ECE 480A3 Project Final Report: Diagnosis of Cardiac Arrhythmias from Electrocardiography Data

Robby Stokoe

May 13, 2016

Abstract

The diagnosis of cardiac arrhythmias can be a tedious process when done by hand and could benefit greatly from computer automation. To this end, an algorithm was developed to distinguish between normal heart beats and abnormal arrhythmic beats in an ECG waveform. First an algorithm was developed to find the location of QRS complexes in the ECG waveform. Principal component analysis was performed using the area around the QRS complex. 20 of the resulting principal components were used to train a simple linear classifier to distinguish between normal and abnormal beats. The classification performed reasonably well with a sensitivity of 85.4% and specificity of 91.7%. More-sophisticated signal processing and classification techniques could be applied to improve these numbers, but the algorithm is a good starting point.

Contents

1 Introduction

Cardiac arrhythmias are a relatively common set of diseases, affecting 3.4% of people in the US [1]. Diagnosis is done by eye from an ECG trace which can be tedious. A common diagnostic technique is to record ECG data from a patient constantly as they go about normal activity. This results in a very long recording which may contain only a few abnormal beats. An algorithm-driven diagnosis could potentially eliminate this tedium and also the possibility of human error if the algorithm is robust enough. Such an algorithm could be used in AEDs, pacemakers, or EEG machines. Even if the algorithm is not as robust as human diagnosis, it could still assist diagnosis by flagging potentially abnormal beats for further scrutiny, allowing a trained cardiologist to quickly zero-in on the relevant beats. This project attempts to develop such an algorithm using signal processing and classification techniques.

2 Data

The data used to develop the signal processing algorithm and train the classifier were obtained from the MIT-BIH Arrhythmia Database which is available on physionet.org [2] . The data and techniques used to obtain it are described as follows:

> 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 annota-

tions for each beat (approximately 110,000 annotations in all) included with the database.

The data and annotations will be instrumental in training and testing the classification algorithm. Figure 2 shows a portion of a recording from the Database with annotations showing normal beats, an atrial premature beat (A) and a premature ventricular contraction (V) [2].

3 Algorithm

The first step of the algorithm will be to identify the locations of QRS complexes in the ECG. This will be done by the Pan-Tompkins algorithm shown schematically in Figure 1. This algorithm works by differentiating the signal, squaring the derivative, and then comparing the resulting squared derivative with some threshold. The location of the maximum absolute value of the signal in the range where the squared derivative is above the threshold is taken to be the location of the R-wave in the QRS complex [4]

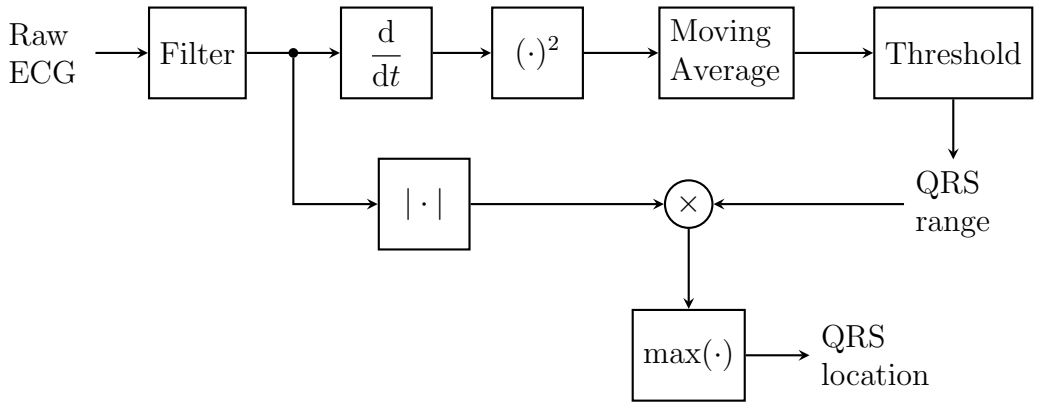

Figure 1: The Pan-Tompkins Algorithm for Finding QRS Complexes

Once the location of the center of the QRS complex is determined, the portion of the signal few hundred milliseconds before and after the QRS complex is extracted for analysis. To train the classifier, a multitude of these signal excerpts were gathered and sorted by normal *vs.* abnormal rhythm.

Principal component analysis was performed on this entire data set to obtain a transformation to be used for classification. Only some of the components of this transformation were used to simplify classification. To obtain

the best discrimination between normal and abnormal beats, the components for which the mean values of the normal and abnormal beats were the most significantly different were used.

4 Implementation

The algorithm was implemented in MATLAB. All code used is included in the appendices starting on page 14 and is also available on github.

4.1 Preliminary Data

The first step was to download ECG data and annotations from Physionet using the `rdsamp()` and `rdann()` functions from Physionet's WFDB toolbox. [5] These functions were wrapped in another function, `fetch()`, to make them easier to use. See page 14 for the full function. Figure 2 shows an example of ECG data directly from physionet with beat annotations. All ECG plots shown were generated by the `plotecg()` function shown on page 17.

The annotations do not always correspond directly with the location of the QRS complex because they correspond to the entire beat rather than a specific feature. So the next step is to determine the exact location of the QRS complexes.

4.2 QRS Complex Detection

The first step in the Pan-Tomkins QRS-Detection algorithm is to filter the signal. This was done by a fourth-order Butterworth bandpass filter with cutoffs at 1 and 50Hz, as well as a notch filter at 60Hz. The full implementation is shown in the `filterecg()` function in section A.2 on page 14. The effect of this filter on an ECG signal is shown in Figure 3.

The rest of the QRS-detection algorithm was implemented in the function `findqrs()` shown in section A.3 on page 15. Figure 4 shows an example of the signals at each stage of the algorithm. The QRS-detection function performed relatively well, accurately finding the location of the QRS complex of most beats. Occasionally it would miss a particularly malformed beat, or find a beat were there was none, especially if the signal was particularly noisy. Figure 5 shows a closer look at the location of the R-wave determined

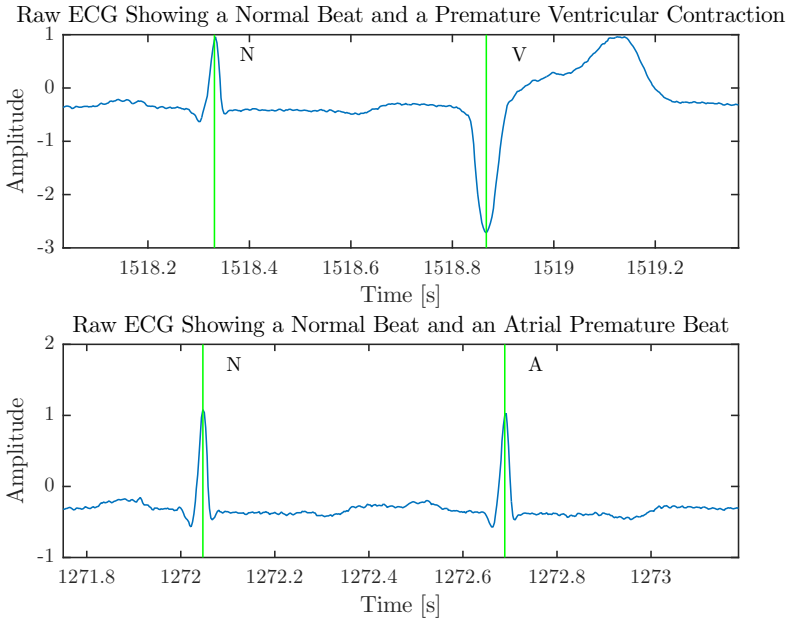

Figure 2: Examples of Raw ECG Signals Showing Normal Beats (N), an Atrial Premature Beat (A), and Premature Ventricular Contraction (V)

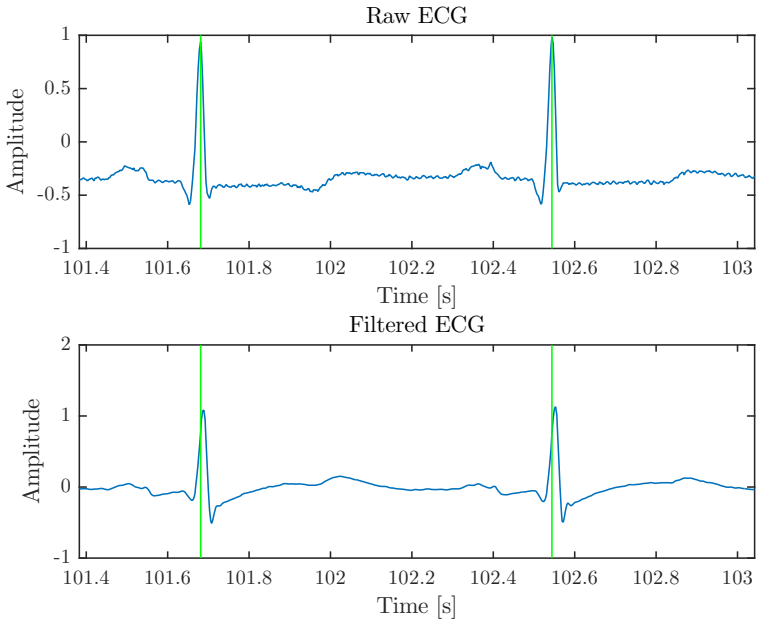

Figure 3: The Effect of `filterecg()` on an ECG Signal

by the algorithm, and an example of some erroneously-detected beats. These figures were all generated by the `figures.m` script on page 22.

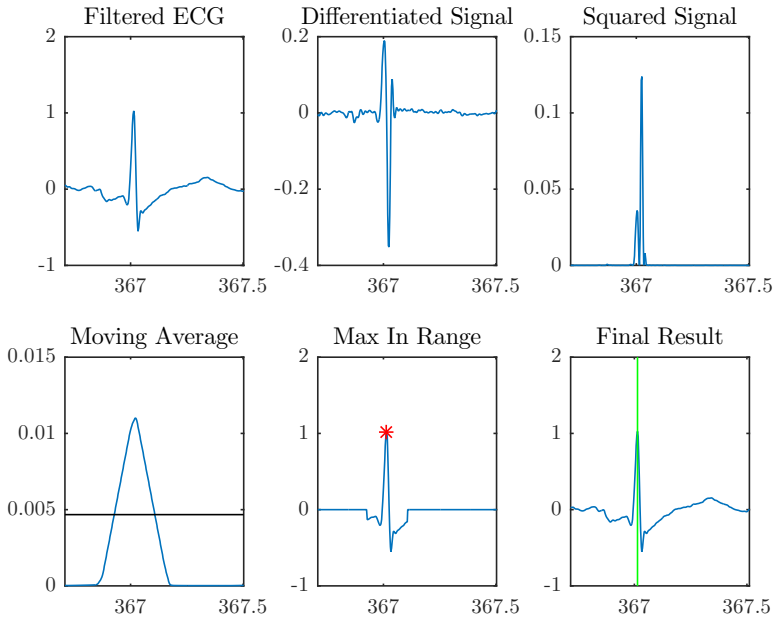

Figure 4: Example Signals at the Various Stages of the Pan-Tompkins QRS-Detection Algorithm

Based on initial investigation of the data, it was decided to develop a classifier to distinguish between normal and atrial premature beats, because these beats deviate little from normal beats, which will give the starting point for the entire classifier, the QRS-detection algorithm, more to work with.

4.3 Data Acquisition

To train the classifier, a large amount of data was collected and processed (see `gather.m` in Section B.1 on page 19). Five recordings from the MIT-BIH ECG database were chosen because they include a large number of atrial premature beats. These recordings were downloaded from physionet (`gather.m` lines 3–10), and processed to extract a 0.6-second excerpt centered around the QRS complex for each normal and abnormal beat (lines 12–48).

For each recording, the locations of normal and abnormal beats were determined from the included annotations (lines 16–20). An equal number

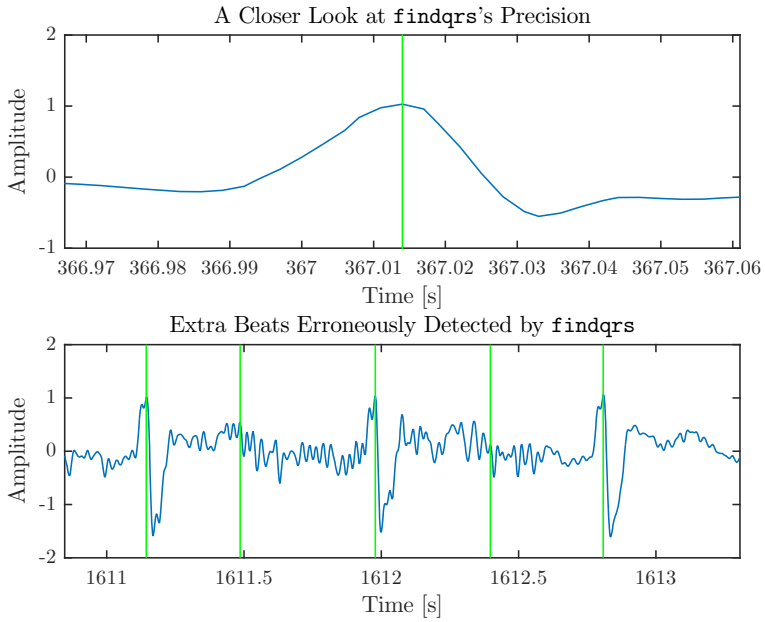

Figure 5: Performance of the QRS-Detection Algorithm

of normal and abnormal beats were randomly selected (lines 21–23) and extracted (lines 25–48).

The extraction process involved finding the QRS complex locations with `findqrs()` (lines 25–29), and extracting the signal 0.3 seconds before and after the QRS complex for both normal (lines 39–46) and abnormal (lines 31–37) beats. The beat excerpts were compiled into two matrices (lines 36 and 45) for subsequent processing. The function `crop()`, shown in Section A.4 on page 16, was created to facilitate extraction of the 0.6-second excerpts around each beat.

4.4 Principal Component Analysis

The data from `gather.m` were further processed before being used to train a linear classifier in `train.m` shown in Section B.2 on page 20. The normal and abnormal beat data were combined into one matrix (`train.m` line 10) and normalized by subtracting the mean of each time point (lines 12–17).

Singular value decomposition was performed on the normalized data (line 19) to find a transformation matrix to transform the data onto an orthogonal

basis of principal components [6]. A visualization of the transformation matrix is shown in Figure 6 along with an example ECG waveform for reference.

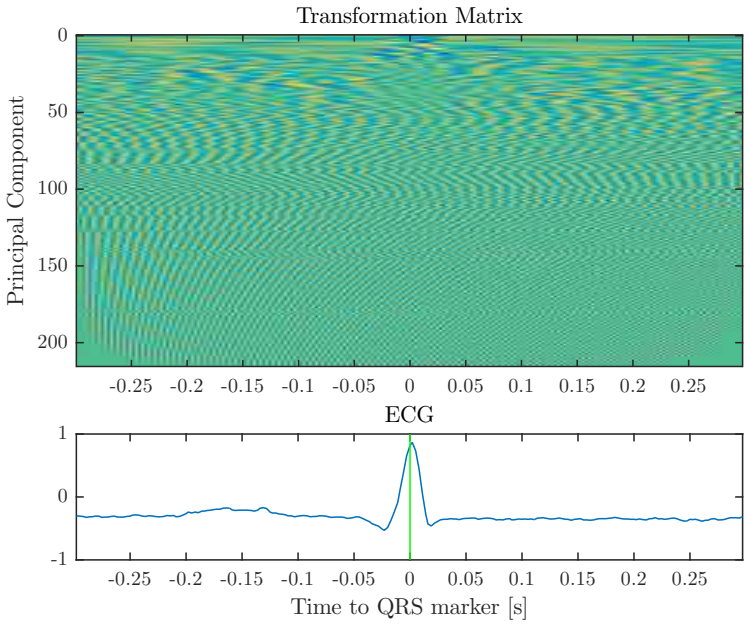

Figure 6: Principal Component Transformation Matrix with Reference ECG

Each row of the matrix produces a different principal component as a linear combination of the value at each sample of the ECG signal. The matrix is arranged so that the first principal component has the highest variance across the data. Figure 7 shows the variances of the first 30 (out of 215) principal components.

To facilitate transformation of the data, a function, `pca_transform()`, was created and is shown on page 17. To aid in visualization, the function `plotpca()` (page 17) plots the data with respect to three given components.

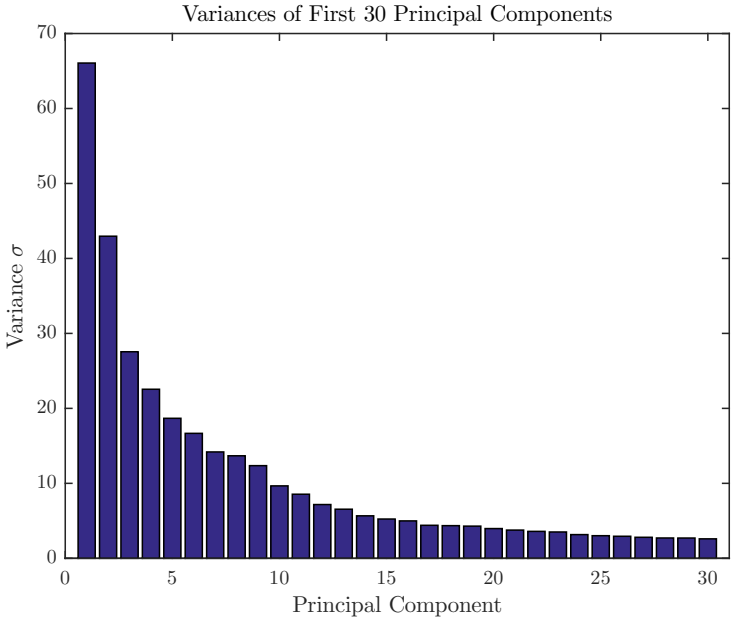

Figure 7: Variances of the First 30 Principal Components

4.5 Component Selection

In a linear classifier, high variance of a particular measurement does not necessarily correspond to better discrimination between two groups of data. It is possible for the two groups to each have a wide distribution that overlaps almost completely with the other. This would result in a large variance, but no ability to categorize the data. On the other hand, it is possible for each group to have a narrow distribution, separated by relatively little from the other group, but significant relative to the standard deviation.

For this reason, the components chosen to classify the data were not those with the most variance. Instead, the mean of each component for the two groups, normal and abnormal, was found (`gather.m` lines 48–62). The difference between these means (line 63) normalized by the variance of that particular component, was used to select the top 20 'most-significant' components to be used for classification (lines 73 and 83).

Figure 8 shows the difference in means for the first 30 Principal components (those with the largest variances). Figure 9 shows the 20 most-significant components (those with the most significant difference between the mean of the normal and abnormal data).

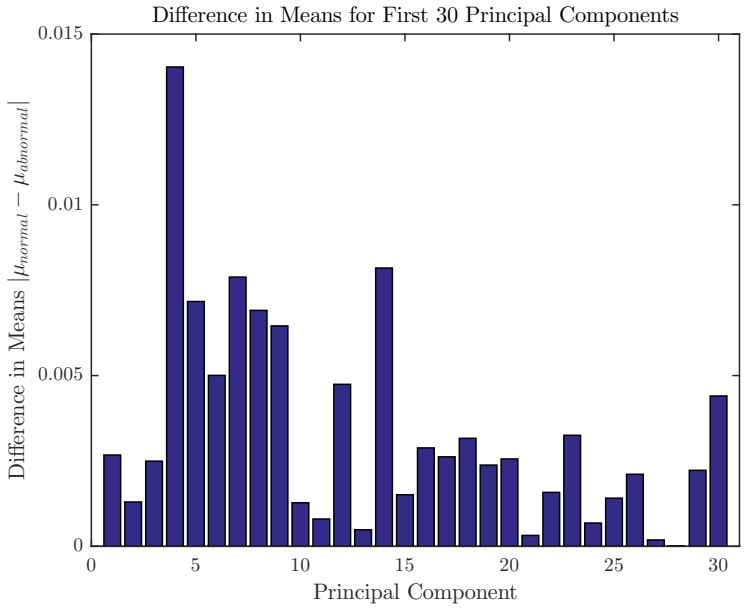

Figure 8: Difference in Means for First 30 Principal Components (Those with the Largest Variance)

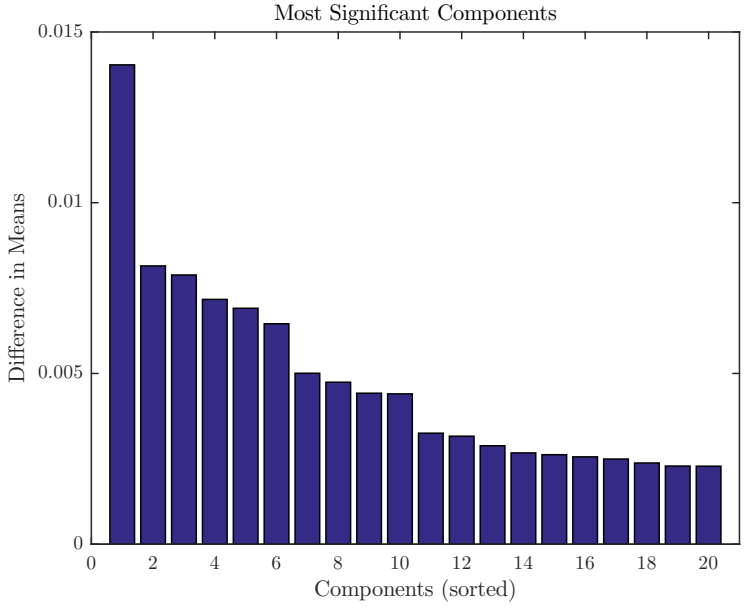

Figure 9: Difference In Means for the First 20 Most-Significant Components (Those with the Largest Difference In Means)

4.6 Classifier Training

The 20 most-significant components were used to train a linear classifier (`train.m` lines 91–97) using the formula $\mathbf{w} = (\mathbf{X}^\top\mathbf{X})^+\mathbf{X}^\top\mathbf{d}$ where $\mathbf{w}$ is a vector of weights for each component, $\mathbf{X}$ is the matrix of data with each column corresponding to a component and each row corresponding to a single point, and $\mathbf{d}$ is a vector of the desired response of the classifier for each row in $\mathbf{X}$ [7]. In this case, $d_i = 0$ for normal beats, and 1 for abnormal beats. Note also that $\mathbf{A}^+$ denotes the pseudo inverse of $\mathbf{A}$ computed by MATLAB's `pinv()` function.

Figure 10 shows the first two most-significant components and the decision boundary for those two components when every other component is equal to zero. Note that since this is a two-dimensional projection of 20-dimensional data, the side of the line on which a point falls on does not necessarily correspond to how it is classified. All 20 dimensions are necessary to classify each point.

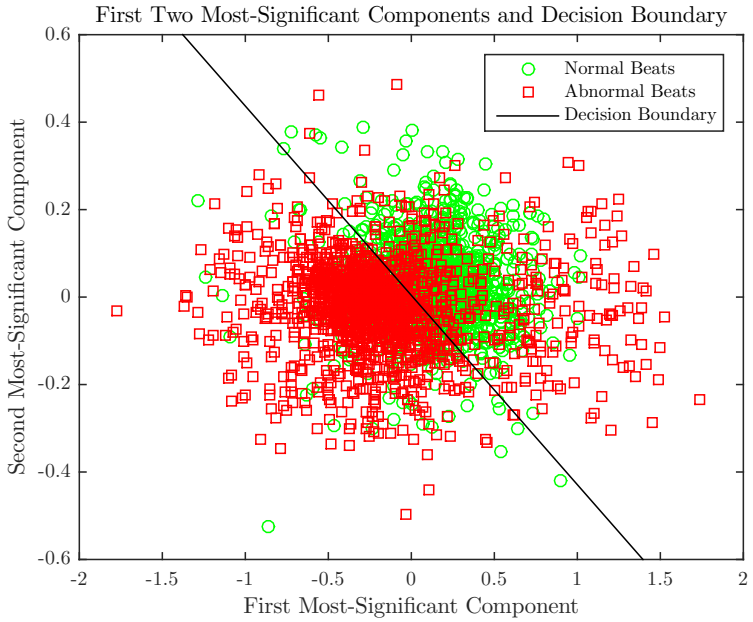

Figure 10: First Two Most-Significant Components and Cross-Section of the Classifier's Decision Boundary

5 Performance

The classifier result was calculated (`train.m` line 117) and compared to the desired result to determine the number of true negatives, false positives, false negatives, and true positives (lines 118–121). These results are shown in the confusion matrix in Figure 5. The true positive rate (sensitivity) and true negative rate (specificity), were also determined. The sensitivity was found to be $\text{TPR} = \frac{\text{true+}}{\text{actual+}} = 85.4\%$, and the specificity was found to be $\text{TNR} = \frac{\text{true−}}{\text{actual−}} = 91.7\%$

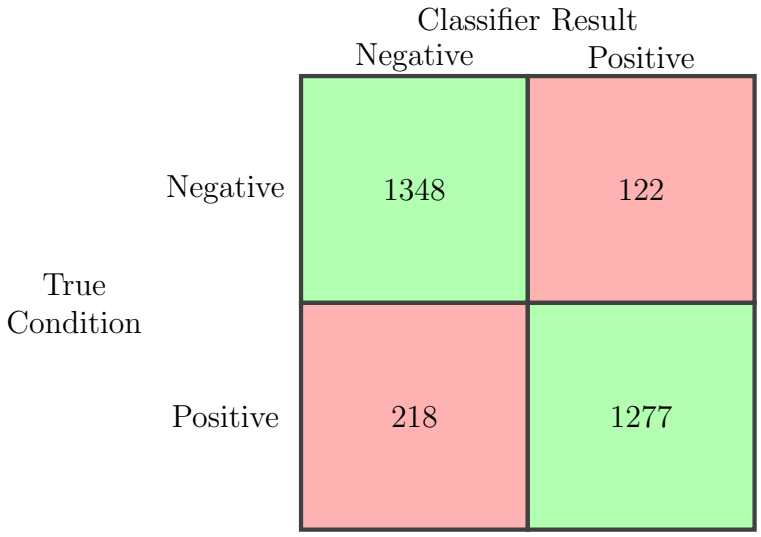

Figure 11: Confusion Matrix For Classifier: Total Cases: 2965, True Negatives: 1348, False Positives: 122, False Negatives: 218, True Positives: 1277

6 Conclusion

The beat classification algorithm performed surprisingly well given its simplicity and how the data are clustered in Figure 10. The sensitivity and specificity are reasonably high but could definitely be improved with more-sophisticated classification techniques, such as a support vector machine or artificial neural network. This algorithm is not particularly remarkable, but is an excellent starting point for future development.

In retrospect, picking an arrhythmia that is very similar to a normal rhythm on an ECG made classification significantly harder. If an arrhythmia presenting a more-drastic difference on the ECG was chosen, the classifier would have likely performed better. An arrhythmia that is too different, though, would have made QRS detection harder, since the algorithm is based on assumptions about what the ECG will look like. There is a trade-off between classifier performance and QRS-detection performance.

A MATLAB Functions

A.1 fetch()

```
function ecg = fetch(url)
%fetch: gets ECG data from physionet
%   usage:  ecg = fetch(url);
%   input:  the physionet url of the physionetdb record to download,
%           e.g. 'mitdb/100'
%   output: an ECG struct containing the 1st signal and annotations if
%           present

    % add WFDB toolbox to path
    addpath('U:\classes\ece\480\wfdb-app-toolbox-0-9-9\mcode');

    % pull data from physionet
    [tm,sig] = rdsamp(url,1);
    [ann,type,subtype,chan,num,comments] = rdann(url,'atr');

    % bundle into struct
    ret.signal = sig;
    ret.time = tm;
    ret.N = length(sig);
    ret.ann = ann(ismember(type,'NLRBAaJSVrFejnE/fQ?'));
    ret.type = type(ismember(type,'NLRBAaJSVrFejnE/fQ?'));
    ret.Nann = length(ret.ann);
    ret.fs = (ret.N - 1) / (tm(ret.N) - tm(1));

    % return ECG struct
    ecg = ret;
end % function
```

A.2 filterecg()

```
function filtered = filterecg(ecg, varargin)
%filterecg: filters an ECG with a fourth-order butterworth filter with a
%passband of 1--50 Hz and a notch at 60 Hz unless otherwise specified
%   usage:  filtered = filterecg(ecg);
%       or: filtered = filterecg(ecg,fl,fh,fn);
%   input:  ECG struct to be filtered
%           high & low cutoff and notch frequencies in Hz (optional
%   output: filtered ECG struct

    % get info from ecg
    sig = ecg.signal;
    fs = ecg.fs;

    switch nargin
        case 1
            fl = 1;     % low cutoff
            fh = 50;    % high cutoff
            fn = 60;    % notch
        case 4
            fl = varargin{1};   % low cutoff
            fh = varargin{2};   % high cutoff
```

```
            fn = varargin{3};   % notch
        otherwise
            error('filterecg: must specify exactly three frequencies');
            return
    end % switch

    % normalize frequencies
    fl = fl * 2 / fs;   % low cutoff
    fh = fh * 2 / fs;   % high cutoff
    fn = fn * 2 / fs;   % notch
    bw = 1 * 2 / fs;    % notch -3dB bandwith

    % calculate filter coefficients
    [bb,ab] = butter(4,[fl,fh],'bandpass'); % bandpass
    [bn,an] = iirnotch(fn,bw);  % notch

    % filter signal
    fsig = filter(bn,an,filter(bb,ab,sig));

    % transfer info to output
    filtered = ecg;

    % change to filtered signal
    filtered.signal = fsig;

end % function
```

A.3 findqrs()

```
function peaks = findqrs(ecg)
%findpeaks: finds the location of qrs complexes in an ecg using the
%Pan-Tompkins algorithm
%   usage:  peaks = findqrs(ecg);
%   input:  ECG struct to find peaks in
%   output: a list of indices of the peaks of the QRS complexes

    % filter with passband of 5-11 Hz
    fecg = filterecg(ecg);
    % fecg = ecg;
    % extract signal and sample frequency
    fsig = fecg.signal;
    fs = fecg.fs;
    % first difference
    sig = filter([1 -1],1,fsig);
    % square
    sig = sig .* sig;
    % moving average
    T = 0.150;                  % width in seconds
    W = round(T*fs);            % width in samples
    b = ones(1,W) / W;          % W-sample average
    sig = filtfilt(b,1,sig);    % compute moving average

    % threshold
    threshold = 1;
    th = sig > threshold * rms(sig);
        % 1 if moving average is above threshold, 0 otherwise
```

```
28      dth = filter([1 -1],1,th);  % derivative of threshold signal:
29          % 1 at start of range, -1 at end
30
31      % get ranges
32      starts = find(dth == 1);    % list of range starts
33      stops = find(dth == -1);    % list of range ends
34
35      Nbeats = min(length(starts), length(stops));    % number of ranges
36
37      % maximum (magnitude) in each range
38      for i = 1:Nbeats
39          excerpt = fsig(starts(i):stops(i));
40          [M,peak] = max(excerpt);
41          peaks(i) = peak + starts(i) - 1;
42      end % for
43  end % function
```

A.4 crop()

```
1   function excerpt = crop(ecg, beats, varargin)
2   % crop:     extracts a portion of an ECG signal around the nth marker in
3   %           the list of annotations
4   %   usage:  excerpt = crop(ecg, beats);
5   %       or: excerpt = crop(ecg, beats, before, after);
6   %   input:  the ECG struct to be cropped
7   %           the number of the beat to extract
8   %           time in seconds before beat to begin excerpt (optional)
9   %           time in seconds after beat to end excerpt (optional)
10  %   output: an ECG struct with the extracted data, including the beat
11  %           annotations
12
13  % offsets
14  switch nargin
15      case 2
16          before  = 0.300;    % time before beat to start excerpt in seconds
17          after   = 0.500;    % time after beat to start excerpt in seconds
18      case 4
19          before = varargin{1};
20          after = varargin{2};
21      otherwise
22          error('crop: must supply 2 or 4 arguments');
23          return
24  end % switch
25
26  % convert times to indices
27  b_offset = floor(before * ecg.fs);
28  a_offset = floor(after * ecg.fs);
29
30  % first and last beats
31  first_beat = max(0,min(beats));
32  last_beat = min(ecg.Nann,max(beats));
33
34  % start and end index
35  s_index = max(1,ecg.ann(first_beat) - b_offset);
36  e_index = min(ecg.N,ecg.ann(last_beat) + a_offset);
37
```

```
38  excerpt = ecg;
39  excerpt.signal = ecg.signal(s_index:e_index);
40  excerpt.time = ecg.time(s_index:e_index);
41  excerpt.N = e_index - s_index + 1;
42  excerpt.ann = ecg.ann(beats) - s_index + 1;
43  excerpt.type = ecg.type(beats);
44  excerpt.Nann = length(excerpt.ann);
45  end %function
```

A.5 pca_transform()

```
1   function T = pca_transform(measurements,means,transform)
2   %pca_transform: transforms a measurement using principal component matrix
3   %       after normalizing to the mean value of each measurement
4   %   usage:  T = pca_transform(measurements,means,transform)
5   %   input:  The raw measurements to transform (m rows x n columns)
6   %           A list of mean values of each measurement for normalization
7   %               (m rows)
8   %           The transformation matrix to apply (m rows x d columns)
9   %   output: The transformed data (n rows x d columns)
10
11      % normalize
12      [m,n] = size(measurements);
13      X = measurements - means * ones(1,n);
14
15      % transform
16      T = X.' * transform;
17  end % function
```

A.6 plotpca()

```
1   function plotpca(norm,abnorm,d1,d2,d3)
2   plot3(norm(:,d1),norm(:,d2),norm(:,d3),'go', ...
3       abnorm(:,d1),abnorm(:,d2),abnorm(:,d3),'ro')
4   end % function
```

A.7 plotecg()

```
1   function plotecg(ecg, varargin)
2   % plotecg: plots data from ECG struct with annotations
3   % marked
4   %   usage:  plotecg(ecg,options)
5   %   input:  the ecg struct to plot
6   %       options:
7   %           'Annotate': adds text labels if present
8   %           'Latex':    uses Latex interpreter for all text
9   %           'NoTitles': omits title and axis labels
10  %           'NoMarks':  omits beat annotation location markers
11  %   output: none
12
13      plot(ecg.time,ecg.signal);
14      xlim([min(ecg.time), max(ecg.time)]);
15      ys = ylim();
16
```

```
    if nargin > 1 && ismember('Latex',varargin)
        texify(gcf);
    end %latex if
    if ismember('Annotate',varargin) && isfield(ecg,'type')
        ty = ys(2) - (ys(2) - ys(1)) * 0.1;
        x = ecg.time(ecg.ann) + 0.05;
        y = ty * ones(ecg.Nann,1);
        strings = cellstr(ecg.type);
        if ismember('Latex',varargin)
            text(x,y,strings,'Interpreter','Latex');
        else
            text(x,y,strings)
        end %latex if
    end %annotation if
    if ~(nargin == 0 || ismember('NoTitles',varargin))
        title('ECG');
        xlabel('Time [s]');
        ylabel('Amplitude');
    end %title if
    if ~(nargin == 0 || ismember('NoMarks',varargin))
        hold on;
        for i = 1:ecg.Nann
            hold on;
            x = ecg.time(ecg.ann(i));
            line([x x],ys,'color','green');
        end % for
        ylim(ys);
        hold off;
end % function
```

A.8 texify()

```
function texify(figure)
%texify: makes latex the interpreter for title and axis labels of the
%current axis of a figure
%   usage:  texify(figure)
%   input:  figure handle to texify
%   output: none
    figure.CurrentAxes.TickLabelInterpreter = 'Latex';
    figure.CurrentAxes.Title.Interpreter = 'Latex';
    figure.CurrentAxes.XLabel.Interpreter = 'Latex';
    figure.CurrentAxes.YLabel.Interpreter = 'Latex';
end % function
```

B MATLAB Scripts

B.1 gather.m

Downloads and sorts data with which to train the classifier

```
%% gather many ECG signals to perform principal component analysis on

records = {'mitdb/100'; 'mitdb/118'; 'mitdb/209'; ...
    'mitdb/222'; 'mitdb/232'};

for i = 1:length(records)
    ecg(i) = fetch(records{i});
end % for

save('ecg_data','ecg');

%% Extract abnormal beats
abnormal_signals = [];
normal_signals = [];
for i = 1:length(ecg)
    abnormal_beats = find(ecg(i).type == 'A');
    normal_beats = find(ismember(ecg(i).type,'NLRB'));
    N_abnorm = length(abnormal_beats);  % number of abnormal beats
    N_norm = length(normal_beats);      % number of normal beats
    N_beats = min(N_norm,N_abnorm);     % total number of beats to analyze
    % select an an equal number of normal beats as abnormal beats
    normal_beats = normal_beats(randperm(N_norm,N_beats));
    abnormal_beats = abnormal_beats(randperm(N_abnorm,N_beats));

    % find qrs complexes and filter
    qrs = ecg(i);
    qrs.ann = findqrs(ecg(i));
    qrs.Nann = length(qrs.ann);
    qrs = filterecg(qrs);
    for j = 1:N_beats
        % find the qrs complex closest to the abnormal beat marker
        beat_no = find(abs(qrs.ann - ecg(i).ann(abnormal_beats(j))) < 108);
        % leave off the first and last beats of the signal
        if beat_no ~= 1 | beat_no ~= qrs.Nann
            beat = crop(qrs,beat_no,0.3,0.3);
            abnormal_signals = [abnormal_signals beat.signal];
        end % if

        % repeat for normal beats
        % find the qrs complex closest to the abnormal beat marker
        beat_no = find(abs(qrs.ann - ecg(i).ann(normal_beats(j))) < 108);
        % leave off the first and last beats of the signal
        if all(beat_no ~= 1 | beat_no ~= qrs.Nann)
            beat = crop(qrs,beat_no,0.3,0.3);
            normal_signals = [normal_signals beat.signal];
        end % if
    end % for
end % for

%%
```

```
51
52 % save data
53 save('normal_data','normal_signals');
54 save('abnormal_data','abnormal_signals');
```

B.2 `train.m`

Trains and tests the classifier

```
1  %% Train classifier
2
3  % load data
4  load('normal_data');
5  load('abnormal_data');
6
7  %% PCA
8
9  % combine into one matrix
10 all_signals = [abnormal_signals normal_signals];
11
12 % normalize
13 [m,n] = size(all_signals);
14 means = mean(all_signals,2);    % mean of each row (dimension)
15 save('means','means');
16
17 X = all_signals - means * ones(1,n);
18
19 [U,S,V] = svd(X.');
20 %% plot transform
21 load('ecg_data');
22 excerpt = crop(ecg(1),12,0.3,0.3);
23 excerpt.time = excerpt.time - excerpt.time(excerpt.ann);
24 dim = [0, length(V)];
25
26 figure(5);
27 subplot(3,1,3); plotecg(excerpt,'Latex','NoTitles');
28 title('ECG'); xlabel('Time to QRS marker [s]');
29 subplot(3,1,[1,2]);
30 imagesc(excerpt.time,dim,V.')
31 title('Transformation Matrix');
32 % xlabel('Time to QRS marker [s]');
33 ylabel('Principal Component');
34 texify(gcf);
35 %% plot variances
36 variances = diag(S);
37 figure(6);
38 bar(variances(1:30));
39 xlim([0,31]);
40 title('Variances of First 30 Principal Components');
41 xlabel('Principal Component');
42 ylabel('Variance $\sigma$');
43 texify(gcf);
44
45 %% save transform
46 save('transform','V');
```

```
47
48  %% Means
49
50  % transform data separately
51  T_norm = pca_transform(normal_signals,means,V);
52  T_abnorm = pca_transform(abnormal_signals,means,V);
53
54  %%
55  % PCA sorts dimensions by variance which is not necessarily what is desired
56  % for classification.  Instead, use dimensions that give the
57  % most-significant difference between normal and abnormal beats
58
59  % find relative difference in mean values of each dimension normalized by
60  % the variance of each dimension
61  norm_means = mean(T_norm);
62  abnorm_means = mean(T_abnorm);
63  norm_diff = abs(norm_means - abnorm_means) ./ diag(S).';
64
65  figure(7);
66  bar(norm_diff(1:30));
67  xlim([0,31]);
68  title('Difference in Means for First 30 Principal Components');
69  xlabel('Principal Component');
70  ylabel('Difference in Means $|\mu_{normal}-\mu_{abnormal}|$');
71  texify(gcf);
72  %%
73  [B,I] = sort(norm_diff, 'descend');
74
75  % Plot dimensions and difference in means
76  figure(8);
77  bar(B(1:20)); title('Most Significant Components')
78  xlabel('Components (sorted)'); ylabel('Difference in Means');
79  xlim([0 21]);
80  texify(gcf);
81
82  % keep only 20 most-varying dimensions
83  V_sub = V(:,I(1:20));
84  save('V_sub','V_sub');
85  norm_means = norm_means(I(1:10));
86  save('norm_means','norm_means');
87  abnorm_means = abnorm_means(I(1:10));
88  save('abnorm_means','abnorm_means');
89
90  %% Train Classifier
91  T_sub = pca_transform(all_signals,means,V_sub);
92  X = [T_sub ones(size(T_sub,1),1)]; % add ones
93  % desired response
94  d = [ones(size(abnormal_signals,2),1); zeros(size(normal_signals,2),1)];
95
96  % solve for classifier weights
97  w = pinv(X.' * X) * X.' * d;
98
99  %% plot decision line
100 T_norm_sub = pca_transform(normal_signals,means,V_sub);
101 T_abnorm_sub = pca_transform(abnormal_signals,means,V_sub);
102
103 figure(9);
```

```
plot(T_norm_sub(:,1),T_norm_sub(:,2),'og',T_abnorm_sub(:,1),T_abnorm_sub(:,2),'sr');
xs = xlim(); ys = ylim();
xx = linspace(xs(1), xs(2), 64);
yy = -w(1)/w(2)*xx - (w(end)-0.5)/w(2);
line(xx,yy,'color','k');
xlim(xs); ylim(ys);
legend({'Normal Beats','Abnormal Beats','Decision Boundary'},'Interpreter','Latex');
title('First Two Most-Significant Components and Decision Boundary');
xlabel('First Most-Significant Component');
ylabel('Second Most-Significant Component');
texify(gcf);

%% Test Classifier
d_out = X * w > 0.5;
total = length(d)
true_negatives = sum(d + d_out == 0)
false_positives = sum(d - d_out == -1)
false_negatives = sum(d - d_out == 1)
true_positives = sum(d + d_out == 2)

percent_true_negatives = sum(d + d_out == 0) / total * 100
percent_false_positives = sum(d - d_out == -1) / total * 100
percent_false_negatives = sum(d - d_out == 1) / total * 100
percent_true_positives = sum(d + d_out == 2) / total * 100

sensitivity = true_positives / (true_positives + false_negatives) * 100
specificity = true_negatives / (true_negatives + false_positives) * 100
```

B.3 figures.m

Generates figures for this report

```
%% Create figures

%% Load data
load('ecg_data');
ecg = ecg(1);

%% Raw ECG
figure(1);
subplot(2,1,1); plotecg(crop(ecg,1906:1907),'Latex','Annotate');
title('Raw ECG Showing a Normal Beat and a Premature Ventricular Contraction');
subplot(2,1,2); plotecg(crop(ecg,1603:1604),'Latex','Annotate');
title('Raw ECG Showing a Normal Beat and an Atrial Premature Beat');

%% Filtered ECG
figure(2);
subplot(2,1,1); plotecg(crop(ecg,126:127),'Latex');
title('Raw ECG');
fecg = filterecg(ecg);
subplot(2,1,2); plotecg(crop(fecg,126:127),'Latex');
title('Filtered ECG');

%% findqrs Stages
figure(3);
```

```
% filtered
subplot(2,3,1); plotecg(crop(fecg,457),'Latex','NoTitles','NoMarks');
title('Filtered ECG');
% difference
qrs = fecg;
qrs.signal = gradient(qrs.signal);
subplot(2,3,2); plotecg(crop(qrs,457),'Latex','NoTitles','NoMarks');
title('Differentiated Signal');
% square
qrs.signal = qrs.signal .^ 2;
subplot(2,3,3); plotecg(crop(qrs,457),'Latex','NoTitles','NoMarks');
title('Squared Signal');
% moving average
T = 0.150;                  % width in seconds
W = round(T*qrs.fs);        % width in samples
b = ones(1,W) / W;          % W-sample average
qrs.signal = filtfilt(b,1,qrs.signal);    % compute moving average
subplot(2,3,4); plotecg(crop(qrs,457),'Latex','NoTitles','NoMarks');
title('Moving Average');
th = rms(qrs.signal);
xs = xlim();
hold on;
line(xs,[th, th],'Color','Black');
hold off;
% max
qrs.signal = (qrs.signal > th) .* fecg.signal;
subplot(2,3,5); plotecg(crop(qrs,457),'Latex','NoTitles','NoMarks');
title('Max In Range');
qrs.ann = findqrs(ecg);
hold on;
x = fecg.time(qrs.ann(457));
y = fecg.signal(qrs.ann(457));
plot(x,y,'r*');
hold off;
% final result
fecg.ann = findqrs(ecg);
fecg.Nann = length(fecg.ann);
subplot(2,3,6); plotecg(crop(fecg,457),'Latex','NoTitles');
title('Final Result');

%% findqrs Performance
figure(4)
subplot(2,1,1); plotecg(crop(fecg,457,0.05,0.05),'Latex');
title('A Closer Look at \verb`findqrs`''s Precision');
load('ecg_data')
qrs = filterecg(ecg(2));
qrs.ann = findqrs(ecg(2));
qrs.Nann = length(qrs.ann);

subplot(2,1,2); plotecg(crop(qrs,2002:2006),'Latex');
title('Extra Beats Erroneously Detected by \verb`findqrs`');
```

References

[1] "Current estimates from the national health interview survey, 1995," Centers for Disease Control and Prevention, Tech. Rep. 199, 1995.

[2] R. G. Moody, George B; Mark, "Mit-bih arrhythmia database," 1992. [Online]. Available: http://dx.doi.org/10.13026/C2F305

[3] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C. K. Peng, and H. E. Stanley, "PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. E215–220, Jun 2000.

[4] J. Pan and W. J. Tompkins, "A real-time qrs detection algorithm," *IEEE Transactions on Biomedical Engineering*, vol. BME-32, no. 3, pp. 230–236, March 1985.

[5] I. Silva and G. B. Moody, "An open-source toolbox for analysing and processing PhysioNet databases in MATLAB and octave," *Journal of Open Research Software*, vol. 2, sep 2014. [Online]. Available: http://dx.doi.org/10.5334/jors.bi

[6] J. Shlens, "A tutorial on principal component analysis," *CoRR*, vol. abs/1404.1100, 2014. [Online]. Available: http://arxiv.org/abs/1404.1100

[7] J. Wilson, "Classification demo," 2016, unpublished.