ECG Heartbeat Classification

By Saydain Sheikh

May 13, 2024

CPE 646-A: Pattern Recognition & Classification

Abstract:

This dataset is composed of two collections of heartbeat signals derived from two famous datasets in heartbeat classification, the MIT-BIH Arrhythmia Dataset and The PTB Diagnostic ECG Database.

Objective: The number of samples for both collections is large enough to train machine learning models such as Random Forest, XGBoost and Block Based Neural Network (BbNN). The ECG project is aiming to develop machine learning models to accurately classify ECG heartbeat signals into few categories based on detected anomalies. This, in turn, helps in early detection of cardiac issues such as Arrhythmia, which is crucial for appropriate timely intervention.

Tools and Approach: This project utilizes Python in Jupyter Notebook Environment for data processing and machine learning, implementing several in-built libraries such as NumPy, Pandas for data manipulation, Matplotlib and Seaborn for data visualization, and Scikit-Learn and TensorFlow for building and training the classification models. The underlying approach involves preprocessing of ECG data using domain knowledge, feature extraction and then applying unsupervised learning techniques.

Results and Outcome: The Random Forest achieved an accuracy of 91.126% with precision value 74% and recall value 99% for Normal beats that indicates high reliability. The best performing model has been XGBoost Classifier with an accuracy of 96.66% with precision value 99% and recall value 97% for Normal Beats. Additionally, a Neural Network model was built from scratch that had the accuracy of 82% with the precision value 95% and recall value 99% for

the Normal beats and this model had different activation function such Leaky ReLU, Tanh, and sigmoid functions.

Keywords: ECG Signals, Random Forest, XGBoost Classifier, Neural Network, Butter Low-Pass Function

Introduction:

Background Information: Electrocardiography (ECG) is an essential diagnostic tool used in assessing the electrical and muscular functions of the heart. An ECG records the electrical signals that is produced each time the heart beats, providing a comprehensive view of the heart's rhythm and electrical activity. ECG is essential for identifying irregular heartbeats such as Arrhythmia. It can detect other conditions such as atrial fibrillation, atrial flutter, ventricular tachycardia, and bradycardia, which can have serious health impact if not treated timely. It also reveals patterns indictive of a current or past heart strokes, and these ECG traces can show which part of the heart muscle is affected that needs to be examined. In short, An ECG is a non-invasive, quick, and cost-effective tool that plays a critical in diagnosing and managing heart diseases. It provides valuable insights into the heart's function and helps in early detection and treatment, improving patient outcomes.

Problem Statement: The problem we are trying to solve is to learn the patterns from ECG Signals and classify them into several categories such as Normal Beats, Supraventricular Ectopy Beats, Ventricular Ectopy Beats, Fusion Beats, and Unclassifiable Beats. Although ECG plays a critical role in identifying heart diseases, the interpretation of the results can be prone to human error that could lead to misdiagnosis, delayed treatment, or unnecessary interventions. Therefore, the main problem we addressed is the need for an accurate, reliable, and automated system to classify ECG abnormalities, reducing the reliance on subjective analysis and improving diagnostic accuracy.

Scope: This project focuses on building an automation system for the Classification of ECG abnormalities using advanced machine learning algorithms. The system will be trained to

identify and classify various types of ECG abnormalities, especially focusing on Arrhythmias that includes: Atrial fibrillation, Atrial flutter, Ventricular tachycardia and Bradycardia. This project will go through the building, testing, and validation of the ECG machine learning system, ensuring high standards of accuracy and reliability.

Tools and Methodology:

Data Collection: The data for ECG Classification was collected from the Kaggle website at this link: https://www.kaggle.com/datasets/shayanfazeli/heartbeat/data

Content

Arrhythmia Dataset

- Number of Samples: 109446
- Number of Categories: 5
- Sampling Frequency: 125Hz
- Data Source: Physionet's MIT-BIH Arrhythmia Dataset
- Classes: ['N': 0, 'S': 1, 'V': 2, 'F': 3, 'Q': 4]

The PTB Diagnostic ECG Database

- Number of Samples: 14552
- Number of Categories: 2
- Sampling Frequency: 125Hz
- Data Source: Physionet's PTB Diagnostic Database

Data Preprocessing: The ECG data was cleaned using Low-Pass filter, which helps remove High-Frequency noise while retaining the important Low-Frequency components related to unusual heart Activity. The Butter Lowpass Filter Function takes parameter such as the cutoff Frequency, sampling rate (which is more than the double the cutoff Frequency), and filter order, and returns the filtered signal.

Demonstrating The Butter Low-Pass Filter Function:

Preprocessing

```
# Splitting Target and other values
ecg_train_target = ecg_train[187]
ecg_test_target = ecg_test[187]
train_val = ecg_train.drop(187,axis=1)
test_val = ecg_test.drop(187,axis=1)
```

Low-Pass Filter: In the context of ECG Signals, it helps remove High-Frequency noise while retaining the important Low-Frequency components related to unusual heart activity.

Butter_Lowpass_Filter Function: This Function takes parameters such as the cutoff Frequency, sampling rate, and filter order, and returns the filtered signal.

```
from scipy.signal import butter, filtfilt
# Filtering for Train Values
ecg_data = ecg_train
ecg_signal = train_val
ecg_label = ecg_train_target
sampling_rate = 300
def butter_lowpass_filter(data, cutoff_freq, fs, order=5):
    freqi = 0.5*fs
    normal_cutoff = cutoff_freq/freqi
    b,a = butter(order, normal_cutoff, btype='low', analog=False)
    filt_data = filtfilt(b,a,data)
    return filt_data
```

Before Passing the Training value into Butter Lowpass Filter Function:

```
train_val
```

	0	1	2	3	4	5	6	7	8	9	...	177	178	179	180	181	182	183	184	185	186
0	0.977941	0.926471	0.681373	0.245098	0.154412	0.191176	0.151961	0.085784	0.058824	0.049020	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.960114	0.863248	0.461538	0.196581	0.094017	0.125356	0.099715	0.088319	0.074074	0.082621	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	1.000000	0.659459	0.186486	0.070270	0.070270	0.059459	0.056757	0.043243	0.054054	0.045946	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.925414	0.665746	0.541436	0.276243	0.196133	0.077348	0.071823	0.060773	0.066298	0.058011	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.967136	1.000000	0.830986	0.586854	0.356808	0.248826	0.145540	0.089202	0.117371	0.150235	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
87549	0.807018	0.494737	0.536842	0.529825	0.491228	0.484211	0.456140	0.396491	0.284211	0.136842	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
87550	0.718333	0.605000	0.486667	0.361667	0.231667	0.120000	0.051667	0.001667	0.000000	0.013333	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
87551	0.906122	0.624490	0.595918	0.575510	0.530612	0.481633	0.444898	0.387755	0.322449	0.191837	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
87552	0.858228	0.645570	0.845570	0.248101	0.167089	0.131646	0.121519	0.121519	0.118987	0.103797	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

After Passing the Training value into Butter Lowpass Filter Function:

```
# Filtering for Train Values
filt_ecg_signal
```

```
array([[ 9.78025359e-01,  9.31228125e-01,  6.72655870e-01, ...,
        -1.09241887e-08,  5.64004720e-09, -4.45968278e-10],
       [ 9.60060339e-01,  8.57851804e-01,  4.71493020e-01, ...,
         2.68216982e-06, -9.78011243e-07, -5.00657850e-07],
       [ 1.00008352e+00,  6.55272004e-01,  1.93779589e-01, ...,
        -5.08565457e-09,  3.38196512e-09, -1.34206895e-09],
       ...,
       [ 9.06053988e-01,  6.26892529e-01,  5.91530715e-01, ...,
        -1.21672499e-06,  3.25671883e-07,  4.04099652e-07],
       [ 8.58253894e-01,  6.70658402e-01,  8.00159391e-01, ...,
        -2.01228176e-07,  7.49996511e-08,  3.51241515e-08],
       [ 9.01512019e-01,  8.46941636e-01,  7.98770636e-01, ...,
        -7.81099988e-07,  4.98316305e-07, -1.74452632e-07]])
```

Further Data Preprocessing was done through Resampling and Normalization. For Resampling, we used Synthetic Minority Oversampling Technique, SMOTE for short, which addresses imbalanced datasets by oversampling the minority class by synthesizing new values from the existing values. For Normalizing our datasets, we used MinMaxScaler to transform the features of a dataset to given range, which is between 0 and 1, where the largest occurring point has the maximum and vice versa.

Model Development: The Models tested were Random Forest Classifier, XGBoost Classifier and our Original Neural Network with activation function such as Leaky ReLU, Tanh, and Sigmoid Function.

Random Forest:

Random Forest

```
[1]: from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
# Use Random Forest Regressor
# Initialize the Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=50, random_state=123)

# Train the classifier using extracted features and corresponding labels
rf_classifier.fit(ecg_train_scaled, y_res)


# Predict labels for test data
y_pred_test = rf_classifier.predict(ecg_test_scaled)

# Calculate accuracy on the test set
accuracy_test = accuracy_score(y_res_test, y_pred_test)
print("Test Accuracy:", accuracy_test)
```

```
Test Accuracy: 0.9112595209184237
```

Model Evaluation for Random Forest:

```
# Generate classification report for Random Forest
report = classification_report(y_res_test, y_pred_test)
print("Classification Report:\n", report)
```

```
Classification Report:
               precision    recall  f1-score   support

         0.0       0.74      0.99      0.85     18118
         1.0       0.99      0.82      0.90     18118
         2.0       0.93      0.95      0.94     18118
         3.0       0.98      0.82      0.89     18118
         4.0       1.00      0.98      0.99     18118

    accuracy                           0.91     90590
   macro avg       0.93      0.91      0.91     90590
weighted avg       0.93      0.91      0.91     90590
```

XGBoost Classifier:

Model building before Hyperparameter Tuning:

XGBoost Classifier

```
import xgboost as xgb
from sklearn.metrics import accuracy_score, classification_report
```

```
model1 = xgb.XGBClassifier(
    objective = 'multi:softmax',
    max_depth=5,
    learning_rate=0.01,
    n_estimators=100
)
```

Model Evaluation before Hyperparameter Tuning:

```
predict = model1.predict(filt_ecg_signal_test)
```

```
xgb_accur = accuracy_score(ecg_test_target, predict)
print(f"XGBoost Accuracy: {xgb_accur:.5f}")
```

```
XGBoost Accuracy: 0.83962
```

```
xgb_report=classification_report(ecg_test_target,predict)
print(xgb_report)
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.83      0.90     18118
         1.0       0.29      0.77      0.42       556
         2.0       0.68      0.85      0.76      1448
         3.0       0.11      0.88      0.19       162
         4.0       0.77      0.94      0.85      1608

    accuracy                           0.84     21892
   macro avg       0.57      0.86      0.62     21892
weighted avg       0.92      0.84      0.87     21892
```

Model Building After Hyperparameter Tuning:

```
# After Hyperparameter Tuning

model3 = xgb.XGBClassifier(
    objective = 'multi:softmax',
    max_depth=7,
    learning_rate=0.2,
    n_estimators=100
)
```

Model Evaluation after Hyperparameter Tuning:

```
predict2 = model3.predict(filt_ecg_signal_test)
```

```
xgb_accur = accuracy_score(ecg_test_target, predict2)
print(f"XGBoost Accuracy: {xgb_accur:.5f}")
```

```
XGBoost Accuracy: 0.96656
```

```
xgb_report2=classification_report(ecg_test_target,predict2)
print(xgb_report2)
```

```
              precision    recall  f1-score   support

         0.0       0.99      0.97      0.98     18118
         1.0       0.63      0.80      0.71       556
         2.0       0.91      0.94      0.92      1448
         3.0       0.55      0.83      0.66       162
         4.0       0.98      0.98      0.98      1608

    accuracy                           0.97     21892
   macro avg       0.81      0.90      0.85     21892
weighted avg       0.97      0.97      0.97     21892
```

Our Neural Network:

Model Building:

```
]: # Creating another Neural Network with different Activation Function
   # Create the neural network model
   model2 = Sequential()
   model2.add(Dense(64, activation = 'leaky_relu', input_shape=(X_train2.shape[1],)))
   model2.add(Dense(32, activation='tanh'))
   model2.add(Dense(1, activation='sigmoid'))

   # Compile the model
   model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

   # Train the model
   model2.fit(X_train2, y_train2, epochs=15, batch_size=32, validation_data=(X_test2, y_test2))

   # Evaluate the model
   loss, accuracy = model2.evaluate(X_test2, y_test2)
   print(f'Test accuracy: {accuracy:.2f}')
```

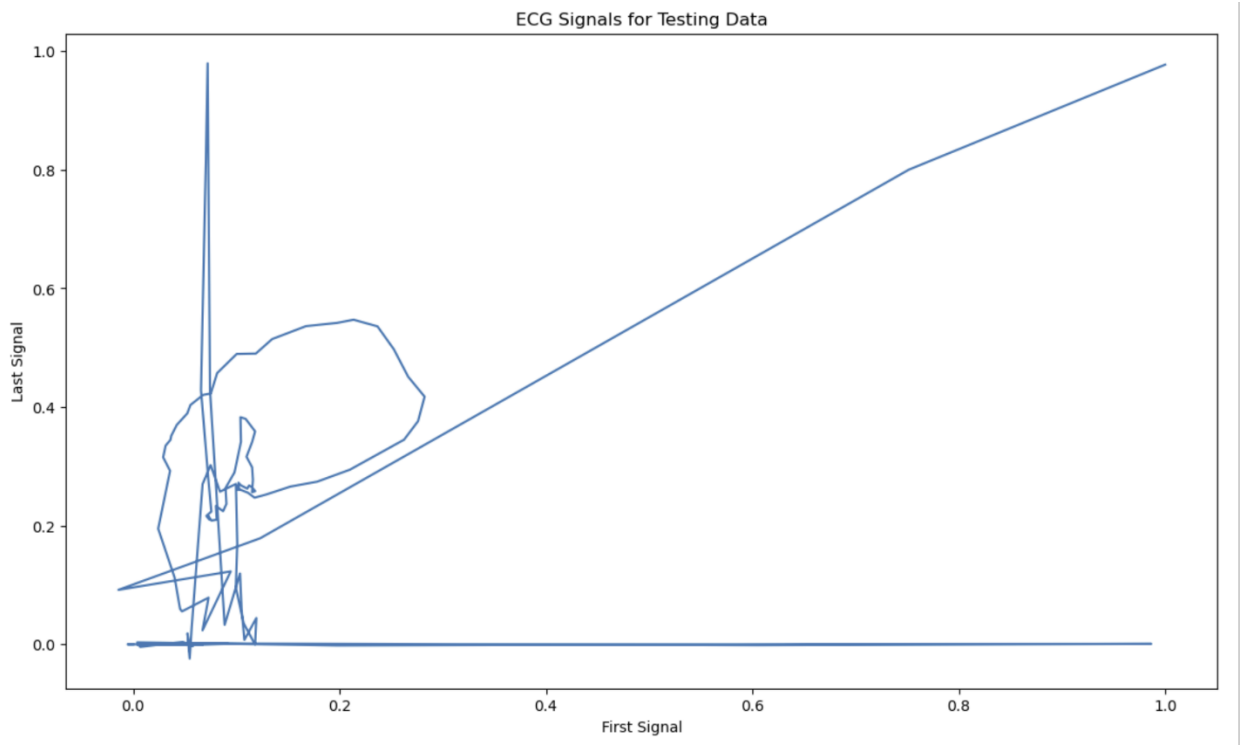
```
Epoch 1/15
548/548 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.7636 - loss: 0.0000e+00 - val_accuracy: 0.8207 - val_loss: 0.0000e+00
Epoch 2/15
548/548 ━━━━━━━━━━━━━━━━━━━━ 0s 764us/step - accuracy: 0.8149 - loss: 0.0000e+00 - val_accuracy: 0.8219 - val_loss: 0.0000e+00
Epoch 3/15
548/548 ━━━━━━━━━━━━━━━━━━━━ 0s 775us/step - accuracy: 0.8222 - loss: 0.0000e+00 - val_accuracy: 0.8148 - val_loss: 0.0000e+00
Epoch 4/15
548/548 ━━━━━━━━━━━━━━━━━━━━ 0s 766us/step - accuracy: 0.8157 - loss: 0.0000e+00 - val_accuracy: 0.8201 - val_loss: 0.0000e+00
Epoch 5/15
548/548 ━━━━━━━━━━━━━━━━━━━━ 0s 774us/step - accuracy: 0.8212 - loss: 0.0000e+00 - val_accuracy: 0.8175 - val_loss: 0.0000e+00
Epoch 6/15
548/548 ━━━━━━━━━━━━━━━━━━━━ 0s 768us/step - accuracy: 0.8172 - loss: 0.0000e+00 - val_accuracy: 0.8194 - val_loss: 0.0000e+00
Epoch 7/15
548/548 ━━━━━━━━━━━━━━━━━━━━ 0s 776us/step - accuracy: 0.8176 - loss: 0.0000e+00 - val_accuracy: 0.8171 - val_loss: 0.0000e+00
Epoch 8/15
548/548 ━━━━━━━━━━━━━━━━━━━━ 0s 779us/step - accuracy: 0.8215 - loss: 0.0000e+00 - val_accuracy: 0.8187 - val_loss: 0.0000e+00
Epoch 9/15
548/548 ━━━━━━━━━━━━━━━━━━━━ 0s 801us/step - accuracy: 0.8266 - loss: 0.0000e+00 - val_accuracy: 0.8228 - val_loss: 0.0000e+00
Epoch 10/15
548/548 ━━━━━━━━━━━━━━━━━━━━ 0s 773us/step - accuracy: 0.8201 - loss: 0.0000e+00 - val_accuracy: 0.8182 - val_loss: 0.0000e+00
Epoch 11/15
548/548 ━━━━━━━━━━━━━━━━━━━━ 0s 768us/step - accuracy: 0.8234 - loss: 0.0000e+00 - val_accuracy: 0.8107 - val_loss: 0.0000e+00
Epoch 12/15
548/548 ━━━━━━━━━━━━━━━━━━━━ 0s 778us/step - accuracy: 0.8154 - loss: 0.0000e+00 - val_accuracy: 0.8219 - val_loss: 0.0000e+00
Epoch 13/15
548/548 ━━━━━━━━━━━━━━━━━━━━ 0s 772us/step - accuracy: 0.8234 - loss: 0.0000e+00 - val_accuracy: 0.8153 - val_loss: 0.0000e+00
Epoch 14/15
548/548 ━━━━━━━━━━━━━━━━━━━━ 1s 954us/step - accuracy: 0.8194 - loss: 0.0000e+00 - val_accuracy: 0.8182 - val_loss: 0.0000e+00
Epoch 15/15
548/548 ━━━━━━━━━━━━━━━━━━━━ 0s 780us/step - accuracy: 0.8212 - loss: 0.0000e+00 - val_accuracy: 0.8228 - val_loss: 0.0000e+00
137/137 ━━━━━━━━━━━━━━━━━━━━ 0s 617us/step - accuracy: 0.8300 - loss: 0.0000e+00
Test accuracy: 0.82
```

Evaluation for NN Model:

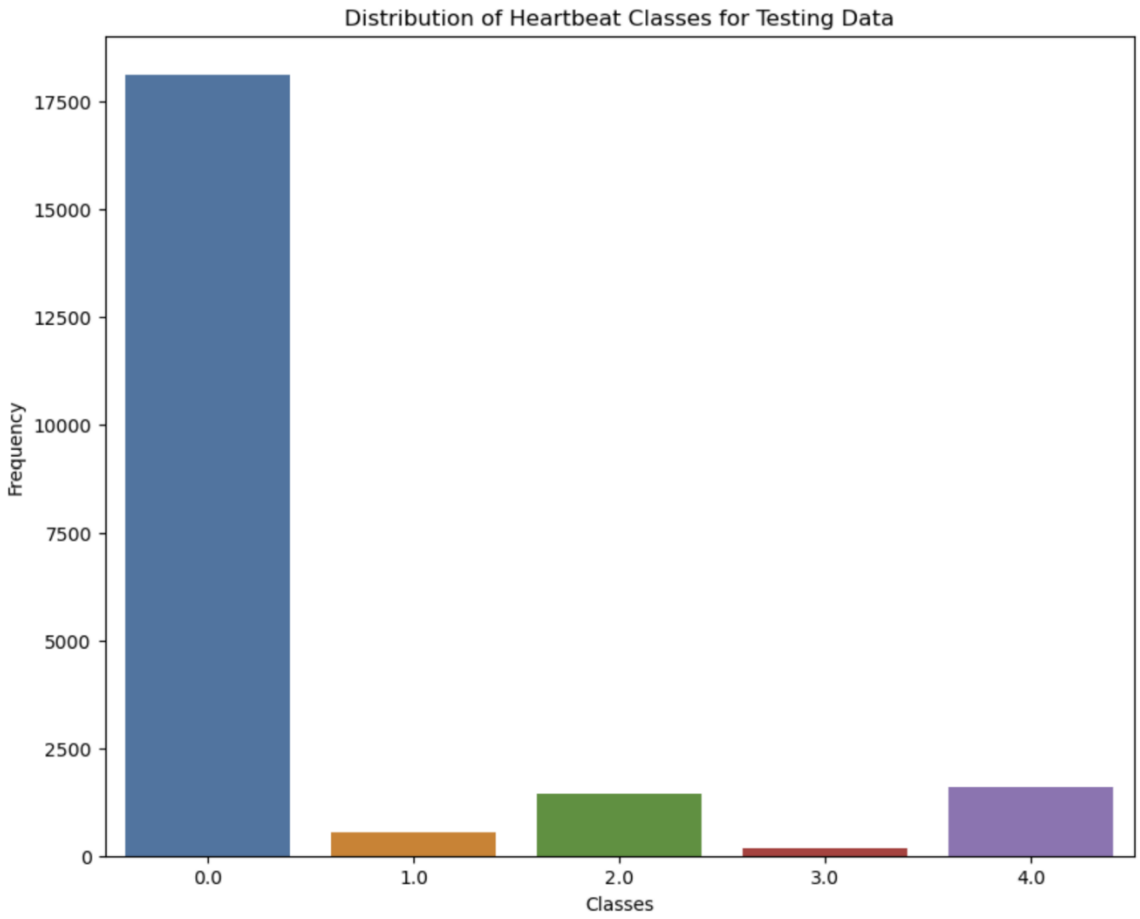
```
]: # Evaluate the model
y_pred = (model2.predict(X_test2) > 0.5).astype("int32")

# Generate the classification report
report = classification_report(y_test2, y_pred)

# Print the classification report
print(report)
```

```
137/137 ━━━━━━━━━━━━━━━━━━━━ 0s 886us/step
              precision    recall  f1-score   support

         0.0       0.95      0.99      0.97      3625
         1.0       0.02      0.09      0.03       106
         2.0       0.00      0.00      0.00       289
         3.0       0.00      0.00      0.00        35
         4.0       0.00      0.00      0.00       324

    accuracy                           0.82      4379
   macro avg       0.19      0.22      0.20      4379
weighted avg       0.79      0.82      0.80      4379
```

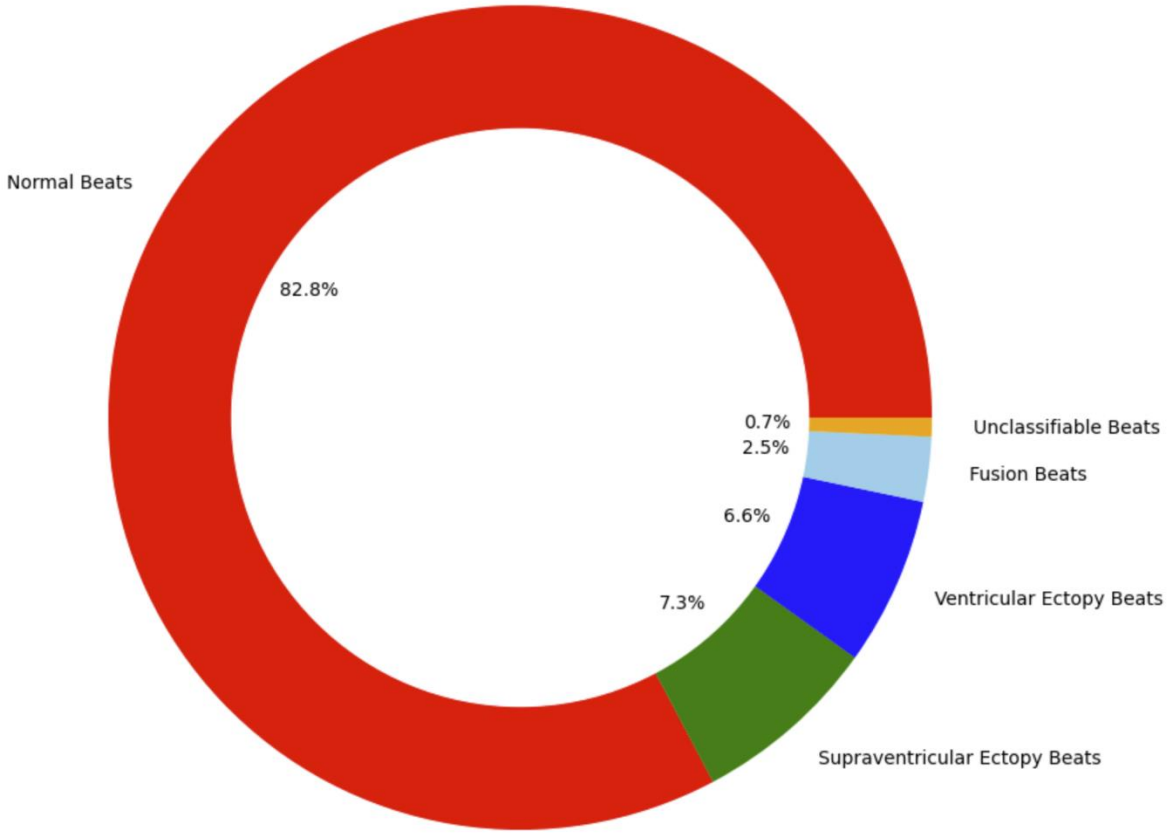
Visualization:



This graph represents the relation between the first ECG Signal and the Last ECG Signal, which seems to be going in a loop between 0.0 and 0.2 and after that they both seems to have a linear relationship.



These are the different classes for heartbeats for the test data where “Normal Beats” has the highest frequency and the one with lower frequency is “Fusion Beats” .



For the above Pie Chart, most of the space has been covered by “Normal beats” and the least space has been taken by “Unclassifiable Beats”.

Conclusion:

The result of the XGBoost Classifier has been the best with the accuracy of 96.66% and it even has a great recall value of 97% , which is a crucial factor for detecting unusual heartbeats through ECG signals. This model can help us identify the Electrical Signals responsible for Arrhythmia and it can be bolstered more in the future to deal with even larger dataset that comes from real life. Although, the accuracy from our original Neural Network Model has not been the best by giving an accuracy of 83%, but with a better computer system and a Higher GPU, which my computer lacks for now, could result in a better accuracy with a high performing computer.

References:

[1] https://www.frontiersin.org/articles/10.3389/fncom.2020.564015/full

[2] https://ieeexplore.ieee.org/document/4359186

[3] https://www.kaggle.com/code/ismaelelhussein/ecg-classification-ml-dl-comparative-study