CSM 6420- Machine Learning for Intelligent System Assignment -2

Dabir Hasan Rizvi (dhr8)

Abstract—The electrocardiogram is a vital tool for detecting and monitoring cardiovascular disease (ECG). Cardiovascular disease (CVD) is the biggest cause of death in the globe. In some cases, the patient's ECG is continually monitored to detect various arrhythmic abnormalities. Atrial fibrillation (AF), in instance, is the most common cardiac arrhythmia, with a prevalence of 1-2 percent in the general population, and it can be fatal if left untreated. Essentially, I want to use two machine learning approaches to classify each recording in the challenge database into one of four categories: normal, AF, other, and noise [1]. The first is a feature-based approach (Random Forest Classifier), and the second is an end-to-end approach (Convolution Neural Network).

I. INTRODUCTION

One of the current issues in the field of cardiac computing is detecting atrial fibrillation (AF) from electrocardiogram (ECG) recordings. The AliveCor device was used to capture ECG data, which were then made accessible for the assignment [2]. On the Physionet Challenge server, an open database containing 188 selected features from 13062 single lead ECG segments and their annotations was utilised for feature-based training. The databases contained four types of ECG recordings: atrial fibrillation (A), normal sinus rhythm (N), other rhythms (O), and noisy recordings (~). A closed database of 4000 ECG recordings was used for testing.

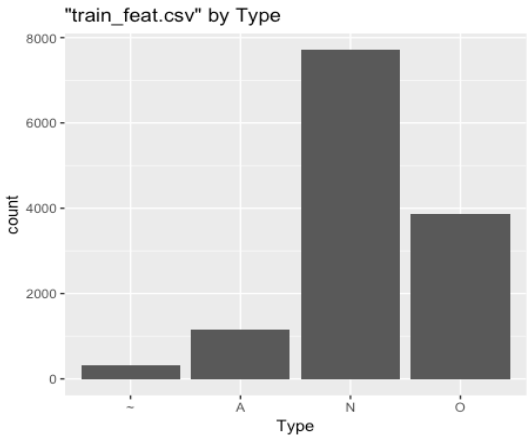


Figure 1: Class distribution of feature training dataset.

The training dataset's class distribution is indicated in Figure 1. It represents a dictionary with the key being the class value (i.e., ~, A, N, O) and the count being the number of ECG segments generated for the 'Type' value in the training dataset. To explore the data, I first described the dataset and observed various numerical factors such as mean, minimum, maximum and standard deviation values of the dataset. I checked the data types, and I observed the values such as 'Type' was an object, and the ECG readings were either in int64 or float64 which could lead to problem later and I needed to standardize. Then I used seaborn library to make statistical graphics. Seaborn was used to graphically check the null values of each column in the data set and to deal with outliners to eliminate them further.

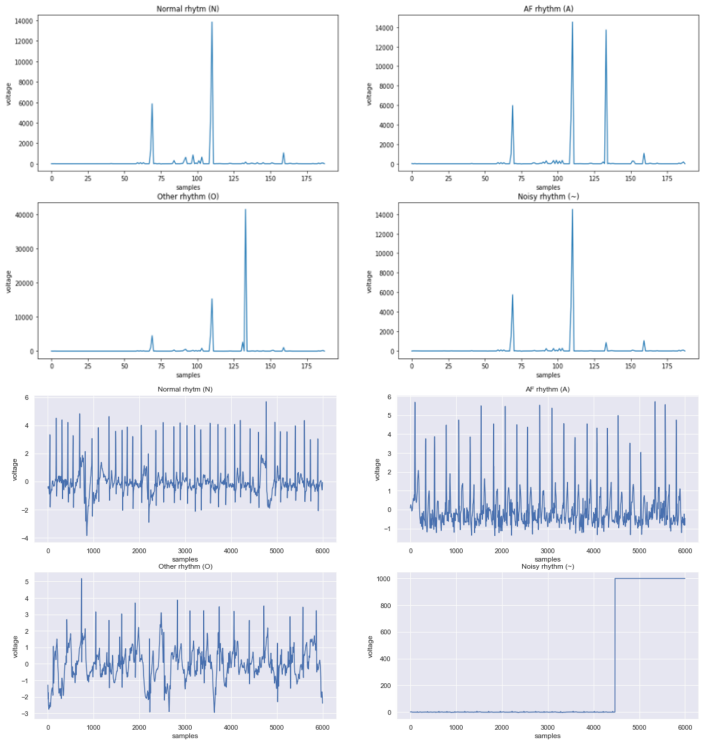


Figure 2: Types of ECG recording and their sample distribution in feature data (above) and signal data (below)

Figure 2 shows the various types of samples utilised in the training dataset, with the 188-feature generated from the voltage value of ECG readings.

II. PRE-PROCESSING DATA

Pre-processing data is a technique for converting the raw data into a usable and efficient format. Pre-processing was used to do the following tasks [3]:

A. Data Cleaning

It is done to manage the irrelevant and missing part in the data.

- Missing Data: This situation arises as the training and testing dataset had few missing values. To observe this, I used seaborn library and check for null values, and I ran a loop across the columns of the data to check for the missing data and to further replace them if found.
- Outliners: An outlier is a value that deviates abnormally from other values in a population's random sampling. I was able to observe outliners for each column graphically with the use of seaborn library and replaced them with arbitrary values.
- Noisy data: Machines cannot analyze noisy data because it is meaningless. It can be caused by poor data collecting, data input problems, etc. Extracted features of ECG Signals recorded using non-medical grade equipment with a single lead are naturally noisy. Due to breathing, a low frequency component is also present in the data set. The signals can be substantially corrupted by the user's movement during recording, the sensor device's voltage fluctuation, or inappropriate contact between the subject's body and the sensor electrodes. Before feature extraction and classification, it is necessary to locate and discard the noisy regions of the signal. Binning method or the organized data was split and by comparing each segment, a trend was observed, and data was replaced with its mean value. Other techniques to reduce a noisy data including clustering the similar type data of data into a group, outliners are then observed, or a linear/multiple regression function is used to make the data smoother [4].

B. Data Transformation

This is done to convert the data into appropriate forms suitable for further process.

- Normalization - This is done to scale the values to a particular range, and it was observed with the outliners to get rid of inconsistent and abnormal data and replace them. I used NumPy library to convert all the inconsistent numerical values in the training data type from either int 64 or float 64 to float 32.
- Attribute Selection – It chooses a new attribute from the available options to aid the cleaning process even more.
- Discretization – Interval levels or conceptual levels replace raw numerical values in data. This method could be used to improve data transformation.
- Hierarchy – This is used to convert hierarchy of a data from lower to higher. For example: A city can be converted into country. As our data did not have any hierarchy. This technique didn't suit the ECG data set [3].

C. Dimensionality Reduction

Data mining is a technique for dealing with large amounts of data. We employ a data reduction technique to make it easier. When dealing with large amounts of data, analysis becomes more difficult. Its goal is to improve storage efficiency while lowering data storage and analysis expenses. I investigated possible linear metrics that are noise-resistant to retrieve the underlying information present in ECG records [5].

- Overfitting occurs when the number of features in a dataset equals or exceeds the number of observations stored in the dataset. It is vital to use either regularization or dimensionality reduction techniques. Regularization can help minimize the possibility of overfitting. The dimensionality of a dataset is equal to the number of variables used to represent it. Employing Feature Extraction techniques alternatively can provide additional benefits, such as:
    - Improvements in accuracy.
    - Reduced possibility of overfitting
    - Increase the training speed.
    - Data visualization is improved.
    - The model's explainability significantly improves.

Feature Extraction is a method for decreasing the number of features in a dataset. It generates new features from existing ones and then discarding the original features. From a combination of the original set, a summarized version of the original features can be generated. The original set of features should then be able to summarize most of the information in the new reduced set of features. Feature selection is a method for reducing the number of features in a dataset. Feature Selection differs from Feature Extraction as it seeks to rank the importance of existing features in the dataset. It rejects the less important features. I opted to partition the data into features (X) and labels (y) before feeding it into the Machine Learning models. Following that, I partitioned the input data into train and test sets, and then train and test using a Random Forest Classifier [5].

Some better techniques for feature extraction include:

- Principle Components Analysis (PCA) - PCA is a linear dimensionality reduction approach. PCA is an unsupervised learning technique, it is unconcerned about data labels and is only interested in variation. When we use PCA, we take our original data as input and aim to discover a combination of input features that can best summarize the original data distribution such that its original dimensions are reduced. By looking at pair wised distances, PCA can maximize variances while decreasing reconstruction error [5].
- Independent Component Analysis (ICA) - Independent Component Analysis is a linear dimensionality reduction method that uses a combination of independent components as input data and attempts to correctly identify each of them removing all the unnecessary noise. If both their linear and nonlinear dependences are zero, two input features can be called independent [5].
- Linear Discriminant Analysis (LDA) - Linear Discriminant Analysis is a machine learning classifier and supervised learning dimensionality reduction approach. LDA attempts to achieve the distance between each classes mean while minimizing the spread within that class [5].

III. Methodology

A. Feature-Based Approach

When creating a predictive model, feature selection is the process of minimizing the number of input variables. To model an agent's competency, Feature Based Modelling employs attribute value machine learning algorithms by developing a model that describes the links between the characteristics of the agent's actions and the circumstances in which they are carried out. Most prior attempts to model agent competencies aimed to create process models, or models of the internal processes that hide those competences. Alternatively, feedback agent modelling depicts an agent's abilities without attempting to characterize the internal processes that result in those capabilities.

The models created can be used to directly map the relationships between the agent's inputs and outputs. Feature Based Modelling is a type of input-output agent modelling based on attribute-value machine learning. The producing models of agent competences with high predictive accuracy is possible within realistic computing limits. In the domain of elementary subtraction, FBM has shown to be highly accurate in predicting students' precise responses. It is thus possible to employ FBM in an interactive situation since FBM models can be changed in computational time spans measured in CPU seconds. As a result, FBM can be used in an interactive situation [6].

Random forest is a supervised machine learning algorithm that is commonly used to solve classification and regression problems. It creates decision trees from various samples, using the majority vote for classification and the average for regression. One of the most essential characteristics of the Random Forest Algorithm is it can manage data sets including both continuous and categorical variables in regression and classification. For classification difficulties, it produces superior results [7].

To distinguish four different types of ECG rhythms, I used a random forest classifier. Because individual decision trees tend to overfit, I employed bootstrap aggregated (bagged) decision trees, which aggregate the predictions from an ensemble of decision trees. The random forest algorithm uses a random subset of features at each decision split, which aim to minimize decision tree correlation. A hyperparameter is a control parameter for the learning process. Hyperparameter optimization or tuning is the challenge of selecting a collection of appropriate hyperparameters for a learning algorithm in machine learning. A hyperparameter can answer questions about the degree of polynomial feature that can be used in the model, maximum depth to be used for decision tree, etc. Cross-validation is used to avoid overfitting the classification model by training it on equal-size subsamples of the training data and testing it on subsample. Cross-validation is a method of testing machine learning models that involves training various models on subsets of the available data input and assessing them on the complementary subset. Other classifiers (Support vector machine, discriminant analysis, decision trees) were used to compare the classification results, but the random forest classifier produced by far the best classification accuracy [8]. For defining the model, the parameters that I used are:

- n_estimators – **1200** – the number of trees in the forest.
- criterion – **'gini'** – This is tree specific, based on the tree classifier I developed, and it is a function to measure the quality of a split.
- max-depth – **None** – Denotes the maximum depth of tree. I choose default settings because I want to nodes to expand until all leaves are pure.
- min_samples_split, min_samples_leaf – **default** – I choose the default setting as it denotes the minimum number of samples used to split an internal node and leaf node respectively.
- max_features – **sqrt** – Represents the number of features that can be considered for the best split.
- Bootstrap – **false** - Bootstrap samples are used when building trees. The whole dataset is used to build each tree when false. It is used for random forest classifier.
- random_state – **30 –** Controls randomness of the bootstrapping of the samples used when building the tree. Sampling of features is also done when looking for best split. I chose 30 features at random for each node.

I used default configurations for other parameters as they didn't influence the model's configuration and this setting yield a great result for me with an accuracy of roughly (~83% using random forest classifier).

B. End-to-End Approach

Convolutional and Recurrent Neural Networks are examples of traditional deep supervised learning algorithms. CNNs have the limitations that it operates on grid-like structures. Because of qualities like translation invariance, parameter sharing, and sparse connectivity, CNNs are particularly popular in the field of computer vision [9].

The introduction of Residual Networks was a recent advancement streamlined training and enhanced accuracy of deeper Convolutional Neural Networks (ResNet). ResNets render feature maps from shallower layers available at later stages by using shortcut identity connections. Rajpurkar et al. [10] used a 34-layer ResNet to classify 30-second single-lead ECG segments into 14 separate categories. This approach takes raw ECG segments as input and outputs classifications without the need for hand-engineered features.

I proposed an ECG classification approach based on convolutional neural networks for end-to-end approach in this assignment. Convolution layers are added below the input layer and above the hidden and output layers in CNNs. Figure 1 shows the Deep convolutional neural network employed in this assignment, which includes a single convolution layer followed by a fully connected network to output layer. Each filter is convolved with the input before being activated nonlinearly (sigmoid) to yield k feature maps of size (m q + 1) x 1. The suggested CNN accepts n stacked beat vectors of length m as input. The convolutional layer will have k filters (or kernels) of size q x n, where q is less than the input vector's length [1].

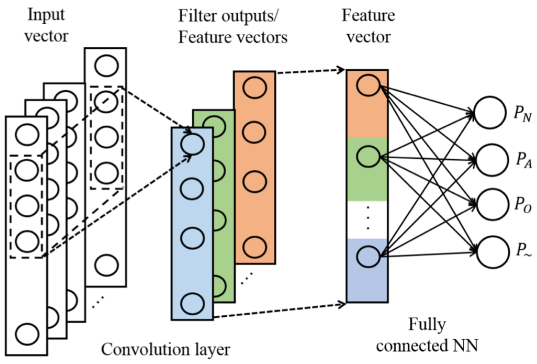


Figure 3: CNN Architecture [1]

Multiple filter results are then combined to create a single feature vector. We utilised a single fully linked layer with SoftMax activation after the convolution layer. The layers in a densely connected multilayer neural network are equivalent to those in a regular multilayer neural network. We used an architecture inspired by keras.io.

After each recording has been pre-processed, the signals are separated into segments and fed into a two-dimensional deep learning convolutional neural network (CNN).

A CNN's conventional architecture consists of components like:

- Convolution Layer: A CNN's basic building block is the convolution layer. Most of the computationally difficult lifting is done by this layer. A 2 -Dimensional Convolution Layer is used for this model. Feature maps are used to organize the convolution layers. Convolution's main goal is to extract characteristics from the input ECG signals.
- Activation Function: The activation function is used to map nonlinearity into data. The Rectified linear unit (ReLu) [is employed as a layer for activation function in this assignment.
- Pooling Layer: Pooling, also known as down sampling, is a network feature and computational complexity reduction procedure. In this assignment, the max-pooling operation is used. A 2- Dimensional Pooling layer is used for this model. Max-pooling reduces the size of the feature map by only producing the maximum number in each kernel.
- Kernel and Filter Size: Keras Conv2D is a two-dimensional convolution layer that generates a tensor of outputs by winding a convolution kernel with the layers input. The dimensions of the kernel are determined by this parameter. This value must be an odd value. It specifies the height and width of the 2D convolution window as an integer or a tuple/list of two numbers. Kernel size (5) also refers to the filter size (32) that convolve around the feature map and increases gradually, whereas stride regulates how the filter convolve around the feature map. The stride is the amount by which the filter slides.
- Padding: The padding parameter of the Keras Conv2D class has two options: 'valid' or 'same.' By setting the value to ‘same’, I kept the volume's spatial dimensions such that the output volume size matched the input volume size.
- Training: In this analysis, the Adam optimizer is used with a batch size of 64. These parameters are adjusted to provide the best results. All other parameters were set to default for this approach [10].

IV. Results and Conclusion

This assignment tackles the difficult task of consistently detecting abnormal cardiac rhythms in a large taxonomy of rhythms. The training dataset included single lead, noisy ECG recordings that lasted only a few seconds. The classification accuracy of ECG recordings contaminated with artefacts and noise was enhanced using a hybrid strategy of time, frequency, linear, and nonlinear feature extraction techniques. In both the temporal and frequency domains, we analysed various parameters to characterize the behaviour of ECG rhythms [8].

As seen in figure 4, a confusion matrix is plotted for the random forest classifier. Which is a summary of prediction results on a classification problem. A random forest classifier is trained over the remaining 70% of the training data utilizing decision trees and random feature selection at each node. Then, at every node, the entropy measure is utilised to choose which feature to split on. I employed a hybrid framework model in this assignment, in which I integrated base-level and meta-level features to build hybrid feature vectors, which were then put into a single learning algorithm to classify. The purpose of this assignment was to underline the importance of correctly classifying acceptable recordings. F1-scores were derived from only the four primary classes to limit the effective weight allocated to noisy recordings. As a result, the overall F1 score was calculated as follows:

Overall F_1-score is given by:

$$F1 = 2\frac{p \cdot r}{p + r} \text{ where } p = \frac{tp}{tp + fp}, \quad r = \frac{tp}{tp + fn}$$

F_{1N} was calculated for Normal sinus rhythm, $F_{1\sim}$ for noise segments and F_{1A}, F_{1O} were calculated for AF and Other rhythms, respectively. F_1 scores obtained from the training dataset is seen in figure 4. The findings imply that the categorization model is generalizable and does not overfit.

	precision	recall	f1-score	support
A	0.83	0.77	0.80	812
N	0.86	0.93	0.89	5405
O	0.78	0.69	0.73	2700
~	0.64	0.39	0.49	227
accuracy			0.83	9144
macro avg	0.78	0.70	0.73	9144
weighted avg	0.83	0.83	0.83	9144

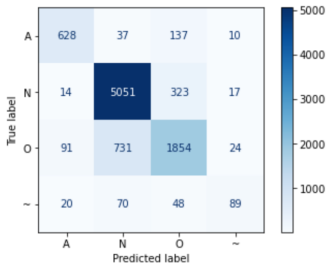
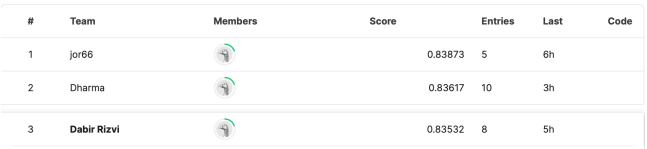


Figure 4: Confusion matrix and classification report

The result uploaded in the Kaggle challenge is shown in figure 5. Where I achieved 83.5232% accuracy for test sample of 1200 cases.



#	Team	Members	Score	Entries	Last	Code
1	jor66		0.83873	5	6h	
2	Dharma		0.83617	10	3h	
3	**Dabir Rizvi**		0.83532	8	5h	

Figure 5: Ranking for Kaggle Challenge.

I provided an end-to-end approach using Convolution Neural Network to solve this assignment. I tweaked the convolutional layer's weights to extract class-specific features that increase the underlying classifier's performance. The solution's strength is in learning several classes' discriminative features. Additionally, the low-complexity execution makes it ideal for real-time applications. To categorize rhythms from brief ECG segments, I presented a comparison of a feature-based versus a deep learning strategy. Deep learning algorithms can categorize brief ECG records, according to the results. More research is needed to improve the predictive capacity of other classes, which could help to improve the overall F1-score.

REFERENCES

[1] B. S. Chandra, C. S. Sastry, S. Jana, and S. Patidar, "Atrial fibrillation detection using convolutional neural networks," in 2017 Computing in cardiology (CINC), 2017, pp. 1–4.

[2] J. A. Behar, A. A. Rosenberg, Y. Yaniv, and J. Oster, "Rhythm and quality classification from short ECGs recorded using a mobile device," in 2017 Computing in Cardiology (CinC), 2017, pp. 1–4.

[3] Deepak Jain, "Data Pre-processing in Data Mining" Geeks for geeks. Retrieved from: https://www.geeksforgeeks.org/data-preprocessing-in-data-mining/

[4] S. Datta et al., "Identifying normal, AF and other abnormal ECG rhythms using a cascaded binary classifier," in 2017 Computing in cardiology (cinc), 2017, pp. 1–4.

[5] Pier Paolo Ippolito, "Feature extraction techniques," Towards Data Science. Retrieved from: https://towardsdatascience.com/feature-extraction-techniques-d619b56e31be

[6] G. I. Webb and M. Kuzmycz, "Feature Based Modelling: A methodology for producing coherent, consistent, dynamically changing models of agents' competencies," User modeling and user-adapted interaction, vol. 5, no. 2, pp. 117–150, 1995.

[7] Sruthi E R, "Understanding Random Forest," Analytics Vidhya. Retrieved from: https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

[8] R. Mahajan, R. Kamaleswaran, J. A. Howe, and O. Akbilgic, "Cardiac rhythm classification from a short single lead ECG recording via random forest," in 2017 Computing in Cardiology (CinC), 2017, pp. 1–4.

[9] F. Andreotti, O. Carr, M. A. Pimentel, A. Mahdi, and M. De Vos, "Comparing feature-based classifiers and convolutional neural networks to detect arrhythmia from short segments of ECG," in 2017 Computing in Cardiology (CinC), 2017, pp. 1–4.

[10] Rajpurkar P, Hannun AY, Haghpanahi M, Bourn C, Ng AY. Cardiologist-Level Arrhythmia Detection with Convolutional Neural Networks. arXiv170701836 2017.

APPENDIX

Link to all codes and explanation with all the libraries: https://github.com/dabirrizvi/CSM6420-

A. Data Exploration

For data exploration I first took the feature training data and printed some part of it in figure 5 using data.head() to understand the data clearly. Then I checked for types so that I can perform type conversion before preprocessing the data, the data was mixed randomly with some having float 64 data type and some int 64, but I needed to convert it to float 32 for smoother operation and I also looked for the missing values and null values in data using a heatmap from seaborn library as we can see in figure 6. I used boxplot, a feature of seaborn library to eliminate irrelevant or corrupt data and set a range for the data shown in figure 7. Then I visualized different types of data, the result of the code in figure 8 is shown in figure 2. Using seaborn library and matplotlib library in python, I was able to visualize the data clearly and observe the requirements and be able to process the requirements and find a solution through a graphical representation made the whole process sensible and interactive. Basic libraries such as NumPy and Pandas was also used to visualize and perform basic operations.

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns

In [2]: df = pd.read_csv('train_feat.csv')

In [3]: df.head()
```

Out[3]:

	ID	Type	F0	F1	F2	F3	F4	F5	F6	F7	...	F178	F179	F180	F181	F182	F183	F184	F185	F186	F18
0	B00000	O	-7	9	2	0	1	3	0.018661	0.033087	...	0.261922	5.374833	-0.244051	6	0	66.518847	0	67.529544	70.754717	0.93
1	B00001	O	16	0	18	1	1	3	0.340250	0.542241	...	0.334393	11.151970	2.338377	8	0	66.518847	2	94.637224	153.061225	1.28
2	B00002	O	-27	27	0	0	0	0	0.065454	0.024133	...	0.118206	6.579651	1.270140	14	0	92.592593	0	108.794198	122.950820	0.68
3	B00003	N	-24	24	0	0	0	0	0.015118	0.008164	...	0.043204	17.917860	3.454199	2	0	80.645161	0	83.160083	86.206897	0.76
4	B00004	N	-10	10	0	0	0	0	0.015908	0.018113	...	0.183005	5.296747	-0.352397	0	0	61.983471	0	66.777963	133.928571	1.00

5 rows × 190 columns

```
In [4]: #Basic information about numerical values in the dataset
        df.describe()
```

Out[4]:

	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	...
count	13062.000000	13062.000000	13062.000000	13062.000000	13062.000000	13062.000000	13062.000000	13062.000000	13062.000000	13062.000000	...
mean	-5.010642	9.290002	4.520671	0.120655	0.573725	2.300031	0.096836	0.126552	0.132732	0.064016	...
std	13.191543	8.316621	7.053515	0.882754	1.890958	3.865684	0.112028	0.169311	0.186497	0.095972	...
min	-55.000000	0.000000	0.000000	-3.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
25%	-14.000000	2.000000	0.000000	0.000000	0.000000	0.000000	0.022209	0.019392	0.023584	0.008272	...
50%	-6.000000	7.000000	1.000000	0.000000	0.000000	1.000000	0.050370	0.042847	0.048759	0.018907	...
75%	2.000000	15.000000	6.000000	0.000000	0.000000	3.000000	0.145311	0.193612	0.163994	0.106128	...
max	48.000000	55.000000	46.000000	24.000000	29.000000	38.000000	1.352993	2.061968	2.236467	4.371038	...

Figure 5: Data Input and understanding the data

```
In [5]: #to check the types of data in the dataset and to further convert all training data into float32.
        df.dtypes
```

Out[5]:

```
ID       object
Type     object
F0        int64
F1        int64
F2        int64
          ...
F183    float64
F184      int64
F185    float64
F186    float64
F187    float64
Length: 190, dtype: object
```

```
In [6]: #To check null values of few column

        cols=df.columns
        sns.heatmap(df[cols].isnull())
```

Figure 6: Checking types of data and null values using seaborn library.

```
In [7]: #To deal with outliners if we want for any column
        sns.boxplot(x=df.F0)
        #df=df[df['F0']<50]

Out[7]: <AxesSubplot:xlabel='F0'>
```

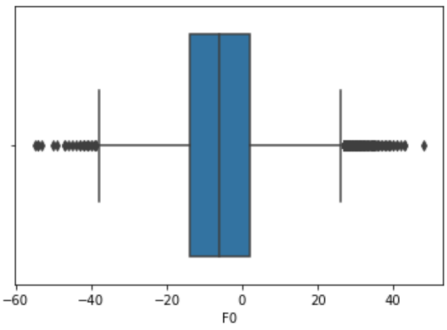


Figure 7: To check for outliners

```
In [98]: # To check different type of samples

         fig, axes = plt.subplots(2, 2, figsize = (20,10))

         sns.lineplot(x = samples, y = x_0, ax = axes[0][0])
         axes[0][0].set_title("Normal rhytm (N)")
         axes[0][0].set_ylabel("voltage")
         axes[0][0].set_xlabel("samples")
         sns.lineplot(x = samples, y = x_1, ax = axes[0][1])
         axes[0][1].set_title("AF rhythm (A)")
         axes[0][1].set_ylabel("voltage")
         axes[0][1].set_xlabel("samples")
         sns.lineplot(x = samples, y = x_2, ax = axes[1][0])
         axes[1][0].set_title("Other rhythm (O)")
         axes[1][0].set_ylabel("voltage")
         axes[1][0].set_xlabel("samples")
         sns.lineplot(x = samples, y = x_3, ax = axes[1][1])
         axes[1][1].set_title("Noisy rhythm (~)")
         axes[1][1].set_ylabel("voltage")
         axes[1][1].set_xlabel("samples")
```

Figure 8: Visualizing the data to understand better.

B. Data Cleaning

For Data Cleaning as we can see in figure 9, I first dropped duplicate values if found in the data. Then I replaced all infinite and null values for better accuracy. Then I checked for any missing values in the dataset. Then I converted all the ECG readings to float 32 data type to be used for training purpose when modelling.

```
In [90]: # To drop if duplicate rows are used
         df=df.drop_duplicates()

In [91]: #Replace all infinite and null values
         df.replace([np.inf, -np.inf], np.nan, inplace=True)
         df.fillna(999, inplace=True)

In [92]: # Classify the signals
         print("The numbers of ECG signals: {} and each signal has: {} samples".format(df.shape[0], df.shape[1] - 2))
         print("The categorical classes of ECG: {}".format(df["Type"].unique()))
         print("Numbers of signals for each class")
         print(df["Type"].value_counts())

         The numbers of ECG signals: 13062 and each signal has: 188 samples
         The categorical classes of ECG: ['N' 'A' 'O' '~']
         Numbers of signals for each class
         N    7721
         O    3857
         A    1160
         ~     324
         Name: Type, dtype: int64

In [93]: # To check for missing columns for training data
         missing_data = df.isnull()
         columns_missing_data = []
         for column in df.columns.values.tolist():
           column_missing_data = missing_data[column].value_counts()
           if len(columnn_missing_data) == 2:
             print(column)
             print(columnn_missing_data)
             print("")
             columns_missing_data.append(column)

         if len(columns_missing_data) == 0:
           print("none of the columns have missing values")
         else:
           print("These columns have missing data:")
           print(columns_missing_data)

         none of the columns have missing values
```

Figure 9: Replacing and finding irrelevant and missing values in the dataset.

C. Modelling / Training

The modelling part for feature-based approach is shown in figure 10 and 12 where I first split the data and applied the random forest classifier to train the data. I also build a decision tree (Figure 11) and did a grid search on the data as seen in figure 14. The classifier and the submission file format for Kaggle challenge was done in figure 13, to predict the accuracy and the result of the challenge is shown in figure 5.

```
In [95]: # To split the training and testing data
         df.drop(columns = ["ID", "Type"], axis = 1, inplace = True)
         X = np.asanyarray(df)

In [96]: #Print X array
         X

Out[96]: array([[ -1.        ,   5.        ,   4.        , ...,  72.11538462,
                   81.52173913,   0.96      ],
                [ 10.        ,   1.        ,  11.        , ...,  72.94832827,
                   89.28571429,   0.984     ],
                [-16.        ,  16.        ,   0.        , ...,  68.33712984,
                   70.75471698,   0.904     ],
                ...,
                [-18.        ,  18.        ,   0.        , ...,  64.        ,
                   66.96428571,   0.984     ],
                [-16.        ,  16.        ,   0.        , ...,  68.80733945,
                   70.75471698,   0.896     ],
                [  5.        ,   1.        ,   8.        , ...,  72.46376812,
                  105.6338028 ,   1.24      ]])

In [100]: # Converting all training data to float32 type
          X = X.astype(np.float32)

In [101]: #Splitting the training and testing data and printing its shape
          from sklearn.model_selection import train_test_split

          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.7, random_state = 21, stratify=y)
          print(X_train.shape, y_train.shape)
          print(X_test.shape, y_test.shape)

          (3918, 188) (3918,)
          (9144, 188) (9144,)
```

Figure 10: Splitting the training and testing data

```
In [118]: # Building Decision Tree
          from sklearn.tree import DecisionTreeClassifier
          dt = DecisionTreeClassifier(criterion = 'gini', random_state = 30)
          dt.fit(X_train, y_train)
          dt_pred_train = dt.predict(X_train)

In [119]: import graphviz
          from sklearn.tree import export_graphviz
          from IPython.display import SVG

          def visualise_tree(treeclf):
              dot = export_graphviz(treeclf, filled=True, rounded=True)
              graph = graphviz.Source(dot)
              display(SVG(graph.pipe(format="svg")))

In [ ]: # Your code here
        #DecisionTreeClassifier has got an attribute "estimators_", which is the list of individual decision trees in the fo
        #Randomly pick two trees to visualise.
        visualise_tree(rfc.estimators_[100])
        visualise_tree(rfc.estimators_[120])
```

Figure 11: Building a decision tree from training and testing data

```
# Create a random forest classifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, plot_confusion_matrix
rfc = RandomForestClassifier(n_estimators=1200, random_state=30, bootstrap = False, criterion='gini')
rfc.fit(X_train, y_train)
y_test_pred = rfc.predict(X_test)

# Check the confusion matrix
disp = plot_confusion_matrix(rfc, X_test, y_test,
                             display_labels=None,
                             cmap=plt.cm.Blues)

# To get a classification report.
print(classification_report(y_test, y_test_pred))
```

Figure 12: Creating a random forest classifier

```
#SVM
from sklearn.svm import SVC

clf=SVC(probability = True, C=2.57)

clf.fit(X,y.ravel())

proba=clf.predict(testing_data.values)
results=pd.DataFrame(proba,columns=["Predicted"])
results.index=testing_data.index.values
results.index.names=["ID"]
results.to_csv("submission2.csv")
```

```
#RANDOM FOREST
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=1200, random_state=30, bootstrap = False, criterion='gini')

clf.fit(X,y.ravel())

proba=clf.predict(testing_data.values)
results=pd.DataFrame(proba,columns=["Predicted"])
results.index=testing_data.index.values
results.index.names=["ID"]
results.to_csv("submissionRF2.csv")
```

Figure 13: Performing Random Forest Classifier and SVM and extracting the result for submission.

```
In [116]: # Hyper parameter tuning using grid searchGrid search
          # Utility function to report best scores
          def report(results, n_top=3):
              for i in range(1, n_top + 1):
                  candidates = np.flatnonzero(results['rank_test_score'] == i)
                  for candidate in candidates:
                      print("Model with rank: {0}".format(i))
                      print("Mean validation score: {0:.3f} (std: {1:.3f})"
                            .format(results['mean_test_score'][candidate],
                                    results['std_test_score'][candidate]))
                      print("Parameters: {0}".format(results['params'][candidate]))
                      print("")

          from sklearn.model_selection import GridSearchCV
          # use a full grid over all parameters
          param_grid = {'n_estimators': [1,100], "max_depth" : [2, 8]}

          # run grid search
          grid_search = GridSearchCV(rfc, param_grid=param_grid, cv=10)
          grid_search.fit(X_train, y_train)
          report(grid_search.cv_results_)
```

```
Model with rank: 1
Mean validation score: 0.812 (std: 0.012)
Parameters: {'max_depth': 8, 'n_estimators': 100}

Model with rank: 2
Mean validation score: 0.755 (std: 0.023)
Parameters: {'max_depth': 8, 'n_estimators': 1}

Model with rank: 3
Mean validation score: 0.745 (std: 0.008)
Parameters: {'max_depth': 2, 'n_estimators': 100}
```

Figure 14: Performing grid search for feature based model

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 2612, 6000, 1)]   0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 2612, 6000, 32)    832
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 1306, 3000, 32)    0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 1306, 3000, 32)    25632
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 653, 1500, 32)     0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 653, 1500, 64)     51264
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 326, 750, 64)      0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 326, 750, 64)      102464
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 163, 375, 64)      0
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 163, 375, 128)     204928
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 81, 187, 128)      0
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 81, 187, 128)      409728
_________________________________________________________________
dropout_8 (Dropout)          (None, 81, 187, 128)      0
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 81, 187, 256)      819456
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 40, 93, 256)       0
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 40, 93, 256)       1638656
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 20, 46, 256)       0
_________________________________________________________________
dropout_9 (Dropout)          (None, 20, 46, 256)       0
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 20, 46, 512)       3277312
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 10, 23, 512)       0
_________________________________________________________________
dropout_10 (Dropout)         (None, 10, 23, 512)       0
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 10, 23, 512)       6554112
_________________________________________________________________
flatten_2 (Flatten)          (None, 117760)            0
_________________________________________________________________
dense_6 (Dense)              (None, 64)                7536704
_________________________________________________________________
dropout_11 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_7 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_8 (Dense)              (None, 4)                 132
=================================================================
Total params: 20,623,300
Trainable params: 20,623,300
Non-trainable params: 0
_________________________________________________________________
```

Figure 15: Model Architecture using CNN for training signal data

```
from tensorflow.keras.optimizers import Adam


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

# Train the model, iterating on the data in batches of 64 samples
history = model.fit(X_train, y_train, epochs=15, batch_size= 64, validation_split=1/6)
```

Figure 16: Training the model using Adam

The result was submitted using Random Forest Classifier and for Neural network-based model, the architecture was built using CNN as seen in figure 15 and to train the model, Adam optimizer was used as seen in figure 16.