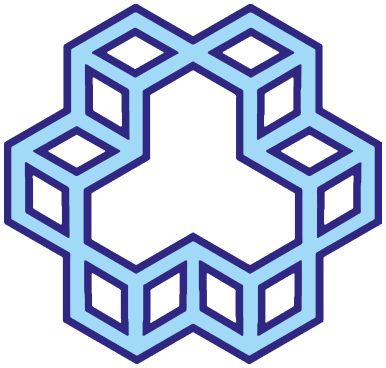
[Path=./font/, Scale=1.3]IRLotus [Scale=1]Times New Roman



K. N. Toosi University of Technology

Faculty of Electrical Engineering — Control Engineering Group

Fundamentals of Intelligent Systems Final Project — Detection and Classification of Cardiac Arrhythmias from ECG Signals

Name	Mahdi Khalilzadeh Shabestari
Student ID	9932213
Date	December 2024

Contents

List of Figures

List of Tables

Listings

1 Introduction

1.1 ECG Signal

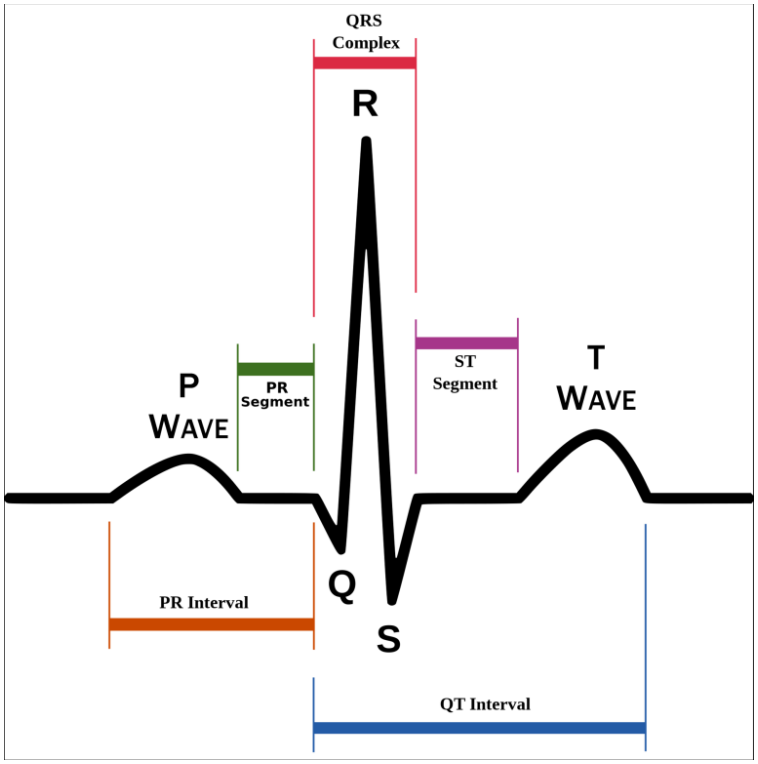


Figure 1: Normal ECG signal and its features [1]

The electrocardiogram (ECG) is a recording of the heart's electrical activity measured by electrodes attached to the skin. It provides information about heart rhythm, electrical conduction, and overall cardiac health. The ECG consists of repeating cycles, each including the P wave, QRS complex, and T wave. The P wave represents atrial depolarization; the QRS complex reflects ventricular depolarization; and the T wave corresponds to ventricular repolarization. These components are visible in Figure 1. Studying these features is essential for diagnosing cardiac diseases such as arrhythmia, ischemia, and heart failure [2].

The ECG signal has specific properties that are important for accurate analysis and processing. One key property is the sampling rate, typically between 250 and 1000 samples per second. The voltage amplitude generally ranges from 0.5 to 5 mV, reflecting the heart's electrical activity. The recording duration depends on the acquisition type; for example, a standard ECG is recorded for several seconds, whereas a Holter ECG collects 24 hours of data. Due to factors such as patient movement, power-line interference, and other electrical disturbances, ECG requires preprocessing and denoising [2].

1.2 Cardiac Arrhythmias and the Importance of Their Detection

Cardiac arrhythmias are disorders of the normal heart rhythm that can arise from problems in the heart's electrical conduction system. They include rhythms that are too fast, too slow, or irregular, and can be caused by cardiovascular disease, genetic issues, stress, certain medications, or electrolyte imbalances. Some arrhythmias—such as atrial fibrillation and premature ventricular contractions—may be benign, whereas others—such as sustained ventricular tachycardia and ventricular fibrillation—are very dangerous and can lead to sudden cardiac arrest [b4].

Early detection of arrhythmias is crucial to prevent serious complications. The primary diagnostic method is the electrocardiogram (ECG), which records the heart's electrical activity and reveals abnormal patterns. Manual analysis of ECG is time-consuming and prone to human error. Therefore, machine learning and deep neural networks are now widely used for automated arrhythmia detection, improving both accuracy and speed. These models can identify complex ECG patterns and classify each beat into different arrhythmia categories [b4].

A key application is the integration of arrhythmia detection models into smart medical devices and patient monitoring systems. Devices such as smartwatches and wearable monitors can continuously record ECG and issue alerts upon detecting arrhythmias. This capability helps cardiac patients and at-risk individuals seek prompt medical care and avoid severe outcomes. Continued development and improvement of these models can transform prevention, diagnosis, and treatment of heart disease, reducing mortality from dangerous arrhythmias [b4].

1.3 Project Objective

The goal of this project is to detect cardiac arrhythmias from beats recorded in ECG signals using the MIT-BIH Arrhythmia database. ECG signals are analyzed so that each beat is examined individually to determine whether it exhibits an arrhythmia. Another objective is to select the best machine-learning model for accurate arrhythmia detection. By comparing different models, we aim to find an optimal method for automatically classifying heartbeats into the five main arrhythmia classes.

Cardiac arrhythmias are among the most important medical problems and, if not detected and treated quickly, can lead to serious complications and even sudden death. Early detection of arrhythmias via ECG is an effective approach to preventing cardiac complications. The proposed model in this project can be integrated into personal care devices and medical monitoring systems so that, upon arrhythmia detection, the patient is promptly informed and can consult a physician. Such a system can serve as an early-warning tool for individuals at risk of heart disease.

2 Introduction to the MIT-BIH Arrhythmia Dataset

The MIT-BIH Arrhythmia database [b5] is one of the most reputable and widely used resources for ECG signal analysis and arrhythmia detection. It was developed at the Beth Israel Hospital Arrhythmia Laboratory in Boston, USA, and has been available to researchers since 1980. Its primary purpose is to establish a global standard for evaluating and developing ECG analysis and arrhythmia detection algorithms.

2.1 Dataset Specifications and Structure

The MIT-BIH database contains 48 ECG records, each consisting of 30 minutes of two-channel electrocardiogram data from 47 real patients. The set includes both normal subjects and patients with various arrhythmias. Data were selected from 4000 twenty-four–hour recordings: 23 were chosen at random, and 25 were chosen to include clinically significant and rare arrhythmias.

Technical specifications of the ECG signals:

- **Number of channels:** two ECG channels per record, including one standard lead and one additional selected lead
- **Sampling rate:** 360 samples per second per channel, enabling high-resolution analysis
- **Digital resolution:** 11 bits over a 10 mV range, ensuring signal quality
- **Record duration:** 30 minutes, covering a broad range of cardiac activity
- **Available formats:** WFDB, CSV, and MATLAB for use across software environments
- **Expert annotations:** over 110,000 heartbeats with labels reviewed and confirmed by cardiologists
- **Recorded leads:** per record, one channel is Modified Lead II (MLII) and the second is one of V1, V2, V5, or V6. In some cases, combinations such as Limb Lead II (LL) and V1 appear depending on the patient.

2.2 Annotations and Labeling

One of the most important features of MIT-BIH is its precise, expert labeling for arrhythmia detection. In this dataset:

- Each heartbeat is labeled individually to indicate the type of arrhythmia or normal rhythm.
- Labels were assigned by cardiology experts and have very high accuracy.
- Timing (in sample indices) for each beat is provided in text files accompanying the signals.
- Beyond beat type, some files indicate noise levels, unreliable segments, and special rhythm events.
- Annotation files include `.atr`, `.hea`, and `.txt`, with precise timing and type of each heartbeat.

2.3 Types of Recorded Arrhythmias

The MIT-BIH database includes various arrhythmias grouped into five main classes:

Table 1: Main arrhythmias in the dataset

N	Normal beat	A normal beat observed in healthy individuals
A	APB	Atrial premature beat arising from abnormal atrial activity
V	PVC	Premature ventricular contraction due to sudden ventricular activity
F	Fusion beat	Fusion of a ventricular beat with a normal beat
Q	Unclassifiable beat	A beat that does not fall into standard categories

In addition to the main groups, the database contains annotations for rarer arrhythmias as well:

Table 2: Other arrhythmia types recorded in the dataset

L	LBBB	Left bundle branch block due to conduction obstruction in the left pathway
R	RBBB	Right bundle branch block indicating impaired conduction in the right pathway
J	JPB	Junctional premature beat originating from the AV node
S	SVPB	Supraventricular premature beat caused by abnormal activity above the ventricles
E	VEB	Ventricular escape beat occurring when SA node activity is reduced

This dataset is recognized as an international standard for evaluating and developing ECG analysis and arrhythmia detection algorithms.

3 Preprocessing

3.1 Loading the Dataset

To prepare the dataset for model training, we load it into Google Colab. The zipped signal files and their arrhythmia event labels are downloaded from the dataset website and uploaded to Google Drive. We then use `gdown` to fetch the archive in Colab:

```
!gdown 1jndKYtmHxRq1I7wNeKXuEQ7t0WYukfyu -O mitbih_database.zip
!unzip mitbih_database.zip -d mitbih_database
```

Code 1: Code for downloading the dataset

Next, the following code iterates over all files, reads them, and separates the signal files from the labels:

```
# Separate records and annotation files
for name in filenames:
    filename, file_extension = os.path.splitext(name)
    if file_extension == '.csv':
        records.append(path + filename + file_extension)
    if file_extension == '.txt':
        annotations.append(path + filename + file_extension)
```

Code 2: Code for separating files

Below is an example of a label file:

Time	Sample #	Type	Sub	Chan	Num	Aux
0:00.050	18	+	0	0	0	(N
0:00.214	77	N	0	0	0	
0:01.028	370	N	0	0	0	
0:01.839	662	N	0	0	0	
0:02.628	946	N	0	0	0	
0:03.419	1231	N	0	0	0	
0:04.208	1515	N	0	0	0	
0:05.025	1809	N	0	0	0	
0:05.678	2044	A	0	0	0	

Figure 2: An example of annotations

As shown—and as described in the dataset documentation—the label for each beat is recorded at the sample index corresponding to the R-peak of that beat.

3.2 Signal Quality Improvement

The figure below shows one thousand samples from a record in the dataset:

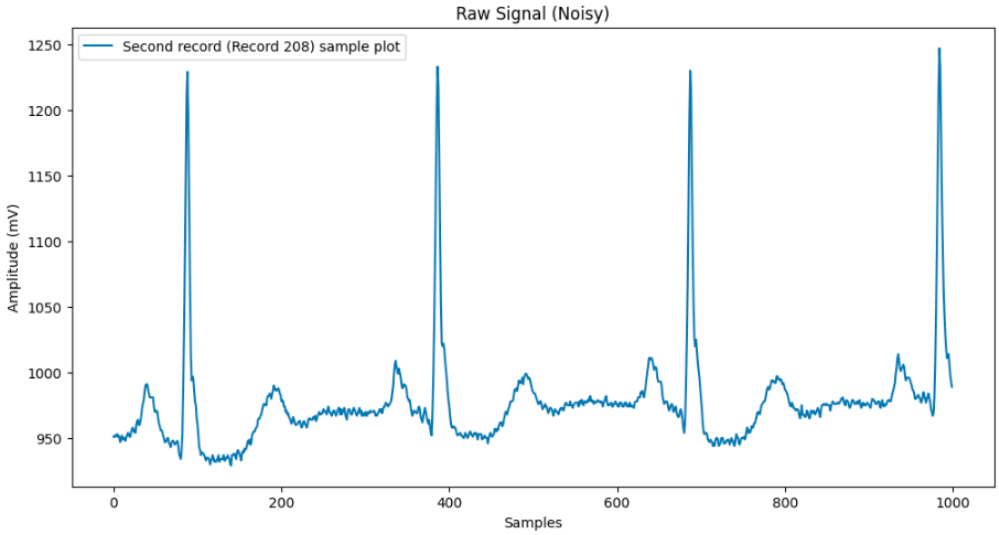


Figure 3: Raw signal

```python
def smooth_ecg(ecg_signal, window_size=10, fs=360):
    kernel = np.ones(window_size) / window_size
    denoised_signal = np.convolve(ecg_signal, kernel, mode='same')
    return denoised_signal

def notch_filter60Hz_ecg(ecg_signal, notch_freq=60, fs=360, quality_factor=30):
    # Design Notch filter
    w0 = notch_freq / (fs / 2)  # Normalize frequency
    b, a = iirnotch(w0, quality_factor)
    # Apply the Notch filter
    filtered_signal = filtfilt(b, a, ecg_signal)
    return filtered_signal

def notch_filter30Hz_ecg(ecg_signal, notch_freq=30, fs=360, quality_factor=30):
    # Design Notch filter
    w0 = notch_freq / (fs / 2)  # Normalize frequency
    b, a = iirnotch(w0, quality_factor)
    # Apply the Notch filter
    filtered_signal = filtfilt(b, a, ecg_signal)
```

```
    return filtered_signal
```

Code 3: Filters

The effect of each stage on the signal is shown here:

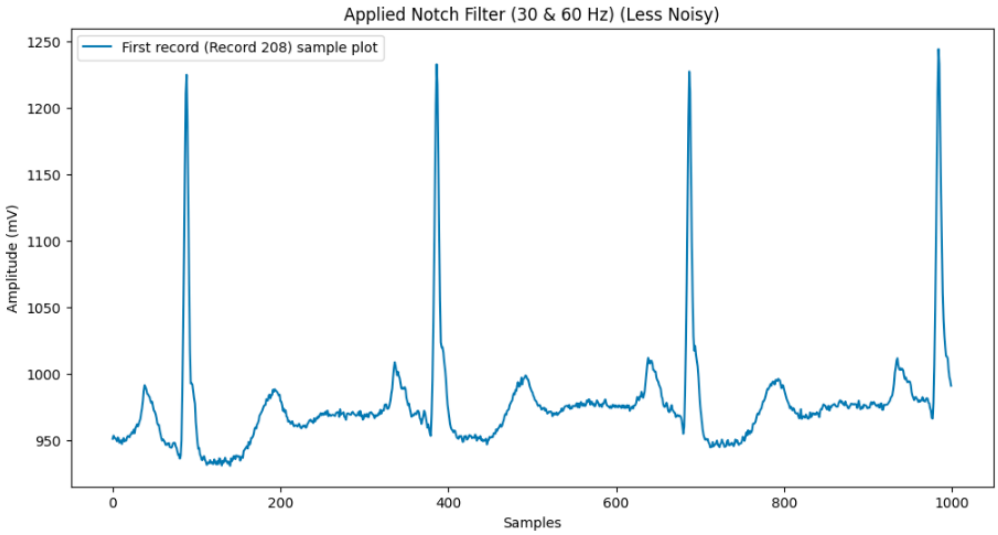


Figure 4: Signal after applying the notch filter

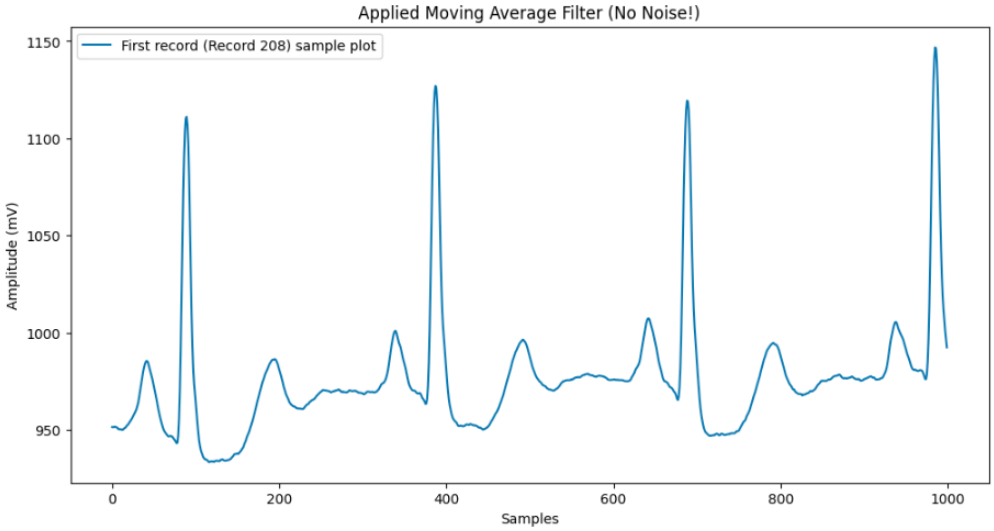


Figure 5: Signal after notch and moving average

Finally, Z-score normalization is applied. The function is given below:

```
def z_score_normalize(ecg_signal):
    ecg_signal = np.array(ecg_signal)
    mean = np.mean(ecg_signal)
```

```
    std = np.std(ecg_signal)
    normalized_signal = (ecg_signal - mean) / std
    return normalized_signal
```

Code 4: Z-Score normalization

The final improved signal is:

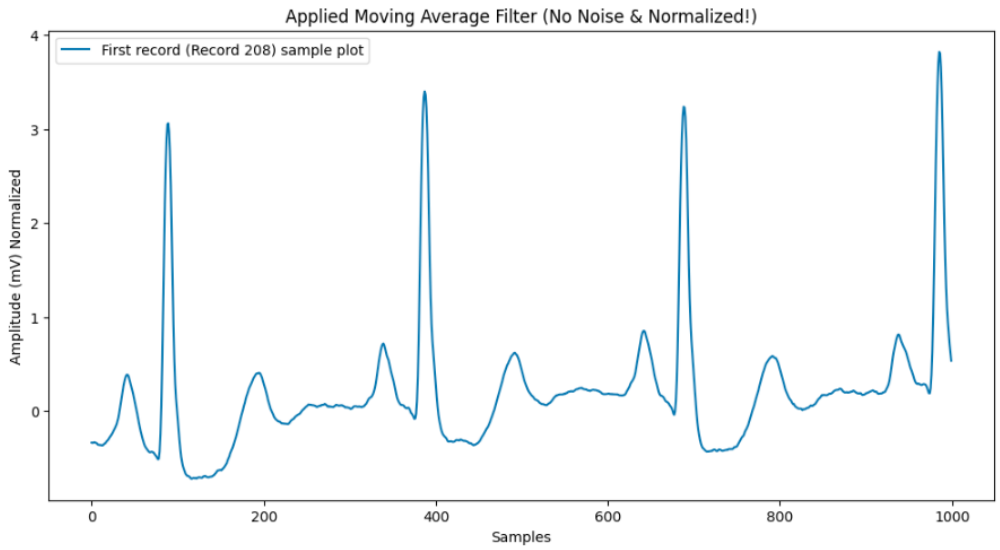


Figure 6: Final improved signal

3.3 Beat Segmentation

```
# Define parameters
beat_size_afterR = 160
beat_size_beforeR = 90
fs = 360  # Sampling frequency (Hz)

# ECG beat classes
classes = ['N', 'V', 'A', 'L', 'R', 'F']
#Normal | Premature Ventricular Contraction | Premature Atrial Contraction
    | Left Bundle Branch Block | Right Bundle Branch Block | Beat Fusion
n_classes = len(classes)
count_classes = [0] * len(classes)

```

```
12
13 X = list()
14 y = list()
15
16 # Process each ECG record
17 for r in range(len(records)):  # Iterate over all records
18     signals = []
19
20     # Read ECG signal from CSV file
21     with open(records[r], 'rt') as csvfile:
22         reader = csv.reader(csvfile, delimiter=',', quotechar='|')  # Read
       CSV file
23         row_index = -1
24         for row in reader:
25             if row_index >= 0:
26                 signals.append(int(row[1]))  # Second column contains the
       lead data
27             row_index += 1
28
29     # Remove ECG signal Noise
30     signals = Remove_Noise(signals)
31     # Normalize signal using z-score
32     signals = z_score_normalize(signals)
33
34     # Extract beats based on annotations
35     with open(annotations[r], 'r') as fileID:
36         data = fileID.readlines()
37         for d in range(1, len(data)):  # Skip header
38             splitted_anotation = data[d].split()  # Split by whitespace
39             if len(splitted_anotation) < 3:
40                 continue  # Skip invalid lines
41             pos = int(splitted_anotation[1])  # Extract Sample Index
       directly
42             arrhythmia_type = splitted_anotation[2]  # Extract Arrhythmia
       label
43
```

```
            if arrhythmia_type in classes:  # Check if beat type is relevant
                arrhythmia_index = classes.index(arrhythmia_type)
                count_classes[arrhythmia_index] += 1  # Count beats per class

                if beat_size_afterR <= pos < (len(signals) - beat_size_afterR):
                    beat = signals[pos - beat_size_beforeR: pos + beat_size_afterR]  # Extract beat

                    X.append(beat)
                    y.append(arrhythmia_index)
```

Code 5: Beat segmentation

A sample segmented beat labeled N is shown below:

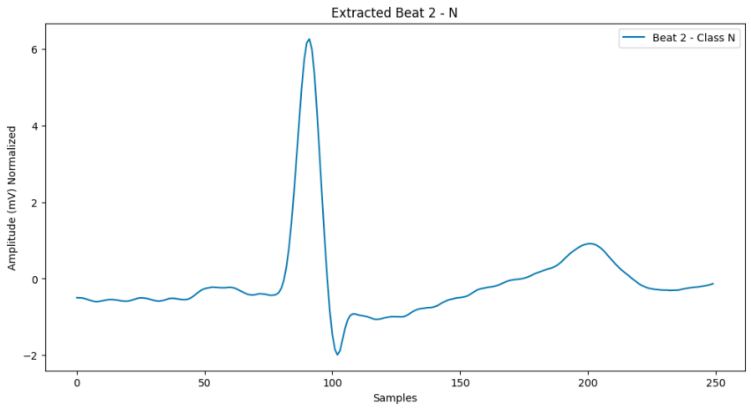


Figure 7: An example of a segmented beat

3.4 Dataset Balancing

The pie chart of class counts before balancing is:

As shown, the distribution is imbalanced, which would bias model predictions. To address this, we apply under/over-sampling to obtain 5000 samples per class.

The post-balancing distribution is:

Finally, an example beat from each class is shown:

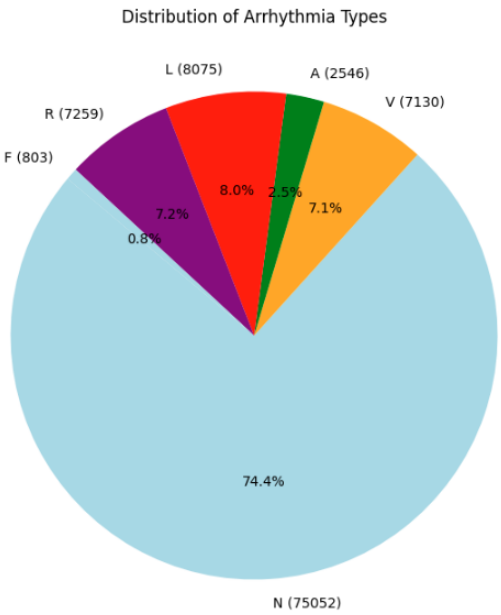


Figure 8: Class-count distribution before balancing

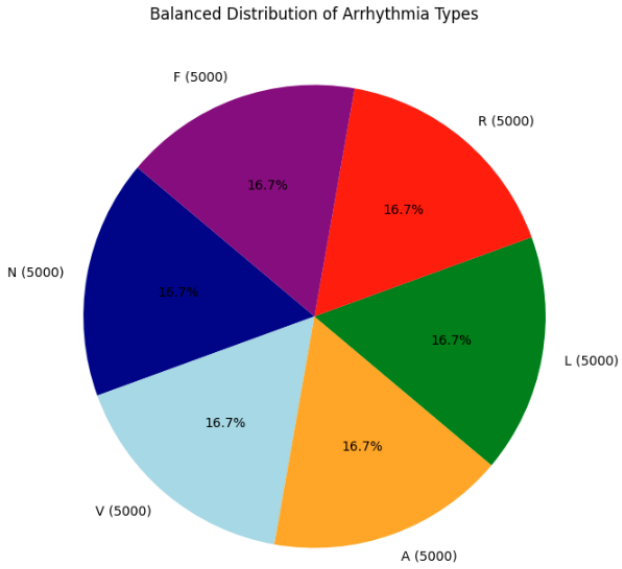


Figure 9: Class-count distribution after balancing

4 Implemented Models

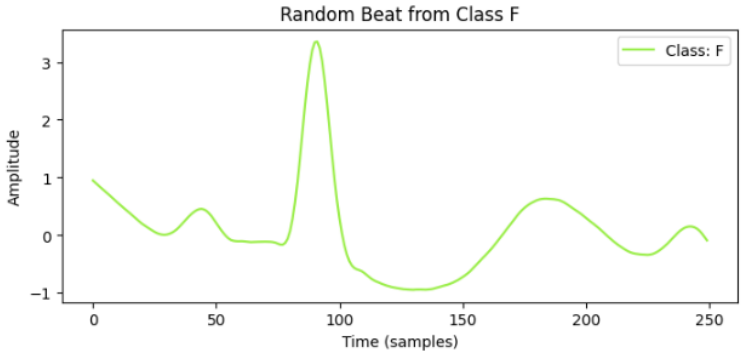


Figure 10: Example beat — class F

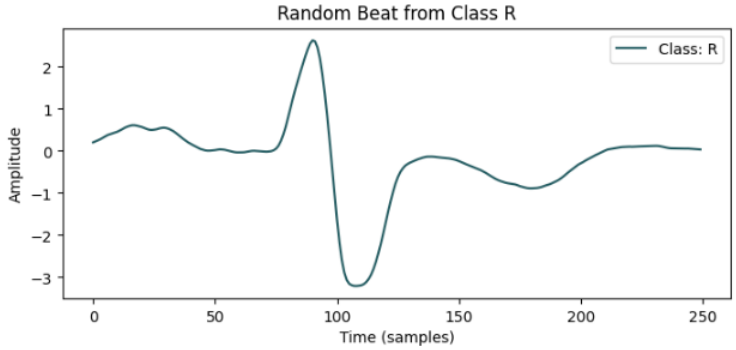


Figure 11: Example beat — class R

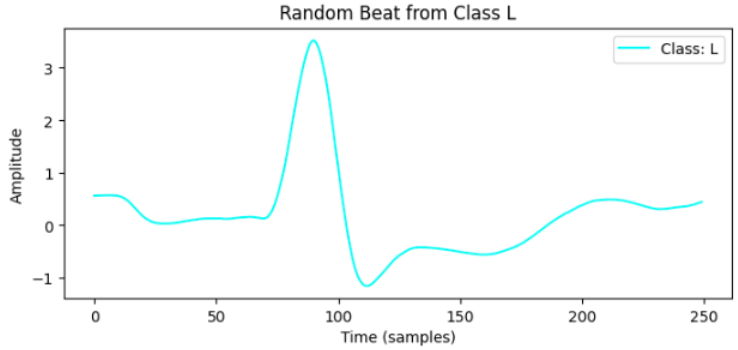


Figure 12: Example beat — class L

4.1 Model 1: Logistic Regression

4.1.1 Why Linear/Logistic Regression? Strengths and Weaknesses

Logistic regression is one of the simplest and most widely used machine-learning models for classification. In this project, logistic regression was selected because:

- **Simplicity and interpretability:** easy to implement; weights indicate each feature's influence.

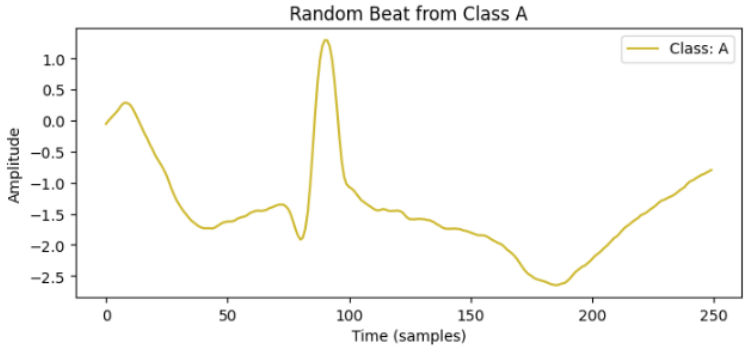


Figure 13: Example beat — class A

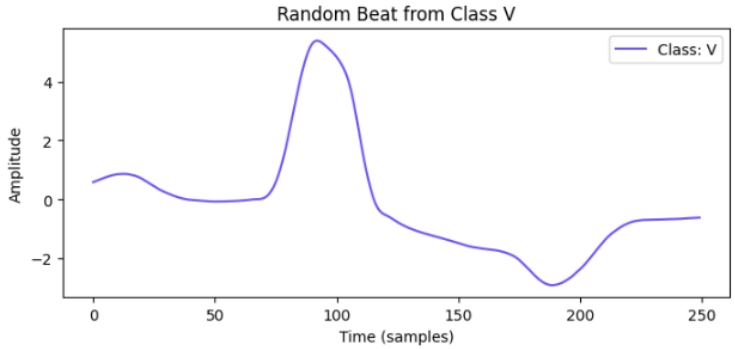


Figure 14: Example beat — class V

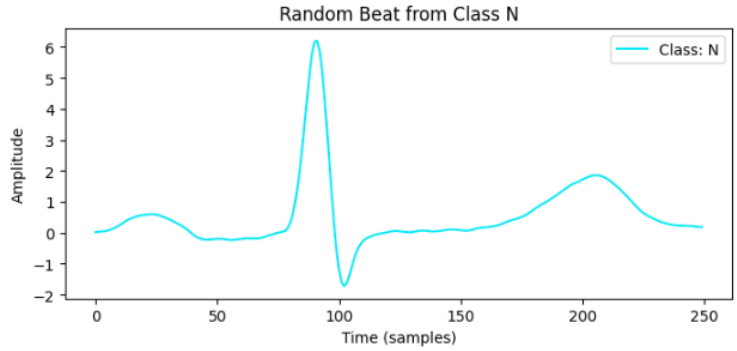


Figure 15: Example beat — class N

- **Fast training:** much faster than more complex models such as neural networks.
- **Generalization:** when the relationship is approximately linear in feature space, performance is reasonable on new data.

However, linear/logistic regression has limitations:

- **Cannot model complex relationships:** assumes linear decision boundaries, whereas medical signals often contain nonlinear patterns.

- **Sensitivity to outliers:** noisy or outlying data can degrade performance.
- **Weak on high-dimensional, complex dependencies:** when inputs interact nonlinearly, performance suffers.

Here, logistic regression is used as a *baseline* to compare with more sophisticated models (e.g., deep neural networks).

4.1.2 Implementation and Code Explanation

```
X_train_flat = X_train.reshape(X_train.shape[0], -1)
X_test_flat = X_test.reshape(X_test.shape[0], -1)
X_val_flat = X_val.reshape(X_val.shape[0], -1)
```

Code 6: Flattening beats into numeric vectors

Next, we define a pipeline with `StandardScaler()` and logistic regression (5000 max iterations):

```
# Logistic Regression Pipeline with Scaling
logistic_model = Pipeline([
    ('scaler', StandardScaler()),  # Normalize input
    ('classifier', LogisticRegression(max_iter=5000, random_state=13))
])

# Train Model
logistic_model.fit(X_train_flat, y_train)
```

Code 7: Defining the logistic regression model

4.1.3 Results of Logistic Regression

After training on the prepared data, we obtain:

As shown, accuracy is about 83%.

	precision	recall	f1-score	support
N	0.70	0.74	0.72	1000
V	0.81	0.78	0.79	1000
A	0.87	0.79	0.83	1000
L	0.82	0.87	0.84	1000
R	0.92	0.95	0.93	1000
F	0.88	0.85	0.86	1000
accuracy			0.83	6000
macro avg	0.83	0.83	0.83	6000
weighted avg	0.83	0.83	0.83	6000

Figure 16: Classification report — logistic regression

The confusion matrix is:

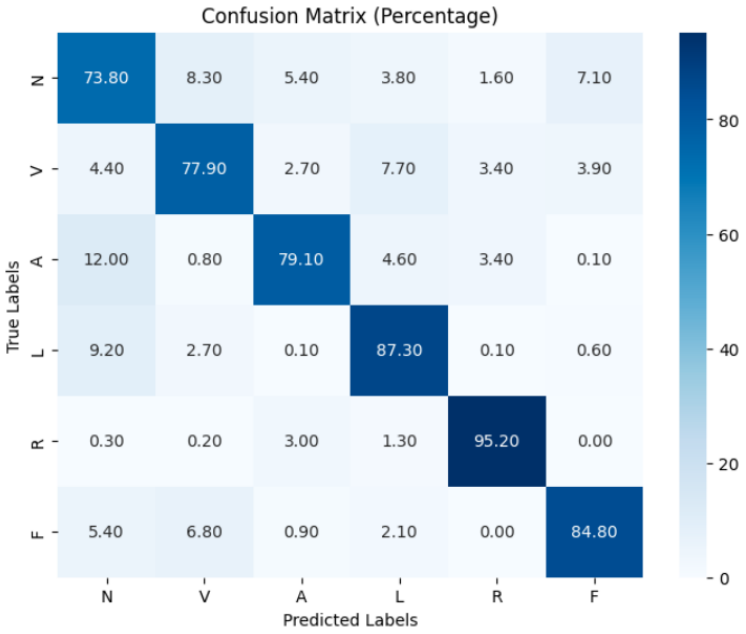


Figure 17: Confusion matrix — logistic regression

We observe weaker performance on normal beats and PVC beats.

ROC curves:

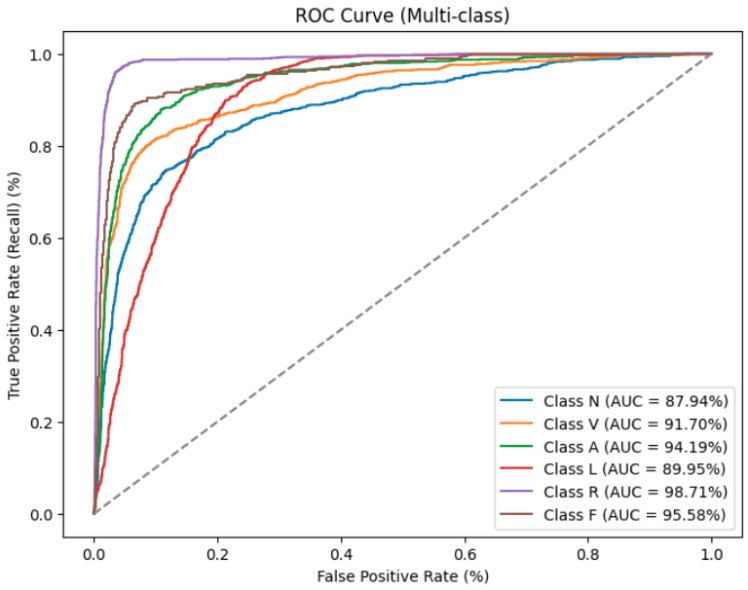


Figure 18: ROC curves — logistic regression

4.2 Model 2: Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) designed for sequential data and time series. Standard RNNs suffer from the vanishing gradient problem as sequence length increases, making long-range dependency learning difficult. LSTMs address this with memory cells and gating mechanisms, enabling learning of long-term dependencies [b7].

Each LSTM unit has three gates: input, output, and forget gates. Using sigmoid activations, these gates control information flow within the cell. The forget gate determines how much past information to retain or discard; the input gate controls how much new information to store; and the output gate decides how much of the cell state to expose as output. These mechanisms allow LSTMs to retain important information over time and discard irrelevant content [b7].

Due to this structure, LSTMs are effective in speech recognition, machine translation, time-series analysis, and NLP. Their ability to learn both long- and short-term dependencies makes them powerful for modeling sequential data [b7]. In this section, we implement an LSTM-based classifier for ECG arrhythmia detection, leveraging their strength on time-series signals.

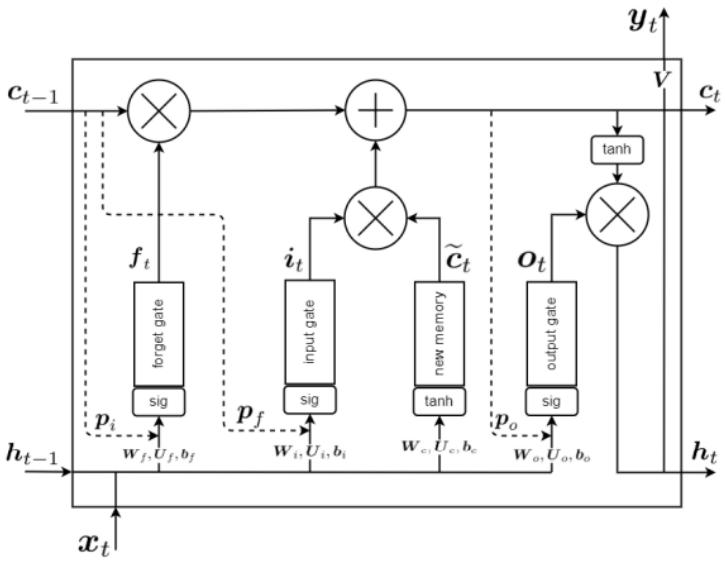


Figure 19: Structure of an LSTM cell [**b7**]

4.2.1 Why LSTM? Strengths and Weaknesses

Advantages:

- Learns long-term temporal dependencies; extracts time-dependent ECG features.
- Robust to noise relative to classical models.
- Reduces need for hand-crafted features.
- Typically achieves higher accuracy on biomedical sequences.

Disadvantages:

- Requires larger training datasets.
- Longer training time and higher computational cost.
- Often benefits from GPU acceleration.

4.2.2 Implementation and Code (LSTM)

Preprocessing: After shaping the data to `(n_samples, sequence_length, n_features)`, class labels are one-hot encoded.

```
# Reshape data to fit LSTM (samples, timesteps, features)
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_val = X_val.reshape(X_val.shape[0], X_val.shape[1], 1)
```

```
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

# Convert labels to one-hot encoding
num_classes = 6  # 6 classes
y_train = to_categorical(y_train, num_classes)
y_val = to_categorical(y_val, num_classes)
y_test = to_categorical(y_test, num_classes)
```

Code 8: Preparing data for LSTM

Model architecture:

- *Input:* ECG sequences as time-sampled vectors; shape `(n, L, 1)`.
- *LSTM layers:* an initial LSTM (64 units) with `return_sequences=True`, followed by a second LSTM (32 units).
- *Dense layers:* a dense layer (64 units), then a softmax output with 6 neurons (for 6 classes).

Optimization:

- Loss: categorical cross-entropy.
- Optimizer: Adam.
- Trained for 400 epochs with validation each epoch.

Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 250, 64)	16,896
batch_normalization_2 (BatchNormalization)	(None, 250, 64)	256
dropout_3 (Dropout)	(None, 250, 64)	0
lstm_3 (LSTM)	(None, 32)	12,416
batch_normalization_3 (BatchNormalization)	(None, 32)	128
dropout_4 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 32)	1,056
dropout_5 (Dropout)	(None, 32)	0
dense_3 (Dense)	(None, 6)	198

Figure 20: LSTM model architecture

Model code:

```
def build_lstm_model(input_shape, num_classes):
    model = Sequential([
        LSTM(64, return_sequences=True, input_shape=input_shape),
```

```
        BatchNormalization(),
        Dropout(0.3),

        LSTM(32, return_sequences=False),
        BatchNormalization(),
        Dropout(0.3),

        Dense(32, activation='relu'),
        Dropout(0.3),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

Code 9: LSTM model implementation

4.2.3 LSTM Results

After training for 400 epochs on the prepared data, the LSTM required significantly more training time but achieved:

Classification Report:

	precision	recall	f1-score	support
0	0.97	0.96	0.96	1000
1	0.99	0.97	0.98	1000
2	0.97	0.97	0.97	1000
3	1.00	0.99	0.99	1000
4	0.99	0.99	0.99	1000
5	0.97	1.00	0.98	1000
accuracy			0.98	6000
macro avg	0.98	0.98	0.98	6000
weighted avg	0.98	0.98	0.98	6000

Figure 21: Classification report — LSTM

Accuracy is about 98%.

Training curves (accuracy and loss) per epoch:

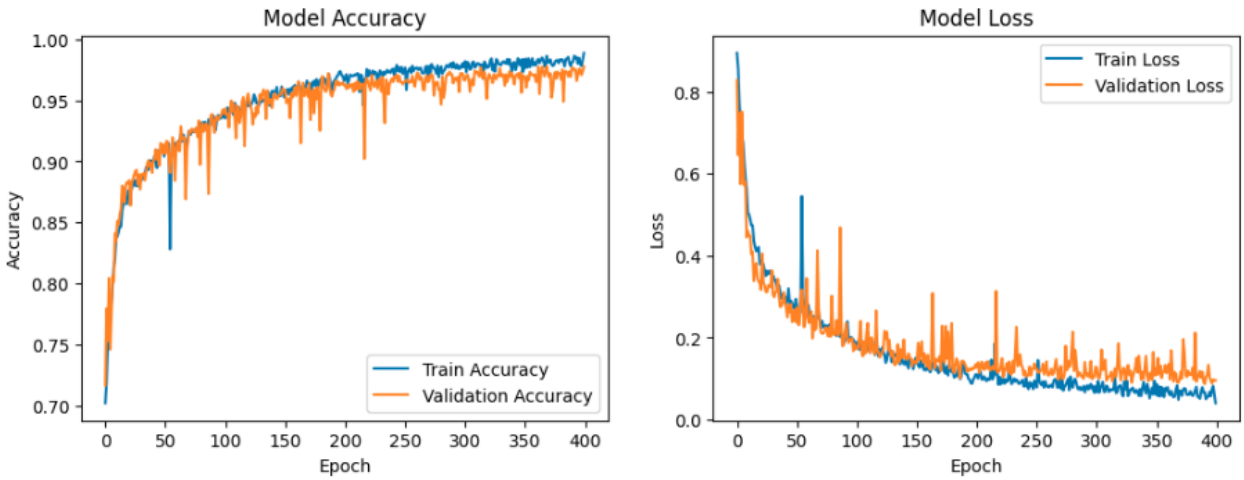


Figure 22: Loss and accuracy curves — LSTM

Confusion matrix for the LSTM (substantially better than classical models on time-series ECG):

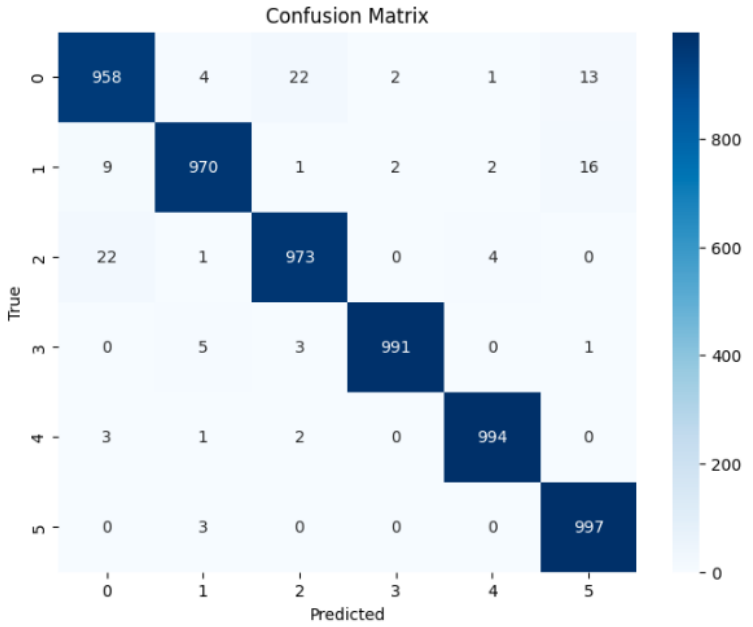


Figure 23: Confusion matrix — LSTM

5 Conclusion

In summary, for ECG analysis and classification, models that handle sequential data are preferable. Classical models such as logistic regression or SVM, while effective in many ML tasks, lack the ability to learn temporal dependencies inherent in physiological signals and thus underperform on ECG. In contrast, RNNs—and especially LSTMs—are designed for sequences and can learn hidden temporal patterns in ECG to classify arrhythmias accurately.

Our experiments show the LSTM substantially outperforms classical models. This advantage stems from its long-term memory, enabling it to capture hidden dependencies across time. However, our data came from a single dataset and experiments were in-dataset. For stronger generalization, models should be evaluated on more diverse datasets with varied patients and realistic conditions.

For clinical use, a model must combine high accuracy with sufficient processing speed, generalization, and robustness to real-world noise. Complex models like LSTM/CNN require larger training sets and more compute, which can be challenging on wearable or remote monitoring devices. Future work can explore optimized, lighter architectures (e.g., GRU or hybrid CNN–LSTM) to balance accuracy with runtime and hardware efficiency. Overall, LSTM is an effective solution for ECG arrhythmia detection, but further optimization is needed for deployment in healthcare and medical devices.

6 Important Links

GitHub.

Google Drive.

Google Colab.

References

[1] Machine Learning course page.

[2] G. N. Bhat, A. K. Ghatol, and A. S. Bhide, "ECG Signal Feature Extraction Trends in Methods and Applications," *BioMedical Eng. OnLine*, vol. 22, no. 1, pp. 1–25, 2023.

[3] O. el B'Charri, L. Rachid, W. Jenkal, and A. Abenaou, "The ECG Signal Compression Using an Efficient Algorithm Based on the DWT," *Int. J. Adv. Comput. Sci. Appl.*, vol. 7, pp. 1–7, Mar. 2016, doi: 10.14569/IJACSA.2016.070325.

[4] U. R. Acharya et al., "Automatic Detection of Arrhythmias Using Machine Learning and Deep Learning Algorithms," *Biomed. Signal Process. Control*, vol. 68, p. 102164, 2021, doi: 10.1016/j.bspc.2021.102164.

[5] G. B. Moody and R. G. Mark, "The Impact of the MIT-BIH Arrhythmia Database," *IEEE Eng. Med. Biol. Mag.*, vol. 20, no. 3, pp. 45–50, 2001.

[6] S. Khurshid, S. H. Choi, L.-C. Weng, E. Y. Wang, L. Trinquart, E. J. Benjamin, P. T. Ellinor, and S. A. Lubitz, "Frequency of Cardiac Rhythm Abnormalities in a Half Million Adults," *Circ. Arrhythm. Electrophysiol.*, vol. 11, no. 7, p. e006273, 2018.

[7] B. Ghojogh and A. Ghodsi, "Recurrent Neural Networks and Long Short-Term Memory Networks: Tutorial and Survey," arXiv:2304.11461, Apr. 2023.