DETECTING HEART ANOMALIES USING HEARTBEAT SOUND

A Project as a Course requirement for
Master of Sciences in Data Science and Computing

ASHWIN PRAKASH
19227

SRI SATHYA SAI INSTITUTE OF HIGHER LEARNING
(Deemed to be University)

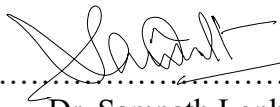Department of Mathematics and Computer Science
Muddenahalli Campus

APRIL 2021

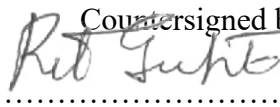## *CERTIFICATE*

This is to certify that this Project / Dissertation titled **Detecting Heart anomalies using Heart beat sound** submitted by **ASHWIN PRAKASH**, 19227, Department of Mathematics and Computer Science, MUDDENAHALLI CAMPUS is a bonafide record of the original work done under my supervision as a Course requirement for the Degree of Master of Sciences in Data Science and Computing.

………………………………..
Dr. Sampath Lonka
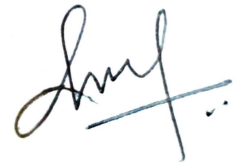Project / Dissertation Supervisor

Countersigned by

…………………………………...
Dr. Rita Gupta
Head of the Department

Place: Muddenahalli

Date: April 20, 2021

DECLARATION

The Detecting Heart anomalies using Heart beat sound was carried out by me under the supervision of Dr. Sampath Lonka, Department of Mathematics and Computer Science, Muddenahalli Campus. It has been carried out as a course requirement for the degree of Master of Sciences in Data Science and Computing. And has not formed the basis for the award of any degree, diploma or any other such title or any other university.
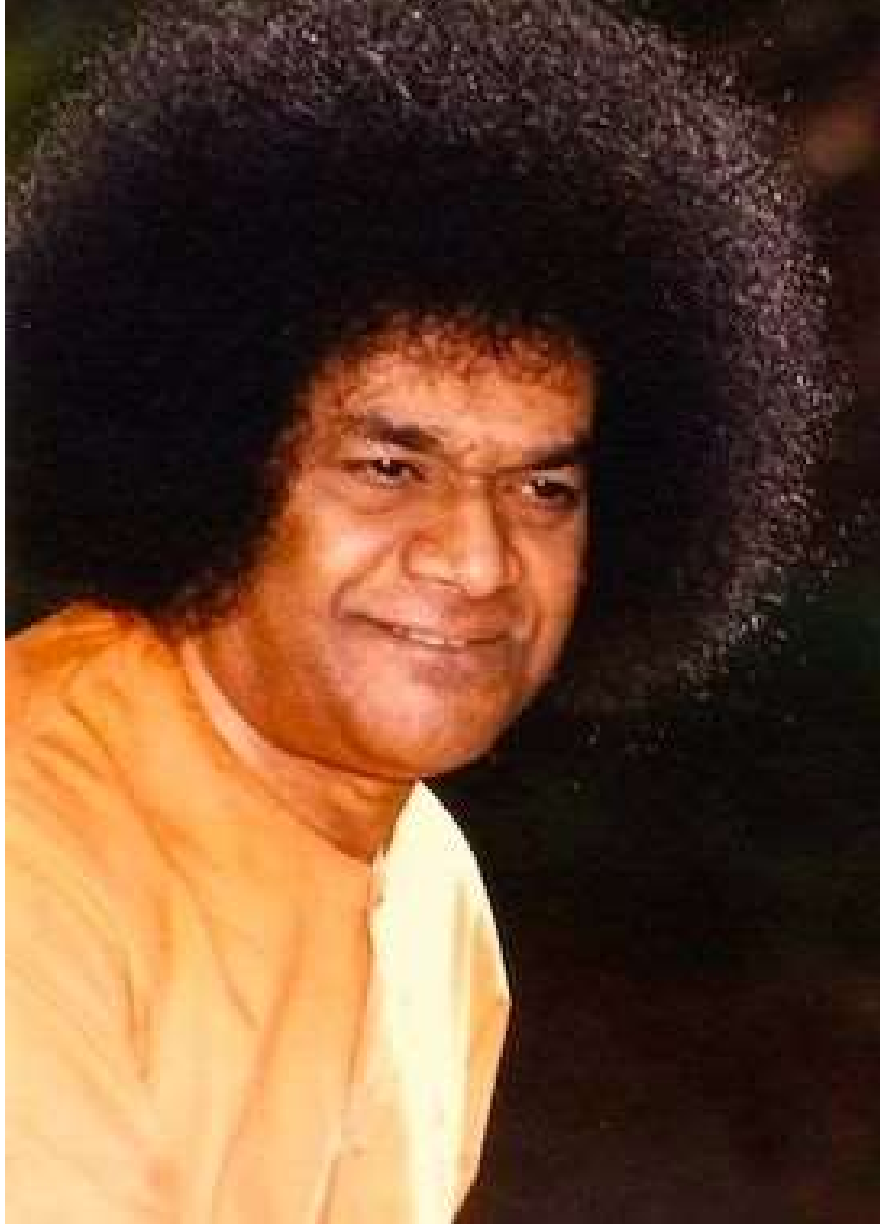
…………………..

Sri Ashwin Prakash
19227
II MDSC
Muddenahalli Campus

Place: Chennai

Date: 15-04-2021

DEDICATED AT THY LOTUS FEET

ACKNOWLEDGEMENT

I.   TABLE OF CONTENTS

II. ABSTRACT

Lub dub Lub dub.. The heart is a very important organ in our body that keeps beating 24/7 pumping out blood for us. It takes in impure blood and pumps out pure blood to be circulated around the Body.

The noise produced by the heart when pumping blood is called Heart beats. In 2004, 17.1 million people died from CVDs, accounting for 29% of all deaths worldwide. Coronary heart disease was responsible for an estimated 7.2 million of these deaths. Any approach that aids in the detection of heart attack symptoms can be extremely beneficial.

Machine learning researchers are particularly interested in the problem because it requires the classification of audio sample data, and distinguishing between groups of interest is difficult. Data is collected in real-world environments and often includes various types of background noise.

Differences in heart sounds that lead to different heart symptoms can be very subtle and difficult to distinguish. Classifying this type of data successfully necessitates the use of extremely robust classifiers. Despite its medical importance, machine learning and deep learning are yet to be used in this application.

1. INTRODUCTION

The only thing that differentiates a human brain with a computer is the ability to THINK and take decisions accordingly. Differences in heart sounds that lead to different heart symptoms can be very subtle and difficult to distinguish. Classifying this type of data successfully necessitates the use of extremely robust classifiers. Despite its medical importance, machine learning and deep learning are yet to be used in this application. Cardiovascular disorder is one of the leading causes of death in the world. The number of deaths caused by CVD (Cardiovascular Diseases) is steadily increasing, and this trend is projected to continue. Advanced procedures for detecting and treating CVDs are still in demand.

This has the potential to revolutionise the way heart diseases are detected an identified, affecting countless lives as well as the healthcare industry. The aim of this project is to provide a dependable and effective process in which the first step of screening can be done at hospitals by doctors using automated stethoscopes, as well as at home by patients using a mobile computer and an app to record the Heart Sounds.

This research shows how to process audio data and model Machine Learning and Deep Learning algorithms using various techniques. The information gathered is not free of environmental noise.

The discrimination between different classes of audio is not easy, and it necessitates the use of sophisticated Machine Learning and Deep Learning strategies to deal with the discriminators. Despite its importance in the medical world, this is an underdeveloped domain in the field of Machine Learning and Deep Learning.

This Project was taken up by Sri. MVSSS Durgesh (19230) and myself as a group project under the guidance of our Professor. Dr. Sampath Lonka.

The project Code is available on my GitHub page [1] .

2. IMPLEMENTATION ON A SAMPLE DATASET

Firstly, we worked on a sample dataset called Taxi route and implemented a few models on it.

1. The summary of the work is given below
2. We received a dataset named Taxi routes -Data.csv [2] having 12 features and 29,999 records.
3. We segregated all the features and Performed Data Cleaning.
4. All the Data inconsistencies were changed and Data was brought into a useable format.
5. All the features were extracted in order to model them.
6. The models applied were
- Decision Trees
- Ada Boost
- Other Ensemble methods.
7. Conclusion: Relatively good predictions are made on the given dataset. Many assumptions were made on the dataset and hyper parameter tuning is done with the models to come to this conclusion. We used the most appreciated algorithms in the ML scientific group to get to these conclusions.

3. SIMPLE WORKFLOW AND EVALUATION METRICS

Heartbeat Sound Signal → Preprocessing → Feature Extraction → Classification Model → Accuracy
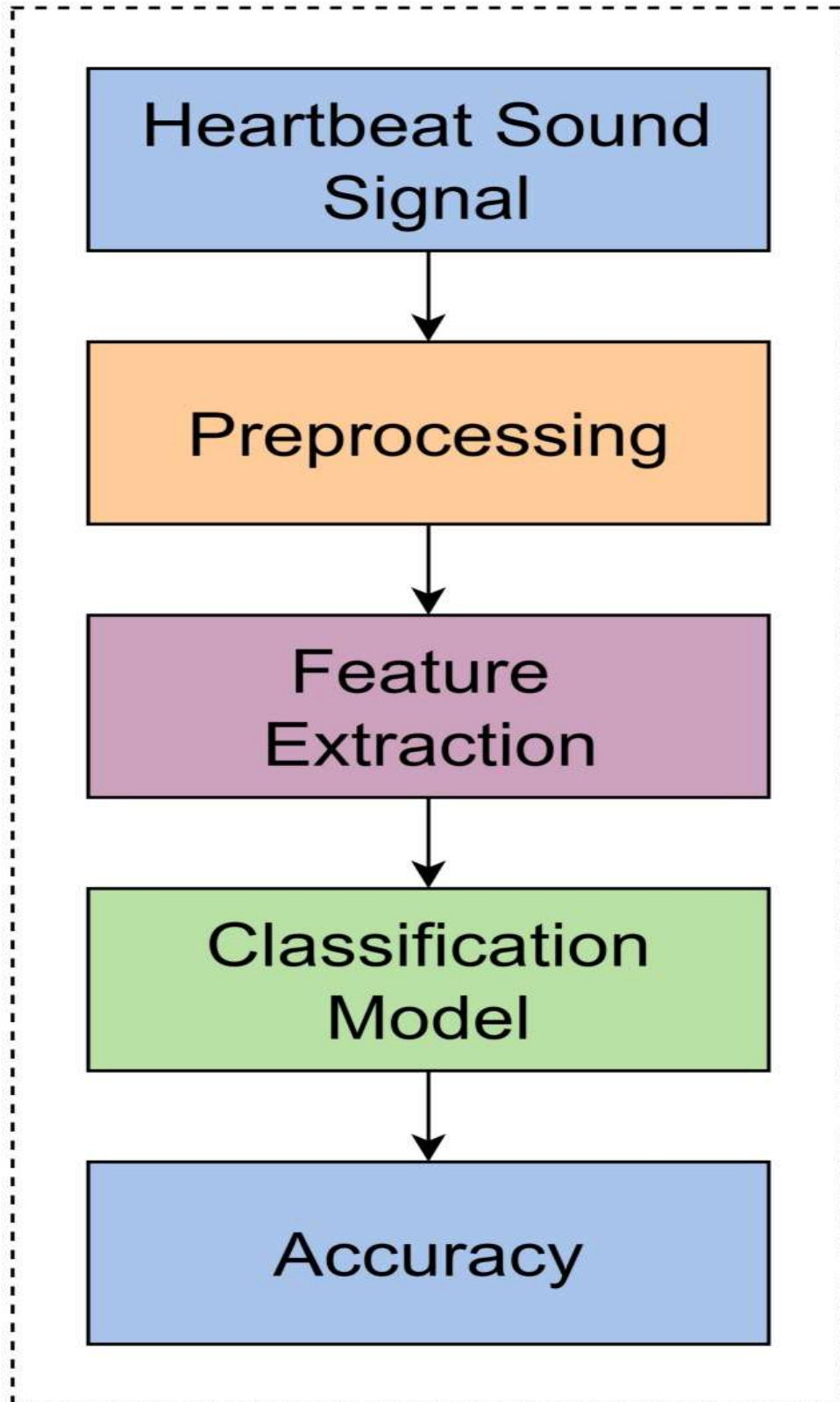
The project begins with audio files being read. After that, the audio files are processed to remove certain key features. After extracting the functions, we model the data to suit the Machine Learning and Deep Learning models. We transfer the data to some predefined computer algorithms in the fourth level. We also develop our own CNN and LSTM models from scratch. The best algorithm is then chosen based on the various metrics we described earlier. We use several web integration frameworks to merge the concept with the web tool in the next step. In the final level, we deploy it on Heroku's free tier so that it can be accessed from anywhere.

3.1. EVALUATION METRICS

Evaluation metrics are metrics that are used to quantify the accuracy and reliability of the model. The model built is a classification model. Accuracy score is the most common metric used for classification problems.

Initial model selection is performed using accuracy score, in addition to other metrics like Precision, Recall and F1 Score.

3.1.1. CONFUSION MATRIX:

Confusion matrix is a visualization matrix used in classification problems. The value in each element of the matrix corresponds to the number of predictions that fall into a specific category. Each element in that matrix is represented with a pair of numbers (i,j) where i is the row number and j is the column number. Number i corresponds to the actual class value and number j corresponds to the predicted class value.

3.1.2. ACCURACY:

Accuracy is defined as the ratio of correctly predicted observation by total number of observations.

$$Accuracy = \frac{\text{Correct Predictions}}{\text{Total Predictions}}$$

3.1.3. PRECISION:

Precision score of a class 'A' is the ratio of correctly predicted observations to all the observations that are predicted as class 'A'. From the confusion matrix, the

precision score of ith class is the value of ith element of ith column divided by the sum of all the elements of ith column.

3.1.4. RECALL:

Recall score of a class 'A' is the ratio of correctly predicted observations to all the observations that belong to class 'A'. From the confusion matrix, the recall score of ith class is the value of ith element of ith row divided by the sum of all the elements of ith row.

3.1.5. F1 SCORE:

F1 score is the weighted average of Precision and Recall. F1 score is more useful than accuracy score when the classes are not balanced.

$$\text{F1 score} = 2 \ * \ \frac{\text{Recall} * \text{Precision}}{\text{Recall} + \text{Precision}}$$

4. LITERATURE REVIEW

4.1. TECHNOLOGY

The Python programming language is the primary technology used in this project. The entire programme is written in the Python programming language. This is due to the fact that Python has a large number of common libraries that can be used at different levels of a Data Science project pipeline. The whole programme is run on a Mi laptop with an Intel i3 10th generation processor and 8GB of RAM. The laptop is equipped with a INTEL® UHD Graphics. Deep learning models include the use of a GPU. Jupyter notebooks or VScode editors are used to write all of the code. We use Flask and Heroku cloud for deployment of the model.

Image Source: https://medium.com/unitechie/deploying-a-flask-application-to-heroku-b8814beaa954

4.2. DATASET

This Dataset [3] was created for a machine learning task that involved classifying heartbeat sounds. The data came from two places: (A) the general public through the iStethoscope Pro iPhone app, and (B) a hospital clinic trial using the DigiScope digital stethoscope.

On classification, The value Counts are:

```
data['type'].value_counts()

normal      563
murmur      265
artifact    120
extrastole   64
extrahls     37
Name: type, dtype: int64
```
```
data['type'].value_counts()

normal      351
murmur      129
extrastole   46
artifact     40
extrahls     19
Name: type, dtype: int64
```

TRAIN DATA   TEST DATA

This information was gathered for a machine learning task to classify heartbeats. Audio files of various lengths were obtained from two separate sources and are included in the data. There are five different types of heartbeats in the results. They are:

- Normal:
    There are natural, safe heart sounds in this category. We should expect heart rates ranging from 50 to 140 beats per minute because the data were gathered through age ranges.

- Murmur:
    Between the "lub" and "dub" of heart beats, a heart murmur can sound like a whooshing noise. They may be signs of some very severe heart conditions.

- Extra Heartbeat:
    An extra heart rhythm is a "lub" or "dub" in the heartbeat. They don't have to be deadly, but they can be dangerous under certain circumstances. Detecting this kind of heartbeat will be very beneficial to both the patient and the doctor.

- Artifact:

  There may be a wide range of sounds captured when capturing heartbeats in this category. When capturing the pulse signal, they may have anything from echos to music and noise. If this is accurately identified, the doctor will be instructed to retrieve the pulse once more.

- Extrasystole:

  These types of sounds may happen on occasion, and the heart rate can miss a beat. It's likely that an additional or missed "lub" or "dub" will appear. It does not have to be a symptom of an illness, since it is a natural occurrence in both adults and infants. However, any excess systole may be the cause of heart failure.

Audio files are processed in two different ways. They are:

- Unclipped audio file:

  All the above-mentioned features are extracted for complete audio file without clipping it into specific duration. The features are then stored in a separate csv file.

- Clipped audio file:

  Audio file is cut into several 3 seconds audio clips. The above-mentioned audio features are extracted for each 3 second audio clips. Each 3 second audio clip becomes a separate audio file and has a separate row in the csv file. This is done because some Deep Learning models do not accept variable length input. Hence, audio has to be clipped to fixed length to use the deep learning models.

```
data.shape

(1049, 36)
```

```
data.shape

(585, 36)
```

4.3. LIBRARIES USED

We used the following libraries in the project.

4.3.1. NUMPY

Numpy is a Python library for manipulating multidimensional vectors and their operations. Numpy arrays are 50 times faster than Python's built-in lists. This is why, wherever possible, numpy objects are used. The version of numpy used in this project is 1.18.5.

4.3.2. PANDAS

Pandas is a Python library that allows you to work with large datasets. It has well-defined functions for data cleaning, analysis, and exploration. Data is stored in DataFrames and Series, two well-defined artefacts. It has a well-defined layout that makes it very user-friendly. The version of Pandas used in this project is 1.1.4.

4.3.3. LIBROSA

Librosa is a Python package for music and audio processing. It provides the required building blocks for constructing frameworks for retrieving music information. The librosa library is used in this project to extract low-level features such as zero crossings and spectral centroid, as well as high-level features such as MFCC. For this project, we're using version 0.8.0.

4.3.4. SCIKIT LEARN

Scikit Learn is a Python library that includes a number of machine learning algorithms as well as other data modelling techniques. In this project, sklearn is heavily used during the data modelling stage. For this project, we're using version 0.23.2.

4.3.5. TENSOR FLOW 2

TensorFlow is an open-source machine learning tool that runs from start to finish. It has a large, scalable ecosystem of software, databases, and community resources that enable researchers to advance the state-of-the-art in machine learning and developers to quickly create and deploy Machine Learning applications. Version 2.3.2 is used in this project.

4.3.6. OS

In Python, the OS module has features for communicating with the operating system. Python's basic utility modules include OS. This module allows you to use operating system-dependent features on the go. The OS module is mostly used to read audio files from different directories.

4.4. FEATURE EXTRACTION

The extraction of features is a critical step in analysing and discovering relationships between various objects. Since the audio data generated by the models cannot be explicitly interpreted by the models, function extraction is used to translate it into a format that can be understood.

4.4.1. ZERO CROSSING RATE

The rate of sign shifts along a signal, i.e., the rate at which the signal changes from positive to negative or back, is known as the zero crossing rate. Both speech recognition and music information retrieval have made extensive use of this capability.

4.4.2. SPECTRAL CENTROID
It is measured as the weighted mean of the frequencies present in the sound and shows where the sound's "centre of mass" is located.

4.4.3. SPECTRAL ROLLOFF
The frequency below which a certain proportion of the overall spectral energy is lost is known as spectral rolloff.

4.4.4. MFCC — Mel-Frequency Cepstral Coefficients

MFCC stands for Mel Frequency Cepstral Coefficient. It is a scaling feature that scales frequencies that humans can perceive more clearly. Humans can identify the change of pitch at lower frequencies than at higher frequencies. This MFCC uses Discrete Cosine Transformation and log transformation in frequency domain so that it scales down to what humans can clearly identify the differences of the sound. For this data set, we extracted 30 MFCC values for each window of audio data.

4.5. REDUCING FEATURE SPACE OF MFCC

MFCCs are a critical aspect of audio files. However, they are so large that a basic machine learning algorithm would be incapable of handling them. As a result, we can use Auto encoders to reduce the registered MFCC's feature space. The MFCCs that are extracted are 130 X 30 pixels in size.

This gives it a total of 3900 features. As compared to the number of samples obtained, this is a large number. As a result, it's critical that we use certain strategies to reduce the feature space. Auto encoders have shown to be extremely useful in reducing the size of feature spaces of very wide dimensions.

We are reducing the feature space from 3900 to 30 dimensions using an auto encoder. Our machine learning models will be used to validate these reduced dimensional space features

```
Model: "encoder"

Layer (type)                     Output Shape              Param #
=================================================================
encoder_input (InputLayer)       [(None, 130, 30, 1)]      0

encoder_conv_layer_1 (Conv2D     (None, 130, 30, 32)       320

encoder_relu_1 (ReLU)            (None, 130, 30, 32)       0

encoder_bn_1 (BatchNormaliza     (None, 130, 30, 32)       128

encoder_conv_layer_2 (Conv2D     (None, 65, 15, 64)        51264

encoder_relu_2 (ReLU)            (None, 65, 15, 64)        0

encoder_bn_2 (BatchNormaliza     (None, 65, 15, 64)        256

encoder_conv_layer_3 (Conv2D     (None, 65, 15, 64)        102464

encoder_relu_3 (ReLU)            (None, 65, 15, 64)        0

encoder_bn_3 (BatchNormaliza     (None, 65, 15, 64)        256

encoder_conv_layer_4 (Conv2D     (None, 65, 15, 64)        102464

encoder_relu_4 (ReLU)            (None, 65, 15, 64)        0

encoder_bn_4 (BatchNormaliza     (None, 65, 15, 64)        256

encoder_conv_layer_5 (Conv2D     (None, 65, 15, 64)        102464

encoder_relu_5 (ReLU)            (None, 65, 15, 64)        0

encoder_bn_5 (BatchNormaliza     (None, 65, 15, 64)        256

flatten_6 (Flatten)              (None, 62400)             0

encoder_output (Dense)           (None, 30)                1872030
=================================================================
Total params: 2,232,158
Trainable params: 2,231,582
Non-trainable params: 576
```

The encoder reduces the input's size, and the decoder attempts to reconstruct it using the reduced function space. As a result, it's a very adaptable unsupervised model. The user has the option of reducing the input to a certain number of measurements. As we can see, the auto encoder has two models in it, encoder model and decoder model. Encoder is giving an output of 60 neurons. It means that the encoder is condensing 3900 dimension feature space into 60 dimension feature space. This is a great reduction in the number of dimensions. We are also avoiding the curse of dimensionality. This autoencoder has a total of 8,322,045 parameters. Number of dimensions has been reduced by 65 times but this reduction also has an error associated with it. This autoencoder is trained using mean squared error. We can only identify the loss when we actually build a machine learning model using this latent space.

```
Model: "decoder"

Layer (type)                          Output Shape              Param #
==================================================================
decoder_input (InputLayer)            [(None, 30)]              0

decoder_dense (Dense)                 (None, 62400)             1934400

reshape_2 (Reshape)                   (None, 65, 15, 64)        0

decoder_conv_transpose_layer          (None, 65, 15, 64)        102464

decoder_relu_1 (ReLU)                 (None, 65, 15, 64)        0

decoder_bn_1 (BatchNormaliza          (None, 65, 15, 64)        256

decoder_conv_transpose_layer          (None, 65, 15, 64)        102464

decoder_relu_2 (ReLU)                 (None, 65, 15, 64)        0

decoder_bn_2 (BatchNormaliza          (None, 65, 15, 64)        256

decoder_conv_transpose_layer          (None, 65, 15, 64)        102464

decoder_relu_3 (ReLU)                 (None, 65, 15, 64)        0

decoder_bn_3 (BatchNormaliza          (None, 65, 15, 64)        256

decoder_conv_transpose_layer          (None, 130, 30, 64)       102464

decoder_relu_4 (ReLU)                 (None, 130, 30, 64)       0

decoder_bn_4 (BatchNormaliza          (None, 130, 30, 64)       256

decoder_conv_transpose_layer          (None, 130, 30, 1)        577

sigmoid_layer (Activation)            (None, 130, 30, 1)        0
==================================================================
Total params: 2,345,857
Trainable params: 2,345,345
Non-trainable params: 512


Model: "autoencoder"

Layer (type)                          Output Shape              Param #
==================================================================
encoder_input (InputLayer)            [(None, 130, 30, 1)]      0

encoder (Functional)                  (None, 30)                2232158

decoder (Functional)                  (None, 130, 30, 1)        2345857
==================================================================
Total params: 4,578,015
Trainable params: 4,576,927
Non-trainable params: 1,088
```

Encoder is giving an output of 30 neurons and Decoder takes it as input and builds a matrix of the input size.

4.6. MACHINE LEARNING MODELS

We used the following Machine Learning algorithms in our project.

4.6.1. RANDOM FOREST

Random forest is a versatile, easy-to-use machine learning algorithm that, in most cases, produces excellent results even without hyper-parameter tuning. Because of its simplicity and versatility, it is also one of the most widely used algorithms. Random forest is a learning algorithm that is supervised. It creates a "forest" out of an ensemble of decision trees, which are normally educated using the "bagging" technique. The bagging method's basic premise is that combining different learning models improves the final outcome.

4.6.2. GRADIENT BOOSTING

The resulting algorithm, gradient boosted forests, outperforms random forest when a decision tree is the weak learner. It constructs the model in the same stage-by-stage manner as other boosting models, but it broadens the scope by allowing optimization of every differentiable loss function

4.6.3. STACKING

Stacked Generalization, also known as Stacking, is an ensemble machine learning algorithm. The advantage of stacking is that it can combine the strengths of a number of high-performing models to make predictions that outperform any single model in the ensemble on a classification or regression assignment.

4.6.4. NAÏVE BAYES

It's a classification method based on Bayes' Theorem and the principle of predictor freedom. A Naive Bayes classifier, in basic words, implies that the existence of one function in a class is irrelevant to the presence of any other feature.

4.6.5. SVM CLASSIFIER

The "Support Vector Machine" is a supervised machine learning algorithm that can be applied to classification and regression problems. It is, however, mostly used to solve classification problems. Each data object is plotted as a point in n-dimensional space in the SVM algorithm, with the value of each feature being the value of a specific coordinate. Then, by locating the hyper-pl, we conduct classification. Then we classify the data by locating the hyper-plane that separates the two groups.

4.7. DEEP LEARNING MODELS

We used the following Deep Learning algorithms in our project.

4.7.1. CONVOLUTIONAL NEURAL NETWORKS

Since they often provide positive outcomes, Convolutional Neural Networks (CNNs) have been extensively used in the field of audio detection and classification. The design is based on a mel-spectrogram representation of the input audio frames, and it has been shown to be accurate in environmental sound classification (ESC) with high accuracy.

4.7.2. LONG SHORT-TERM MEMORY

Long short-term memory (LSTM) is a deep learning architecture that uses an artificial recurrent neural network (RNN). LSTM has feedback relations, unlike normal feedforward neural networks. It can handle not only individual data points (such as audio files), but also entire data sequences.

4.7.3. BI-DIRECTIONAL LSTM

Bidirectional LSTMs are a type of LSTM that can be used to increase model performance in sequence classification problems. Bidirectional LSTMs train two instead of one LSTM on the input sequence in problems where all timesteps of the input sequence are visible.

4.7.4. CNN BASED AUTO ENCODERS

An autoencoder is a kind of unsupervised artificial neural network that learns efficient data coding. An autoencoder's goal is to train the network to disregard signal "noise" in order to learn a representation (encoding) for a collection of data, usually for dimensionality reduction. In addition to the reduction hand, the autoencoder learns a reconstructing side, in which it attempts to produce new data. Along with the reduction side, the autoencoder learns a reconstructing side, in which it attempts to produce a representation as similar to its original input as possible from the reduced encoding, hence its name.

4.8. PYTHON FLASK

Flask is a Python-based microweb platform. It's referred to as a microframework because it doesn't require any specific resources or libraries. Object-relational mappers, type validation, upload management, various transparent authentication technologies, and other framework-related tools all have extensions. We Use this for Front-end Deployment.

4.9. HEROKU CLOUD

Python applications can be easily deployed and scaled with Heroku. If you choose Django or Flask frameworks, Heroku allows you to create stuff the way you want with the tools you want.

5. MODEL OUTPUTS WITHOUT CLIPPING

5.1. RANDOM FOREST

How Random Forest algorithm works?
1. Randomly select **K** features from total **m** features where K << m.
2. Among the **K** features, calculate the node **d** using the best split point.
3. Split the node into daughter nodes using the best split.
4. Repeat the 1 to 3 **steps** until **l** number of nodes has been reached.

Random Forest Classifier

```
forest = RandomForestClassifier(max_depth=8,n_estimators=700, max_samples=0.7,
forest.fit(X_train, y_train)
```

```
RandomForestClassifier(class_weight='balanced', max_depth=8, max_samples=0.7,
                       n_estimators=700, oob_score=True, random_state=31)
```
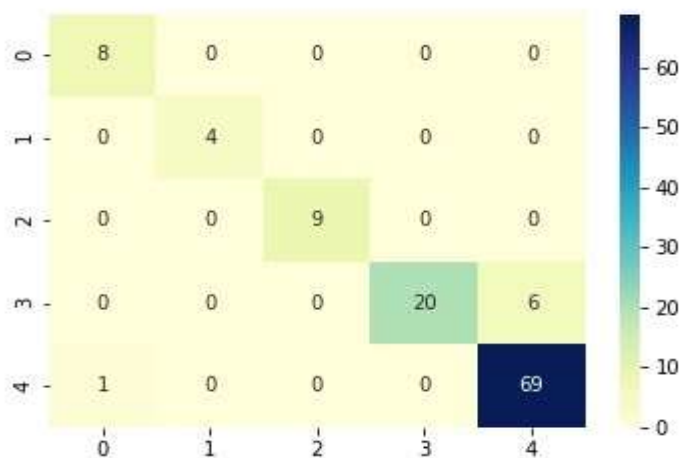
```
y_pred = forest.predict(X_test)
accuracy_score(y_test, y_pred)
```

```
0.9401709401709402
```

```
forest.oob_score_
```

```
0.9401709401709402
```

CONFUSION MATRIX

Random Forest Classifier is giving an accuracy of **94%**.

5.2. GRADIENT BOOSTING

How Gradient Boosting algorithm works?
1. Calculate the average of the target label.
2. Calculate the residuals.
3. Construct a decision tree.
4. Predict the target label using all of the trees within the ensemble.
5. Compute the new residuals.
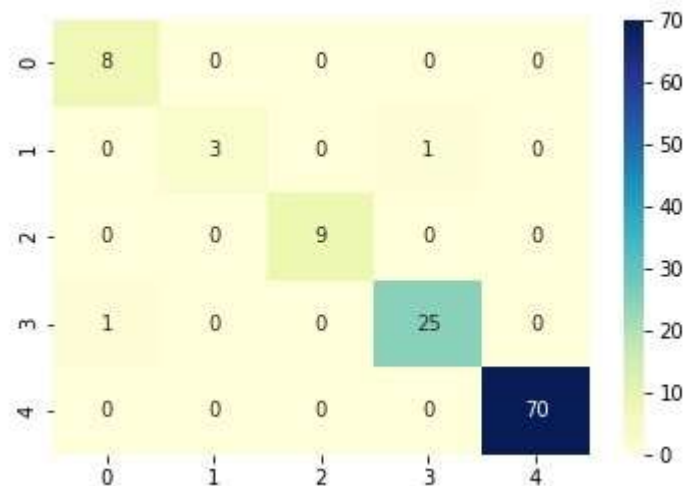
Gradient Boosting Algorithm

```
gradient = GradientBoostingClassifier(random_state=31)
gradient.fit(X_train, y_train)
```

```
GradientBoostingClassifier(random_state=31)
```

```
y_pred = gradient.predict(X_test)
accuracy_score(y_test, y_pred)
```

0.9829059829059829

CONFUSION MATRIX

The Gradient boosting classifier is giving an accuracy of **98%** . This makes this a very good model.

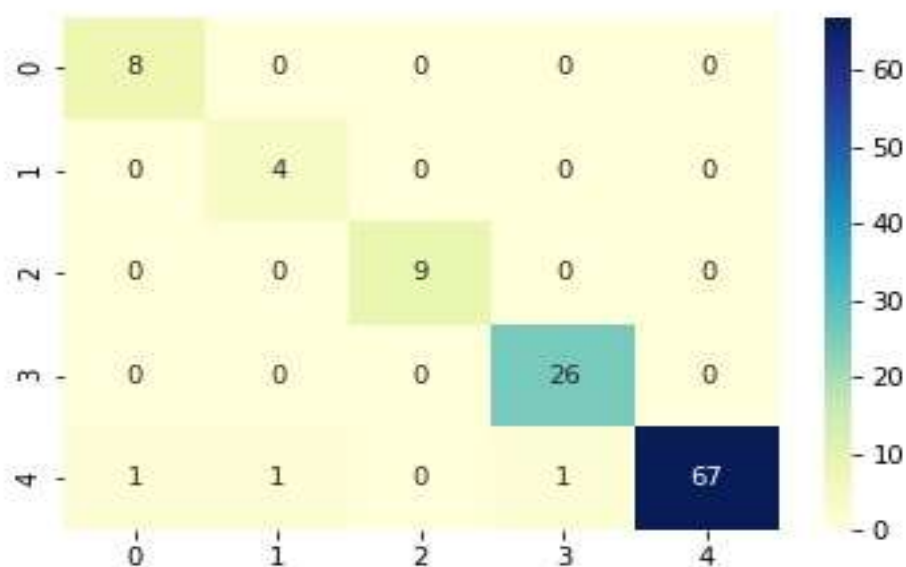## 5.3. STACKING

How Stacking algorithm works?
1. Learn first-level classifiers based on the original training data set. We have several choices to learn base classifiers.
2. Construct a new data set based on the output of base classifiers.
3. Learn a second-level classifier based on the newly constructed data set.

```
StackingClassifier(cv=5,
                   estimators=[('rf',
                                RandomForestClassifier(class_weight='balanced',
                                                       max_depth=8,
                                                       max_samples=0.7,
                                                       n_estimators=700,
                                                       oob_score=True,
                                                       random_state=31)),
                              ('gba',
                               GradientBoostingClassifier(random_state=31)),
                              ('svc',
                               SVC(C=2, class_weight='balanced',
                                   kernel='linear', probability=True,
                                   random_state=31)),
                              ('naive_bayes', GaussianNB())],
                   final_estimator=LogisticRegression(class_weight='balanced',
                                                      max_iter=1000),
                   n_jobs=4, passthrough=True)
```

```
y_pred = stacking.predict(X_test_scaled)
accuracy_score(y_test, y_pred)
```
0.9743589743589743

CONFUSION MATRIX

The Stacking classifier is giving an accuracy of **97%** .

5.4. NAÏVE BAYES

How NAÏVE BAYES algorithm works?
1. Calculate the prior probability for given class labels.
2. Find Likelihood probability with each attribute for each class.
3. Put these values in **Bayes** Formula and calculate posterior probability.
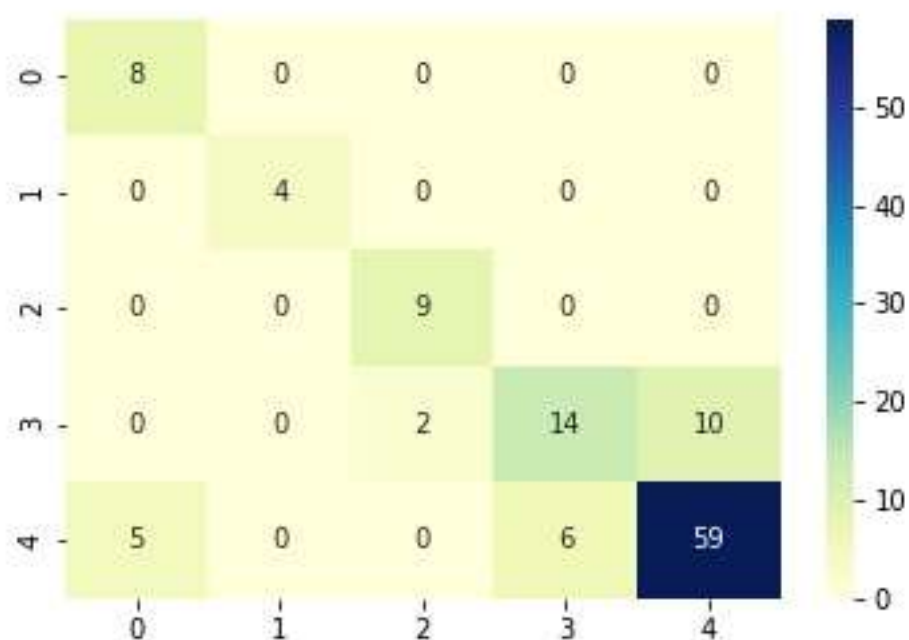
Gaussian Naive Bayes

```
gaussian_nb = GaussianNB()
gaussian_nb.fit(X_train_scaled, y_train)
```

GaussianNB()

```
y_pred = gaussian_nb.predict(X_test_scaled)
accuracy_score(y_test, y_pred)
```

0.8034188034188035

CONFUSION MATRIX

The Naïve Bayes classifier is giving an accuracy of **80%**.

5.5. SVM CLASSIFIER

How SVM CLASSIFIER algorithm works?
1. Each data object is plotted as a point in n-dimensional space in the SVM algorithm, with the value of each feature being the value of a specific coordinate.
2. Then, by locating the hyper-plane, we conduct classification.
3. Then we classify the data by locating the hyper-plane that separates the two groups.
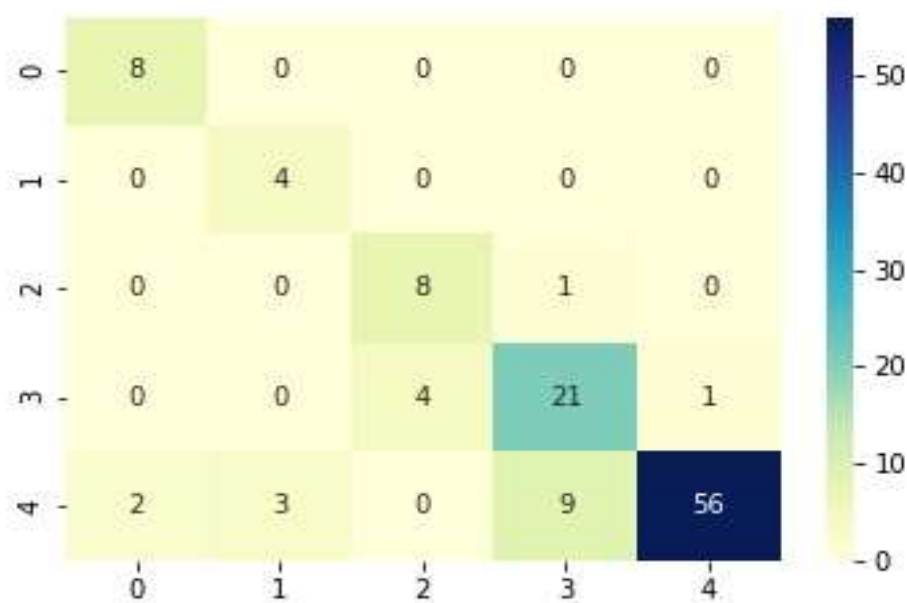
Support Vector Classifiers

```
svc = SVC(kernel = 'linear', C=2, probability=True, random_state=31, class_weight='balanced')
svc.fit(X_train_scaled, y_train)

SVC(C=2, class_weight='balanced', kernel='linear', probability=True,
    random_state=31)
```

```
y_pred = svc.predict(X_test_scaled)
accuracy_score(y_test, y_pred)

0.8290598290598291
```

CONFUSION MATRIX

The SVM classifier is giving an accuracy of **83%**.

6. MODEL OUTPUTS WITH CLIPPING

6.1. RANDOM FOREST

We have implemented the same steps as in 5.1 on data with clipping.

Random Forest Classifier

```
forest = RandomForestClassifier(max_depth=8,n_estimators=700, max_samples=0.7,
forest.fit(X, y)
```

```
RandomForestClassifier(class_weight='balanced', max_depth=8, max_samples=0.7,
                       n_estimators=700, oob_score=True, random_state=31)
```
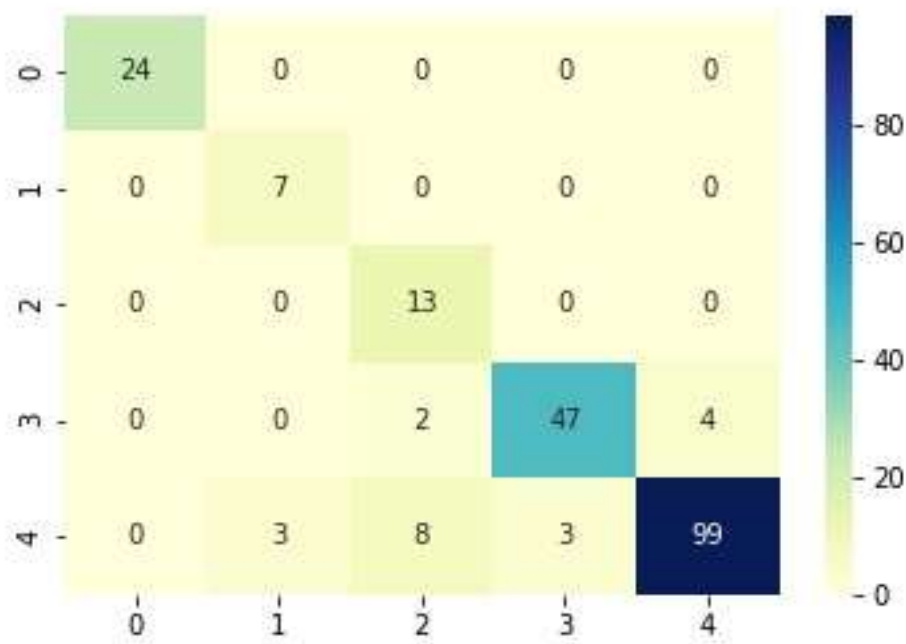
```
y_pred = forest.predict(X_test)
accuracy_score(y_test, y_pred)
```

```
0.9047619047619048
```

```
forest.oob_score_
```

```
0.7550047664442326
```

CONFUSION MATRIX

The Random Forest is giving an accuracy of **75%**.

6.2. STACKING
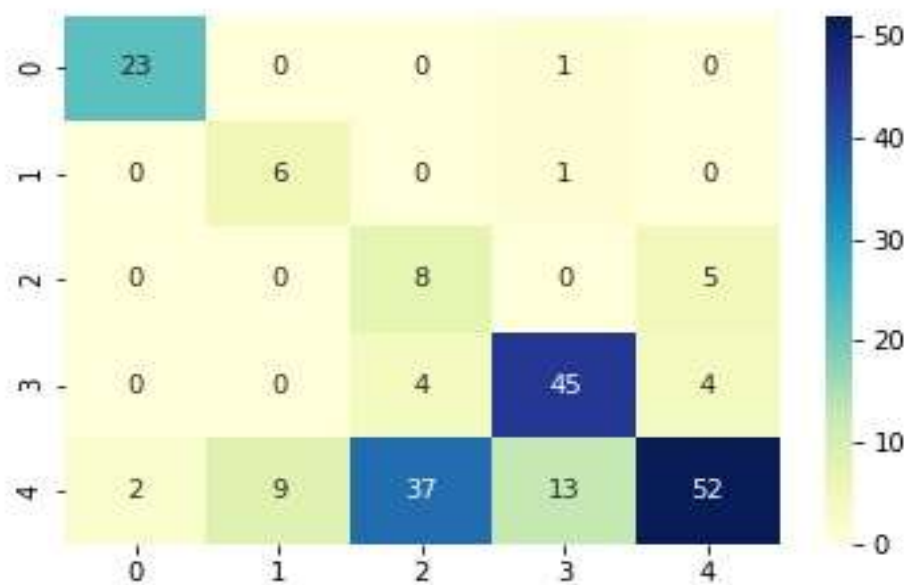
We have implemented the same steps as in 5.3 on data with clipping.

```
StackingClassifier(cv=5,
                   estimators=[('rf',
                                RandomForestClassifier(class_weight='balanced',
                                                       max_depth=8,
                                                       max_samples=0.7,
                                                       n_estimators=700,
                                                       oob_score=True,
                                                       random_state=31)),
                               ('gba',
                                GradientBoostingClassifier(random_state=31)),
                               ('svc',
                                SVC(C=2, class_weight='balanced',
                                    kernel='linear', probability=True,
                                    random_state=31)),
                               ('naive_bayes', GaussianNB())],
                   final_estimator=LogisticRegression(class_weight='balanced',
                                                      max_iter=1000),
                   n_jobs=4, passthrough=True)
```

```
y_pred = stacking.predict(X_test_scaled)
accuracy_score(y_test, y_pred)
```

0.638095238095238

CONFUSION MATRIX

The Stacking Classifier is giving an accuracy of **64%** .

6.3. NAÏVE BAYES

We have implemented the same steps as in 5.4 on data with clipping.
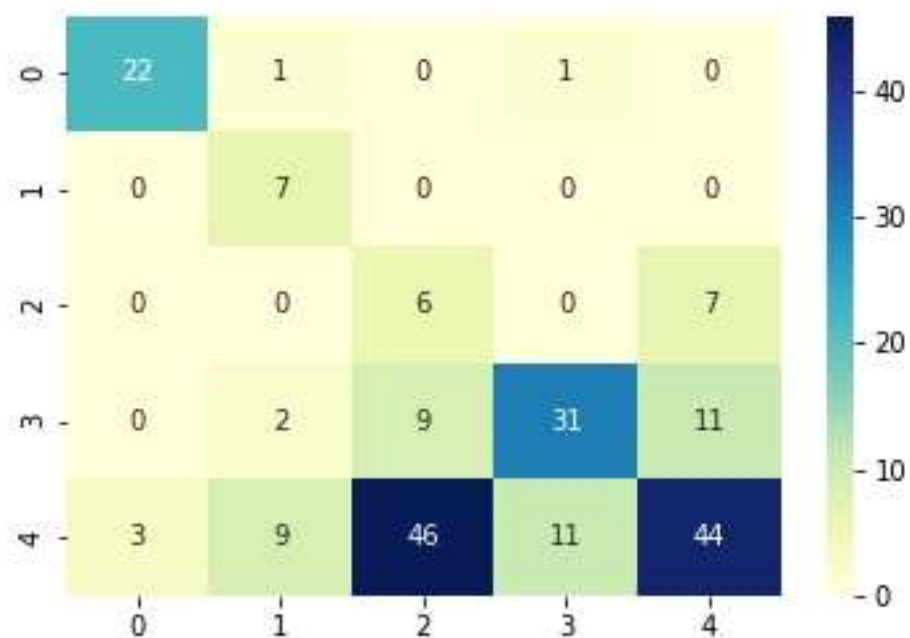
Gaussian Naive Bayes

```
gaussian_nb = GaussianNB()
gaussian_nb.fit(X_train_scaled, y_train)
```

```
GaussianNB()
```

```
y_pred = gaussian_nb.predict(X_test_scaled)
accuracy_score(y_test, y_pred)
```

```
0.5238095238095238
```

CONFUSION MATRIX

The Naïve Bayes classifier is giving an accuracy of **52%**.

6.4. SUPPORT VECTOR CLASSIFIER

We have implemented the same steps as in 5.5 on data with clipping.
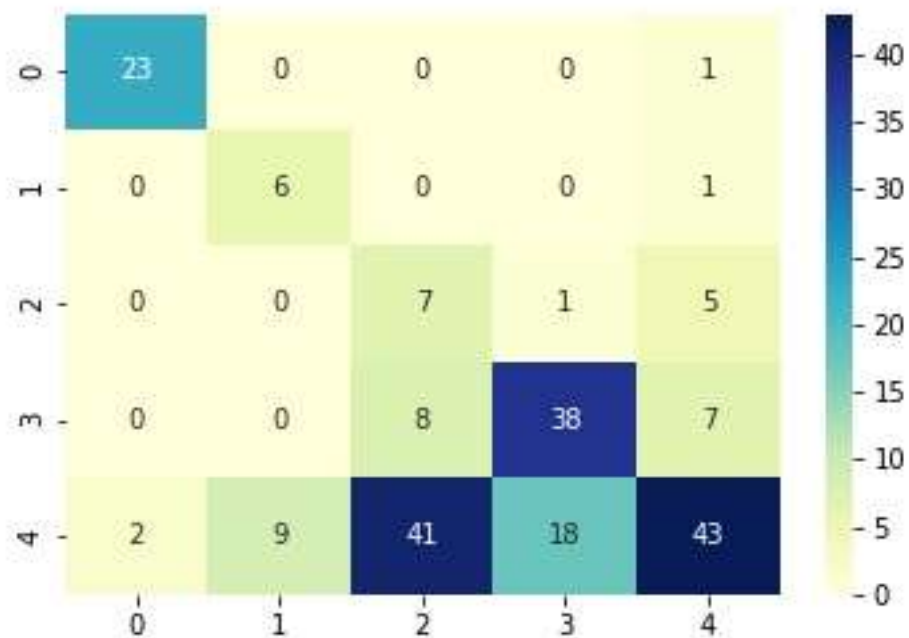
Support Vector Classifiers

```
svc = SVC(kernel = 'linear', C=2, probability=True, random_state=31, class_weight='balanced')
svc.fit(X_train_scaled, y_train)

SVC(C=2, class_weight='balanced', kernel='linear', probability=True,
    random_state=31)
```

```
y_pred = svc.predict(X_test_scaled)
accuracy_score(y_test, y_pred)
```

0.5571428571428572

CONFUSION MATRIX

The SVM classifier is giving an accuracy of **56%**.

7. DEEP LEARNING MODELS

7.1. CONVOLUTIONAL NEURAL NETWORKS

Convolution Neural Network [10] is a type of neural network that uses mathematical operation convolution. Convolution neural network is a locality sensitive architecture. This can be used here because our matrices contain temporal data.

```
7/7 - 0s - loss: 0.8704 - accuracy: 0.6952
Accuracy on test set is: 0.6952381134033203
Error on test set is: 0.870423436164856


21/21 - 0s - loss: 0.2975 - accuracy: 0.9717
Accuracy on train set is: 0.9716840386390686
Error on train set is: 0.29746130108833313


6/6 - 0s - loss: 0.9450 - accuracy: 0.6845
Accuracy on validation set is: 0.6845238208770752
Error on validation set is: 0.9449868202209473
```

From this output, we can see that the architecture learnt well on train data but it is unable to generalize on the unseen data. This is a problem of overfitting.

We implemented an architecture that contains two convolution layers, two max pooling layers, two dense layers, two batch normalization layers and a dropout layer.

The first layer is a Convolution layer that contains 32 convolution kernels of size 3X1. This is because we know that each column in that sample matrix corresponds to a single MFCC. Each MFCC is independent of the other. Hence, kernels are not spanned across columns.

We then applied a 2D max pooling layer. Here again, the window size is 2X1 because of the reason stated earlier. This is followed by a Batch Normalization layer.

We now again apply a convolution layer with 64 kernels of each size 3X1. Then, again, we apply a max pooling layer and a batch normalization layer.

We now have a matrix of size 32X30X64. We now flatten the matrix so that we can connect it to the dense layers. We connect it to a dense layer of 32 neurons followed by a dropout layer and final output layer of 5 neurons. Each of these 5 neurons represent a class of heartbeat. All the layers of the architecture have ReLU activation functions except for the final layer that has softmax activation function.

We used Adam optimizer with sparse categorical cross entropy as loss function. We trained the model for 60 epochs with batch size of 64.

```
Layer (type)                    Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)               (None, 130, 30, 32)       128

max_pooling2d_2 (MaxPooling2    (None, 65, 30, 32)        0

batch_normalization_2 (Batch    (None, 65, 30, 32)        128

conv2d_3 (Conv2D)               (None, 63, 30, 64)        6208

max_pooling2d_3 (MaxPooling2    (None, 32, 30, 64)        0

batch_normalization_3 (Batch    (None, 32, 30, 64)        256

flatten_4 (Flatten)             (None, 61440)             0

dense_5 (Dense)                 (None, 32)                1966112

dropout_2 (Dropout)             (None, 32)                0

dense_6 (Dense)                 (None, 5)                 165
=================================================================
Total params: 1,972,997
Trainable params: 1,972,805
Non-trainable params: 192
```

CONFUSION MATRIX
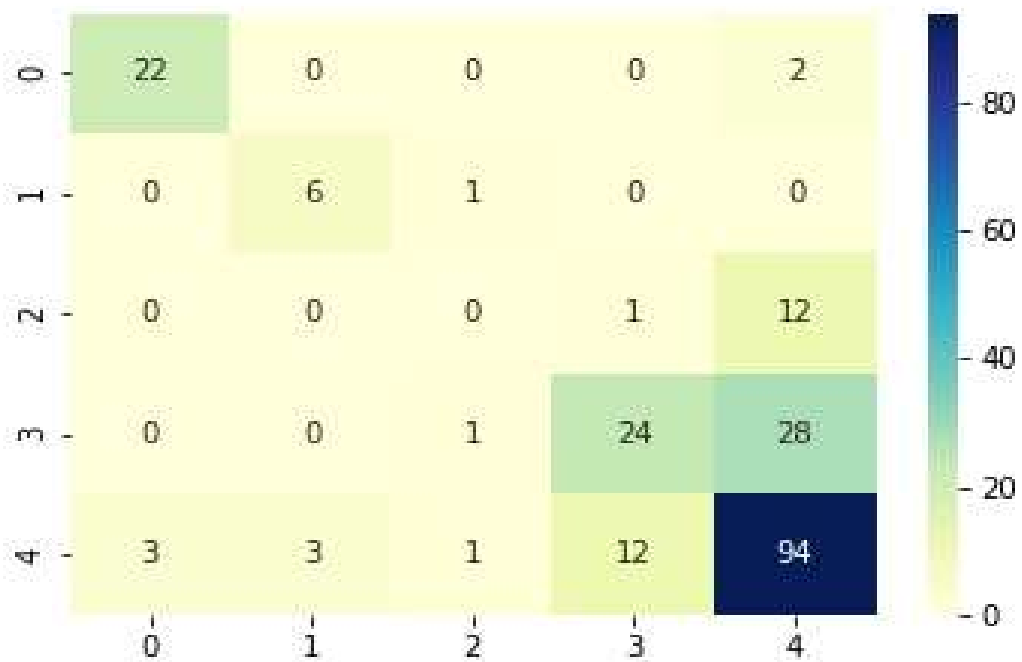
7.2. LONG SHORT TERM MEMORY

LSTM is an RNN based Architecture.

```
Layer (type)                     Output Shape        Param #
=================================================================
lstm_4 (LSTM)                    (None, 130, 100)    52400

bidirectional_2 (Bidirection     (None, 130, 200)    160800

bidirectional_3 (Bidirection     (None, 130, 200)    240800

lstm_7 (LSTM)                    (None, 100)         120400

dense_7 (Dense)                  (None, 64)          6464

dropout_3 (Dropout)              (None, 64)          0

dense_8 (Dense)                  (None, 5)           325
=================================================================
Total params: 581,189
Trainable params: 581,189
Non-trainable params: 0
```

The first layer is an LSTM layer. This LSTM layer tries to remember from previous 100 values. In the temporal axis. Then we have a bi directional LSTM which learns 100 values from the past and 100 values from the future. This is followed by another bi directional layer. We then have another LSTM layer.

We connected the last LSTM layer with a Dense layer with 64 neurons. We then introduced a Dropout layer with dropout percentage fixed at 30. It is finally connected to the output layer that contains 5 neurons, each representing a class of output.

There are a total of 581,189 parameters that this network is learning.

It uses Adam optimizer with learning rate of 0.0001 and loss as sparse categorical cross entropy. It is trained for 100 epochs with a batch size of 32 samples.

```
7/7 - 0s - loss: 1.0731 - accuracy: 0.7429
Accuracy on test set is: 0.7428571581840515
Error on test set is: 1.07306551933288857
```

```
21/21 - 0s - loss: 0.1449 - accuracy: 0.9568
Accuracy on train set is: 0.9567809104919434
Error on train set is: 0.14486941695213318
```

```
6/6 - 0s - loss: 1.1397 - accuracy: 0.6905
Accuracy on validation set is: 0.6904761791229248
Error on validation set is: 1.13969087600708
```

From the output, we can see that the accuracy of LSTM is not very encouraging. There could be many reasons for this. One reason could be that there is not much information that can be learnt from the temporal property of the matrix.

CONFUSION MATRIX
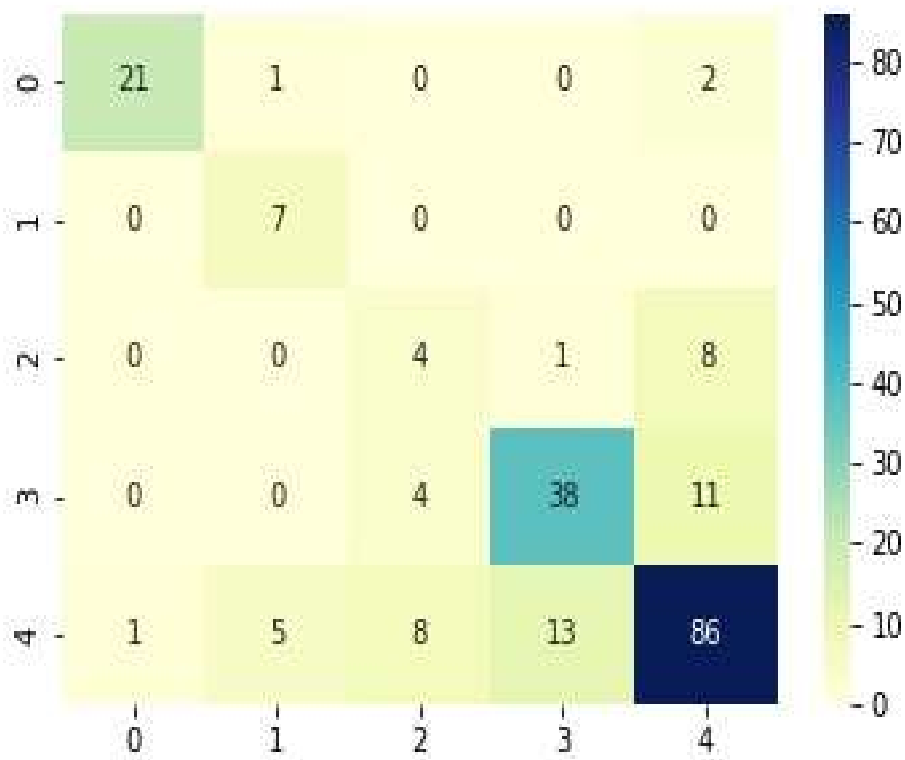
Summary

We now compare the two models.

|	CNN based Model	RNN based Model
Number of parameters	1,972,997	581,189
Number of Epochs ran	60	100
Train Accuracy	0.971	95.6%
Validate Accuracy	0.684	69.04%
Test Accuracy	0.695	74.28%

8. A NOVEL APPROACH

After building various models, we could identify that the maximum that we could reach is 98% by a Gradient Boosting model. In an effort to break this barrier, we propose a new approach for this problem.

We divide the problem statement into two different stages. The first stage deals with identifying if the heartbeat is normal heartbeat or diseased heartbeat. In this stage we train a binary classification model.

The second stage deals with identifying the type of heart disease. We train models in this stage using only the four subcategories of the data. We do not train on the normal heartbeat data. Hence, we can pass data to this model only if the model in the previous stage does not classify a healthy heartbeat as unhealthy.

Here also we need to try with the two different csv files. Even if in the previous experiments, we realised that models worked better with one csv file, we cannot confirm that the same result occurs with a different target variable.
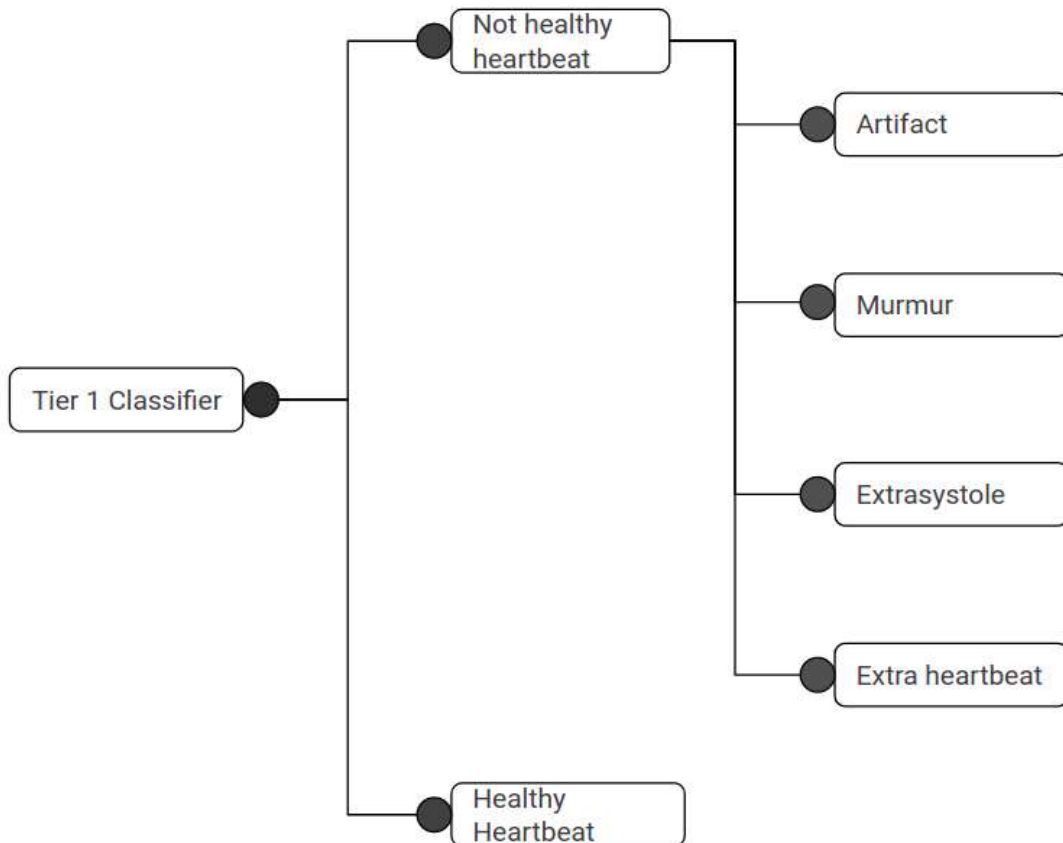
Different algorithms like Random Forest, Gradient Boosting and Support Vector Machines are implemented on both the datasets and at both the levels. We will now see the results of some of the algorithms that are implemented on different levels.

8.1. TIER 1

Tier 1 is a binary classification model. It differentiates between healthy heartbeat and unhealthy heartbeat. The below table gives us the accuracies of the models that are implemented in tier 1.

	Features.csv	Features_without_clips.csv
Random Forest Classifier	98.47%	95.91%
Gradient Boosting Classifier	99.6%	99.31%
Support Vector Classifier	92.01%	87.75%

For both the datasets, Gradient Boosting Classifier gives better results when compared with other Classifiers. Let us take a deeper look into the results of both the Gradient Boosting Classifiers.
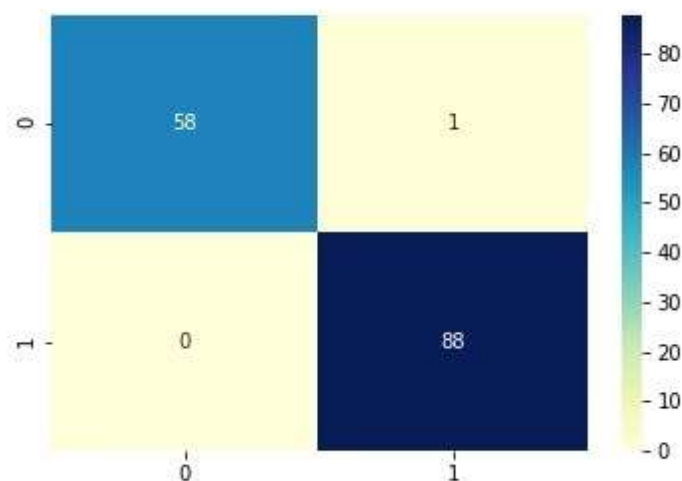
The above confusion matrix corresponds to the Gradient boosting Classifier built using Features_without_clips.csv dataset. It has an accuracy of 99.31% and the only misclassified data tells that some unhealthy heartbeat is classified as healthy heartbeat.
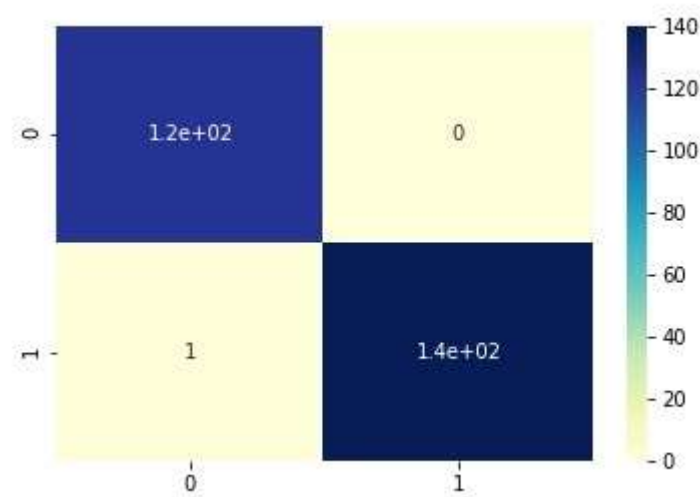
The above confusion matrix corresponds to the Gradient Boosting classifier built using Features.csv dataset. It has test accuracy of 99.6%. The issue here is that healthy heartbeat is classified as unhealthy heartbeat. So, the healthy heartbeat audio clip will be passed to the next tier. But in the next tier, no models are trained for healthy heartbeat. So, we cannot accept this model.

As a result, we will choose the Gradient Boosting Classifier model built using the Features_without_clips.csv data file.

8.2. TIER 2

Tier 2 is a multi class classification problem that focuses on identifying the type of non healthy heartbeat. It categories the heartbeat in one of the 4 different categories. All models in this tier are built using only the four categories.

The below table gives accuracies of the models that are built using the Features_witout_clips.csv file. This is because the previously selected model is built on that dataset.

Models	Accuracy
Random Forest	89.83%
Gradient Boosting	94.91%

As we can see from the table, Gradient Boosting Classifier performed better when compared to the Random Forest Classifier. Let us take a look at the confusion matrix of gradient Boosting Classifier.
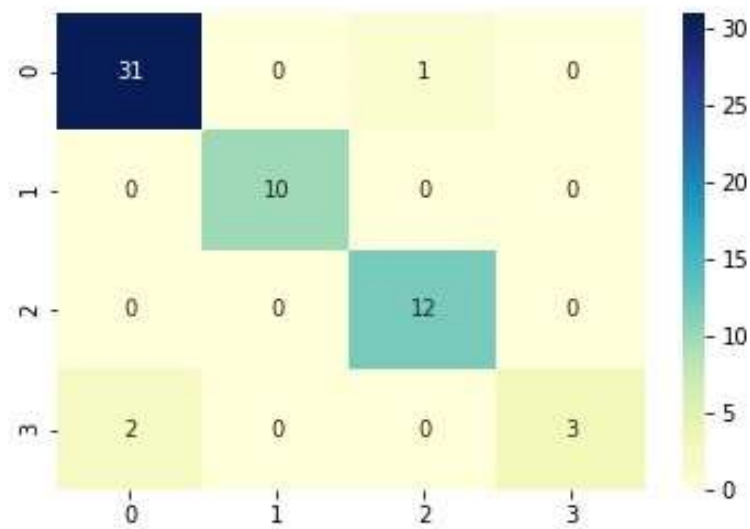
Class 0, 1, 2 and 3 correspond to the types Murmur, Artifact, Extra Systole and Extra heartbeat respectively. 2 out of 5 audible files of extra heartbeat is classified incorrectly. It shows that the model is not very good at classifying the extra heartbeat sounds.
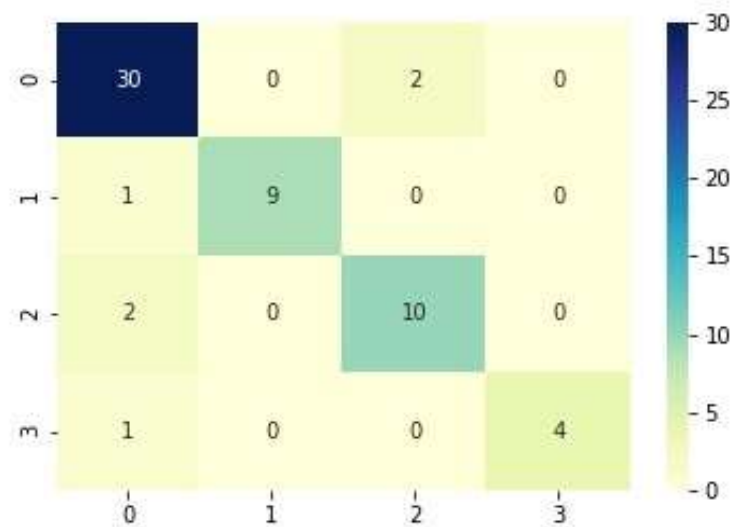
The above confusion matrix corresponds to the Random Forest Classifier. Here, we can observe that Random Forest Classifier performed well for the class that Gradient Boosting Classifier could not classify. But, it has got the tendency of classifying few of the samples across classes as Murmur.

So, we will try to combine both the models using a stacking classifier. We create an ensemble stacking classifier. The first layer contains the two classifiers discussed earlier and the meta learner is again a Gradient Boosting Classifier.

```
StackingClassifier(cv=5,
                   estimators=[('rf',
                                RandomForestClassifier(class_weight='balanced',
                                                        oob_score=True,
                                                        random_state=9)),
                               ('gba',
                                GradientBoostingClassifier(random_state=9))],
                   final_estimator=GradientBoostingClassifier(random_state=9),
                   n_jobs=4, passthrough=True)
```

```
y_pred = stacking.predict(X_test)
accuracy_score(y_test, y_pred)
```

```
0.9491525423728814
```

The accuracy of this model is 94.91%. It is a good model when compared to the other two models. Its accuracy is the same as that of the Gradient Boosting Classifier. Let us take a look into the confusion matrix.
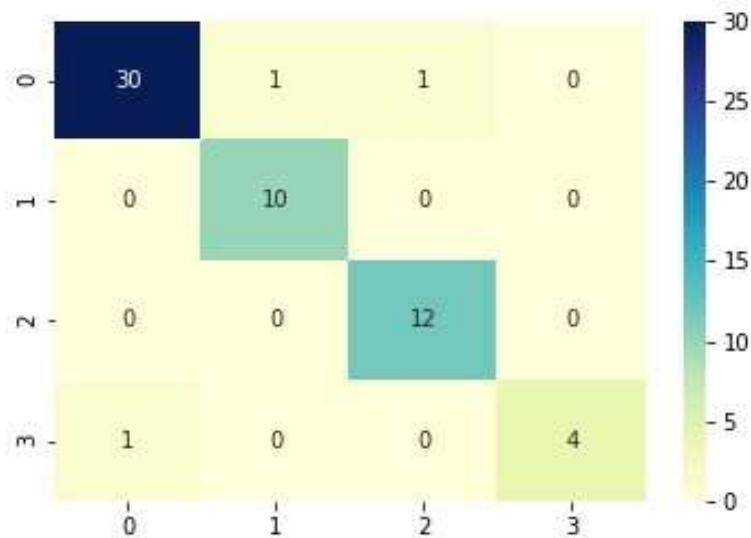
The confusion matrix gives a better understanding of how good the classifier is. The accuracy of this model is the same as that of the Gradient Boosting Classifier but this model works much better with the class 3. One in every 32 samples of murmur is predicted as Artifact. It calls for the physician to record the heartbeat once again. Hence there is not much cost involved in the mis classification. Hence, from the confusion matrix, we can finalize that Stacking Classifier performs better than Gradient Boosting and Random Forest classifier.

Summary
In conclusion, we built a two tier model that first classifies a heartbeat as healthy and unhealthy. It then classifies the heartbeat that is predicted as unhealthy into

different types of heartbeats. We trained the models on the features extracted from the complete audio file.

In tier 1, we used Gradient Boosting Classifier and in tier 2, we used Stacking Classifier that is made up of Random Forest and Gradient Boosting classifier.

Results

We have two best models from the tens of models that are built and experimented on. We will now have a comparison of both the models. One model is a direct model built on Features_without_clips.csv file. A gradient boosting model was built and it gave an accuracy of 98.29%. The model confusion matrix also shows good signs of a healthy model.
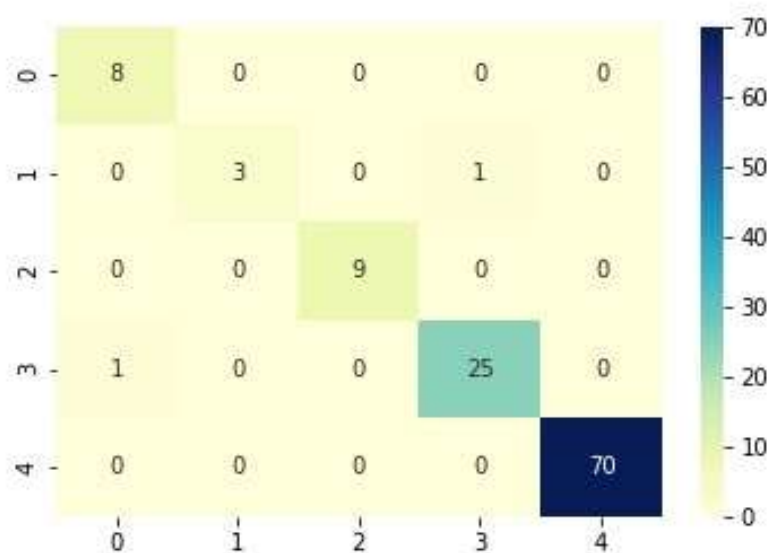
The train data gave a complete accuracy of 100% and only 2 out of 117 are classified incorrectly. Out of the two wrongly classified observations, one observation is wrongly classified as an artifact. It means that the person is going to retake the experiment. It is a loss less misclassification when seen is a practical aspect. So, this model is considered as the best model.

The novel model architecture that we designed also gave very good results and they are only one step behind this model. Hence the novel model will be considered as the second-best model.

9. DEPLOYMENT

Any model's primary goal is to be used by people. It's pointless to use the right one if it's not held in use. As a result, we created a user interface and integrated the best model with it. We then published our model using Heroku Cloud's free tier cloud service. This model can be used on any computer from anywhere in the world.

9.1.   PYTHON FLASK

The web interface is developed using a python framework called flask.
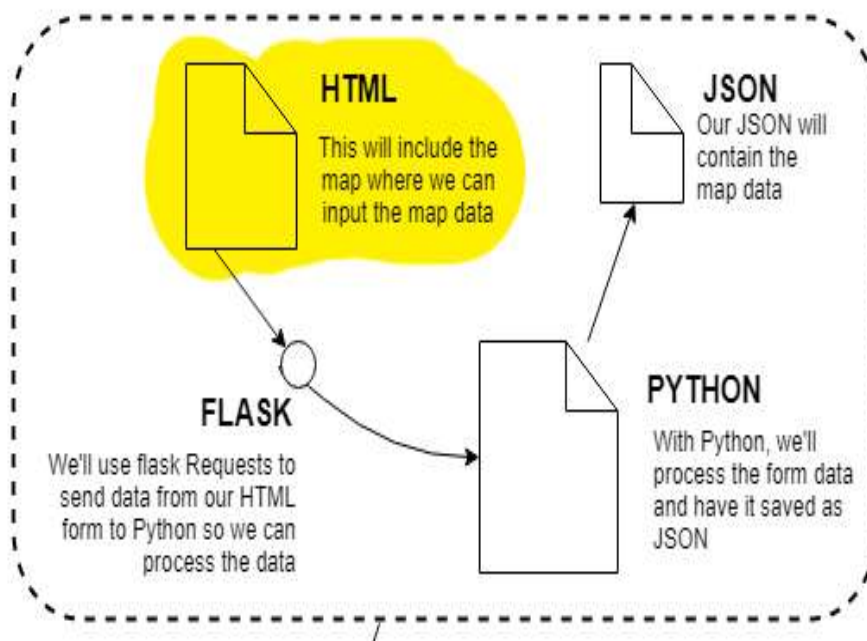
9.2.   HEROKU CLOUD

The web framework and the model is built. They should be deployed using a cloud service so that they can be used from anywhere in the world. We chose Heroku Cloud Platform because of the advantages it provides over other cloud platforms for a python developer.

Heroku also provides free tier services that are very helpful for the projects that are in the development stage.
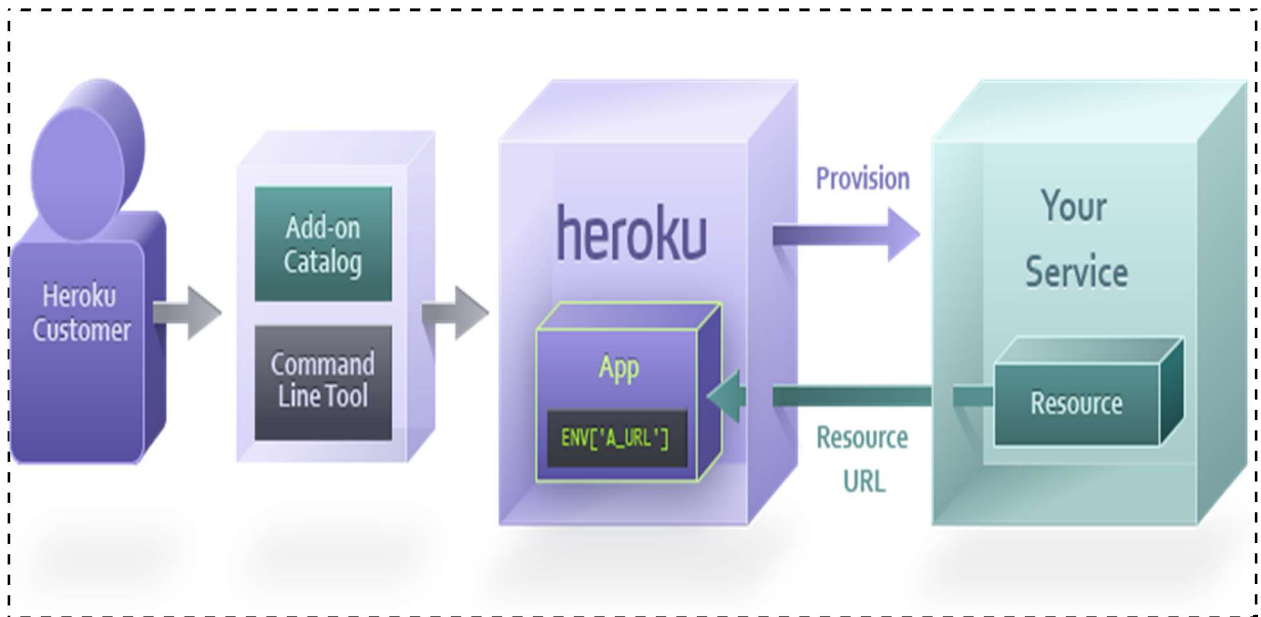
Few of the advantages include:

- Allows the developer to focus on code instead of infrastructure
- Support form Modern Open Source Languages
- Beginner and start-up friendly

10. MY CONTRIBUTIONS

My Project partner Sri. MVSSS Durgesh (19230) contributed towards the Data Pre-processing, Machine Learning Algorithms and I Contributed towards Deep Learning Models and Deployment of Project.

11.   CONCLUSION & FUTURE WORK

11.1. CONCLUSION

This project developed Machine Learning and Deep Learning models for heartbeat sound classification. This procedure can effectively monitor the heartbeat signal and provide information that can be used to determine whether or not additional treatment is required. Normal, Murmur, Extra Heartbeat, Artifact, and Extra-Systole are the five sections in which the data is split The noise is filtered out of the heartbeat sound signal. Down-sampling reduces the dimension of the heartbeat sound signal wave to extract further discriminative characteristics, while data framing transforms the sampling frame rate of each audio file into a fixed-size frame rate. In this analysis, the Gradient Boosting model was applied to Dataset, which had the maximum Test accuracy of 99.31. Our approach is more competitive and effective, as shown by the experiment.

11.2. FUTURE WORK

We'll use spectrogram techniques and other deep learning techniques to get more discriminative functions. This model can be used in apps like the iStethoscope Pro iPhone app, which allows the heartbeat to be registered and labelled directly.

12. REFERENCES

1. Ashwin Prakash - Detecting Heart Anomalies using Heartbeat sound - https://bit.ly/3sugeCI

2. Original Problem Statement - http://www.peterjbentley.com/heartchallenge/#taskoverview

3. Taxi-route dataset link-https://www.kaggle.com/c/nyc-taxi-trip-duration/data.

4. Dataset link - Heartbeat Sounds

5. Valerio Velardo YouTube series on Audio data classification - https://www.youtube.com/playlist?list=PL-wATfeyAMNrtbkCNsLcpoAyBBRJZVInf

6. Scikit learn documentation - https://scikit-learn.org/0.23/user_guide.html

7. Librosa documentation - https://librosa.org/doc/latest/index.html

8. Numpy documentation - https://numpy.org/doc/1.18/

9. Corey Schafer YouTube series on Python Flask - https://www.youtube.com/playlist?list=PL-osiE80TeTs4UjLw5MM6OjgkjFeUxCYH

10. Random forest Algorithm - https://builtin.com/data-science/random-forest-algorithm

11. Stacking Algorithm - https://machinelearningmastery.com/stacking-ensemble-machine-learning-with-python

12. Gradient Boosting Algorithm - https://towardsdatascience.com/understanding-gradient-boosting-machines

13. Deep learning Algorithms - https://www.datarobot.com/wiki/deep-learning/

14. Heartbeat Sound Signal Classification using Deep Learning - https://www.researchgate.net/publication/337050256_Heartbeat_Sound_Signal_Classification_Using_Deep_Learning

15. Automatic Classification of Periodic Heart Sounds using Convolution Neural Network - https://www.researchgate.net/publication/323394068_Automatic_Classification_of_Periodic_Heart_Sounds_Using_Convolutional_Neural_Network

16. Recognizing Abnormal Heart Sounds Using Deep Learning - http://ceur-ws.org/Vol-1891/paper2.pdf

17. A Meta Analysis of research in Random Forest for Classification - https://www.researchgate.net/publication/312486161_A_meta-analysis_of_research_in_random_forests_for_classification

18. Human heart sounds classification using Ensemble Methods - https://www.researchgate.net/publication/316472009_Human_Heart_Sounds_Classification_using_Ensemble_Methods

19. A better auto encoder for Image: Convolutional Auto Encoder - http://users.cecs.anu.edu.au/~Tom.Gedeon/conf/ABCs2018/paper/ABCs2018_paper_58.pdf

20. A Deep Convolutional Auto Encoder with Embedded Clustering - https://www.researchgate.net/publication/327995458_A_Deep_Convolutional_Auto-Encoder_with_Embedded_Clustering

21. A Convolutional Auto Encoder Approach for Feature Extraction in Virtual Metrology - https://www.sciencedirect.com/science/article/pii/S2351978918311399

22. Recent Advances on Support Vector Machines Research - https://www.researchgate.net/publication/254321762_Recent_advances_on_support_vector_machines_research

23. A novel approach to handle class imbalance in machine learning - https://www.ijert.org/a-novel-approach-to-handle-class-imbalance-in-machine-learning