Matlab Code For Ecg Classification Using Knn

INTRODUCTION TO ECG CLASSIFICATION USING K-NEAREST NEIGHBORS IN MATLAB

The electrocardiogram (ECG) remains one of the most essential tools in cardiology for diagnosing heart conditions. With the increasing availability of digital health data, automated classification of ECG signals using machine learning has become a critical area of research. Among the many algorithms available, the K-Nearest Neighbors (KNN) classifier offers a simple yet powerful approach for heartbeat classification. When implemented in MATLAB, it provides researchers and engineers with a flexible environment to preprocess signals, extract features, and build predictive models. This article explores the complete workflow for developing **Matlab Code For Ecg Classification Using Knn**, from data preparation to performance evaluation. Whether you are a student, a data scientist, or a biomedical engineer, this guide will help you understand the process and implement an effective ECG classifier.

UNDERSTANDING ECG SIGNAL CLASSIFICATION

ECG signals record the electrical activity of the heart over time. These signals contain characteristic waveforms – P wave, QRS complex, and T wave – that reflect different phases of the cardiac cycle. Abnormalities in these waveforms can indicate arrhythmias, myocardial infarction, or other cardiac disorders. Manual analysis of long ECG recordings is time-consuming and prone to human error. Automated classification systems can assist clinicians by labeling heartbeats as normal or abnormal, and even identifying specific arrhythmia types.

The goal of ECG classification is to assign each heartbeat (or a segment of the signal) to a predefined class. Common classes include normal beats, premature ventricular contractions (PVC), atrial premature beats, and various conduction blocks. A robust classifier must handle the natural variability in ECG signals caused by patient anatomy, electrode placement, and noise. MATLAB, with its Signal Processing Toolbox and Statistics and Machine Learning Toolbox, provides an ideal platform for developing such classifiers.

WHY CHOOSE K-NEAREST NEIGHBORS FOR ECG CLASSIFICATION?

KNN is a non-parametric, instance-based learning algorithm. It classifies a new data point based on the majority class among its K nearest neighbors in the feature space. Several factors make KNN attractive for ECG classification:

- **Simplicity:** The algorithm is easy to understand and implement. No complex training phase is required; the model simply stores the training data.
- **Non-linearity:** KNN can capture complex decision boundaries without assuming any underlying distribution.
- **Multi-class capability:** ECG classification often involves multiple heartbeat types. KNN naturally handles multi-class problems.
- **Interpretability:** By examining the nearest neighbors, one can understand why a particular heartbeat was classified in a certain way.

However, KNN also has limitations: it is sensitive to the choice of K and distance metric, and it can be computationally expensive with large datasets. Proper feature extraction and dimensionality reduction are therefore essential when writing **Matlab Code For Ecg Classification Using Knn**.

DATASET PREPARATION FOR ECG CLASSIFICATION

Before writing any MATLAB code, you need a reliable ECG dataset. The most widely used public dataset is the MIT-BIH Arrhythmia Database, available from PhysioNet. This database contains 48 half-hour recordings of two-channel ambulatory ECG, with beat annotations verified by experts. Each recording typically includes normal beats, PVCs, paced beats, and other arrhythmias. To use this data in MATLAB:

1. Download the records (e.g., record 100, 101, etc.) along with the annotation files.
2. Use the **rdsamp** and **rdann** functions (from the WFDB toolbox for MATLAB) to read the signals and annotations.
3. Segment the continuous ECG into individual heartbeats using the R-peak locations provided in the annotations.
4. Extract a window of samples around each R-peak (e.g., 100 ms before and 200 ms after) to capture the entire QRS complex and T wave.

For a balanced classifier, it is crucial to handle class imbalance. Normal beats typically dominate arrhythmia datasets. Techniques such as random undersampling, oversampling (SMOTE), or using class weights can be applied later in the code.

PREPROCESSING ECG SIGNALS IN MATLAB

Raw ECG signals contain noise from muscle activity, powerline interference, and baseline wander. Effective preprocessing improves the feature quality and classification accuracy. Typical steps in MATLAB include:

- **Filtering:** Apply a bandpass filter between 0.5 Hz and 50 Hz using `designfilt` and `filtfilt` to remove low-frequency drift and high-frequency noise. A notch filter at 60 Hz (or 50 Hz) eliminates powerline interference.
- **Baseline removal:** Use median filtering or polynomial fitting to correct baseline wander.
- **Normalization:** Scale each heartbeat segment to have zero mean and unit variance. This reduces inter-subject variability.

Here is a snippet of how you might apply a bandpass filter in MATLAB:

```
fs = 360; % sampling frequency of MIT-BIH
[b, a] = butter(4, [0.5 50]/(fs/2), 'bandpass');
filtered_ecg = filtfilt(b, a, raw_ecg);
```

Preprocessing must be applied consistently to both training and test sets to ensure valid classification.

FEATURE EXTRACTION FOR ECG HEARTBEAT CLASSIFICATION

Choosing informative features is the most critical step for KNN performance. Features should capture morphological and temporal characteristics of each heartbeat. Common feature types include:

Time-Domain Features

- R-R interval: time between consecutive R-peaks
- QRS duration
- Amplitude of R peak, Q and S waves
- ST segment elevation or depression
- Area under the QRS complex

Frequency-Domain Features

- Power spectral density in different frequency bands (e.g., very low frequency, low frequency, high frequency)
- Ratio of low frequency to high frequency power (sympathetic/parasympathetic balance)

Wavelet Features

- Discrete wavelet transform coefficients at various scales – these capture transient patterns in the signal
- Approximation and detail coefficients from levels 3–5

Statistical Features

- Mean, variance, skewness, kurtosis of the heartbeat segment
- Entropy measures (Shannon, log energy) to quantify complexity

In MATLAB, you can write a function that loops through each beat and computes these features. For example, to compute R-R interval:

```
rr_intervals = diff(r_peak_indices) / fs;
```

To compute wavelet coefficients:

```
[C, L] = wavedec(beat_segment, 5, 'db4');
```

The resulting feature matrix should have rows representing beats and columns representing features. It is common practice to normalize features to zero mean and unit variance so that the Euclidean distance used in KNN is not dominated by features with larger scales.

IMPLEMENTING KNN CLASSIFICATION IN MATLAB

Once you have the feature matrix (X) and label vector (Y), implementing KNN is straightforward using the Statistics and Machine Learning Toolbox. The key steps in your **Matlab Code For Ecg Classification Using Knn** are:

1. Split the data into training and testing sets (e.g., 70% training, 30% testing). Use `cvpartition` for stratified sampling to preserve class proportions.
2. Create a KNN model:
   ```
   mdl = fitcknn(X_train, Y_train, 'NumNeighbors', 5, 'Distance',
   'euclidean');
   ```
3. Predict on test data:
   ```
   Y_pred = predict(mdl, X_test);
   ```
4. Evaluate performance using confusion matrix, accuracy, precision, recall, and F1-score.

You can tune hyperparameters such as `NumNeighbors` and `Distance` (e.g., 'cityblock', 'cosine', 'correlation') using cross-validation. For example:

```
mdl = fitcknn(X_train, Y_train, 'OptimizeHyperparameters', {'NumNeighbors',
'Distance'},              'HyperparameterOptimizationOptions',
struct('AcquisitionFunctionName', 'expected-improvement-plus'));
```

This automatic optimization helps you find the best K value, which is typically between 3 and 15 for ECG data. A smaller K reduces bias but increases variance; a larger K smooths the decision boundary. Since ECG classes can be overlapping, a moderate K often works well.

HANDLING CLASS IMBALANCE IN ECG DATA

In real-world ECG datasets, normal beats outnumber abnormal beats by a large margin. Without

addressing this imbalance, a KNN classifier might simply predict the majority class and still achieve high accuracy. To build a clinically useful classifier, you must consider imbalance mitigation techniques. Within the context of **Matlab Code For Ecg Classification Using Knn**, you can:

- **Resample the training set:** Use `randsample` to undersample the majority class or use the `smote` function from the MATLAB File Exchange for synthetic oversampling.
- **Use weighted KNN:** The `fitcknn` function allows a 'Cost' matrix to penalize misclassification of minority classes more heavily.
- **Adjust the decision threshold:** In binary or multi-class scenarios, you can shift the probability threshold or use a custom distance weighting.

For multi-class problems, you can also reassign weights based on the inverse class frequency. This ensures that beats from rare arrhythmia types contribute more to the distance calculation.

PERFORMANCE EVALUATION AND METRICS

After running your KNN classifier, you must evaluate its performance thoroughly. Do not rely only on overall accuracy, as it can be misleading in imbalanced datasets. Instead, calculate per-class metrics. In MATLAB:

```
confMat = confusionmat(Y_test, Y_pred);
accuracy = sum(diag(confMat)) / sum(confMat(:));
precision = diag(confMat) ./ sum(confMat, 2);
recall = diag(confMat) ./ sum(confMat, 1)';
f1 = 2 * (precision .* recall) ./ (precision + recall);
```

Visualize the confusion matrix using `heatmap` to identify which classes are frequently confused. Common misclassifications in ECG studies include normal beats being mistaken for bundle branch blocks, or premature beats confused with normal beats due to similar morphology. Such analysis can guide further feature engineering.

OPTIMIZING THE MATLAB CODE FOR LARGER DATASETS

KNN is a lazy learner, meaning it does not build a model during training but stores all training data. When you have thousands of heartbeats, prediction time can become slow. To speed up your **Matlab Code For Ecg Classification Using Knn**, consider these strategies:

- **Use K-d trees:** MATLAB's `fitcknn` automatically uses a K-d tree for Euclidean distance when the number of features is less than 10. For higher dimensions, an exhaustive search is used. Set `'NSMethod', 'kdtree'` if appropriate.
- **Reduce feature dimensionality:** Apply PCA (`pca` function) or select only the most informative features using `fscmrmr` or `relieff`. This reduces computation and may improve accuracy by removing noise.
- **Use a subset of training data:** If the dataset is extremely large, consider using prototype selection to keep only representative training samples.

Another important optimization is to code efficiently: avoid loops over beats when computing features by using vectorized operations. For example, compute all R-R intervals at once with `diff` and apply the same preprocessing to the entire ECG vector before segmentation.

CASE STUDY: CLASSIFYING NORMAL VS. PVC BEATS

To illustrate the entire pipeline