THE CLASSIFICATION OF PVCs USING FILTER BANK FEATURES, INDUCTION OF DECISION TREES AND A FUZZY –RULE–BASED SYSTEM

by

OLIVER WIEBEN

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
(ELECTRICAL AND COMPUTER ENGINEERING)

at the

UNIVERSITY OF WISCONSIN-MADISON

1996

ABSTRACT

The classification of heart beats is an important component for automated arrhythmia monitoring devices and reported algorithms have their shortcomings. This study describes two different classifiers for the identification of premature ventricular contractions (PVCs) in surface ECGs. A decision tree algorithm based on inductive learning from a training set and a fuzzy-rule-based classifier are explained in detail. Traditional features for the classification task are extracted by analyzing the heart rate and morphology of the heart beats from a single lead. In addition, a novel set of features based on the use of a filter bank is presented. Filter banks allow for time-frequency dependent signal processing with low computational effort. The performance of the classifiers is evaluated on the MIT-BIH Database following the AAMI recommendations. The decision tree algorithm had a sensitivity of 85.29% and a positive predictivity of 85.23%, while the sensitivity of the fuzzy-rule-based system was 81.34% and the positive predictivity 80.64%.

I. INTRODUCTION

Correct classification of heart beats is the foundation for ECG monitoring systems such as in intensive care, automated analysis of long-term recordings, arrhythmia monitors, or even in cardiac defibrillators. Counting the occurrence of PVCs, also referred to as ventricular ectopic beats, is of particular interest to support the detection of ventricular tachycardia and to evaluate the regularity of the depolarization in the ventricles. For example, the risk of sudden death for patients with a structural heart disease is higher with an increased occurrence of PVCs [1].

PVCs are extra ventricular contractions originating in an ectopic pacemaker in the ventricles. They are mostly caused by enhanced automaticity or reentry mechanisms in the propagation of the electrical impulse in the conduction system of the heart. They are usually characterized by an atypically wide and bizarre QRS complex and a shorter than average R-R interval from the previous R wave (see Figure 1), while the underlying rhythm does not change [2]. The amplitude of the peak for a PVC often differs in magnitude and is sometimes of opposite polarity in comparison to the peak of the QRS complex in the underlying rhythm. Furthermore, a full compensatory pause commonly follows a PVC. PVCs may occur singly or in groups of two or more (ventricular group beats). A ventricular tachycardia exists if three or more consecutive premature ventricular contractions are present, occurring at a heart rate greater than 100 beats per minute.

Unfortunately, variability in morphology and heart rate from patient to patient and even for the same patient, noise present in the signal, or other arrhythmias make correct classification a challenging task, and reported results leave room for improvement. Usually the first step in a classification process is the preprocessing of measured data before features (or ‘attributes’) are extracted to perform the actual classification. The key for a successful classifier is a well chosen set of features, which allow for discrimination between the desired classes. Different classification approaches based on features such as rate, shape and correlation with templates are proposed in the literature for classification of surface electrocardiographs (ECGs). Wang [3] describes different methods based on a heuristic analysis of features in general. Rappaport et al. [4] evaluate quantitatively morphology analysis, clustering algorithms, and different normalization procedures, concluding that the choice of the normalization process for the features derived from ECGs from different patients is more crucial for the classification results than the type of classifier. A more recent paper [5] compares self-learning techniques (in particular: an artificial neural network and an induction algorithm approach) with a classifier based on statistical analysis for differentiating between supraventricular and ventricular tachycardia. Other approaches include artificial neural networks using complete templates of the QRS complex as features [6-9]. Jenkins and Caswell [10] summarize classification methods including features, such as those related to rate and morphology, for the detection of ventricular tachycardia in electrograms. Most of these could be used for surface ECGs as well.

An evaluation of the performance of algorithms reviewed in the literature was difficult because the authors used different databases or their own data sets and different learning strategies for their algorithms. Even if the same database was chosen, the considered number of records and the sizes of the training and test sets differed. The *Association for the Advancement of Medical Instrumentation* (*AAMI*) recommends standards for the development of classifiers based on a training set and beat-by-beat testing of the performance of arrhythmia monitoring algorithms to avoid such confusion [11].

The purpose of this study is to introduce a new set of features based on a filter bank concept and to compare two methods for beat classification. The contribution of features extracted from a filter bank to a successful classification of PVCs is evaluated within a set of features based on the heart rate and morphology. Afonso et al. [12-14] suggest the use of a filter bank to address different issues of monitoring systems, such as signal enhancement and beat detection. This approach offers the possibility of processing the original signal in subbands representing different frequency ranges in the signal. Therefore, the filter bank allows for the completion of multiple tasks in real time with a single set of filters. In addition, it offers the opportunity for frequency-dependent signal processing at a downsampled rate.

The two classifiers, a decision tree and a fuzzy-rule-based system, differ widely in their implementation and representation of knowledge. The induction of a decision tree [15] represents a self-learning algorithm, which analyzes a classified training set to develop a successful strategy. On the other hand, the fuzzy logic approach is a rule-based system which takes advantage of knowledge implemented by a human expert. Fuzzy logic allows the use of linguistic variables to formulate rules, which are intuitively plausible. It also avoids fixed thresholds or distance measures and therefore accounts for the noise and uncertainties in measured data. The principles of neural networks and fuzzy logic can be combined [16].

II. Data Selection and Feature Extraction

A. Data Selection

The ECG records from long-term Holter recorders analyzed in this study are from the *MIT-BIH Arrhythmia Database* [17], which contains 48 records. This database was chosen because of its variety of rhythms, waveforms, noise, and artifacts, which represent characteristics of ECG recordings of different qualities and beat types. The records include PVCs with typical attributes. However, a significant quantity of unpredictable abnormalities make the correct identification of PVCs a difficult task. Moody et al. [18] summarize information on alternative applicable annotated databases for similar purposes. In comparison to the popular database of the *American Heart Association* (*AHA*) [19], which includes a great number of different premature ventricular contractions, Moody and Mark [20] state "Our observation, confirmed by those of many other users of both databases, suggest that arrhythmia detectors consistently perform better using the AHA Database, independent of which (if either) was used for training. Weaknesses in detector design are often exposed by the rhythms, the beat morphologies, and the noise in the MIT-BIH Database."

The original analog recorded signals were bandpass filtered before being digitized by the producer of the database at a rate of 360 samples per second. The corner frequencies of the bandpass were at 0.1 and 100 Hz to prevent the loss of recoverable frequency components of the original recordings and to avoid foldover problems. The upper channel, a modified limb lead II (MLII) in all of the analyzed records except for record 114, was chosen, because normal QRS complexes usually have a large positive R wave in this lead.

The *AAMI* standard recommends the exclusion of records with paced beats (records 102, 104, 107, and 217) from the performance evaluation of beat-by-beat testing of arrhythmia monitoring algorithms. As a result, this leaves a total of 44 records with a length of approximately 30 minutes each. Premature ventricular contractions and ventricular escape beats are grouped in the class *P* of positive instances. An annotated normal beat, left and right bundle branch block beat, atrial premature and aberrated atrial premature beat, nodal (junctional) premature beat, supraventricular premature beat, or an atrial or nodal (junctional) escape beat are referred to as a ‘non-PVC’ or negative instance, belonging to the class *N*. The *AAMI* protocol excludes ventricular fusion and unknown (unclassifiable) beats from the performance table but does not exclude these records.

B. Preprocessing of the Data

The annotations in the database for the locations of R waves are used as an ideal beat detector. Unfortunately not all beat labels are located precisely at the actual peak position due to shifting in copying processes and manually inserted labels while creating the database. An algorithm searching for the local maximum or minimum, depending on the direction of the heart axis at the specific beat, locates the correct sample number to indicate the real location and magnitude of the peak.

Characteristic variables in the ECGs such as the current heart rate or the magnitude of the R wave for the same beat type may vary significantly from patient to patient. They may even vary for the same patient, when something in the environment of the patient changes, e.g. arising from a resting position, exercising, or mental influences. A normalization process must be used to compensate for these differences in order to feed the classifier with comparable features. The choice of the normalization process is crucial because it has a great impact on the results of the classifier [4]. We chose to establish a reference or normal signal by averaging the specific variable for the first eight beats and then adapting the buffer of eight measurements with current normal beats. A feature of a beat is considered to represent normal values, whenever an incoming measurement lies within $\pm 15\%$ of the mean of the buffer or $\pm 15\%$ of the last measurement added to the buffer. The whole buffer is overwritten if all the measurements of the last eight beats lie within $\pm 15\%$ of the mean of these eight measurements to adapt, for example, to a change in the direction of the heart axis or abrupt changes in the heart rate.

C. Heart Rate Features

Three features related to the heart rate are extracted from the ECG recordings using the corrected peak locations: 1) the normalized R-R interval *norm-*RR_0 between the preceding and the current R waves (based on the coupling interval RR_0), 2) the ratio RR_1*-to-*RR_0 obtained by dividing the R-R interval RR_1 between the current and the following R waves by the previous R-R interval RR_0 (see Figure 1), and 3) the *irregularity* of the heart rate.

The normalized R-R interval is calculated as the ratio of RR_0 divided by the mean of the last eight R-R intervals considered to be normal. Typically this ratio fluctuates slightly near a value of 1 for a normal heart rhythm and is significantly smaller than 1 if a PVC occurs [21]. The feature RR_1*-to-*RR_0 is usually a good measurement for a single PVC occurring between non-PVCs but causes a delay of one beat for the classification in a real time application. The *irregularity* of the heart rate is calculated as the ratio of the standard deviation s over the mean $\overline{RR}$ of the last 8 beats and is expressed as a percentage [22]. It is also referred to as the relative variability or coefficient of variation. The *irregularity* is calculated with a delay of one beat as well to include the R-R interval after the analyzed beat RR_1 in the calculation:

$$\overline{RR} = \sum_n RR_n$$

$$s^2 = \frac{\sum_n (RR_n - \overline{RR})^2}{n-1}$$

$$irregularity = \frac{s}{\overline{RR}}$$

D. Morphological Features

Four morphological features were chosen to represent information about the shape of the QRS complexes: 1) amplitude, 2) peakdirection, and 3) width of the largest local minimum or maximum nearest the annotation, and 4) the normalized peak-to-peak distance within the QRS complex.

An offset in the ECG records must be considered before determining the magnitude of the peak nearest the annotation as the the maximal deflection between baseline and peak. The mean of the last 360 samples (one second) is subtracted from the current peak to account for the baseline offset. The normalized amplitude *norm-amp* is derived as the ratio of the amplitude over the mean of the last eight normal amplitudes and is always represented as an absolute value. The *peakdirection* indicates whether or not the actual peak points in the same direction as the normal peaks.

A wide QRS complex (longer than 120 ms) is characteristic for PVCs, but it is difficult to determine the fiducial points in an ECG with noise, artifacts or rhythm disturbances present. Therefore, the width of the peak *peakwidth* as a correlated measurement to the QRS width is approximated. This width is derived by the time interval that the distance between the signal and the baseline exceeds 15% of the maximum magnitude of the peak. The interval is limited to 61 samples starting 20 samples before the occurrence of the maximum deflection.

The peak-to-peak distance is found as the difference between the maximum and minimum of the signal within an interval of 20 samples before and 49 samples after the annotated beat. This leaves 136 ms after the peak of the R wave to include the S wave and a fraction of any abnormal ST segment. Normalization of the peak-to-peak distance *norm-peak-to-peak* followed the same rules as the normalization process for the actual R-R interval and the amplitude of the R wave.

E. Features extracted from a Filter Bank

Filter banks allow for the separation of a signal $x(n)$ into different subbands, each representing the signal content in a certain frequency range. A set of analysis filters $H_k(z)$, ($k = 0, 1, 2, \ldots, M-1$) is designed to decompose a signal into M subbands, which can be downsampled without losing information of the original signal as long as the downsampling factor is smaller than M. A maximally decimated filter bank, where the downsampling factor equals the number of subbands, is used in the experiment setup (see Figure 2).

The downsampled signals $w_k(m)$, where $m = M{\cdot}n$, change their magnitude with every Mth sample of the original signal. Each subband may be processed individually or used to extract information related to different frequency ranges. Upsampling of the downsampled and processed signals in the subbands $wp_k(m)$ and filtering with a set of synthesis filters $F_k(z)$ leads to output subbands $o_k(n)$ of the same sampling frequency as the input signal of the filter bank $x(n)$.

An unprocessed downsampled signal ($wp_k(m) = w_k(m)$) can be perfectly reconstructed with an adequate set of filters [23] by adding up all M subbands $o_k(n)$ to get $\hat{x}(n)$. The design of the filters compensates for aliasing, amplitude and phase distortion in the subbands, and the output sequence is identical to the input sequence except for the delay due to filtering. An efficiently implemented filter bank decomposes and reconstructs M datapoints only for every Mth incoming datapoint $x(M{\cdot}n)$. Thus, the use of a filter bank including M filters with a filter length of L requires the same computational effort as only one filter of length L without downsampling. Possible processing and extraction of information in the downsampled subbands could be advantageous for real time processing algorithms.

The filters implemented in this filter bank were developed by the lapped orthogonal transform (LOT) [24]. The filter bank contains $M = 32$ filters with equal bandwidths and each of a length $L = 2{\cdot}M = 64$. Due to the frequency range of the digitized signal from 0 to 180 Hz, the bandwidth of every subband is 180 Hz / 32 = 5.625 Hz. The sampling frequency of each downsampled subband is 360 Hz/M = 11.25 Hz.

The analysis filter of the first subband (subband 0) is a low-pass filter with a cutoff frequency at 5.625 Hz. The next 30 subband filters are bandpass filters, and the filter of the last subband (subband 31) is a high-pass filter with a cutoff frequency at 174.375 Hz (see Figure 3). The even subbands have symmetric analysis and synthesis filters, and the uneven subbands have antisymmetric filters. All filters have linear phase to allow independent analysis in the subbands with the same delay for fiducial points. They are orthogonal to ensure that the energy of the signal is preserved in each subband. All filters have a finite impulse response, and the attenuation of the highest sidelobes is greater than -20 dB [14].

The idea behind the use of different subbands is to recognize the PVCs by their energy distribution over the frequency range. Intuitively, the wider shape of the PVCs compared to normal beats with sharper peaks, leads to the assumption that the low-frequency energy is greater than the high-frequency energy. Thakor et al. [25] present a power spectral analysis of ECG waveforms, in which the relative power spectra of the QRS complex has a high amplitude between 5-25 Hz. Clayton et al. [26] evaluated the frequency spectrum of ventricular tachycardia and ventricular fibrillation with a Fast Fourier Transform (FFT). They used consecutive one second epochs. They conclude that differences in the frequency spectrum, such as the mean dominant frequency and the width and proportional size of the peak in the spectrum, allow differentiation of different ventricular arrhythmias.

The time interval between two samples in the downsampled subbands is only 88.89 ms. This leaves a representation of a normal QRS complex (80 ms) by two samples, which is insufficient for a detailed analysis of the shape of the waveform. Therefore, all seven features using the filter bank are extracted from upsampled subbands. Let the actual energy $e_k(n)$ of a subband k be represented by the squared actual amplitude $o_k(n)$ in the subband (see Figure 3):

$$e_k(n) = o_k^2(n).$$

The mean value of the amplitude during the previous second was subtracted from the amplitude in subband 0 to account for the dc component. We implemented a moving window integrator (MWI), which adds up the squared amplitudes over a rectangular window with a length of 122 ms (see Figure 4). The maximum energy E_k in a subband is then represented by the peak in the output of the MWI within ± 35 samples of the annotated beat location. The maximum energy $E_{x,x+1}$ in adjacent subbands was determined by the same procedure after summing up the two outputs $o_k(n)$ and $o_{k+1}(n)$. The feature set consists of 5 ratios of maximum energies in different adjacent subbands and 2 different widths of the MWI for the energy of the lowest subbands $E_{0,1}$. Two parameters *MWI-15* and *MWI-30* represent the time that the actual energy is above 15% or respectively 30% of the maximum energy. The search for the threshold crossings is limited to 35 samples before and 125 samples after the annotated beat. The energy ratios are calculated with respect to the energy $E_{0,1}$, which represents the sum of the energy of the signal in subbands 0 and 1 corresponding to a frequency range from 0 to 12.25 Hz. The five ratios are $E_{2,3}/E_{0,1}$, $E_{4,5}/E_{0,1}$, $E_{6,7}/E_{0,1}$, $E_{8,9}/E_{0,1}$, and a weighted ratio $E_{0,1}/E_{07}$ of most of these energies combined:

$$\frac{E_{0,1}}{E_{07}} = \frac{E_{0,1}}{E_{0,1} + 2E_{2,3} + 10E_{4,5} + 100E_{6,7}}$$

where E_{07} is the weighted sum of the energies in all subbands from 0 through 7. The weighting factors had to be considered because of the smaller signal amplitudes in the higher subbands. $E_{0,1}$ is in both the numerator and denominator of the ratios to bound the result to values between 0 and 1.

III. Classification Methods

Two different methods for the classification task are proposed: the induction of decision trees and a rule-based fuzzy logic classifier. Decision trees learn from the properties in features of a training set, consisting of classified examples, without a priori knowledge (except the choice of the features). The fuzzy logic classifier is a so called knowledge-based expert system, where the rules are formulated by an human expert. An analysis of the features in the training set may only help to fine-tune these rules, which are implemented using linguistic variables.

A. Classification with Induction of Decision Trees

Induction means to extract knowledge about the behavior of features in a set of given examples which are related to different classes and to apply this knowledge to unseen instances. A decision tree provides a kind of representation of acquired knowledge. The advantages of decision trees are that they are fairly robust, capable of dealing with noisy and even incomplete data, and result in a simple classifier with easy to interpret decision rules [15].

A decision tree is represented by two elements: nodes and leaves (see Figure 5). A node represents a test being performed on a feature and contains branches for the possible outcomes. The branches each lead to another node or a leaf, which determines the affiliation to a class. The classification process starts in the node at the roof of the tree, performs tests in successivbe nodes and takes the appropriate branches until categorization to a class in a leaf.

The number of branches which separate the root from the most distant leaf determines the level of the decision tree.

We used the decision tree algorithm *MC4* from the *MLC++* library proposed by Kohavi et al. [27]. *MC4* stands for the *MLC++* implementation of C4.5, an algorithm developed by Quinlan [28]. The stages in creating the final decision tree include the construction of an initial tree and a subsequent pruning of the branches to simplify decision rules which overfit the data.

The criteria to generate the initial decision tree [29, 15] are based on an analysis adopted from information theory. In general, the information I conveyed by the answer to a question depends on the probabilities of the m possible answers $P(x_{\mathrm{i}})$ and is measured in bits:

$$I(P(x_1),\ldots,P(x_m)) = \sum_{i=1}^{m} -P(x_i)\log_2 P(x_i)\,.$$

Each node in the decision tree splits the training set into branches. Let n be the number of positive instances in the class N and p the number of negative instances of the class P in the training set S. Then the probability that an arbitrary instance belongs to the class P is simply $p/(p+n)$, while the probability of belonging to class N is $n/(p+n)$. Therefore, the estimated information required to find the correct answer (class membership) of an instance is:

$$I\left(\frac{p}{p+n},\frac{n}{p+n}\right) = -\frac{p}{p+n}\log_2\left(\frac{p}{p+n}\right) - \frac{n}{p+n}\log_2\left(\frac{n}{p+n}\right).$$

Now let F be the feature in the node at the root of the tree with the m discrete values $\{F_1, F_2, ..., F_m\}$. The feature divides the training set into subsets $\{S_1, S_2,..., S_m\}$ (branches in the tree), where each subset S_i contains p_i positive and n_i negative instances. The estimated information needed to find the correct class in the subset S_i is $I(p_i, n_i)$ bits. An arbitrary example has the probability $(p_i+n_i)/(p+n)$ to be in the subset S_i. The expected information remained to identify the class memberships for an example is derived by averaging the information for the subsets weighted with their probability of occurrence:

$$remainder(F) = \sum_{i=1}^{m} \frac{p_i + n_i}{p+n} I(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}) .$$

Therefore, the *gain* in information due to partitioning the examples in the different subsets of feature F is the difference between the information needed at the node and the remaining average information at the branches:

$$gain(F) = I(\frac{p}{p+n}, \frac{n}{p+n}) - remainder(F) .$$

This *gain* in information could be used as a criterion for evaluating the significance of the features, but it fails in evaluating the effectiveness of branches which only have a small number of instances. A form of normalization compensates for the unbalanced evaluation of the features. The information content *split info* of dividing into subsets is measured, instead of evaluating the information indicating the class memberships themselves:

$$split\ info(F) = -\sum_{i=1}^{m} \frac{p_i + n_i}{p+n} \log_2(\frac{p_i + n_i}{p+n}) .$$

Now the *gain ratio* (the ratio of *gain* over *split info*) evaluates the contribution of the branching to the classification process:

$$gain\ ratio(F) = \frac{gain(F)}{split\ info(F)} .$$

The *gain ratio* indicates the performance of the features at different nodes well.

If the features are continuous and not discrete, thresholds for the subsets must be defined. In this case the examples in the training set are sorted for each feature by the numerical value of the specific feature, and the *gain ratio* is evaluated for each possible threshold in between neighboring data points. The optimal threshold is determined by the maximum *gain ratio*.

The *gain ratio* is calculated for each of the features, and the order of the examples in the training set does not influence the result of the algorithm. The decision tree is generated by placing the feature with the highest *gain ratio* at the root of the tree. Each subset or branch induces a new decision tree evaluated by the same criterion. The general principle of inductive learning, often called Ockham's razor, is that "The most likely hypothesis is the simplest one that is consistent with all observations."[29]. Thus, the search for the perfect decision tree is determined either when all examples in the training set are correctly classified, when more branches do not increase the accuracy of the algorithm on the training set, or when the complexity of the tree exceeds certain thresholds.

Considering the fact that the developed decision tree should perform on unseen test data, the resulting classifier should not adapt too specifically on single examples but rather on trends in the training set, otherwise it overfits the data while losing generalization. Two independent procedures help to prevent an overfitting: finding criteria to stop the growing process of the tree and pruning of the derived tree. Parameters in *MC4* allow limiting the number of levels in the tree and determining a minimum amount of instances *min instances*, that must trickle down at least two branches of a node. This amount is derived for each node over a weighting factor *split weight*, which represents the percentage of instances divided by the number of classes. If the number of instances present in a node is either very high or very low, the calculated *min instances* may not be adequate. To avoid this problem, *min instances* is bounded on both sides. In case it exceeds the adjusted maximum or falls below a minimum, it will take on the bounding value. Pruning of the decision tree involves the removal of branches, which do not contribute significantly to the performance, for the sake of less complex results. It may cause less accuracy in the classification on the training set but a higher accuracy on test sets, because of the generalization in the parameters.

B. Classification with a Fuzzy Logic System

Zadeh [30] introduced the theory of fuzzy sets, where the membership of objects to classes is a matter of degree. This is an extension of the conventional crisp set theory, where partial memberships are not possible. A Fuzzy Logic System (FLS) maps crisp inputs, such as a feature vector, nonlinearly into crisp outputs. It uses fuzzy sets and fuzzy logic, rather than crisp logic with hard thresholds, to tolerate imprecison and noise in the nature of the data. Fuzzy logic allows for implementation of rules by an expert using linguistic variables. The advantages of a FLS are that it is robust and cost-effective in the implementation of existing knowledge. It contains the four components fuzzification, rules, inference engine, and defuzzification [31] (see Figure 6).

The process of fuzzification is a form of quantization, where crisp numbers are mapped into fuzzy sets. The degree of membership of an input feature to fuzzy set is evaluated by the membership functions of the fuzzy sets. Fuzzy sets are defined within the range of the feature (universe of discourse) and associate a degree of similarity between the value of the feature and the fuzzy subset. More than one subset may be defined for each feature and the degree of membership to a set varies from 0 to 1 (see Figure 7). The relation between fuzzy sets can be evaluated with operators such as the union (*or*) or the intersection (*and*) by different mathematical methods such as determining the maximum or minimum or the product of the degrees of membership in each set.

Rules in a FLS have the form of IF-THEN statements:

IF *antecedent* THEN *consequent* ,

where the antecedent represents a fuzzy set or relations between fuzzy sets and the consequent assigns a fuzzy set to the ouput, again within a range of 0 to 1 (see Figure 8). Rules are imbedded using linguistic variables and map fuzzy sets into fuzzy sets. As they are all evaluated in parallel, the order of rules is unimportant.

The inference engine determines the degree of activation of each rule based on the values of the antecedents. It also combines the usually equally weighted rules which may have coincidental or contradictory consequents. The results derived in this stage are fuzzy ouput sets, which is not a desirable output format of the system.

The defuzzifier transforms the fuzzy sets into crisp numbers, because the output of a fuzzy system has to be in a scalar form. Popular defuzzification methods are the centroid of area method, mean of maximum method, and largest and smallest of maximum method.

The design of a fuzzy logic classifier for the implementation of rules requires the choice of features and membership functions for the fuzzification, as well as combining the rules and defuzzification of the output sets. Adjusting the inference engine and the parameters of the membership functions is usually done arbitrarily. Visual inspection of the features (scatter plots) can be used to explore structure in the data [32]. Likewise unsupervised learning algorithms such as cluster analysis of the features can be used to adjust the membership functions [32].

The classifier was implemented using the *Fuzzy Logic Toolbox for use with MATLAB* [33]. All membership functions associated with the features were chosen to be generalized bell curve functions:

$$f(x) = \frac{1}{1 + \left| \frac{x - c}{a} \right|^{2b}}$$

The parameter *c* locates the maximum of the symmetric bell curve, while *b* changes the slope. The distance from the center to both sides of the bell, where the value of the function drops to 0.5, is determined by *a*. The bell membership function has the advantage of being flexible, and therefore adaptable, to different desired shapes (see Figure 7). Table I summarizes all fuzzy input sets and their parameters. A total of nine features were chosen to reduce the complexity of the system. The two filter bank features *MWI-15* and $E_{0,1}/E_{07}$ were selected because their characteristics were similar to *MWI-30* and the other energy ratios. The implementation of the membership functions required fine-tuning with scatter plots from the training set. The rules implemented in the system are listed in Table II. The fuzzy output *beat type* is defined on a universe of discourse from 0 to 1. Two triangular waveforms, namely *non-PVC* and *PVC*, with a width of 0.5 are centered at 0.25 (*non-PVC*) and 0.75 (*PVC*). The intersection of fuzzy sets is implemented as the minimum operator and the union as the maximum operator. The contribution of all rules is accredited for as the sum of each rule’s output set. The defuzzification reduces the degree of membership to the sets *non-PVC* and *PVC* to a single number with the center of area method. If that number is larger than or equal to 0.5, the incoming beat will be defined as a PVC, otherwise it will be classified as a non-PVC.

C. Training and Testing of the Classifiers

The *AAMI* standard recommends methods for the training and testing practice as well as the presentation of performance results. The test should report the performance of the algorithm to new data and, therefore, the training and test set must be strictly separated from each other. Different methods, such as cross validation, are used to evaluate classification performance in the field of pattern recognition. However, following the *AAMI* recommended standards the first five minutes of each record are used for training purposes while the last 25 minutes are used for testing in the *MIT-BIH Database*. Only the training set should be used to extract information on rhythm analysis, morphology of beats, cluster plots, tuning the algorithm, etc.

The first eight beats of each record are used as a warm-up period for the normalization process of the heart rate and do not contribute to the training set. Obviously the feature RR_1-*to*-RR_0 and the *irregularity* of the heart rate cannot be calculated for the last beat in a set either. Only 24 of the records (records 105, 106, 108, 109, 114, 116, 118, 119, 200, 202, 203, 205, 207, 208, 210, 213, 214, 215, 219, 221) include PVCs in the first five minutes. These records were considered as the training set for the decision tree algorithm with 1080 PVCs and 8089 non-PVCs. In addition, features for the last beat in each record are missing in the test set because of the reasons mentioned above. The algorithms take advantage of the first five minutes as the warm-up period to establish normalized features and thus, no beats in the beginning of the test sets are missing.

Segments of ventricular flutter or fibrillation show properties differing from other heart rhythms and are missing characteristic QRS complexes. According to the *AAMI* protocol, they are excluded from the statistics as well. The standard analysis software in the *MIT-BIH Database* includes these considerations and was used to report the test results. The test set includes 5,900 PVCs and 77,394 non-PVCs.

IV. RESULTS

Comparing the classifications of the algorithm with the annotations of the database beat-by-beat could result in four different categories: a PVC could be picked up by the algorithm either as a PVC, which is referred to as a true positive (*TP*) classification, or as a non-PVC, which is referred to as a false positive (*FP*) event. A beat erroneously classified as a PVC by the algorithm represents a false negative (*FN*) classification, while a non-PVC correctly detected as such is a true negative (*TN*) reading.

The sensitivity (*Se*) for detection of premature ventricular contractions is defined as the fraction of correctly classified PVCs divided by total number of PVCs present in the data set:

$$Se = \frac{TP}{TP + FN}$$

The fraction of beats correctly classified as PVCs divided by all beats classified as PVCs is the positive predictivity (*+P*):

$$+P = \frac{TP}{TP + FP}\,.$$

The results reported on the 44 records are given as gross statistics, which means that they are derived by summing up *FN*, *FP*, *TN*, and *TP* in the records.

Decision Tree

The numbers of nodes in the generated decision trees were varied to evaluate the relationship between the complexity of the tree and the positive predictivity and sensitivity of the classification. The *split weight* was varied between 0 and 5 to influence the growth of the decision tree and *min instances* was bounded between 2 and 25 instances. Table III shows the results of this experiment with a fixed pruning factor. A split weight of 0.2 achieved the most balanced classification in terms of sensitivity (85.29%) and positive predictivity (85.23%) for all beats. Table IV shows the beat-by-beat statistics for all 44 records.

Fuzzy-Rule-Based Classifier

The results for the best performance of the Fuzzy Logic classifier are summarized in Table V. The achieved sensitivity is 81.34% and the positive predictivity is 80.64%. Experiments using a fuzzy c-mean clustering algorithm [34] for adjusting the membership functions to the training set resulted in a worse performance.

V. Discussion

The ratio of the energy in higher subbands to lower subbands was found to be significantly lower for most PVCs compared to non-PVCs. The width of a moving window integrator around the peak is an additional criteria for the separation of PVCs from non-PVCs.

Therefore, features from a filter bank support the classification process, although they cannot be extracted in the downsampled subbands. Time-frequency dependent features could be used for other analysis tasks in ECG monitoring as well, including identification of paced beats or rhythms such as ventricular fibrillation. Filter banks with higher attenuation in the sidelobes reduce the aliasing, but the trade-off would be an increased filter length. Longer filters prolong the latency in the system for other tasks such as signal enhancement and beat detection. The same issues must be considered when a filter bank with more subbands, and therefore smaller bandwidths, is desired. The construction of filter banks with uneven bandwidths would allow for higher frequency resolutions in certain frequency ranges. The lower subbands, which contain most of the ECG signal, could be split into finer ranges than the subbands for higher frequencies. The problem in the design of these filters is to maintain the properties of linear phase, orthogonality and perfect reconstruction.

The decision tree algorithm used features related to the heart rate and morphology in the highest levels of the tree. They appear to be more discriminative in the differentiation between PVCs and non-PVCs. The algorithm produced robust classification schemes, which differed little for changes in the pruning and *split weight* parameters. Each leaf in a decision tree could be interpreted as a rule with hard thresholds. The level of the leaf in the tree indicates the number of tests to be performed for the final classification.

The fuzzy-rule-based classifier did not perform as well as the decision tree classifier. Of significance is that it only uses 9 features and 15 rules with a maximum of two fuzzy sets in each of the antecedents. Fine-tuning of the fuzzy logic system by changing membership functions or adding rules turned out to be difficult due to the nonlinear behavior of the classifier. The classifier is robust as well and has a more balanced performance averaged over the 44 records (74.58% sensitivity and 66.54% positive predictivity) than the reported decision tree classifier (80.47% sensitivity and 58% positive predictivity).

Both classifiers behave similarly for the records in the database and leave room for improvement in their performance. Records with atrial fibrillation and atrial premature beats (records 201, 202, 203, and 219), where the heart rate and R-R intervals are irregular, increase the number of *FN* and *TN* classifications. This is due to the fact that the classifiers emphasize the heart rate features, which are good discriminators in the majority of the records. Almost all of the fusion beats (623 out of 638) were picked up as non-PVCs by the fuzzy-rule-based classifier, while the decision tree classified 490 fusion beats as non-PVCs and 148 as PVCs.

The results for the decision tree algorithm increased significantly when trained and tested on a selected subset of the 44 records. However, the purpose of the study was to evaluate the performance of the algorithms on a large number of different records to identify the strengths of the classifiers as well as their limitations. The most important factors in a classification process are the choice of features and their normalization. More features such as the cross-correlation with different beat templates or an analysis of the complete ST segment and T wave could be used as additional criteria. Finding better locations of fiducial points could be helpful as well. A feedback loop from the classifier to the process of normalization of the features would result in more accurate features. Incorporation of the information from additional leads of the ECG would probably result in a better performance. Besides possible electrode problems (e.g. dead segments in the signal) the ECG may have vary small amplitude changes in one lead. The waveforms of PVCs may appear to be similar to non-PVCs in one lead while differences may be excentuated in a second lead. Moody and Mark [35] report a 42% reduction in number of *TN* classifications when analyzing two ECG leads simultaneously instead of just one lead. The problem with a multilead analysis is to find criteria to determine which lead to choose when there are contradictory results.

REFERENCES

[1] M. Hamdan and M. Scheinman, "Current approaches in patients with ventricular tachyarrhythmias," *Med. Clin. North Am.*, vol. 79, no. 5, pp. 1097-1120, Sept. 1995.

[2] R. J. Huszar, *Basic Dysrhythmias: interpretation & managment*. St. Louis: Mosby, 1994.

[3] J. Y. J. Wang, "Application of pattern recognition techniques to QRS complex classification - a review," in *Computers in Cardiology 1983.* Los Alamitos, CA: IEEE Computer Society Press, 1983, pp. 77-82.

[4] S. H. Rappaport, L. Gillick, G. B. Moody, and R. G. Mark, "QRS morphology classification: quantitative evaluation of different strategies," in *Computers in Cardiology 1982.* Los Alamitos, CA: IEEE Computer Society Press, 1982, pp. 33-38.

[5] W. R. Dassen, V. L. Karthaus, J. L. Talmon, R. G. Mulleneers, J. L. Smeets, and H. J. Wellens, "Evaluation of new self learning techniques for the generation of criteria for differentiation of wide-QRS tachycardia in supraventricular tachycardia and ventricular tachycardia," *Clin. Cardiol.*, vol. 18, pp. 103-108, Sept. 1995.

[6] H. S. Chow, G. B. Moody, and R. G. Mark "Detection of ventricular ectopic beats using neural networks," in *Computers in Cardiology 1992.* Los Alamitos, CA: IEEE Computer Society Press, 1992, pp. 659-662.

[7] Y. H. Hu, W. J. Tompkins, J. L. Urrusti, and V. X. Afonso, "Applications of artificial neural networks for ECG signal detection and classification," *J. Electrocardiol.*, vol. 26, suppl., pp. 66-73, 1994.

[8] R. Silipo, A. Taddei, M. Varanini, and C. Marchesi, "Classification of arrhythmic events in ambulatory electrocardiogram, using artificial neural networks," *Comput. Biomed. Res.*, vol. 28, pp. 305-318, Aug. 1995.

[9] R. Watrous and G. Towell, "A patient adaptive neural network ECG patient monitoring algorithm," in *Computers in Cardiology 1995.* Los Alamitos, CA: IEEE Computer Society Press, 1995, pp. 229-232.

[10] J. M. Jenkins and S. A. Caswell, "Detection algorithms in implantable cardioverter defibrillators," *Proc. IEEE*, vol. 84, no. 3, pp. 428-445, March 1996.

[11] Advancement of Medical Instrumentation (AAMI), "Recommended practice for testing and reporting performance results of ventricular arrhythmia detection algorithms (draft)". Available from: AAMI, 1901 North Fort Myer Drive, Suite 602, Arlington, VA 22209, Jan. 1986.

[12] V. X. Afonso, W. J. Tompkins, T. Q. Nguyen, S. Trautmann, and S. Luo, "Filter bank-based processing of the stress ECG," *Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc.*, vol. 17, pp. 887-888, Sept. 1995.

[13] V. X. Afonso, W. J. Tompkins, T. Q. Nguyen, S. Luo, "Filter bank-based ECG beat detection", *Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc.*, vol. 18, Nov. 1996

[14] V. X. Afonso, W. J. Tompkins, T. Q. Nguyen, K. Michler, S. Luo, "Comparing stress ECG enhancement algorithms: with an introduction to a filter bank based approach," *IEEE Eng. Med. and Biol. Mag.*, vol. 15, no. 3, pp. 37-44, May/June 1996.

[15] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81-106, 1986.

[16] F. M. Ham and S. Han, "Classification of cardiac arrhythmias using fuzzy ARTMAP," *IEEE Trans. Biomed. Eng.*, vol. 43, no. 4, pp. 425-430, April 1996.

[17] *MIT-BIH Arrhythmia Database.* Available from: MIT-BIH Database Distribution, Massachusetts Institute of Technology Room 20A-113, 77 Massachusetts Avenue, Cambridge, MA 02139, Aug. 1988.

[18] G. B. Moody, C. L. Feldman, and J. J. Bailey, "Standards and applicable databases for long-term ECG monitoring," *J. Electrocard.*, vol. 26, suppl., pp. 151-155, 1994.

[19] American Heart Association Database for the Evaluation of Ventricular Arrhythmia Detectors. Available from: ECRI, 5200 Butler Pike, Plymouth Meeting, PA 19462.

[20] G. B. Moody and R. G. Mark, "The MIT-BIH arrhythmia database on CD-ROM and software for use with it," in *Computers in Cardiology 1990.* Los Alamitos, CA: IEEE Computer Society Press, 1991, pp. 185-188.

[21] W. J. Tompkins, *Biomedical Digital Signal Processing: C-Language examples and laboratory experiments for the IBM_PC*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[22] K. L. Ripley, T. E. Bump, and R. C. Arzbaecher, “Evaluation of techniques for recognition of ventricular arrhythmias by implanted devices,” *IEEE Trans. Biomed. Eng.*, vol. 36, no. 6, pp. 618-624, June 1989.

[23] A. K. Soman, P. P. Vaidyanathan, and T. Q. Nguyen, “Linear phase paraunitary filter banks: theory, factorizations and designs,” *IEEE Trans. Signal Processing.*, vol. 41, no. 12, pp. 3480-3495, 1993.

[24] H. S. Malvar, *Signal processing with lapped transforms*. Norwood, MA: Artech House, 1992.

[25] N. V. Thakor, J. G. Webster, and W. J. Tompkins, “Estimation of QRS complex power spectra for design of a QRS filter,” *IEEE Trans. Biomed. Eng.*, vol. 31, no. 11, pp. 702-706, Nov. 1984.

[26] R. H. Clayton, A. Murray, and R. W. F. Campbell, “Objective features of the surface electrocardiogram during ventricular tachyarrhythmias,” Eur. Heart J., vol. 16, pp. 1115-1119, Aug. 1995.

[27] R. Kohavi, D. Sommerfield, and J. Dougherty, “Data mining using MLC++: a machine learning library in C++,” to appear in *Tools in Artificial Intelligence*. Los Alamitos, CA: IEEE Computer Society Press, 1996 (see http://www.sgi.com/Technology/mlc).

[28] J. R. Quinlan, *C4.5 Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers, 1993.

[29] S. J. Russell and P. Norvig, *Artificial Intelligence: A modern approach.* Englewood Cliffs, NJ: Prentice Hall, 1995.

[30] L. Zadeh, “Fuzzy sets,” *Inform. Contr.*, vol. 8, no. 3, pp. 338-352, June 1965.

[31] J. M. Mendel, “Fuzzy logic systems for engineering: a tutorial,” *Proc. IEEE*, vol. 83, no. 3, pp. 345-377, March 1995.

[32] J. C. Bezdek and S. K. Pal, *Fuzzy Models for Pattern Recognition.* New York: IEEE Press, 1992.

[33] N. Gulley and J. S. Jang, *Fuzzy Logic Toolbox for Use with MATLAB.* Available from: The MathWorks Inc., 24 Prime Park Way, Natick, MA 01760-1500, Jan. 1995.

[34] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms.* New York: Plenum Press, 1981.

[35] G. B. Moody and R. G. Mark, “Development and evaluation of a 2-lead ECG analysis program,” in *Computers in Cardiology 1982.* Los Alamitos, CA: IEEE Computer Society Press, 1982, pp. 39-44.

List of Figure Captions

Table I

Linguistic Description of the Fuzzy Input Sets and the Parameters *a*, *b*, and *c* of their generalized Bell Curve Membership Functions.

Table II

Antecedents and Consequents of the 15 Rules Implemented in the Fuzzy Logic System.

Table III

Results of the Decision Tree Algorithm for varying Split Weights. The Number of Nodes, Leaves, and Features Chosen by the Algorithm are Included. Sensitivity (*Se*) and Positive Predictivity (*+P*) for the Training and Test Set are Reported as Gross Statistics.

Table IV

Beat-by-Beat Performance Report for PVC Classification with the Decision Tree Algorithm and a Splitting Weight of 0.2. The Number of True negative (*TN*), False Negative (*FN*), True positive (*TP*), and False Positive (*FP*) Classifications, the Sensitivity (*Se*) and Positive Predictivity (*+P*), and the Gross and Average Statistics are Listed for 44 Records of the MIT-BIH Database.

Table V

Beat-by-Beat Performance Report for PVC Classification with the Fuzzy-Rule-Based System using a total of 9 features and 15 rules. The Number of True negative (*TN*), False Negative (*FN*), True positive (*TP*), and False Positive (*FP*) Classifications, the Sensitivity (*Se*) and Positive Predictivity (*+P*), and the Gross and Average Statistics are Listed for 44 Records of the MIT-BIH Database.

Fig. 1. Premature ventricular contractions (PVCs) in record 116. PVCs are labeled as ‘P’ and non-PVCs are labeled as ‘N’. The beats differ in the shape of the QRS complex and in their R-R intervals. The coupling interval RR_0 is shorter than the interval RR_1 that follows a PVC or the R-R intervals in the underlying rhythm such as RR_{-1}.

Fig. 2. The concept of a maximally decimated filter bank with M subbands. A set of analysis filters $H_k(z)$ decomposes the input signal $x(n)$ into M subbands. The subbands $w_k(m)$ are efficiently represented when downsampled by M and could be individually processed or used for the extraction of information. Upsampling and filtering of the processed subbands leads to the output signals $o_k(n)$, which reconstruct the processed signal $\hat{x}(n)$ after a summation of all subbands.

Fig. 3. Upsampled and interpolated outputs from the filter bank for record 116. The original signal (top row) is split in 32 different subbands, where o_0 (second row) represents a low-pass filtered portion of the original signal. The mean of the preceding 360 samples is subtracted from o_0 to compensate for the offset in the original signal. Rows three and four show the ouputs o_3 and o_5 of the filter bank.

Fig. 4. The energy in adjacent subbands of the filter bank after passing through the moving window integrator. The top row shows the original signal from record 116. The outputs of the moving window integrator after squaring the added amplitudes (o_0+o_1) and (o_4+o_5) are represented in row two and three respectively.

Fig 5. The decision tree created by *MC4* for a split weight of 2.0. The tree has 29 nodes, 15 leaves, 7 levels and uses 10 features. The classification process starts at the root of the tree. An incoming beat travels down the branches of the tree depending on the result of the test on a feature. The procedure ends when the beat arrives at a leaf ‘P’ (PVC) or ‘N’ (non-PVC).

Fig 6. Block diagram for a Fuzzy Logic System for classification. Incoming data are preprocessed and features are extracted before the fuzzification. The inference engine combines the fuzzy sets and the implemented rules to a fuzzy output set. As a final result, the defuzzifier transforms fuzzy output sets into crisp data (classes).

Fig 7. The bell curve membership functions premature, on time, and delayed for the feature *norm-RR*$_0$. The feature could be described using fuzzy sets with linguistic variables. The degrees of membership to these fuzzy sets vary between zero and one.

Fig 8. Nonlinear input to output mapping of a Fuzzy Logic System. The input sets *norm-RR*$_0$ and *RR*$_1$*-to-RR*$_0$ are combined by the rules 1-8 in Table II and evaluate the degree of membership of an incoming beat to the fuzzy set *beat type*.

TABLE I
LINGUISTIC DESCRIPTION OF THE FUZZY INPUT SETS AND THE PARAMETERS *A*, *B*, AND *C* OF THEIR GENERALIZED BELL CURVE MEMBERSHIP FUNCTIONS.

Feature	Fuzzy Input Set	Bell function parameters		
		a	*b*	*c*
norm-RR$_0$	premature	0.29	5	0.5
	on_time	0.115	1.5	1
	delayed	0.8	8	2
RR$_1$*-to-RR*$_0$	short	0.7	5	0
	regular	0.2	2	1
	long	2.5	15	4
irregularity	small	9	2	0
	large	55	7	75
norm-amp	small	0.7	10	1
	average	0.2	2	1
	enlarged	0.2	2.5	1.5
	highly_enlarged	2.3	10	4
peakdirection	identical	0.25	2.5	0
	opposite	0.25	2.5	1
peakwidth	normal	0.06	5	0
	wide	0.9	50	1
norm-peak-to-peak	small	0.6	5	0
	average	0.25	2.5	1
	enlarged	2.1	25	3.5
MWI-15	normal	0.22	6	0
	wide	0.14	10	0.4444
$E_{0,1}/E_{07}$	low	0.3	7	0
	high	0.51	10	1

TABLE II

ANTECEDENTS AND *CONSEQUENTS* OF THE 15 RULES IMPLEMENTED IN THE FUZZY LOGIC SYSTEM.

Rule (#)		*Antecedent*		*Consequent*
1	IF	*RR_1-to-RR_0* is short	THEN	*beat type* is non-PVC
2	IF	*RR_1-to-RR_0* is regular	THEN	*beat type* is non-PVC
3	IF	*irregularity* is small	THEN	*beat type* is non-PVC
4	IF	*norm-RR_0* is on time and *RR_1-to-RR_0* is regular	THEN	*beat type* is non-PVC
5	IF	*norm-RR_0* is delayed and *RR_1-to-RR_0* is regular	THEN	*beat type* is non-PVC
6	IF	*norm-RR_0* is on time and *RR_1-to-RR_0* is short	THEN	*beat type* is non-PVC
7	IF	*norm-RR_0* is delayed and *RR_1-to-RR_0* is short	THEN	*beat type* is non-PVC
8	IF	*norm-RR_0* is premature and *RR_1-to-RR_0* is long	THEN	*beat type* is PVC
9	IF	*norm-amp* is highly enlarged	THEN	*beat type* is PVC
10	IF	*norm-amp* is enlarged and *peakdirection* is opposite	THEN	*beat type* is PVC
11	IF	*norm-amp* is small and *peakdirection* is opposite	THEN	*beat type* is PVC
12	IF	*norm-peak-to-peak* is enlarged	THEN	*beat type* is PVC
13	IF	*peakwidth* is wide	THEN	*beat type* is PVC
14	IF	$E_{0,1}/E_{07}$ is low	THEN	*beat type* is non-PVC
15	IF	*MWI-15* is wide and $E_{0,1}/E_{07}$ is high	THEN	*beat type* is PVC

TABLE III

RESULTS OF THE DECISION TREE ALGORITHM FOR VARYING SPLIT WEIGHTS. THE NUMBER OF NODES, LEAVES, AND FEATURES CHOSEN BY THE ALGORITHM ARE INCLUDED. SENSITIVITY (*SE*) AND POSITIVE PREDICTIVITY (+*P*) FOR THE TRAINING AND TEST SET ARE REPORTED AS GROSS STATISTICS.

				Training set		Test set	
Split weight	Nodes (#)	Leaves (#)	Features (#)	*Se* (%)	+*P* (%)	*Se* (%)	+*P* (%)
0	131	66	14	98.97	97.87	86.08	80.32
0.1	125	63	14	98.97	97.87	86.17	79.67
0.2	119	60	14	98.96	97.04	85.29	85.23
0.3	121	61	13	98.77	97.04	82.76	84.35
0.4	121	61	14	98.68	96.76	85.80	83.84
0.5	105	53	13	98.48	96.02	85.31	84.33
0.6	105	53	14	98.29	95.65	83.51	86.00
0.7	117	49	13	98.02	96.30	84.03	84.96
0.8	85	43	13	97.15	94.81	82.19	85.57
0.9	79	40	14	96.93	93.70	80.85	84.53
1.0	47	24	10	94.11	93.24	81.54	82.98
1.1	29	15	10	92.78	92.78	83.15	82.41
5.0	29	15	10	92.78	92.78	83.15	82.41

TABLE IV

BEAT-BY-BEAT PERFORMANCE REPORT FOR PVC CLASSIFICATION WITH THE DECISION TREE ALGORITHM AND A SPLITTING WEIGHT OF 0.2. THE NUMBER OF TRUE NEGATIVE (*TN*), FALSE NEGATIVE (*FN*), TRUE POSITIVE (*TP*), AND FALSE POSITIVE (*FP*) CLASSIFICATIONS, THE SENSITIVITY (*SE*) AND POSITIVE PREDICTIVITY (+*P*), AND THE GROSS AND AVERAGE STATISTICS ARE LISTED FOR 44 RECORDS OF THE MIT-BIH DATABASE.

Record	*TN* (#)	*FN* (#)	*FP* (#)	*TP* (#)	*Se* (%)	+*P* (%)
100	1900	0	0	1	100	100
101	1520	0	0	0	-	-
103	1727	0	1	0	-	0
105	2078	0	42	29	100	40.85
106	1235	98	0	362	78.7	100
108	1424	3	40	10	76.92	20
109	2061	11	5	21	65.62	80.77
111	1771	0	3	1	100	25
112	2110	0	0	0	-	-
113	1503	0	2	0	-	0
114	1553	3	16	27	90	62.79
115	1636	0	0	0	-	-
116	1915	2	3	96	97.96	96.97
117	1281	0	2	0	-	0
118	1900	12	2	1	7.69	33.33
119	1296	0	0	364	100	100
121	1542	0	16	1	100	5.88
122	2053	0	0	0	-	-
123	1265	1	0	2	66.67	100
124	1312	17	2	30	63.83	93.75
200	1460	36	5	664	94.86	99.25
201	1231	116	89	82	41.41	47.95
202	1824	13	30	2	13.33	6.25
203	1955	61	147	312	83.65	67.97
205	2124	14	0	51	78.46	100
207	1363	13	119	96	88.07	44.65
208	1307	22	2	802	97.33	99.75
209	2501	0	15	1	100	6.25
210	2012	27	17	138	83.64	89.03
212	2283	0	1	0	-	0
213	2234	43	1	152	77.95	99.35
214	1657	13	5	199	93.87	97.55
215	2662	42	0	89	67.94	100
219	1693	14	28	37	72.55	56.92
220	1692	0	1	0	-	0
221	1702	2	1	314	99.37	99.68
222	1907	0	208	0	-	0
223	1734	269	1	186	40.88	99.47
228	1390	2	10	300	99.34	96.77
230	1856	0	1	1	100	50
231	1275	0	2	0	-	0
232	1431	0	53	0	-	0
233	1860	34	2	658	95.09	99.7
234	2287	0	0	3	100	100
Sum	76522	868	872	5032		
Average					80.47	58.00
Gross					85.29	85.23

TABLE V

BEAT-BY-BEAT PERFORMANCE REPORT FOR PVC CLASSIFICATION WITH THE FUZZY-RULE-BASED SYSTEM USING A TOTAL OF 9 FEATURES AND 15 RULES. THE NUMBER OF TRUE NEGATIVE (*TN*), FALSE NEGATIVE (*FN*), TRUE POSITIVE (*TP*), AND FALSE POSITIVE (*FP*) CLASSIFICATIONS, THE SENSITIVITY (*SE*) AND POSITIVE PREDICTIVITY (+*P*), AND THE GROSS AND AVERAGE STATISTICS ARE LISTED FOR 44 RECORDS OF THE MIT-BIH DATABASE.

Record	*TN* (#)	*FN* (#)	*FP* (#)	*TP* (#)	*Se* (%)	+*P* (%)
100	1900	0	0	1	100	100
101	1520	0	0	0	-	-
103	1727	0	1	0	-	0
105	2097	0	23	29	100	55.77
106	1234	75	1	385	83.7	99.74
108	1440	0	24	13	100	35.14
109	2066	17	0	15	46.88	100
111	1772	0	2	1	100	33.33
112	2110	0	0	0	-	-
113	1505	0	0	0	-	-
114	1566	1	3	29	96.67	90.62
115	1636	0	0	0	-	-
116	1911	1	7	97	98.98	93.27
117	1283	0	0	0	-	-
118	1889	6	13	7	53.85	35
119	1296	0	0	364	100	100
121	1543	0	15	1	100	6.25
122	2053	0	0	0	-	-
123	1265	0	0	3	100	100
124	1313	31	1	16	34.04	94.12
200	1463	29	2	671	95.86	99.7
201	1158	124	162	74	37.37	31.36
202	1722	2	132	13	86.67	8.97
203	1776	69	326	304	81.5	48.25
205	2124	27	0	38	58.46	100
207	1476	98	6	11	10.09	64.71
208	1303	138	6	686	83.25	99.13
209	2516	1	0	0	0	-
210	1996	24	33	141	85.45	81.03
212	2284	0	0	0	-	-
213	2222	157	13	38	19.49	74.51
214	1662	7	0	205	96.7	100
215	2662	17	0	114	87.02	100
219	1600	8	121	43	84.31	26.22
220	1691	0	2	0	-	0
221	1695	1	8	315	99.68	97.52
222	1925	0	190	0	-	0
223	1723	193	12	262	57.58	95.62
228	1377	1	23	301	99.67	92.9
230	1857	1	0	0	0	-
231	1277	0	0	0	-	-
232	1462	0	22	0	-	0
233	1858	73	4	619	89.45	99.36
234	2287	0	0	3	100	100
Sum	76242	1101	1152	4799		
Average					74.58	66.54
Gross					81.34	80.64

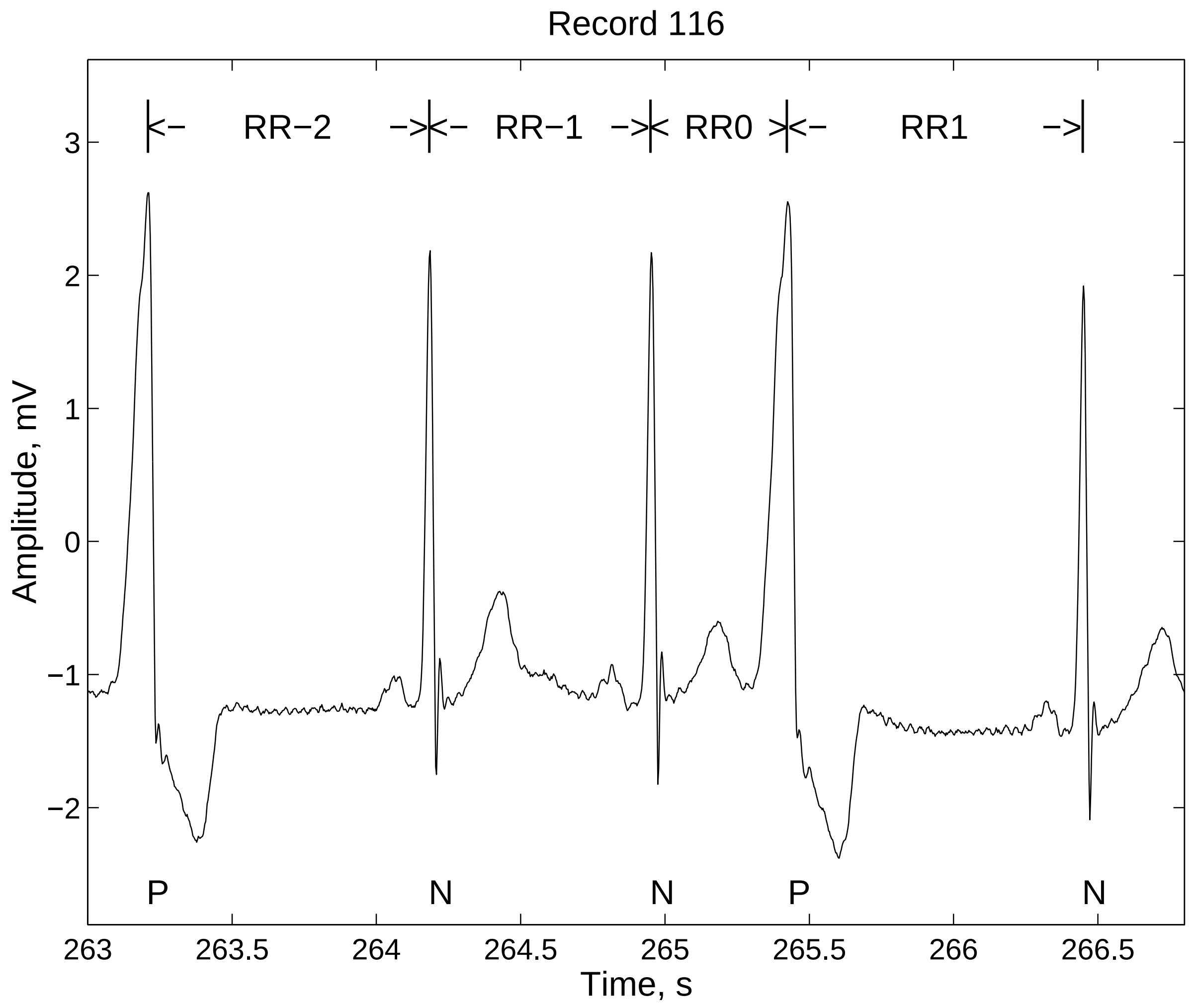

Fig. 1. Premature ventricular contractions (PVCs) in record 116. PVCs are labeled as 'P' and non-PVCs are labeled as 'N'. The beats differ in the shape of the QRS complex and in their R-R intervals. The coupling interval RR_0 is shorter than the interval RR_1 that follows a PVC or the R-R intervals in the underlying rhythm such as RR_{-1}.

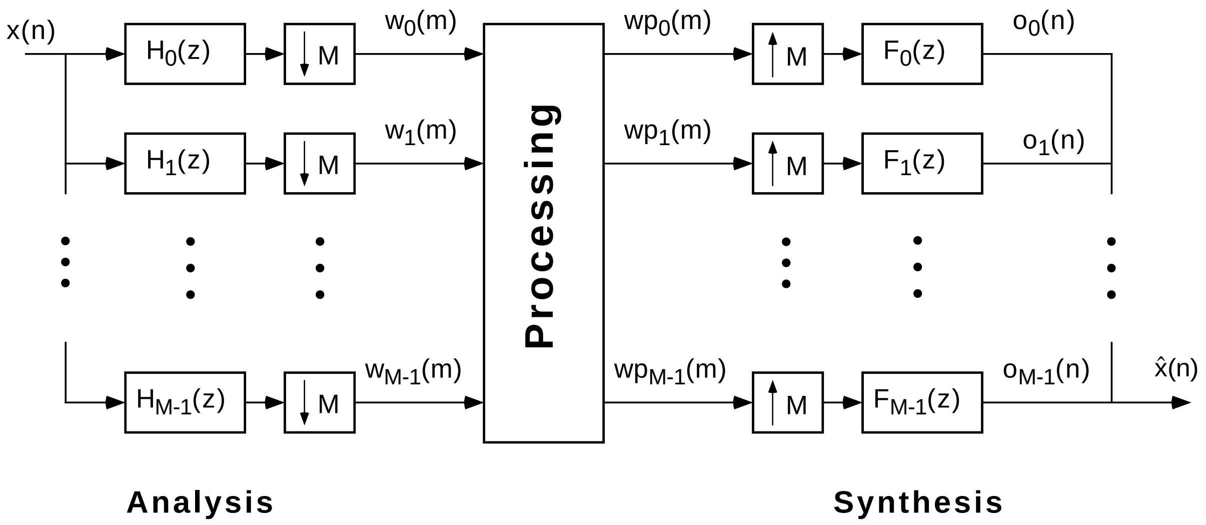

Fig. 2. The concept of a maximally decimated filter bank with *M* subbands. A set of analysis filters $H_k(z)$ decomposes the input signal $x(n)$ into *M* subbands. The subbands $w_k(m)$ are efficiently represented when downsampled by *M* and could be individually processed or used for the extraction of information. Upsampling and filtering of the processed subbands leads to the output signals $o_k(n)$, which reconstruct the processed signal $\hat{x}(n)$ after a summation of all subbands.

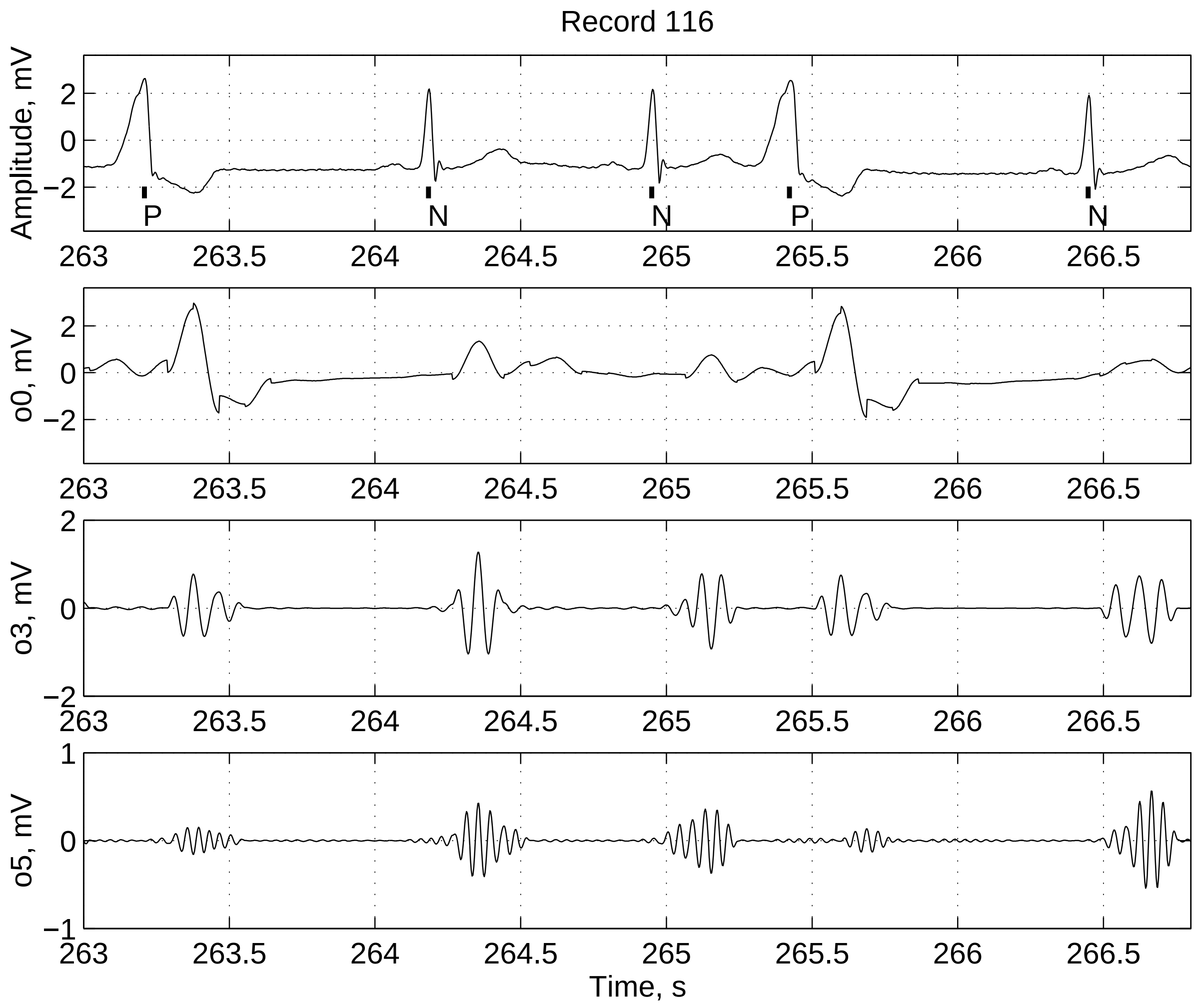

Fig. 3. Upsampled and interpolated outputs from the filter bank for record 116. The original signal (top row) is split in 32 different subbands, where o_0 (second row) represents a low-pass filtered portion of the original signal. The mean of the preceding 360 samples is subtracted from o_0 to compensate for the offset in the original signal. Rows three and four show the ouputs o_3 and o_5 of the filter bank.

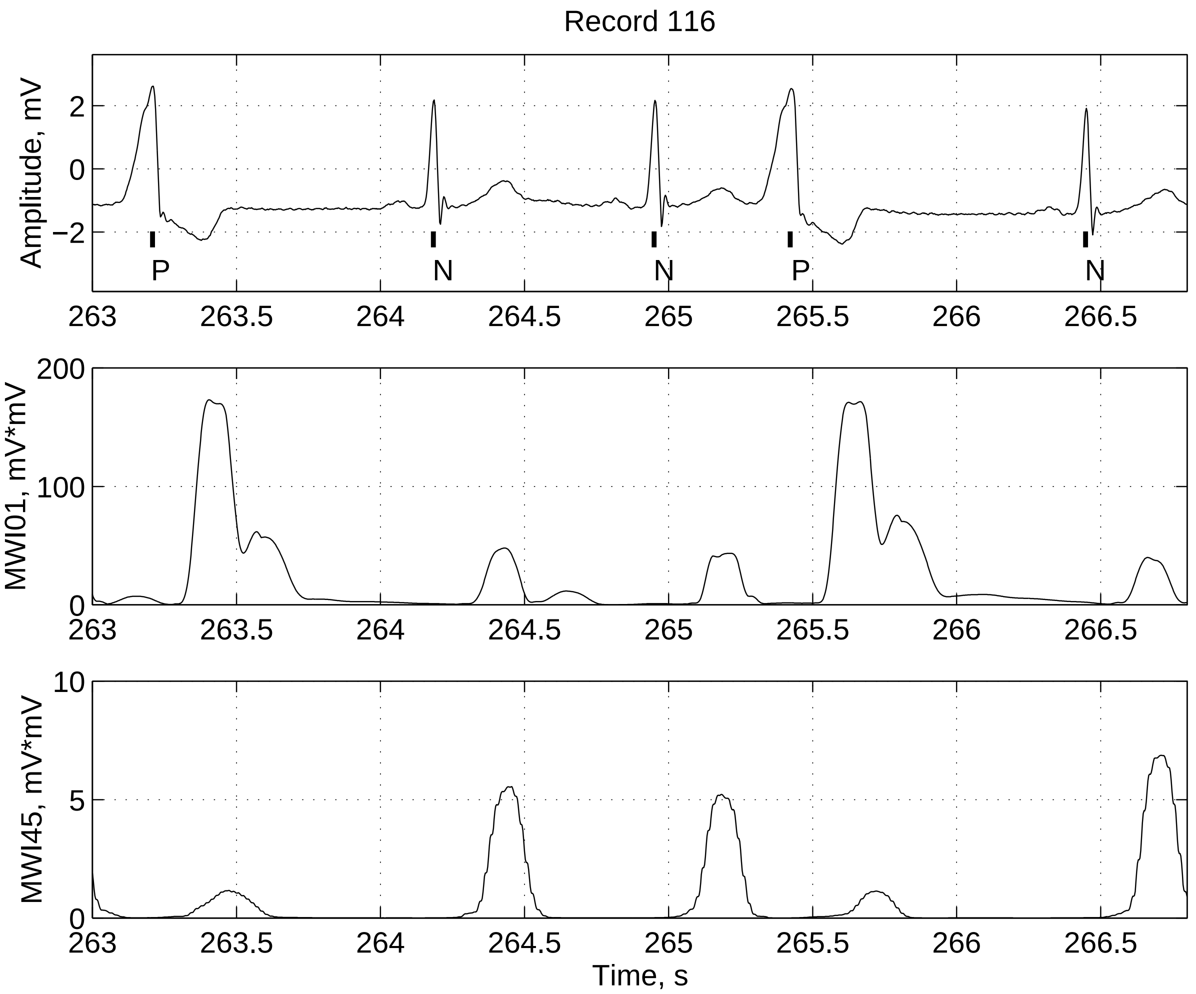

Fig. 4. The energy in adjacent subbands of the filter bank after passing through the moving window integrator. The top row shows the original signal from record 116. The outputs of the moving window integrator after squaring the added amplitudes (o_0+o_1) and (o_4+o_5) are represented in row two and three respectively.

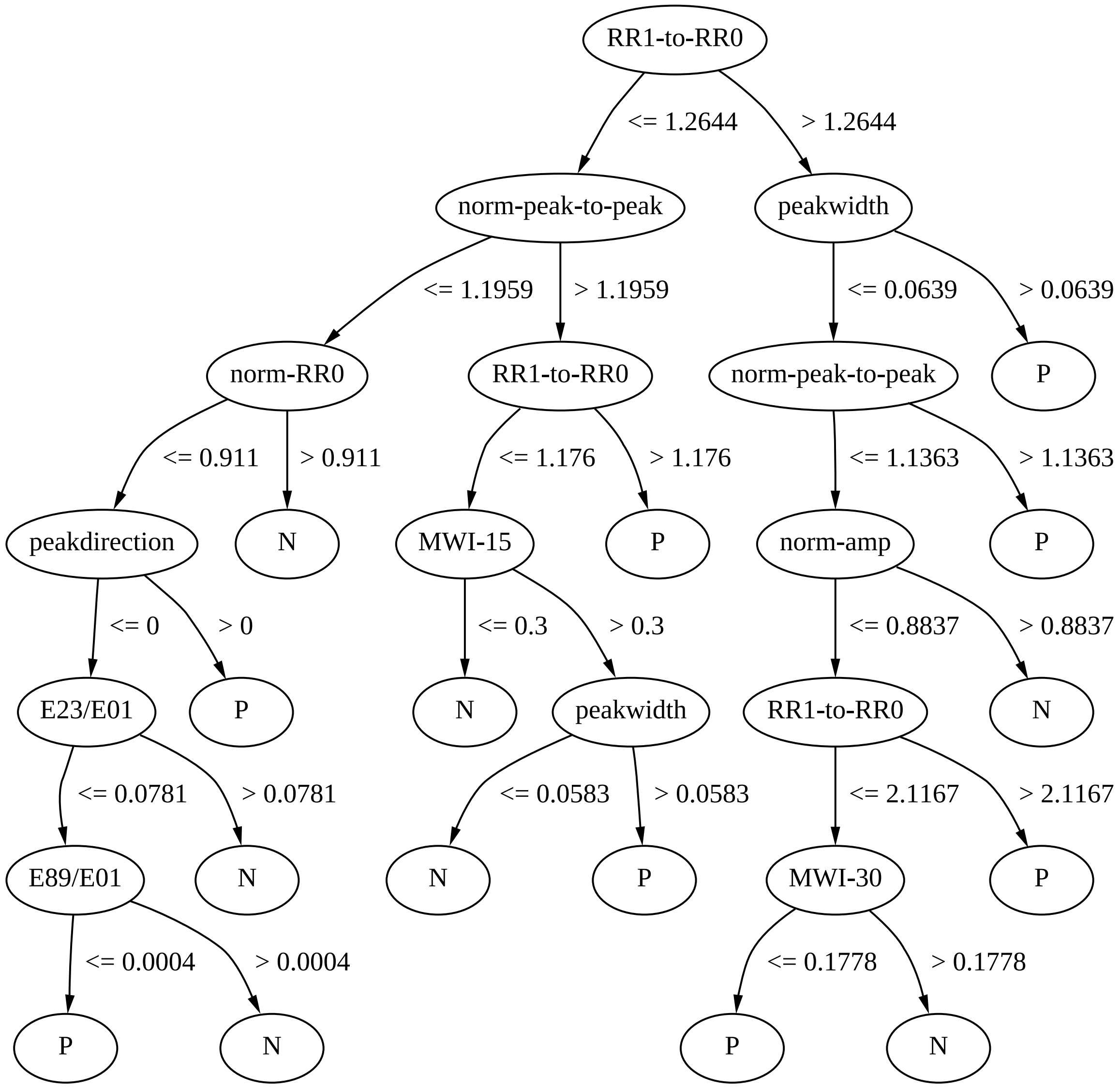

Fig 5. The decision tree created by *MC4* for a split weight of 1.1. The tree has 29 nodes, 15 leaves, 7 levels and uses 10 features. The classification process starts at the root of the tree. An incoming beat travels down the branches of the tree depending on the result of the test on a feature. The procedure ends when the beat arrives at a leaf ‘P’ (PVC) or ‘N’ (non-PVC).

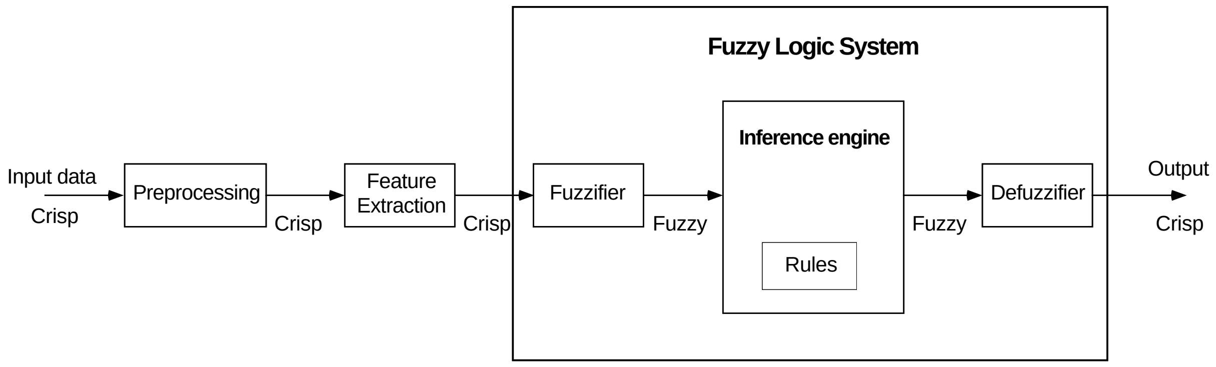

Fig 6. Block diagram for a Fuzzy Logic System for classification. Incoming data are preprocessed and features are extracted before the fuzzification. The inference engine combines the fuzzy sets and the implemented rules to a fuzzy output set. As a final result, the defuzzifier transforms fuzzy output sets into crisp data (classes).

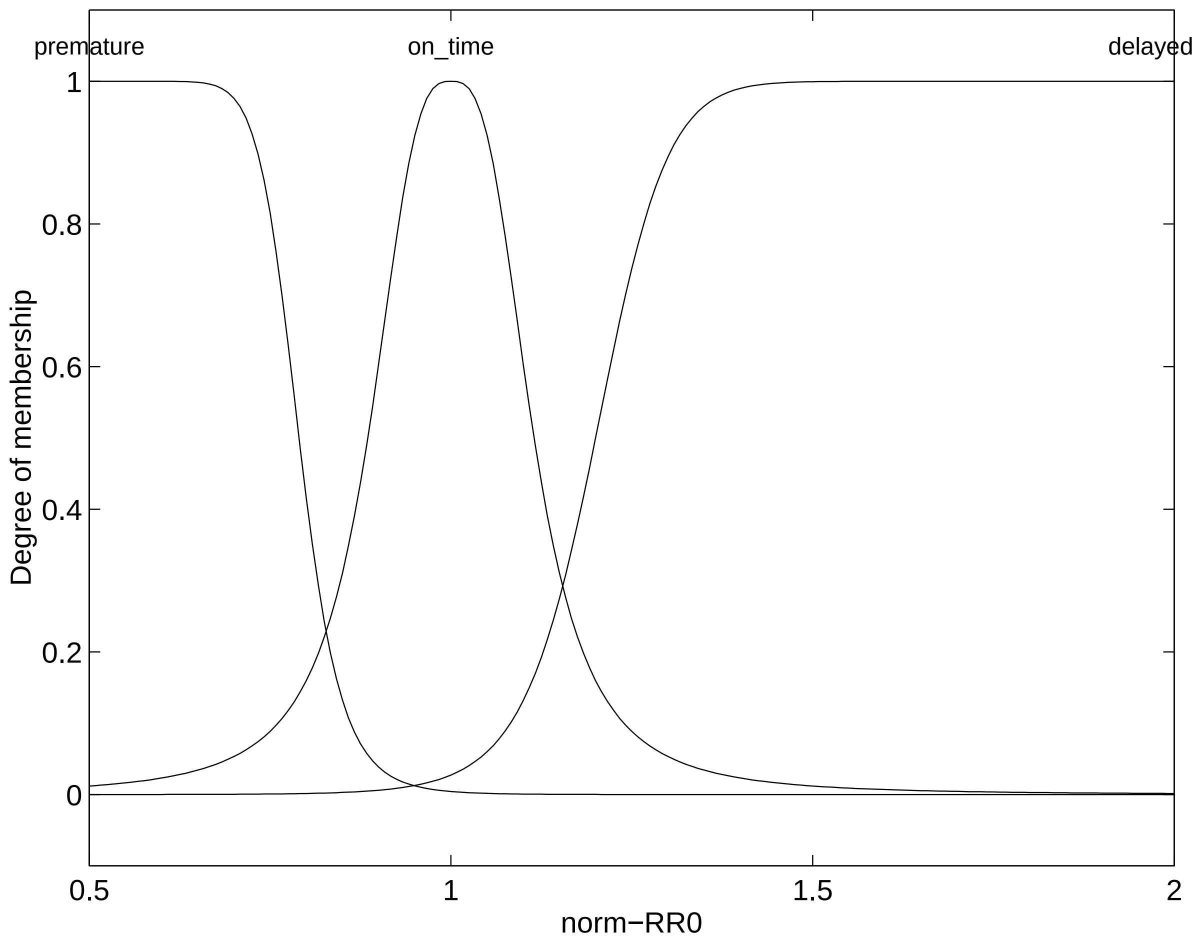

Fig 7. The bell curve membership functions premature, on time, and delayed for the feature *norm-RR*$_0$. The feature could be described using fuzzy sets with linguistic variables. The degrees of membership to these fuzzy sets vary between zero and one.

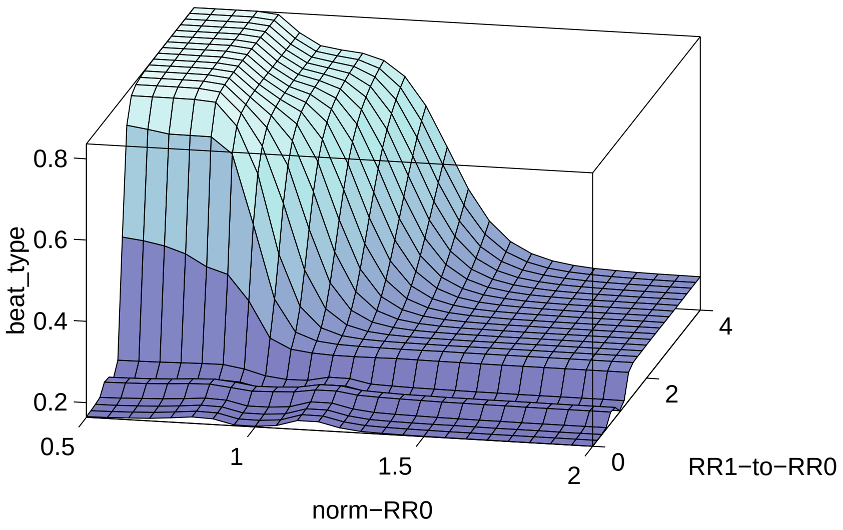

Fig 8. Nonlinear input to output mapping of a Fuzzy Logic System. The input sets *norm-RR*$_0$ and *RR*$_1$*-to-RR*$_0$ are combined by the rules 1-8 in Table II and evaluate the degree of membership of an incoming beat to the fuzzy set *beat type*.

APPENDIX

CODING OF THE ALGORITHMS IN C AND MATLAB

```
/*  created by Oliver Wieben
    FILE: EnergyPeaks.c
    last modified: 23 November 1996

      Usage from Matlab:   [Energies] = EnergyPeaks(ecg, P, idx);
      ecg: signal to be filtered.
      P: each row is a filter.
      idx : Vector that includes all sample numbers with a kind of 'beats'
    Output: signal :  Matrix of [10 rows, # of idx-entries]
           row 0 = energy of subband0-mean
           row 1-9 = energy of subband 1 to 9
           row 10 = energy of (subband0-mean + subband1)
           row 11 = energy of (subband2 + subband3)
           row 12 = energy of (subband4 + subband5)
           row 13 = energy of (subband6 + subband7)
           row 14 = energy of (subband8 + subband9)

   NOTE:
        - Energypeaks is looking for the peak in the calculated energy in the subbands of a filter bank
        - the intervals are limited to +- 35 samples around the annotated beat
       - calculates the mean for the first subband and subtracts it from the first subband to compensate for the
             DC offset
       - the mean for subband 0 is based on last 1 sec of data = 360 samples
       - the windowlength for the energycalculation (ouput of a subband squared) is 122 msec = 44 sample
             points
       - subbandMAX (=14) limits the computationtime;
*/

#include <math.h>
#include <stdlib.h>
#include "mex.h"

#define max(A,B) ((A) > (B) ? (A) : (B));

/* Functions */
/* == matrixD()              */
/* == init_FB()              */
/* == sendMATLAB()           */
/* == analyze()              */
/* == synthesize()           */
/* == controlwindow          */
/* == Energycalc() ==         */
/* == searchmaxima() =       */
/* == run_program()          */
/* == mexFunction()          */

/* Input Arguments */
#define  sig_IN prhs[0]
#define  P_IN  prhs[1]
#define Idx_IN prhs[2]

/* Output Arguments */
#define  Features_OUT plhs[0]
```

```
struct Filter {
      double **Matrix;
      long M;
      long N;
      long N_2;
      };

struct Data_Array {
      double *Array;
      long M;
      long M_2;
      long Pointer;
      };

struct Data_Matrix {
      double **Matrix;
      long M;
      long N;
      long N_2;
      long Pointer;
      double Delay;
      };

/* ============ Define some global variables ============= */

/** --- data from/derived MATLAB (or externally) --- **/
double *sig;      /* pointer to incoming data */
long Nsig;        /* size of *sig */
double *Pmex;     /* size:[P_M x P_L]; each row is an analysis filter;
                      stored columnwise since from MATLAB. */
long P_M, P_L;    /* P_M: # of channels; P_L: filter length */
double *beatsample;  /* pointer to incoming idx */
long beatsamplelength;  /* size of idx */

/** --- data sent to MATLAB --- **/
double *Features;

/** --- internal data  --- **/
long subbandMAX = 15;
long beatsampleindex;
long Featuresnumber;

/* ==================== matrixD ================= */
#ifdef __STDC__
double **matrixD(long rows, long cols)
#else
double **matrixD(rows, cols)
long rows, cols;
#endif
{
   long k;
   double **mat;
   mat  = (double **) calloc((size_t) rows, (size_t) sizeof(double *));
   for (k=0; k<rows; k++)
      mat[k]  = (double *) calloc((size_t) cols, (size_t) sizeof(double));
   return mat;
}

/* ================== init_FB =================== */
#ifdef __STDC__
init_FB(struct Filter *P struct Filter *H, struct Filter *F,
      struct Data_Array *X, struct Data_Matrix *W, struct Data_Matrix *O)
#else
init_FB(P, H, F, X, W, O)
struct Filter *P, *H, *F;
struct Data_Array *X;
struct Data_Matrix *W, *O;
#endif
{
   long i,k,t = 0;

   printf("Initializing of the Filterbank \n");

   /********* init Filters   **********/
   /* set up P */
   /* Convert between matrix formats. */
   /* Matrices are stored columnwise in MATLAB */
   for (i=0, t=0; i<(*P).N; i++)
     for (k=0; k<(*P).M; k++, t++)
       (*P).Matrix[k][i] = Pmex[t];

   /* set up analysis filters, H. */
   for (i=0; i<(*H).M; i++)
     for (k=0; k<(*H).N; k++)
       (*H).Matrix[i][(*H).N-1-k] = (*P).Matrix[i][k];  /* reverse coefficients */
      /* ist hier alles richtig?? */

   /* set up synthesis filters, F. */
   for (i=0; i<(*F).M; i++)
     for (k=0; k<(*F).N; k++)
       (*F).Matrix[i][k] = (*P).Matrix[i][k];

   /********* init Data-Sets   **********/
   /* set up signal buffer, X. */
   (*X).M_2 = (*X).M/2;
   (*X).Pointer = (*X).M_2;

     /* set up decimated subbands, W. */
   (*W).N_2 = (*W).N/2;
   (*W).Pointer = (*W).N_2;
   (*W).Delay = 32.0;

   /* set up filtered upsampled subbands, O **/
   (*O).N_2 = (*O).N/2;
   (*O).Pointer = (*O).N_2;
   (*O).Delay = 63.0;
  }

/* ================== analyze ================== */
#ifdef __STDC__
void analyze(long k, struct Filter H,
             struct Data_Array X, struct Data_Matrix *W)
#else
analyze(k, H, X, W)
long k;
struct Filter H;
struct Data_Array X;
struct Data_Matrix *W;
#endif
{
   long i;
   double tt;

   for (i = 0, tt = 0.; i < H.N; i++)
     tt += H.Matrix[k][i] * X.Array[X.Pointer-i];

     /* store in decimated, k'th subband, W */
   (*W).Matrix[k][(*W).Pointer] = (*W).Matrix[k][(*W).Pointer - (*W).N_2] = tt;
   (*W).Delay = 32.0;  /* line 207 */
}

/* =========== synthesize ================== */
#ifdef __STDC__
void synthesize(long k, long countP_M, struct Filter F,
             struct Data_Matrix W, struct Data_Matrix *O)
#else
synthesize(k,countP_M, F, W, O)
long k, countP_M;
struct Filter F;
struct Data_Matrix W, *O;
#endif
{
   long i;
   double tt;

   for (i = 0, tt = 0.; i < W.N_2; i++)
      tt += F.Matrix[k][countP_M + P_M*i] * W.Matrix[k][W.Pointer - i];

     /* store in upsampled, k'th subband, O */
   (*O).Matrix[k][(*O).Pointer] = (*O).Matrix[k][(*O).Pointer - (*O).N_2] = tt;
   (*O).Delay = 63.0;
}  /* line 200 */

/* ======== controlwindow =============== */
#ifdef __STDC__
void controlwindow(struct Data_Matrix O,struct Data_Array *subband0,
              struct Data_Matrix *Subsout)
#else
```

```
controlwindow( O, subband0, Subsout)
struct Data_Matrix O, *Subsout;
struct Data_Array *subband0;
#endif
{
long k;

   /* store the current reconstructed output of the subband 0 in the array */
   (*subband0).Array[(*subband0).Pointer] =
   (*subband0).Array[(*subband0).Pointer - (*subband0).M_2]=
   O.Matrix[0][O.Pointer];

   /* store the current reconstructed output of the subbands 0-27 in the matrix */
   for (k = 0; k < subbandMAX; k++) {
     (*Subsout).Matrix[k][(*Subsout).Pointer] =
     (*Subsout).Matrix[k][(*Subsout).Pointer - (*Subsout).N_2] =
     O.Matrix[k][O.Pointer];
     }
}

/* ================== Energycalc() =========================== */
#ifdef __STDC__
void Energycalc( struct Data_Array subband0, struct Data_Matrix Subsout,
                            struct Data_Matrix *Energywindow)
#else
void Energycalc( subband0, Subsout, Energywindow)
struct Data_Array subband0;
struct Data_Matrix Subsout, *Energywindow;
#endif
{
   long i, k, windowsize ;
   double sum, mean, energy;

   windowsize = Subsout.N/2;

      /* energy-output for subband 0   */
       /* calculate the mean of the very long window to subtract the DC level*/
       sum = 0.0;
       for(i = 0; i < subband0.M/2; i++)
            sum += subband0.Array[subband0.Pointer-i];
       mean = sum / (subband0.M/2);

       /* calculate the energy incl. subtraction of the DC-level = mean */
       energy = 0.0;
       for(i = 0; i < windowsize; i++)
          energy += (subband0.Array[subband0.Pointer-i]-mean)*(subband0.Array[subband0.Pointer-i]-mean);
       (*Energywindow).Matrix[0][(*Energywindow).Pointer] =
       (*Energywindow).Matrix[0][(*Energywindow).Pointer - (*Energywindow).N_2] =
       energy;

      /* energy-calculation for subbands 1 to 9 (channels 2-10)   */
       for (k = 1; k < 10; k++){
          energy = 0.0;
```

```
    for(i = 0; i < windowsize; i++)
      energy += Subsout.Matrix[k][Subsout.Pointer-i]*Subsout.Matrix[k][Subsout.Pointer-i];
    (*Energywindow).Matrix[k][(*Energywindow).Pointer] =
    (*Energywindow).Matrix[k][(*Energywindow).Pointer - (*Energywindow).N_2] =
    energy;
  }  /* for (k = 1; k < 9; k++) */

/* energy-calculation for subband 10 */
    k = 10;
    energy = 0.0;
    for(i = 0; i < windowsize; i++)
      energy += (subband0.Array[subband0.Pointer-i]-mean+Subsout.Matrix[1][Subsout.Pointer-i])
         *(subband0.Array[subband0.Pointer-i]-mean+Subsout.Matrix[1][Subsout.Pointer-i]);
    (*Energywindow).Matrix[k][(*Energywindow).Pointer] =
    (*Energywindow).Matrix[k][(*Energywindow).Pointer - (*Energywindow).N_2] =
    energy;

/* energy-calculation for subband 11 */
    k = 11;
    energy = 0.0;
    for(i = 0; i < windowsize; i++)
      energy += (Subsout.Matrix[2][Subsout.Pointer-i]+Subsout.Matrix[3][Subsout.Pointer-i])
         *(Subsout.Matrix[2][Subsout.Pointer-i]+Subsout.Matrix[3][Subsout.Pointer-i]);
    (*Energywindow).Matrix[k][(*Energywindow).Pointer] =
    (*Energywindow).Matrix[k][(*Energywindow).Pointer - (*Energywindow).N_2] =
    energy;

/* energy-calculation for subband 12 */
    k = 12;
    energy = 0.0;
    for(i = 0; i < windowsize; i++)
      energy += (Subsout.Matrix[4][Subsout.Pointer-i]+Subsout.Matrix[5][Subsout.Pointer-i])
         *(Subsout.Matrix[4][Subsout.Pointer-i]+Subsout.Matrix[5][Subsout.Pointer-i]);
    (*Energywindow).Matrix[k][(*Energywindow).Pointer] =
    (*Energywindow).Matrix[k][(*Energywindow).Pointer - (*Energywindow).N_2] =
    energy;

/* energy-calculation for subband 13 */
    k = 13;
    energy = 0.0;
    for(i = 0; i < windowsize; i++)
      energy += (Subsout.Matrix[6][Subsout.Pointer-i]+Subsout.Matrix[7][Subsout.Pointer-i])
         *(Subsout.Matrix[6][Subsout.Pointer-i]+Subsout.Matrix[7][Subsout.Pointer-i]);
    (*Energywindow).Matrix[k][(*Energywindow).Pointer] =
    (*Energywindow).Matrix[k][(*Energywindow).Pointer - (*Energywindow).N_2] =
    energy;

/* energy-calculation for subband 14 */
    k = 14;
    energy = 0.0;
    for(i = 0; i < windowsize; i++)
      energy += (Subsout.Matrix[8][Subsout.Pointer-i]+Subsout.Matrix[9][Subsout.Pointer-i])
         *(Subsout.Matrix[8][Subsout.Pointer-i]+Subsout.Matrix[9][Subsout.Pointer-i]);
         (*Energywindow).Matrix[k][(*Energywindow).Pointer] =
         (*Energywindow).Matrix[k][(*Energywindow).Pointer - (*Energywindow).N_2] =
         energy;
}

/* ================== searchmaxima() ============================ */
#ifdef __STDC__
double searchmaxima(long k, struct Data_Matrix Energywindow)
#else
double searchmaxima(k, Energywindow)
long k;
struct Data_Matrix Energywindow;
#endif
{
   long i;
   double maxima;
   maxima = 0.0;
   for(i = 0; i < Energywindow.N/2; i++){
     if (Energywindow.Matrix[k][Energywindow.Pointer-i] > maxima)
       maxima = Energywindow.Matrix[k][Energywindow.Pointer-i];
     } /* if (Energywindow.Matrix[k]... */

   return(maxima) ;
}

/* ===================== sendMATLAB() ======================== */
#ifdef __STDC__
sendMATLAB(long t,  struct Data_Matrix Energywindow)
#else
sendMATLAB(t, Energywindow)
long t;
struct Data_Matrix Energywindow;
#endif
{
  long k;

   /* energy-output for subbands 0-31 (channels 1-32) (only for annotated beats)  */
   if (t-Energywindow.Delay-Energywindow.N/4 == beatsample[beatsampleindex]) {
      for (k = 0; k < subbandMAX; k++)
           Features[beatsampleindex*subbandMAX + k] = searchmaxima(k, Energywindow);
      } /* if (t-Energywindow.Delay-Energywindow.N/4... */
  }

/* ========================= run_program() ======================= */
#ifdef __STDC__
void run_program(void)
#else
run_program()
#endif
{

   long t, subBand, countP_M = 0;
   struct Filter P, F, H;
```

```
 struct Data_Array X, subband0;
 struct Data_Matrix W, O, O_long, Energywindow;

/*********** begin of initializing part  *************/
 beatsampleindex = 0;
 Featuresnumber = 32;

 P.M = F.M = H.M = P_M;
 P.N = F.N = H.N = P_L;
 P.Matrix = (double **) matrixD(P.M, P.N);
 H.Matrix = (double **) matrixD(H.M, H.N);
 F.Matrix = (double **) matrixD(F.M, F.N);

 X.M = P_L * 2;
 X.Array = (double *) calloc((size_t) X.M, (size_t) sizeof(double));
 W.M = O.M = P_M;
 W.N = O.N = P_L / P_M * 2;
 W.Matrix = (double **) matrixD(W.M, W.N);
 O.Matrix = (double **) matrixD(O.M, O.N);

 init_FB(&P, &H, &F, &X, &W, &O);

 /* set up the window for the features, O_long and Energywindow and subband0*/
 O_long.M = subbandMAX;
 O_long.N = 44*2;                    /* windowlength = 44 samples = 122ms */
 O_long.Matrix = (double **) matrixD(O_long.M, O_long.N);
 O_long.N_2 = O_long.N/2;
 O_long.Pointer = O_long.N_2;
 O_long.Delay = O.Delay;

 Energywindow.M = O_long.M;               /* same as O_long*/
 Energywindow.N = 70*2;                   /* window length = 70 samples: 194ms*/
 Energywindow.Matrix = (double **) matrixD(Energywindow.M,Energywindow.N );
 Energywindow.N_2 = Energywindow.N/2;
 Energywindow.Pointer = Energywindow.N_2;
 Energywindow.Delay = O.Delay + O_long.N/4;   /* delay = delay of output (63) +
                                                 + half of window for squaring (22)   */
 subband0.M = 360 * 2;              /* windowlength for the calculation of the mean */
 subband0.Array = (double *) calloc((size_t) subband0.M, (size_t) sizeof(double));
 subband0.M_2 = subband0.M/2;
 subband0.Pointer = subband0.M_2;

/*************** end of initializing part  *****************/
 printf("Beginning to run\n");
 for (t = 0; t < Nsig; t++) {

   if ( (t - Energywindow.Delay - Energywindow.N/4 > beatsample[beatsampleindex]) &&
          (beatsampleindex < beatsamplelength - 1) )
    beatsampleindex++;

     /* Feed data to buffer */
   if (++X.Pointer >= X.M) X.Pointer = X.M_2;
   X.Array[X.Pointer] = X.Array[X.Pointer - X.M_2] = sig[t];
```

```
      /* count modulo countP_M */
    if (++countP_M >= P_M) countP_M = 0;

      /* Downsample if countP_M is 0 */
    if (countP_M == 0) {
      if (++W.Pointer >= W.N) W.Pointer = W.N_2;
      for (subBand = 0; subBand < subbandMAX; subBand++)
        analyze(subBand, H, X, &W);
    } /* if (countP_M == 0) */

    /* Pass through synthesis filters. */
    if (++O.Pointer >= O.N) O.Pointer = O.N_2;
    for (subBand = 0; subBand < subbandMAX; subBand++)
      synthesize(subBand, countP_M, F, W, &O);

    /* Control the window of the data in the last 100 ms. */
    if (++O_long.Pointer >= O_long.N)
     O_long.Pointer = O_long.N_2;
    if (++subband0.Pointer >= subband0.M)
     subband0.Pointer = subband0.M_2;
    controlwindow( O, &subband0, &O_long);

    /* calculate the energy  */
      if (++Energywindow.Pointer >= Energywindow.N)
         Energywindow.Pointer = Energywindow.N_2;
      Energycalc( subband0, O_long , &Energywindow);

    /* Save variables to send to MATLAB */
    sendMATLAB(t, Energywindow);

  } /* for (t = 0; t < Nsig; t++) */
}

/* ========================== mexFunction() ======================= */
#ifdef __STDC__
void mexFunction(int nlhs, Matrix *plhs[],
   int       nrhs, Matrix *prhs[]
   )
#else
mexFunction(nlhs, plhs, nrhs, prhs)
int nlhs, nrhs;
Matrix *plhs[], *prhs[];
#endif
{
   /* Check for proper number of arguments */
   if (nrhs != 3)
      mexErrMsgTxt("Usage: [Energies] = EnergyPeaks(ecg, P, idx);");

   /* Assign pointers to the input parameters */
   sig = mxGetPr(sig_IN);
   Nsig = max(mxGetM(sig_IN), mxGetN(sig_IN));  /* # of rows => channels */
   Pmex  = mxGetPr(P_IN);
   P_M = (long) mxGetM(P_IN);  /* # of rows => channels */
   P_L = (long) mxGetN(P_IN);  /* # of cols. => filter length */
   beatsample = mxGetPr(Idx_IN); /* reads idx in */
   beatsamplelength = (long) mxGetM(Idx_IN);  /* length of idx */


   /* Create a matrix for the return argument */
   Features_OUT = mxCreateFull(15, beatsamplelength , REAL);

   /* Assign pointers to output arguments */
   Features = mxGetPr(Features_OUT);

   /* Do the actual computations in a subroutine */

   run_program();
   printf("Nsig = %ld\n", Nsig);
   printf("length of output vector = %ld\n", beatsamplelength);

   return;
}
```

```
/* createDecisionTree.c                                    *
*  created 10/6/96        by Oliver Wieben      earlier version by Surekha Palreddy        *
*  function call: createDecisionTree MLC_tree_file                                                *
*    where MLC_tree_file represents a file produced    by the Inducer (called Inducer.dot)        *
*    In addition there has to be a file called 'Names' in the same directory               *
*  creates a file mlcTree.ascii, which can be read in matlab as a matrix to be interpreted as a   *
*   decision tree algorithm                                                                       *
*  the 'Names' file:                                                                              *
*    first rows: the classes                                                                      *
*    last rows: the features                                                                      *
*    one class or feature each line                                                               *
*                                                                                                 *
*  the matrix looks the following way:                                                            *
*   -each row[0,1,...N+M-1] has four columns and represents either a node or a leaf.              *
*   -first column is the feature number [1, 2, ...N] or class number [-1, -2, ..-M] of the node or the  leaf.  *
*   -second column is the number of the row, where the left child of the node could be found or zero,   *
*      if there is no left child                                                                  *
*   -third column is the number of the row, where the right child of the node could be found or zero,   *
*      if there is no right child                                                                 *
*    -fourth column is the threshold of the node or zero if it is a leaf                          *
*   expects only three different patterns of entries                                              *
*    1. command line                                                                              *
*    2. one line                                                                                  *
*           node_6 [label="non-PVC"]                                                              *
*    3. three lines in a row ( a node)                                                            *
*           node_5 [label="norm-amp"]                                                             *
*           node_5->node_6 [label="<= 6.05"]                                                      *
*           node_5->node_7 [label="> 6.05"]                                                       *
*  changes 10/30/96:                                                                              *
*   - now it works on pruned decision trees as well                                      */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>

#define MAX_NAMES 20                /* Max number of (classes+features) */
#define CONST_60 60
#define linemax 50
# define nodemax 500
  /* Main function */

main(int argc, char *argv[])
{
   FILE *infp, *outfp;
   char names[MAX_NAMES][CONST_60];
   char actualLine[linemax];
   char buffer[80];
   char Label[40], nodenumber[4], L_child[4], R_child[4], THRESH[20];
   int Nnames, j, l, e_label, e_number, i, e_th, lab, k;
   e_label = 0;
   e_number = 0;
```

```
/* check the input command (functioncall + 1 parameter ) */
  if( argc != 2) {
    fprintf( stderr, "\n\n Usage:  %s MLC_tree_file.dot \n", argv[0]);
    fprintf( stderr, "       saves mlcTree.ascii\n");
    exit(0);
  }

/* Start: Read in from the names of the Features 'Names' file */
  /* Open Names where Class names are stored */
  infp = fopen("Names","r");
  if (infp == NULL) {
     printf("Error occurred opening input file Names.\n");
     exit(1);
  }

  /* Read in class names + feature names into 'Names' (maximum = 12) */
  for (j = 0; j < MAX_NAMES - 1; j++) {
     if (fscanf(infp, "%s", names[j]) == EOF) break;
  }

  /* No. of classes and features present is Nnames */
  Nnames = j;
  fclose(infp);
/* End: Read in from 'Names' file */

/* Open 'INfilename' where Class names are stored */
  infp = fopen(argv[1],"r");
  if (infp == NULL) {
     printf("Error occurred opening input file.\n");
     exit(1);
  }

/* Open 'mlcTreetmp.ascii' where the Matrix will be stored */
  outfp = fopen("mlcTreetmp.ascii","w");
  if (outfp == NULL) {
     printf("Error occurred opening output file DTmatrix.ascii .\n");
     exit(1);
  }

/* -------------The fun part : analyze the Inducer.dot file----------------- */
/* Get rid of the first lines without information on the decision tree */
 fgets(actualLine, linemax, infp);    /* read in first line = comment */
 fgets(actualLine, linemax, infp);    /* read in the next line */
 while ( actualLine[0] != '/')        /* read in until next comment line '.. node 0;..'  */
     fgets(actualLine, linemax, infp);

 for (j = 0; j < nodemax; j++) {                    /* begin of the loop: maximal number of nodes: maxnodes */

    if (fscanf(infp, "%s", buffer) == EOF) break; /* reads in first string in the row */

    /* CHECK IF A COMMENT LINE: */
    if ( strlen(buffer) < 4){
       fgets(actualLine, linemax, infp);
```

```
      if (fscanf(infp, "%s", buffer) == EOF) break; /* reads in first string in the next row */
    }

    /* FIRST ROW AFTER A COMMENT INCLUDES A LABEL*/
    if ( strlen(buffer) < 9){

      /* READ IN THE NUMBER OF THE LEAF OR CHILD */
      e_number = strlen(buffer);
     for (k=5,i = 0; k < e_number; k++,i++) {
          nodenumber[i] = buffer[k];                /* removes 'node_' from the second word */
      }
      nodenumber[i] = '\0';

      /* READ IN THE LABEL OF THE LEAF OR A NODE */
      fgets(actualLine, linemax, infp);
      e_label = strlen(actualLine) - 3;             /* finds the end of the label */
     for (k=9,i = 0; k < e_label; k++,i++) {
          Label[i] = actualLine[k];                 /* removes '[label="' from the second word */
      }

      Label[i] = '\0';
      lab = 0;
      if( strcmp(Label, names[0]) == 0 ) lab = -1;
      if( strcmp(Label, names[1]) == 0 ) lab = -2;
      if( strcmp(Label, names[2]) == 0 ) lab = 1;
      if( strcmp(Label, names[3]) == 0 ) lab = 2;
      if( strcmp(Label, names[4]) == 0 ) lab = 3;
      if( strcmp(Label, names[5]) == 0 ) lab = 4;
      if( strcmp(Label, names[6]) == 0 ) lab = 5;
      if( strcmp(Label, names[7]) == 0 ) lab = 6;
      if( strcmp(Label, names[8]) == 0 ) lab = 7;
      if( strcmp(Label, names[9]) == 0 ) lab = 8;
      if( strcmp(Label, names[10]) == 0 ) lab = 9;
      if( strcmp(Label, names[11]) == 0 ) lab = 10;
      if( strcmp(Label, names[12]) == 0 ) lab = 11;
      if( strcmp(Label, names[13]) == 0 ) lab = 12;
      if( strcmp(Label, names[14]) == 0 ) lab = 13;

/* LEAF OR NODE?  */
      if (fscanf(infp, "%s", buffer) == EOF) break; /* reads in first string in the row */

        /* IT IS A NODE: READ IN THE CHILDS AND THRESHOLD */
        if ( strlen(buffer) >= 9){
          /* FIND THE L_child */
          e_number = strlen(buffer);
           for (k=12+strlen(nodenumber),i = 0; k < e_number; k++,i++) {
              L_child[i] = buffer[k];          /* removes 'node_x->node' from the second line for the node*/
          }
          L_child[i] = '\0';
            fgets(actualLine, linemax, infp);

          /* FIND THE R_child */
           fscanf(infp, "%s", buffer);
          e_number = strlen(buffer);
           for (k=12+strlen(nodenumber),i = 0; k < e_number; k++,i++) {
              R_child[i] = buffer[k];               /* removes 'node_x->node' from the third line for the node */
          }
          R_child[i] = '\0';

          /* FIND THE THRESHOLD */
           fscanf(infp, "%s", buffer);              /* removes '[label=">' */
           fscanf(infp, "%s", buffer);              /* reads '0.08911"]' */
          e_th = strlen(buffer);
           for (k=0,i = 0; k < e_th-2; k++,i++) {
              THRESH[i] = buffer[k];                        /* removes first space and '"]' from the last word */
          }
           THRESH[i] = '\0';

          /* PRINT ALL THE PARAMETERS OF A NODE */
           printf("Node %s: Leftchild= %s Rightchild= %s Threshold= %s\n",nodenumber, L_child, R_child,
THRESH);

            fprintf(outfp, "%s ", nodenumber);
          fprintf(outfp, "%d ", lab);
          fprintf(outfp, "%s ", L_child);
          fprintf(outfp, "%s ", R_child);
          fprintf(outfp, "%s \n", THRESH);

        }                           /* if ( strlen(buffer) >= 8) */

        /* IT WAS A LEAF */
      else {
          /* PRINT ALL THE PARAMETERS OF A LEAF */
           printf("Leaf %s: Label= %s = %d \n", nodenumber, Label, lab);

            fprintf(outfp, "%s ", nodenumber);
            fprintf(outfp, "%d ", lab);
            fprintf(outfp, "%s ", "0");
            fprintf(outfp, "%s ", "0");
            fprintf(outfp, "%s \n", "0");

            fgets(actualLine, linemax, infp);               /* to read the rest of the line here */
      }

      }                             /* if ( strlen(buffer) < 8) */
}                                               /* end of the loop */
 /* -------------End of the fun part : analyze the Inducer.dot file----------------- */
  printf("\n\nNumber of classes and features in [Names]: %i\n", Nnames);
  printf("Number of nodes and leaves: %d\n\n", j);
  fclose(infp);
  fclose(outfp);
}
```

```
function [confMatrix,myAnns] = testDT(TestMatrix, mlcTree);

% created 10/10/96 by Oliver Wieben
% previous version from Surekha Palreddy
% call [confMatrix,myAnns] = testDT(TestMatrix, mlcTree);
% This algorithm implements a decision tree algorithm based on an
%  input file and test data.
% INPUT : TestMatrix =  column 1: sample number;
%                       column 2 true annotation : 0 for NonPVC, 1 for PVC
%                       column 3,4,... the features
%    mlcTree = Decision Tree Matrix; each row represents a node or a leaf
%                       here: Featurenumbers are 1,2,3,4,..
%                       and -1 for NonPVC -2 for PVC
% OUTPUT : confMatrix = the confusion Matrix
%     myAnns = Annotation of the algorithm (sample number + my annotation)

[Nbeats,Nfeatures] = size(TestMatrix);
Data=TestMatrix(:,3:Nfeatures);     % to get rid of first column=correct classification
Nfeatures=Nfeatures-1;
[Nnodes,junk] = size(mlcTree);

myAnns = zeros(size(Data,1),1);

% creates datasubsets D0, D1, D2 ... D(Nnodes-1)
D0 = Data;
for i = 1:Nnodes-1
  eval(['D' int2str(i) '= [] ;']);
end

for i = 1:Nnodes                        % check out every node
   P = ['D' int2str(i-1)];              % e.g. P=D0
   eval(['Pbeats = size(' P ',1) ; end' ]);          % ?
   if (Pbeats > 0)
      % IF IT IS A DECISION NODE
      if(mlcTree(i,2) ~= 0 & mlcTree(i,3) ~= 0)
         feat = mlcTree(i,1);           % find the feature number
         th   = mlcTree(i,4);           % find the threshold
         Lc   = mlcTree(i,2);           % find the line correspondending to the left child
         L = ['D' int2str(Lc)];         % L=e.g. 'D1"
         Rc   = mlcTree(i,3);           % find the line correspondending to the left child
         R = ['D' int2str(Rc)];         % R=e.g. 'D2"
      %find the beats-numbers whose corresponding feature is below the threshold
         eval(['L_Ind = find( ( ' P '(:,feat) < th) | ( ' P '(:,feat) == th) ); ']);
         Label = ['N' int2str(i-1) '_' int2str(Lc) ];            % e.g. Label = N0_1 : left child of Node0 is
found in line 1 of the mlcTree
         eval([Label '=  L_Ind ; ']);                            % e.g. N0_1 = [102 104 ..]
      % same procedure for right child of the tree
         eval(['R_Ind = find( ' P '(:,feat) > th) ; ']);     % find indices with feature < smaller threshold
         Label = ['N' int2str(i-1) '_' int2str(Rc) ];            % e.g. Label = N0_2
         eval([Label '=  R_Ind ; ']);                            % e.g. N0_2 = [103 105 ...]

         eval([L '= ' P '(L_Ind,:); ' ]);                    % e.g. L = D1
         eval([R '= ' P '(R_Ind,:); ' ]);                    % e.g. L = D2
```

```
    end                                   % if(mlcTree(i,2)...
   % IF IT IS A LEAF
   if(mlcTree(i,2) == 0 & mlcTree(i,3) == 0)
      parent = i-1;
      i_hist = ' ';
      while ( parent ~= 0 )
      % previous parent must have the actual line number in left child or right child
        [prevparent,junk] = find(mlcTree(1:parent,2:3) == parent);
        i_hist = [ '(N' int2str(prevparent-1) '_' int2str(parent) i_hist ')' ];   % e.g. i_hist = (NO_1)
                                          % e.g. i_hist = (NO_1)
        parent = prevparent-1;
      end                                          % while ( parent ~= 0 )
      eval(['myAnns' i_hist '= zeros(size' i_hist ',1) - mlcTree(i,1) -1;']);
      % e.g. myAnns(N0_1) = zeros(size(N0_1),1) - the_feature_number;
    end                                   % if(mlcTree(i,2) == 0...
  end                                              % if (Pbeats > 0)
end                                                % for i = 1:Nnodes
% include the sample number as the first column
myAnns=[TestMatrix(:,1) myAnns];

% calculating the confusion matrix
confMatrix = zeros(2,2);
for i = 1:size(TestMatrix,1)                       % start loop for each beat
 % convert from intern classification to extern classification (0=NonPVC, 1 =PVC)
 confMatrix(TestMatrix(i,2)+1,myAnns(i,2)+1) = 1 + confMatrix(TestMatrix(i,2)+1,myAnns(i,2)+1);
end                                                % end loop for each beat
confMatrix
```

```
% Fuzzy Logic System implemented by Oliver Wieben          12/02/96
%  uses 9 input-features for the classification of PVCs

[System]
Name='ow80iesorig'
Type='mamdani'
NumInputs=9
NumOutputs=1
NumRules=15
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='sum'
DefuzzMethod='centroid'

[Input1]
Name='MWI-15'
Range=[0 0.4444]
NumMFs=2
MF1='normal':'gbellmf',[0.22 6 0]
MF2='wide':'gbellmf',[0.14 10 0.4444]

[Input2]
Name='E01/07'
Range=[0 1]
NumMFs=2
MF1='low':'gbellmf',[0.3 7 0]
MF2='high':'gbellmf',[0.51 10 1]

[Input3]
Name='norm-RR0'
Range=[0.5 2]
NumMFs=3
MF1='premature':'gbellmf',[0.29 5 0.5]
MF2='on_time':'gbellmf',[0.115 1.5 1]
MF3='delayed':'gbellmf',[0.8 8 2]

[Input4]
Name='RR1-to-RR0'
Range=[0 4]
NumMFs=3
MF1='short':'gbellmf',[0.7 5 0]
MF2='regular':'gbellmf',[0.2 2 1]
MF3='long':'gbellmf',[2.5 15 4]

[Input5]
Name='irregularity'
Range=[0 75]
NumMFs=2
MF1='small':'gbellmf',[9 2 0]
MF2='large':'gbellmf',[55 7 75]

[Input6]
```

```
Name='peakwidth'
Range=[0 1]
NumMFs=2
MF1='normal':'gbellmf',[0.06 5 0]
MF2='wide':'gbellmf',[0.9 50 1]

[Input7]
Name='norm-amp'
Range=[0 4]
NumMFs=4
MF1='small':'gbellmf',[0.7 10 0]
MF2='average':'gbellmf',[0.2 2 1]
MF3='enlarged':'gbellmf',[0.2 2.5 1.5]
MF4='highly_enlarged':'gbellmf',[2.3 10 4]

[Input8]
Name='peakdirection'
Range=[0 1]
NumMFs=2
MF1='identical':'gbellmf',[0.25 2.5 0]
MF2='opposite':'gbellmf',[0.25 2.5 1]

[Input9]
Name='norm-peak-to-peak'
Range=[0 3.5]
NumMFs=3
MF1='small':'gbellmf',[0.6 5 0]
MF2='average':'gbellmf',[0.25 2.5 1]
MF3='enlarged':'gbellmf',[2.1 25 3.5]

[Output1]
Name='beat_type'
Range=[0 1]
NumMFs=2
MF1='non-PVC':'trimf',[-0.5 0 0.5]
MF2='PVC':'trimf',[0.5 1 1.5]




[Rules]
2 2 0 0 0 0 0 0 0, 2 (1) : 1
```

```
0 1 0 0 0 0 0 0 0, 1 (1) : 1
0 0 2 2 0 0 0 0 0, 1 (1) : 1
0 0 3 2 0 0 0 0 0, 1 (1) : 1
0 0 2 1 0 0 0 0 0, 1 (1) : 1
0 0 3 1 0 0 0 0 0, 1 (1) : 1
0 0 1 3 0 0 0 0 0, 2 (1) : 1
0 0 0 1 0 0 0 0 0, 1 (1) : 1
0 0 0 2 0 0 0 0 0, 1 (1) : 1
0 0 0 0 1 0 0 0 0, 1 (1) : 1
0 0 0 0 0 0 4 0 0, 2 (1) : 1
0 0 0 0 0 0 3 2 0, 2 (1) : 1
0 0 0 0 0 0 1 2 0, 2 (1) : 1
0 0 0 0 0 0 0 0 3, 2 (1) : 1
0 0 0 0 0 2 0 0 0, 2 (1) : 1
```