GENERAL

matlab code for ecg classification using knn

Published: September 3, 2026 | Index ID: #-

MATLAB Code for ECG Classification Using KNN: A Complete Guide

Electrocardiogram (ECG) analysis is a cornerstone of modern cardiac diagnostics. With the rise of machine learning, automated ECG classification has become a powerful tool for early detection of arrhythmias, myocardial infarction, and other heart conditions. Among the plethora of algorithms available, the k Nearest Neighbors (KNN) method stands out for its simplicity, interpretability, and surprisingly robust performance on many biomedical signal datasets.

This comprehensive article walks you through every step of building an ECG classification system in MATLAB, from data acquisition and preprocessing to feature extraction, KNN implementation, hyper parameter tuning, and evaluation. Whether you're a biomedical engineer, a data scientist, or a medical researcher, you'll find practical code snippets, best practice tips, and insights that can be directly applied to your projects.

Why KNN for ECG Classification?

Before diving into the MATLAB code, it's useful to understand why KNN is a popular choice for ECG signal classification:

- Non Parametric Nature – KNN makes no assumptions about the underlying data distribution, which is advantageous when dealing with complex physiological signals.
- Ease of Implementation – With just a few lines of MATLAB code, you can build a working classifier.
- Interpretability – The decision for each sample is based on the majority label of its nearest neighbors, making it easier to explain to clinicians.
- Good Baseline Performance – Despite its simplicity, KNN often achieves competitive accuracy on ECG datasets, especially when paired with thoughtful feature engineering.

However, KNN also has limitations: it can be computationally heavy on large datasets, and its performance depends heavily on the choice of distance metric and the number of neighbors (k). The article will address these challenges by showing how to optimize your implementation.

Dataset Overview

For this tutorial, we'll use the MIT BIH Arrhythmia Database – a widely used benchmark containing 48 half hour ECG recordings from 47 subjects. Each recording is sampled at 360 /Hz and annotated with 48 rhythm classes.

Key attributes of the dataset:

- Sampling Rate: 360 /Hz
- Signal Length: 30 /minutes per record
- Classes: 48 (including normal, supraventricular, ventricular, and miscellaneous arrhythmias)
- File Format: WFDB (WaveForm DataBase) – MATLAB can read these files directly using the wfdb toolbox.

While the full dataset contains 48 classes, for clarity we'll focus on a binary classification problem: *Normal vs. Arrhythmia*. This simplifies the discussion and keeps the code concise, but the same principles apply to multi class scenarios.

Step 1: Setting Up MATLAB Environment

Installing Required Toolboxes

To work with ECG data in MATLAB, you'll need the following toolboxes:

1. Signal Processing Toolbox – For filtering and feature extraction.
2. Statistics and Machine Learning Toolbox – Provides built in functions for KNN and cross validation.
3. Wavelet Toolbox (optional) – Useful for advanced denoising techniques.
4. WFDB Toolbox – Enables reading of WFDB files. You can download it from PhysioNet.

Once installed, add the WFDB toolbox to your MATLAB path:

```
addpath(genpath("path_to_wfdb_toolbox"));
```

Loading the ECG Recordings

Here's a snippet that loads a single ECG record and its annotations:

```
% Load ECG record 100
[ecg, Fs] = rdsamp("100", 1, 0, 3600); % 1 channel, first 3600 samples

% Load annotations
ann = rdann("100", "atr"); % Atrial activity annotations
```

For batch processing, you can loop through the 48 files, concatenating the signals and labels into a single dataset.

Step 2: Preprocessing the ECG Signal

Raw ECG signals often contain baseline wander, powerline interference, and other noise artifacts. Proper preprocessing is critical for reliable feature extraction.

Baseline Wander Removal

Baseline drift can be mitigated using a high pass filter (cut off ~0.5 /Hz). In MATLAB:

```
hpFilt = designfilt("highpassiir","FilterOrder",4,...
        "HalfPowerFrequency",0.5,"SampleRate",Fs);
ecg_clean = filtfilt(hpFilt, ecg);
```

Powerline Interference Suppression

A notch filter at 50 /Hz (or 60 /Hz depending on your locale) removes mains hum:

```
notchFilt = designfilt("bandstopiir","FilterOrder",2,...
         "HalfPowerFrequency1",49,"HalfPowerFrequency2",51,...
         "SampleRate",Fs);
ecg_clean = filtfilt(notchFilt, ecg_clean);
```

Signal Normalization

Standardizing the amplitude helps KNN avoid bias toward signals with higher variance:

```
ecg_norm = (ecg_clean - mean(ecg_clean)) / std(ecg_clean);
```

Segmentation into Beats

For classification, you typically analyze individual heartbeats. Using the annotation points, segment the ECG into beat windows (e.g., 0.6 /s around each R peak):

```
beatWindow = round(0.6 * Fs); % 0.6 seconds
beats = [];
labels = [];
for i = 1:length(ann)-1
    startIdx = ann(i) - beatWindow/2;
    endIdx   = ann(i) + beatWindow/2;
    if startIdx > 0 && endIdx
```

Replace getLabelFromAnnotation with your own mapping function that assigns “Normal” or “Arrhythmia” based on the annotation.

Step 3: Feature Extraction

Feature engineering turns raw waveforms into numerical descriptors that KNN can compare. Common ECG features include:

- Time Domain Features – RR intervals, QRS duration, PR interval, QT interval, heart rate variability metrics.
- Frequency Domain Features – Power spectral density in specific bands (0.04–0.15 /Hz, 0.15–0.4 /Hz).
- Morphological Features – Peak amplitudes, slopes, area under QRS complex.
- Statistical Features – Mean, standard deviation, skewness, kurtosis of the beat waveform.

Below is a concise MATLAB implementation extracting a handful of features:

```
function feat = extractFeatures(beat)
    % Time domain
    feat.RR = diff(findRPeaks(beat)); % RR intervals
    feat.QRS = getQRSDuration(beat);
    feat.PR = getPRInterval(beat);
    feat.QT = getQTInterval(beat);

    % Statistical
    feat.mean = mean(beat);
    feat.std  = std(beat);
    feat.skew = skewness(beat);
    feat.kurt = kurtosis(beat);
```

```
    % Frequency domain
    [pxx,f] = pwelch(beat,[],[],[],Fs);
    feat.lowFreq  = bandpower(pxx,f, [0.04 0.15], "psd");
    feat.midFreq  = bandpower(pxx,f, [0.15 0.4], "psd");
    feat.highFreq = bandpower(pxx,f, [0.4 1.0], "psd");

    % Flatten into vector
    featVec = [feat.RR, feat.QRS, feat.PR, feat.QT, ...
               feat.mean, feat.std, feat.skew, feat.kurt, ...
               feat.lowFreq, feat.midFreq, feat.highFreq];
    featVec = featVec(~isnan(featVec)); % Remove NaNs
end
```

After extracting features for all beats, you'll have a feature matrix X of size n_samples × n_features and a label vector Y.

Step 4: Splitting Data into Training and Test Sets

To evaluate the classifier, divide the data into training and test subsets. A common split is 80/20, but you can also use stratified sampling to maintain class proportions.

```
cv = cvpartition(Y,"HoldOut",0.2);
XTrain = X(training(cv),:);
YTrain = Y(training(cv));
XTest  = X(test(cv),:);
YTest  = Y(test(cv));
```

Step 5: Implementing KNN in MATLAB

MATLAB's fitcknn function provides a straightforward way to train a KNN classifier. The basic syntax is:

```
mdl = fitcknn(XTrain, YTrain, "NumNeighbors", 5, "Distance", "euclidean");
```

Key parameters:

- NumNeighbors (k) – The number of nearest neighbors to consider.
- Distance – Distance metric (euclidean, cityblock, cosine, etc.).
- Standardize – Whether to standardize features before training.

Hyper Parameter Tuning with Cross Validation

Choosing the optimal k and distance metric can significantly impact performance. Use cross validation to search over a grid of values:

```
kVals = 1:10;
distMethods = {"euclidean","cityblock","chebychev"};
```

```
bestAcc = 0;
bestParams = struct();

for k = kVals
    for d = distMethods
        mdl = fitcknn(XTrain, YTrain, "NumNeighbors", k, ...
                    "Distance", d, "Standardize", true, ...
                    "CV", 5);
        acc = 1 - kfoldLoss(mdl);
        if acc > bestAcc
            bestAcc = acc;
            bestParams.k = k;
            bestParams.distance = d;
        end
    end
end

% Train final model with best parameters
finalModel = fitcknn(XTrain, YTrain, "NumNeighbors", bestParams.k, ...
                    "Distance", bestParams.distance, "Standardize", true);
```

After tuning, you can evaluate on the hold out test set:

```
pred = predict(finalModel, XTest);
conf
```

Baca Juga (Artikel Terkait)

- **automaker mini software version v8 erptoboss**
  URL: http://www.dtrainderailed.com/automaker-mini-software-version-v8-erptoboss.pdf
- **automata and computability**
  URL: http://www.dtrainderailed.com/automata-and-computability.pdf
- **automata and languages theory and applications**
  URL: http://www.dtrainderailed.com/automata-and-languages-theory-and-applications.pdf
- **automata and mechanical toys**
  URL: http://www.dtrainderailed.com/automata-and-mechanical-toys.pdf

Tags: #matlab #code #classification #using