Project 1: Analyzing and classifying ECGs

1 Introduction

This programming project is concerned with automatically determining if an ECG is "shockable" or not. For simplicity we are going to only look at ECGs that are either shockable or normal (although, as mentioned in class, there are non-shockable arrythmias that are abnormal).

This assignment should be done individually. You can use any language you want, though we suggest C or Matlab. If your programs cannot be run from the command line you will need to speak to Devin, to make sure he can easily run it. (More details about your implementation are provided below.)

2 Assignment

For our purposes, consider a signal to be a function $f : \mathbb{Z} \to \mathbb{R}$, where $f(i)$ is the value of the i^{th} sample, and $f(j) = 0$ for any j outside the range of available sample information. How you represent a signal in your implementation is up to you, but a one-dimensional array is perfectly sufficient, provided that you know how the array indices correspond with sample indices. This is something that you may have to consider carefully if you choose to implement your own convolution routine.

2.1 Zero-crossings

As a start, write a routine which can compute zero-crossings on a signal, for a given value of "zero." The ability to shift the signal by redefining a value for zero is very important, as the baseline potential on an ECG signal from a live person may not be zero. Additionally, as we saw in class, noise may produce extra zero-crossings that don't reflect the broader behavior of the signal; to compensate for this, you should convolve your signal with a Gaussian to smooth it.

To formalize this notion, let's say the following: your routine, given a signal f, a zero value y_0, and a smoothing parameter σ, should compute a smoothed signal $f' = f * G_\sigma$, where G_σ is a sampled Gaussian kernel

$$G_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-x^2/2\sigma^2}.$$

Your routine should then compute and return the set of all indices i such that $(f'(i-1) - y_0)(f'(i) - y_0) < 0$. Note: some convolution implementations, like Matlab's `conv` function, will change the dimensions of the one-dimensional array containing the signal; be sure that you adjust the values of i accordingly!

2.2 Baseline detection

Your zero-crossing routine is only useful if it's been passed a sensible value of zero. In an ECG signal, the baseline value may vary from patient to patient, and the encoding may vary between records (and, in fact, it does in the training set we're giving you). You should come up with a sensible way to determine this baseline value and implement it. (Note: this can be harder than it might seem! In fact, some of the signals have no sensible baseline.) In addition to computing a baseline value, your implementation should also produce a boolean value indicating whether or not the computed baseline value is valid for the entire signal. There are many possible ways to formulate the problem of finding a baseline and assessing its validity, so you should come up with something sensible and provide some (informal) mathematical justification for your choice. Please include this explanation in your documentation.

2.3 Arrythmia detection

Finally, you should implement an arrythmia detector. In this assignment, we'll assume that all arrythmias are shockable, so your detector, when given an unannotated signal, should just output a (possibly empty) list of intervals where it detects arrythmia. You may want to try multiple approaches, but at least one should involve an implementation of k-NN with some feature description of your choice. Be sure you document your approach.

We've given you a set of annotated training examples, which are described in detail below. You should use these examples to train and test your arrythmia detector. Please document the procedure you used to evaluate your performance and tune your algorithm.

3 Data sets and implementation requirements

In this assignment, you will be asked to deal with records from real ECG machines. Each record contains signals recorded from various leads which measure the potential between two parts of the body of a human subject during a heart test. While a full ECG typically has twelve leads, the ECGs we'll be considering have only two: MLII, from which the signals exhibit the familiar PQRST(U) shape, and another lead (usually V1, V2, or V5), which conveys important information for cardiologists, but which you are welcome to ignore for the purposes of this assignment.

3.1 Data file format

We've pulled data for training and testing from some of the Physionet ECG collections. The records on Physionet are encoded using a fairly complex binary format, so we've converted them to CSV files so that it's easier for you to write code to parse them. Each record has an identifier, e.g. "202" or "sele0104", and comprises two files: a sample file `ID.csv` containing samples from the two leads, and an annotation file `ID.atr.csv` containing annotations.

The sample file contains three columns which are, in order: the sample index, the amplitude measured on the MLII lead, and the amplitude measured on the V1/2/5 lead. Some of the signals are recorded with a sampling frequency of 360 Hz, while others are recorded at 250 Hz. There's no reason you should need to know the sampling frequency of a particular recording to complete this assignment, but you can obtain this information from the Physionet website if you need it.

The annotation file contains seven columns which are, in order: the time (M:SS.SSS) of the sample to which the annotation is attached, the index of the sample to which the annotation is attached, the annotation type, the annotation subtype, the channel to which the annotation is attached (corresponding to columns of values in the sample file), the "number" field of the annotation, and the auxiliary field of the annotation. In this assignment, you will be primarily concerned with the sample index, annotation type, and auxiliary fields. In particular, normal beats are marked (at the R peak) with the "N" annotation; ventricular fibrillation/flutter is (when not otherwise marked with a rhythm annotation) marked at the beginning with a "[" annotation and the end with a "]" annotation, and rhythm classifications are marked with a "+" annotation and a rhythm mnemonic in the auxiliary field: "(N" for normal, "(VFL" for ventricular flutter, "(VF" for ventricular fibrillation, or "(VT" for ventricular tachycardia. *You can probably ignore other annotations.* See Figures 1 and 2 for examples. However, if you find it useful, a complete list of annotation codes is available at: `http://physionet.cps.unizar.es/physiobank/annotations.shtml`.

We've provided a training set consisting of two classes, each containing twenty signals. Signals in the "normal" class are entirely normal. Signals in the "shockable" class exhibit ventricular tachycardia or ventricular fibrillation. You will need to read the annotations for these signals to determine the time periods during which these arrythmias were observed.

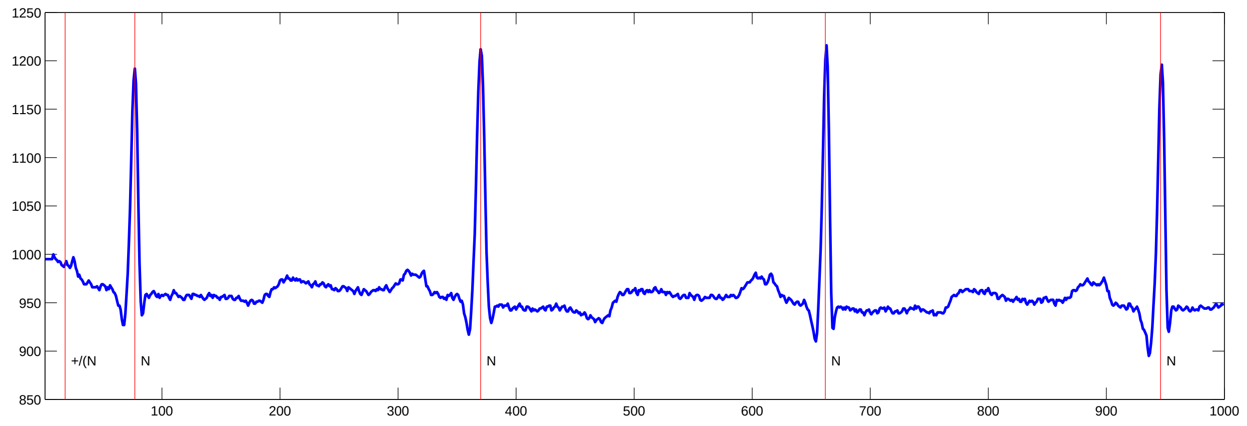

Figure 1: An annotated ECG from a patient exhibiting normal rhythm. This segment is from the beginning of a record; note that a rhythm annotation is given at the beginning of the signal with a “(N” auxiliary value indicating normal rhythm.

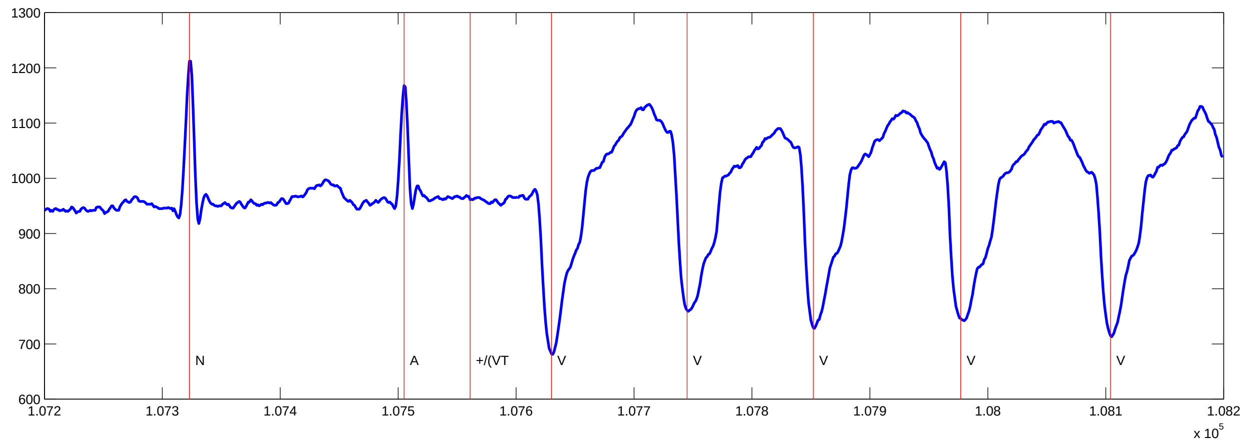

Figure 2: An annotated ECG showing a patient transitioning into ventricular tachycardia. Note how a rhythm annotation is given with a “(VT” auxiliary value to indicate the start of ventricular tachycardia.

3.2 Required interface

While you may write your software using the programming language of your choosing, for ease of evaluation, we'd like your program to support a command-line interface with the following contract (see below for a Matlab alternative):

- When invoked with the command-line arguments

  ```
  --zc --sigma=S --zero=Y /path/to/filename.csv
  ```

  your program should read samples (in the same format as the training set) from the specified file and compute the zero-crossings on the MLII signal as described above. The program should print indices (as integers) to stdout, one index per line (outputting nothing if no crossings are found).

- When invoked with the command-line arguments

  ```
  --baseline /path/to/filename.csv
  ```

  your program should read samples (in the same format as the training set) from the specified file and output an estimated baseline value for the MLII signal. It should output this value (as an integer or floating point number readable by `scanf`) to stdout, followed by a newline, followed by either `0` or `1` indicating whether the baseline is valid for the entirety of the input.

- When invoked with the command-line arguments

  ```
  --shock /path/to/filename.csv --freq=F
  ```

  your program should read samples (in the same format as the training set) from the specified file and look for periods during which a shockable arrythmia is present. For each interval where a shockable arrythmia is detected, it should output (to stdout, as integers) the beginning sample index and ending sample index separated by whitespace and followed by a newline. If no such intervals are detected, the program should not output anything to stdout.

  Note: Your program may assume the signal was sampled at frequency F (Hz), if the `--freq` argument is given; if no `--freq` argument is present, your program may assume the signal was sampled at a frequency of 360 Hz.

Note: your program may write arbitrary output to stderr: we will discard it during testing.

3.3 Required interface (MATLAB)

If you choose to use Matlab for your project, you must provide (in addition to whatever other code you may write) three m-files providing functions with the following interface:

- A file `zc.m` providing a function compatible with this signature:

  ```
  function [indices] = zc(signal, sigma, zero)
  ```

where `signal` contains a column vector containing the sample values from the signal, and `sigma` and `zero` are parameters to the routine described above. The return value `indices` may be in any form, so long `indices(1)` yields the index of the first zero-crossing, `indices(2)` yields the index of the second, and so on.

- A file `baseline.m` providing a function compatible with this signature:

  ```
  function [zero, valid] = baseline(signal)
  ```

  where `signal` is a column vector containing the sample values from the MLII signal.

- A file `shock.m` providing a function compatible with this signature:

  ```
  function [intervals] = shock(signal, frequency)
  ```

  where `signal` is a column vector containing the sample values from the MLII signal and `frequency` is the sampling frequency of the given signal, in Hertz. The return value `intervals` should be a $n \times 2$ matrix, where each `intervals(i, 1)` contains the starting sample index (inclusive) of the i^{th} interval where arrythmia was detected and `intervals(i, 2)` contains the ending sample index (inclusive).

3.4 Submission format

Please submit a zip file your source code, and data files necessary to run your program on novel input, and, if applicable, an executable (preferably for Windows or Linux, 32- or 64-bit; Mac and others negotiable) or build instructions. Feel free to include a README file (in any reasonable format) with an overview of your approach and any special instructions.

3.5 Evaluation

We will evaluate your code based on your design decisions and its performance of your code on novel signals. Don't worry too much about producing excellent results, as long as you can justify your approach.