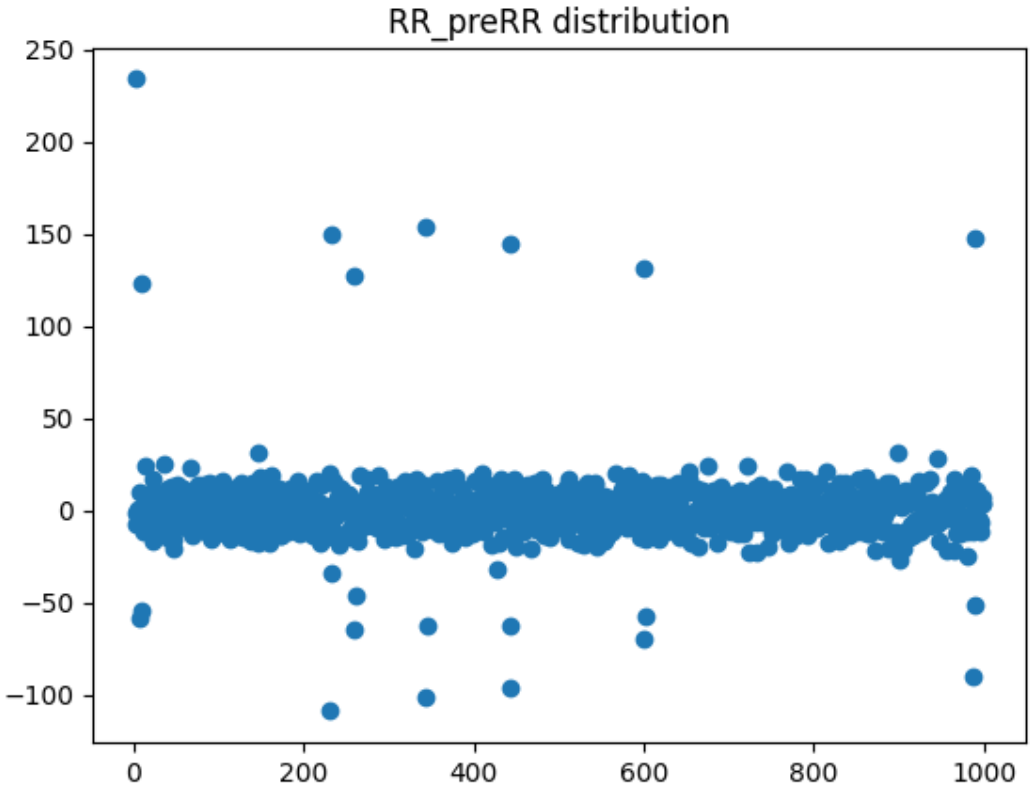
Upload files:

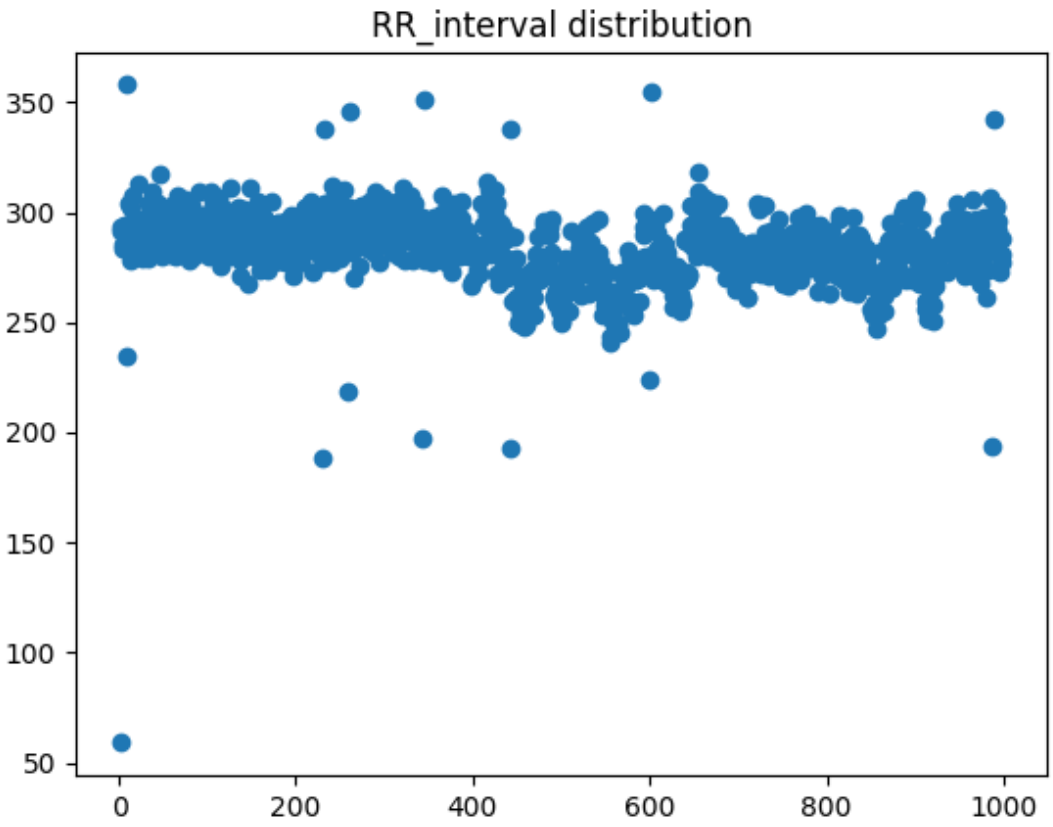


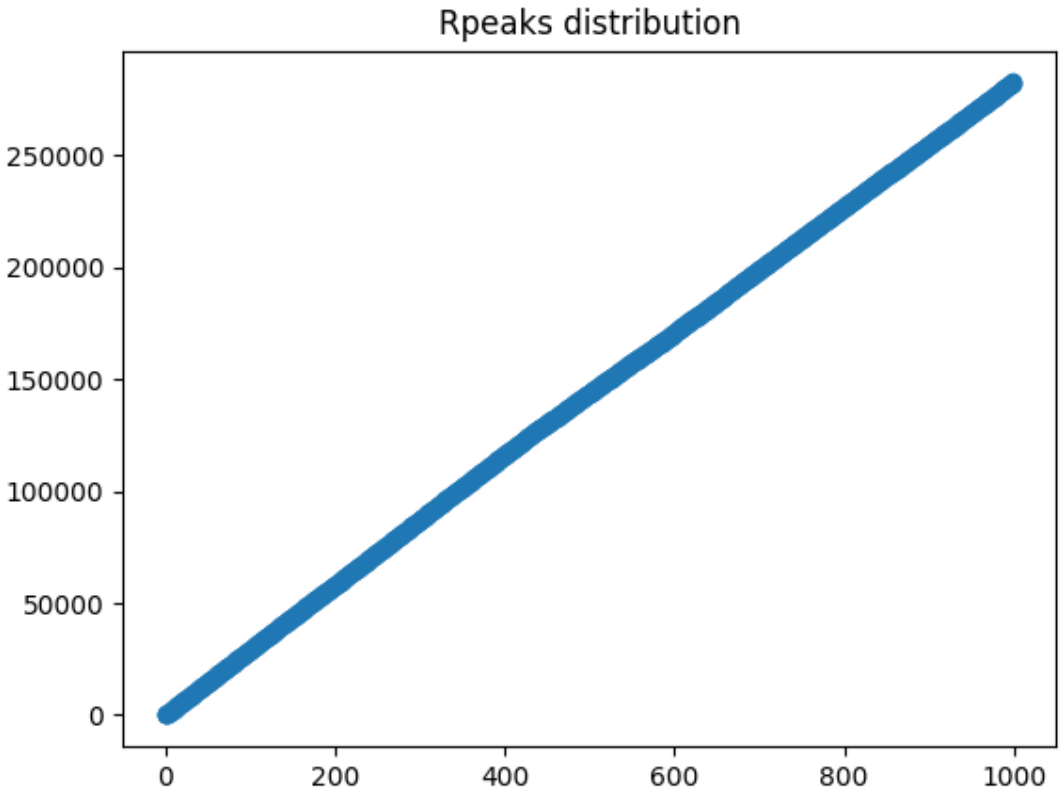
Results for the first 1000 heartbeats:

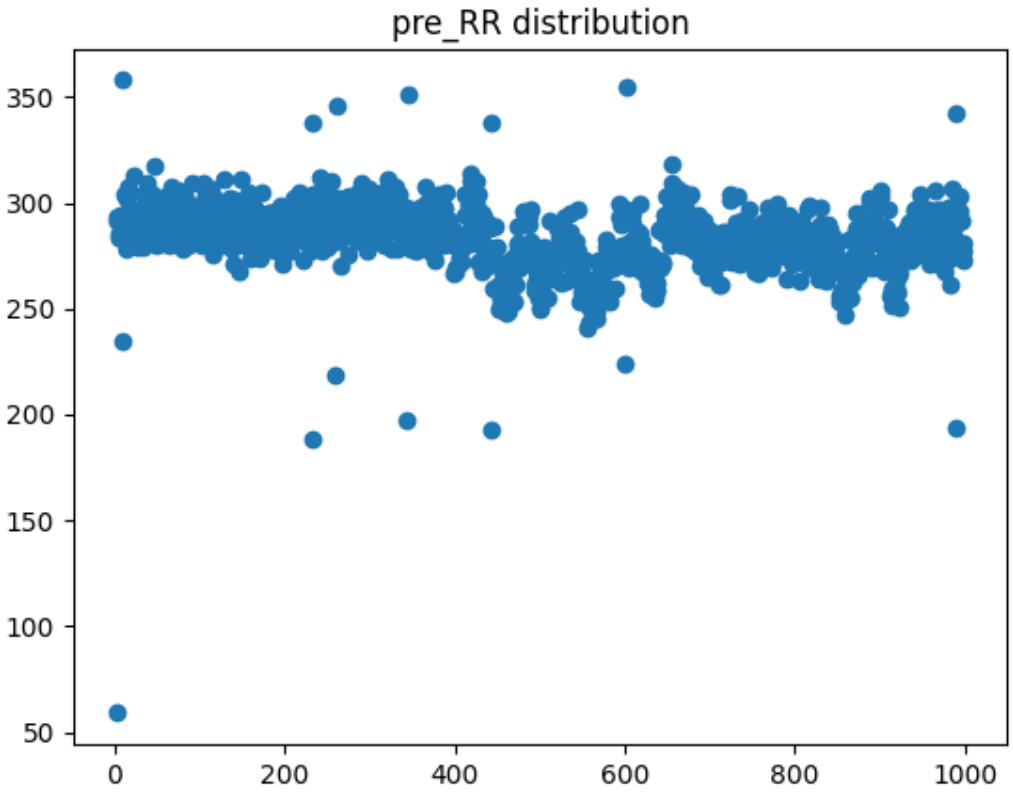


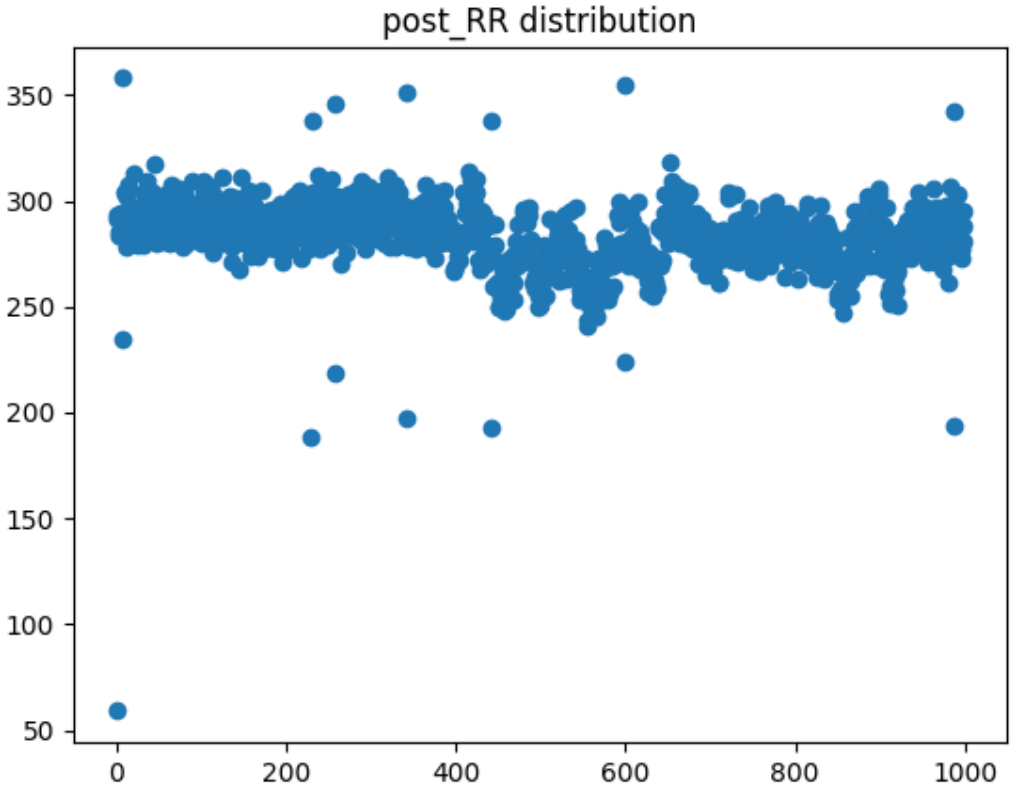

sampling rate distribution
350
300
250
200
150
100
50
0
200
400
600
800
1000

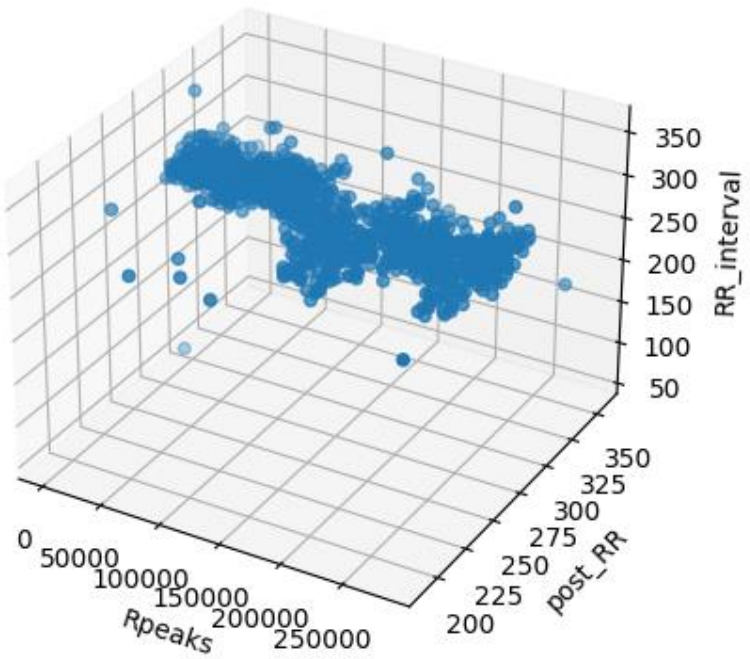

RR_preRR distribution
250
200
150
100
50
0
−50
−100
0
200
400
600
800
1000


RR_interval distribution
350
300
250
200
150
100
50
0
200
400
600
800
1000


Rpeaks distribution
250000
200000
150000
100000
50000
0
0
200
400
600
800
1000


pre_RR distribution
350
300
250
200
150
100
50
0
200
400
600
800
1000


post_RR distribution
350
300
250
200
150
100
50
0
200
400
600
800
1000



Arrhythmia detection interface video

Source code : https://github.com/hassanoubrahim/arrhythmia-detection-

Conclusion :

As the technologies developed there was a great deal of advancement in the field of medical electronics. The diagnosis and analysis by expertise (doctors) was not sufficient to find out the heart disorders and for the increase in cardiac problem cases that are seen by the hospitals every day. Hence, a more advanced, systematic, and procedural fast analysis technique is required, and this project is done for fulfilling the above requirement. To find the disorders of the heart many methods have been adopted so far in research but this project concentrates on developing two new methods namely kNN method and SVM. Based on the evaluation the disorders of the heart are identified. Out of these two methods of study, kNN method gives better analysis of heart disorders.