Discrimination of reflected sound signals

Information Technology (M. Eng.)
Machine Learning
Jagdeep Singh
1317679
jagdeep.singh@stud.fra-uas.de
Sidra Hussain
1318131
sidra.hussain@stud.fra-uas.de
Vishakha Choudhary Babulal
1324497
vishakha.babulal@stud.fra-uas.de

Abstract—**Object discrimination is one of the main branches of machine learning. It is widely applied in many applications such as monitoring, security, RADAR, SONAR, autonomous driving, etc. Identifying and discriminating the object via a convenient and efficient domain is one of the main challenges of data-driven science. With developing technologies and evolving deep learning networks, it is possible to employ object discrimination and identification via a comprehensive set of mediums. This paper demonstrates using an acoustic signal to discriminate between objects that signal is reflected on. The problem statement includes using significant feature extraction and binary classification to distinguish between the objects.**

Keywords—Object Identification, Binary Classification, Loudness, Quadratic Time-Frequency Representation, Maximum Energy, Neural Networks, k-Nearest neighbor algorithm, Logistic Regression, Decision Tree, Random Forest Algorithm, Spectrogram, Receiver Operating Characteristic, F1 Score

I. INTRODUCTION

Machine learning in acoustics is rapidly evolving, with promising results and a bright future [1]. Many sensory systems need signal processing in noisy environments. Especially when the data retrieved is hefty, the procedure of identifying an object through acoustic analysis becomes difficult and prone to error.

This is where signal processing comes into play. Using the current technology of employing a classification algorithm, significant signals can be easily understood, processed, and operated for classification and discrimination.

Although feeding large and robust data into a classifier poses no harm, signal analysis can use a regressive approach by employing energy distributions of a signal rather than original signals. The aim of this approach is to reduce the processing time and get an overview of the nature of the signal received firsthand rather than handling the raw data itself. The energy distributions provide a map of the behavior of the signal and aids in the process of feature extraction for classification. The energy distribution used is a Quadratic Time Frequency representation that provides a map of energy with time and frequency instances.

In this paper, for the classification of the energy distributions of the signal, a binary classifier is used. Binary or binomial classification is the process of assigning elements to one of two classes. This activity is carried out in accordance with a set of classification rules. The computational apparatus for implementing classification algorithms is provided by machine learning theory. The algorithms are Decision trees and Random Forest Algorithm [2].

This paper is organized by first providing a detailed study of Quadratic Time Frequency Representation as our energy distribution spectrum with the classification analysis rates. Following that, Binary classification and its algorithms are discussed with classification analysis rates. In the end, implementation, steps to rebuild the solution (code summited) and results analysis are put forward.

II. QUADRATIC TIME FREQUENCY REPRESENTAION

A. *Quadratic Time Frequency Representation Analysis*

The Short-Time Fourier Transform (STFT) takes a linear approach for a time-frequency representation [3]. It decomposes the signal on elementary components, called, atoms and is represented as:

$$h_{t,w} = h(\tau - t)e^{-j\omega\tau} \tag{1}$$

Each atom is obtained from the window $h(t)$ By a translation in time and a translation in frequency (modulation). The Time Frequency is represented as in the following diagram.

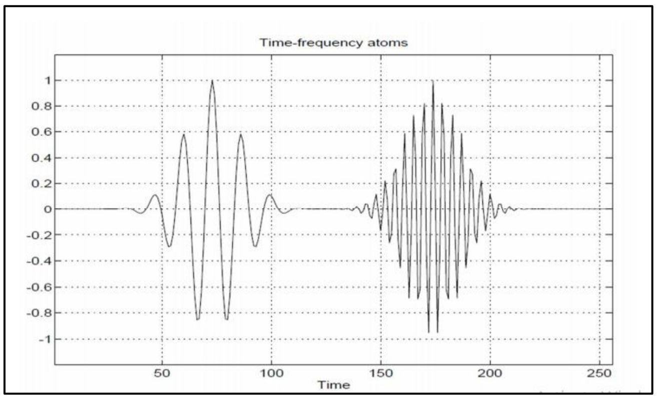


Figure 1 Time Frequency atoms [3]

In the frequency domain, it is represented as:

$$H_{t,\omega} = h(w - \theta)e^{-j\theta t} \tag{2}$$

The STFT can be considered as the result of passing the signal through a bank of band-pass filters whose frequency response is $H(w - \theta)$ and is deduced from a mother filter $H(w)$ by a translation of θ. Each filter in the bank has a constant bandwidth. If we consider the square modulus of the STFT, we get the *Spectrogram*, which is the spectral energy density of the locally windowed signal $s_t = s(\zeta)h(\zeta - t)$.

The spectrogram is a quadratic or bilinear representation. If the energy of the windows is selected to be one, the energy of the spectrogram is equal to the energy of the signal. Thus, it can be interpreted as a measure of the energy of the signal contained in the time-frequency domain centered on the point (t, ω) [3].

B. *The Energy Distributions*

The purpose of the energy distribution is to distribute the energy of the signal over time and frequency. The energy of a signal s(t) can be deduced from the squared modulus of either the signal or its Fourier transform. $Es = \int |s(t)|^2\ dt = \int |s(\omega)|^2\ d\omega.$ $|s(t)|^2$ and $|s(\omega)|^2$ can be interpreted as energy densities in time and frequency. It is natural to look for a joint time and frequency energy density, $\rho_s\ (t, \omega)$ such that $Es = \iint\ \rho_t(t, \omega) dt d\omega$. As the energy is a quadratic function of the signal, the time-frequency energy distributions will be in general quadratic representations [3].

The two properties that an energy density should satisfy are the time and frequency marginal conditions, i.e.: $\rho_s\ (t, \omega) dt = |S(t)|^2$ and $\rho_s(t, \omega)\ d\omega = |S(t)|^2$. If the time-frequency energy density is integrated along with one variable, the result is the energy density corresponding to the other variable. As per the Cohens class, there are many distributions that satisfy the properties mentioned before therefore, it is possible to impose additional constraints to $\rho_t(t, \omega)$ that would result in desirable properties. Among these properties, the covariance principles are of fundamental importance. The Cohen's class is the family of time-frequency energy distributions covariant by translations in time and frequency. The spectrogram is an element of the Cohen's class, since it is a quadratic, time- and frequency-covariant, and preserves energy. Taking the square modulus of an atomic decomposition is only a restrictive possibility to define a quadratic representation [3].

$$x(t) = s(t - t_0) = P_x(t, \omega) = P_s\ (t - t_0, \omega) \tag{3}$$

$$x(t) = s(t) e^{-j\omega_0 t} = P_x\ (t, \omega) = P_s\ (t, \omega - \omega_0) \tag{4}$$

$$P(t, \omega) = |S(\omega)|^2 = |s(t)|^2 \tag{5}$$

C. *Quadratic Time Frequency Representations*

Quadratic Time Frequency Representations are useful methods for analyzing signals that change over time. A power spectrum of a signal is computed by using the Discrete Fourier Transform, which computes the signal in its frequency domain. This spectrum represents the energy of the signal with respect to its frequency. This is important in case of examining the behavior and response of a signal. If the signal in question is a time domain signal this power spectrum can be called a Time Frequency Representation (TFR). And as the power spectrum is estimated by squaring the Fourier Transform, a TFR can be called a Quadratic Time Frequency Representation [4].

D. *Spectrogram*

A spectrogram shows the signal strength of a signal over time at various frequencies. The amplitude of a spectrogram is the third dimension which is basically the brightness or loudness of a sound signal. It shows the energy content of the signal at a specific time and frequency via x and y axis. It aids is observing the energy content at specific frequency levels and how the energy levels change over time [5].

III. Binary Classification

Binary classification is one of the most common and often addressed problems in the field of machine learning. Simply put, it tries to classify an element into one of the two possible categories by measuring a series of attributes. The theoretical apparatus for implementing classification algorithms is provided by machine learning theory. Some of these algorithms are Random Forests and Decision Tree [6].

A. *Decision Tree*

1) Definition: A decision tree is a supervised machine learning classification model with a tree structure. It is used to visually and explicitly portray decisions and decision-making so that non-expert users can understand easily and can be effectively derived from a given dataset. In the decision tree, the data is continuously split according to a specific attribute. The two main decision tree entities are decision nodes, where the data is split into further decision nodes, or leaves, where we get the outcome [7]. Figure 2 depicts a decision tree graph.

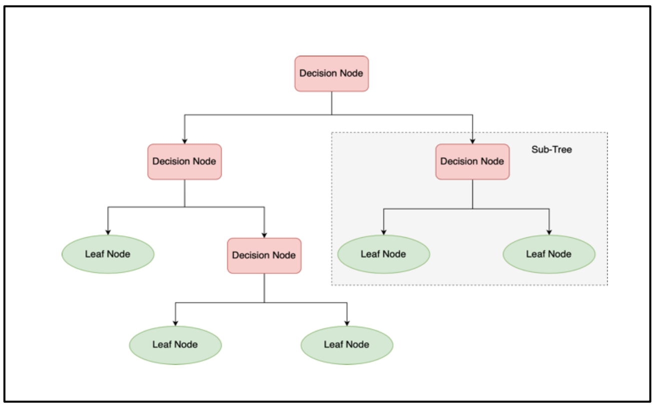


Figure 2 Decision Tree representation [6]

2) Mathematical Implementation: The Decision Tree algorithm works by selecting the best attribute using Attribute Selection Measures(ASM) to split the records. ASM is a heuristic for choosing the best possible splitting criterion for data partitioning. Since it helps determine breakpoints for tuples on a given node, it is also known as the splitting rules. By describing the given dataset, ASM assigns a rank to each function (or attribute). As a splitting attribute, the best score attribute will be chosen. Information Gain, Gain Ratio, and Gini Index are the most commonly used selection criteria [7].

- Entropy gives the degree of randomness or impurity contained within a given dataset.
- Information gain calculates the difference between the entropy in advance of the split and the average entropy after the split following attribute values.

$$Info(D) = -\sum_{i=1}^{m} p_i log_2 p_i \tag{6}$$

where Pi is the probability that an arbitrary tuple in data D belongs to class Ci.

$$Info_A(D) = \sum_{j=1}^{V} \frac{|D_j|}{|D|} X\ Info(D_j) \quad (7)$$

$$Gain(A) = Info(D) - Info_A(D) \quad (8)$$

with $Info(D)$ is the average amount of data needed to determine a tuple's class label in D, $|D_j|/|D|$ as the weight of the j^{th}partition and $Info_A(D)$ represents the expected information required to identify a tuple from D using A's partitioning.

- By using Split Info to normalize the Information gain, the Gain Ratio addresses the problem of bias. The Gain ratio is calculated as follows:

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)} \quad (9)$$

$$SplitInfo_A(D) = -\sum_{j=1}^{V} \frac{|D_j|}{|D|} \times log_2\left(\frac{|D_j|}{|D|}\right) \quad (10)$$

with V as the number of discrete values in attribute A. The attribute having the highest gain ratio is chosen as the splitting attribute.

- Gini Index with formula in equation 8 is used to create split points in the algorithm.

$$Gini(D) = 1 - \sum_{i=1}^{m} {p_i}^2 \quad (11)$$

Variable splits should have a low Gini Index.

3) *Use Cases:* Data analysts often employ Decision Trees in their predictive analysis, such as to develop operations strategies in businesses. They are also a popular machine learning and artificial intelligence tool used as supervisory learning training algorithms (e.g., categorization of data based on various tests, such as 'yes' and 'no' classifications). Overall, decision-making trees are used to solve a variety of problems in a variety of industries. They are used in technology and health sectors to financial planning because of their flexibility [8].

B. *Random Forest*

1) *Definition:* The Random Forest (RF) classifier is a supervised machine learning classification method that uses decision trees as its foundation. A Random Forest is a set of unbiased, unrelated, and de-correlated decision trees. It is thus called a 'random forest' [9]. It's a type of meta-estimator that uses decision-tree classifiers on random subspaces and dataset samples and applies to mean averaging tools to improve prediction accuracy over time while avoiding overfitting [10]. A simple illustration of random forest is presented in Fig. 3.

2) *Mathematical Implementation:* Random Forest works by first selecting random samples from a given dataset using which it constructs a decision tree for each sample. Prediction results from each decision tree is collected, and voting is performed on them. Prediction result with the highest vote is selected as the final result. The idea of variable importance is an implicit feature selection done by RF with a random subspace methodology, and the Gini impurity criterion index assesses it. The Gini index is a measure of variable predictive power in regression or classification based on the impurity reduction theory. It is non-parametric, which means it does not depend on data from a specific type of distribution [11]. The Gini index of a node n in a binary split is determined as follows:

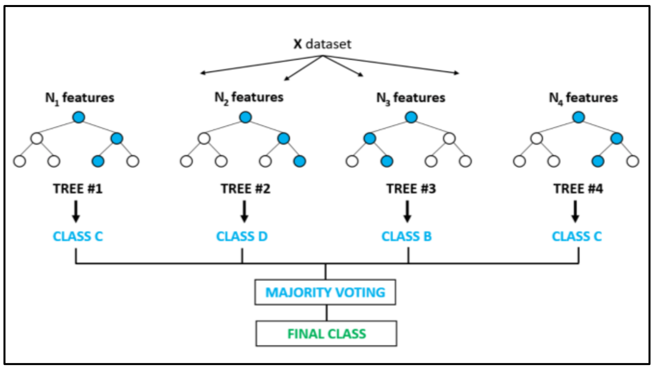


Figure 4 Random Forest Classification [15]

$$Gini(n) = 1 - \sum_{j=1}^{2} (p_j)^2 \quad (12)$$

where p_j is the relative frequency of class j in the node n.

The Gini index should be improved to the greatest extent possible when splitting a binary node. In another explanation, a low Gini (i.e., a more significant decrease in Gini) indicates that a specific predictor function is more critical in dividing the data into two groups. In a classification problem, the Gini index can be used to rank the importance of features [12].

3) *Use Cases:* The Random Forest algorithm can be used for both classification and regression tasks. It can deal with missing values and keep a large proportion of data accurate. Having the ability to work upon an extensive data set with higher dimensionality, RF is used in the banking sector, medicines, stock market, and E-Commerce. For example, to determine whether the customer is loyal or fraudulent, RF analysis can be useful. With RF, it has grown easier to detect and predict the drug sensitivity of a medicine [13].

IV. CLASSIFIER ASSESSMENTS

There are a number of ways to assess the performance of a classifier. Especially when the data set has a binary mode. All the assessments are rooted in one matrix called the confusion matrix.

A confusion matrix is a table that determines how well a classification model (or "classifier") performs on a collection of test data for which the actual values are known. It is a comprehensive and helpful presentation of the classifier performance. Several measures, such as error rate, precision, accuracy, sensitivity, and specificity, are obtained from the confusion matrix [14].

An example of a confusion matrix for a binary classifier with four different combinations of predicted and actual values is depicted in fig. 4. The target variable which needs to classify has two values: Positive or Negative. In the confusion matrix, the columns denote the actual values of the target variable and the rows denote the predicted values of the target variable [14].

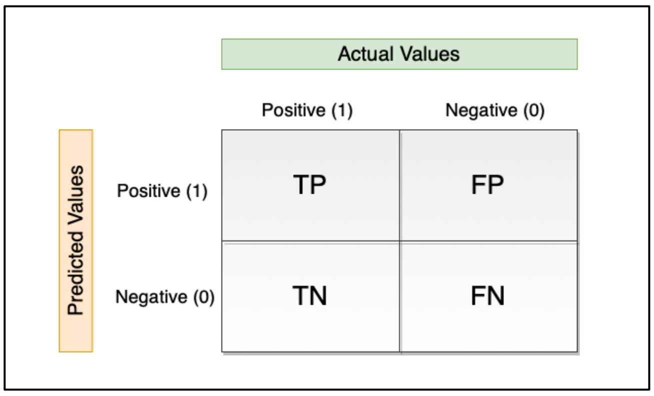


Figure 6 Confusion matrix for Binary Classifier [13]

1) Outcomes of Classification: A binary classifier assigns a positive or negative label to all data instances in a test dataset. True positive (TP), true negative (TN), false positive(FP), and false-negative (FN) are the four outcomes of this classification (or prediction).

- True Positive (TP) when the actual value is positive and predicted value is also positive.
- True Negative (TN) when the actual value is negative, and prediction is also negative
- False Positive (FP) when the actual is negative, but the prediction is positive. It is also referred to as Type 1 error.
- False Negative (FN) when the actual is positive, but the prediction is negative. It is also referred to as Type 2 error.

2) Classification Measure: Using Confusion Matrix, various classification measures are derived which helps in performance analysis of the classification model.

a) Accuracy: The ratio of correctly predicted examples to the total number of examples by the classifier is called accuracy. It simply measures how frequently the classifier delivers the correct prediction.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{13}$$

b) Precision: Precision is a metric that measures how many accurate positive predictions have been made. The ratio of correctly predicted positive examples divided by the total number of positive examples predicted is used to measure it.

$$Precision = \frac{TP}{TP + FP} \tag{14}$$

c) Sensitivity: Also called Recall or True Positive Ratio (TPR) or Hit Rate, Sensitivity is calculated as the ratio of the number of true positives to the sum of the total number of true positives and false negatives.

$$Sensitivity = \frac{TP}{TP + FN} \tag{15}$$

d) Specificity: The specificity is computed by dividing the number of correct negative predictions by the total number of negatives. It is also referred to as True Negative Rate (TNR).

$$Specificity = \frac{TN}{TN + FP} \tag{16}$$

e) F-Measure/F1-Score: F-Measure is a method for combining precision and recall into a single measure that encompasses both. It is calculated as a harmonic mean of Precision and Recall.

$$F1\ Score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{17}$$

$$= \frac{2TP}{2TP + FP + FN}$$

f) False Discovery Rate (FDR): The FDR determines how many of all positive predictions are wrong. It is the ratio of incorrect predicted examples to the total number of positive predicted examples.

$$FDR = \frac{FP}{FP + TP} = 1 - Precision \tag{18}$$

g) Negative Predicted Value (NPV): The ratio of correct negative predicted examples to the total number of negatives gives NPV.

$$NPV = \frac{TN}{TN + FN} \tag{19}$$

h) ROC Curve: An ROC curve (receiver operating characteristic curve) is a graphical representation that shows how well a classification model performs over all the classification thresholds. It is based on two primary evaluation measures – specificity (a measure of the whole negative part of the dataset) and sensitivity (a measure of the whole positive part).

The ROC graph plots 1 – specificity (True Positive Rate) on the x-axis and sensitivity (False Positive Rate) on the y-axis as shown in fig. 5. Evaluation of a logistic regression model is performed many times with different classification thresholds yields the points in an ROC curve [15].

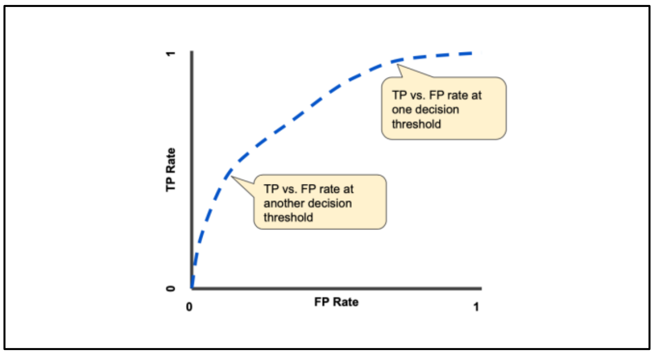


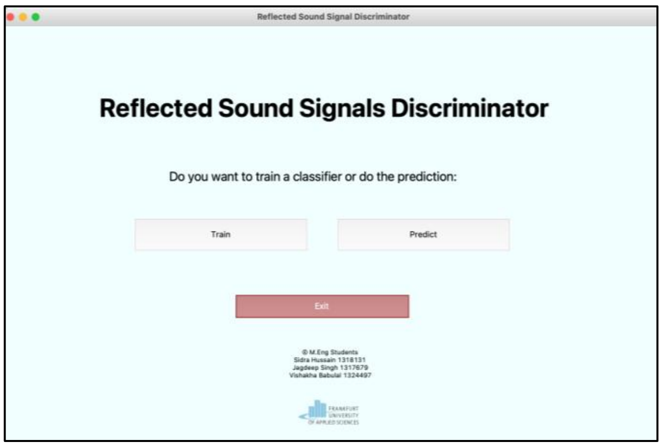
Figure 8 ROC Curve [14]

V. Implementation

The main purpose of this project is to specify if the signal given was reflected off Object 1 or Object 2 which are also considered as their data labels. This section precedes the following subsections that cover the details of implementation of the classifier and it's training.

A. GUI

The graphical user interface (GUI) is implemented using tkinter module of python. It provides options to a user with training a classification model using a new dataset or do the prediction using the existing trained model. Homepage of GUI consists these two options as shown in fig. 9.


Reflected Sound Signal Discriminator

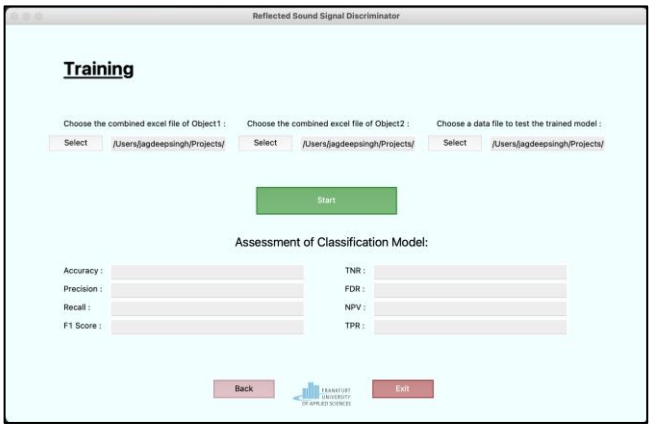
Reflected Sound Signals Discriminator

Do you want to train a classifier or do the prediction:

© M.Eng Students
Sidra Hussain 1318131
Jagdeep Singh 1317679
Vishakha Babulal 1324497

Figure 9 GUI Homepage

If the user decides to train a model from their own dataset, they use the option *"Train"*. Clicking on Train button takes to the Training page (see fig. 10), where it asks the user to input dataset of Object 1 and Object 2 in excel format. It also asks for a test data file to test the model on after training. Using all these inputs from user, training and testing is performed which is explained in section Vb. Two classification models (Random Forest and Decision Tree) are trained and saved with .sav files on the disk.


Reflected Sound Signal Discriminator

Training

Choose the combined excel file of Object1 : /Users/jagdeepsingh/Projects/

Choose the combined excel file of Object2 : /Users/jagdeepsingh/Projects/

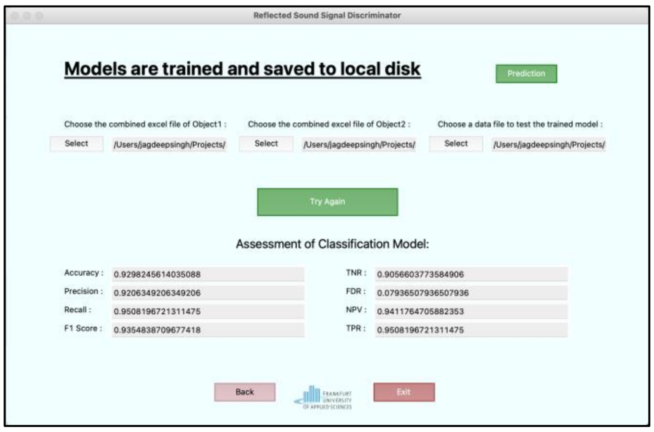
Choose a data file to test the trained model : /Users/jagdeepsingh/Projects/

Assessment of Classification Model:

Accuracy :		TNR :	
Precision :		FDR :	
Recall :		NPV :	
F1 Score :		TPR :	

Figure 10 GUI Model Training Page

Post training completion, notification and the classifier assessment are provided as shown in fig. 11. This assessment consists various classifier performance measures: Accuracy, Precision, Recall, F1 Score, TNR, FDR, NPV and TPR. These trained models are then used in performing the


Reflected Sound Signal Discriminator

Models are trained and saved to local disk

Choose the combined excel file of Object1 : /Users/jagdeepsingh/Projects/

Choose the combined excel file of Object2 : /Users/jagdeepsingh/Projects/

Choose a data file to test the trained model : /Users/jagdeepsingh/Projects/

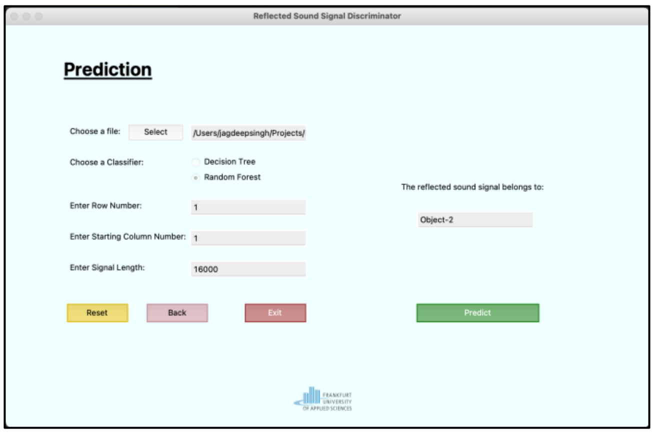
Assessment of Classification Model:

Accuracy :	0.9298245614035088	TNR :	0.9056603773584906
Precision :	0.9206349206349206	FDR :	0.07936507936507936
Recall :	0.9508196721311475	NPV :	0.9411764705882353
F1 Score :	0.9354838709677418	TPR :	0.9508196721311475

Figure 11 Training Completion

prediction which is on Prediction page of GUI. On training page, if the user doesn't choose the "*Train*" option, the code will select an already saved model from the options available on predict page.

In prediction page (see fig 12), the user is asked to input the test data file which needs to be predicted, option to select the desired classifier, row number and column number of the time signal and number of scan points as signal length. After getting all these inputs, Prediction button performs the prediction with the selected model and gives the output. This output provides whether selected time signal (reflected) from test data file belongs to either Object 1 or Object2.


Reflected Sound Signal Discriminator

Prediction

Choose a file: /Users/jagdeepsingh/Projects/

Choose a Classifier: Decision Tree [✓ Random Forest]

Enter Row Number: 1

Enter Starting Column Number: 1

Enter Signal Length: 16000

The reflected sound signal belongs to: Object-2

Figure 12 GUI Prediction Page

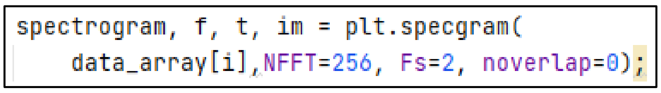
B. Data Conversion

The files received were labeled as Object 1 and Object 2 where Object 1 had total 240 signals and Object 2 had 330 signals. For the purpose of data mining, this raw large data received per object was compiled first in excel files that included all respective signals in respective rows i.e. For Object 1; 240 rows and Object 2; 330 rows.

C. Quadratic Time Frequency Representation

The Quadratic Time Frequency Representation as discussed in section II is a map of energy distributions of a signal. This type of representation also provides the frequency and time at the specific energy distribution.

The function used to generate such a spectrogram is:



```
spectrogram, f, t, im = plt.specgram(
    data_array[i],NFFT=256, Fs=2, noverlap=0);
```

Figure 13 Function to generate Spectrogram

It is imported from library matplotlib.pyplot. which retrieves the signal and according to the NFFT block, which is the number of data points mentioned to calculate FFT at the block, the spectrogram is calculated. The corresponding time and frequency values are returned as 1D arrays. One of the input parameters is the noverlap which is set to 0. It decides the number of data points between the blocks to set to overlap for the process of STFT to prevent spectral leakage. In this application we do not concern with a spectral leakage and set the noverlap to 0. We also decide the default value of data point block 256 for NFFT.

Following illustrates the spectrograms of a signal reflected by object 1 and object 2.

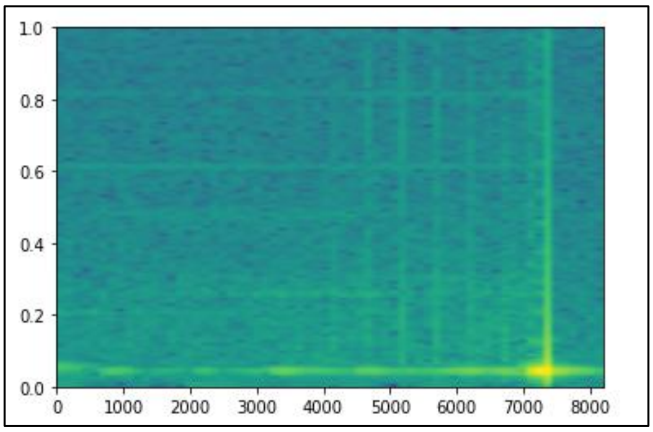


Figure 14 Spectrogram of signal 30 side2-1.6, object 1

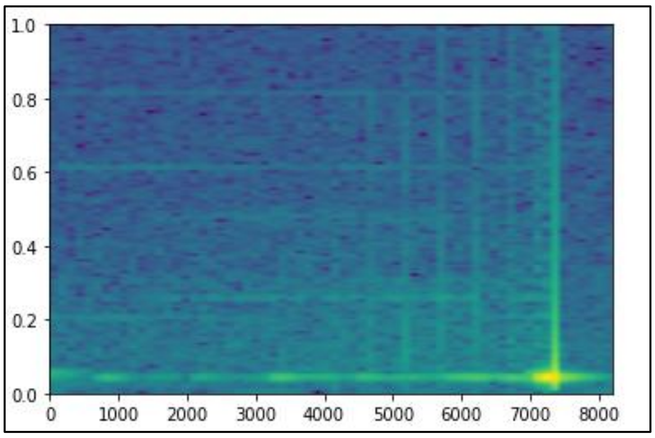
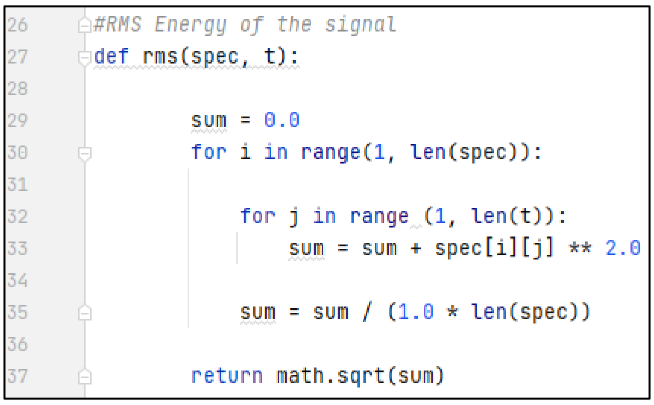


Figure 15 Spectrogram of signal 30, frontyellow- 1.6, object 2

D. Feature Extraction

The above spectrograms provides a 1D array of time, frequency and corresponding loudness or energy density of the signal at the corresponding time and frequency. Using this information per signal we extract the following features.

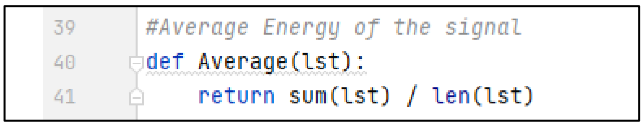
1) *Root Mean Squared Energy:* RMS or Root mean squared energy is the effective energy stored in the signal. It is simply the root squared of the mean of total energy density of the signal. It is useful for signals primarily that deviate from negative to positive values for e.g., a sinusoidal signal. It gives the map of effective energy stored in the waveform over time for such signals. It is calculated by summation of all data points in spectrogram and taking the root mean square of the value. The python math library is imported, and the function is processed over all data points of the spectrogram array. As the spectrogram is a 2D array, it traverses over two for loops to get the total sum of Sxx or Energy density.



```
#RMS Energy of the signal
def rms(spec, t):

        sum = 0.0
        for i in range(1, len(spec)):

            for j in range (1, len(t)):
                sum = sum + spec[i][j] ** 2.0

            sum = sum / (1.0 * len(spec))

        return math.sqrt(sum)
```

Figure 16 Function rms() for RMS energy feature extraction from spectrogram

2) *Average Energy:* The average energy is the first moment of the spectrogram Sxx values. It is simply the mean of total energy distribution across the signal.

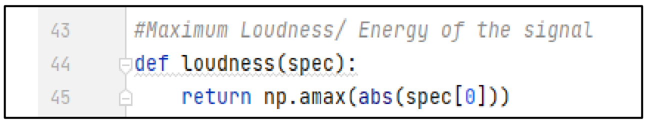


```
#Average Energy of the signal
def Average(lst):
    return sum(lst) / len(lst)
```

Figure 17 Function Average() for Average energy feature extraction from spectrogram

The function is straightforward and uses library math of python.

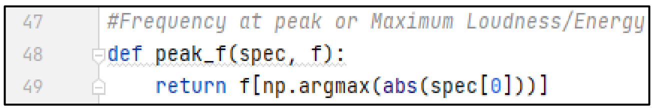
3) *Maximum Loudness:* Maximum Loudness or maximum energy density is the maximum point of energy distribution in the spectrogram. In our case it is the instant at which the sound signal hits the object and by the concept of acoustics the sound on point of impact has the highest energy distribution. This maximum energy density point is the reason echo takes place in acoustics which reflects back to the incident or receiver. In the project this point is found in the spectrogram Sxx energy density points array via using amax() function of library numpy.



```
#Maximum Loudness/ Energy of the signal
def loudness(spec):
    return np.amax(abs(spec[0]))
```

Figure 18 Function loudness() to extract maximum loudness from spectrogram

4) *Peak Frequency:* This feature corresponds to the peak frequency at the time of impact to the object. This frequency corresponds to the point of maximum loudness. It in turn provides a feature of the object property the signal is incident on. According to the surface of the object the energy lost at the point of impact will in turn affect the frequency of



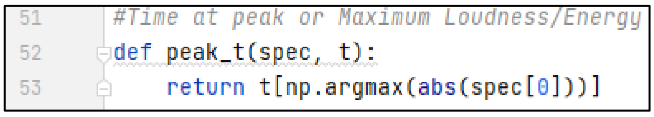
```python
#Frequency at peak or Maximum Loudness/Energy
def peak_f(spec, f):
    return f[np.argmax(abs(spec[0]))]
```

Figure 19 Function peak_f() to extract maximum peak at point of impact from spectrogram

the signal. Thus, this feature can refer to mapping the surface property of the object.

The function is straightforward and uses argmax which is an attribute of numpy arrays in python to find the index of maximum spectrum value, which in this case spectrum denotes Sxx value or energy density value in the spectrogram. As spectrogram is a 2D array the notation "[0]" allows traversal in all rows of the numpy array. At the index of maximum spectral value, the index of maximum frequency point is received as the peak frequency.

5) *Propagation Delay:* This feature corresponds to time taken by the signal to travel back from the point of reflection to the point of transmission or vice versa It is simply calculated by noting the time of impact or time of maximum energy density or time at which peak frequency occurs. The distance of the object from the transmission of signal can help in differentiating the object.

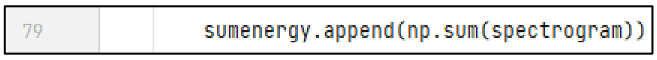


```python
#Time at peak or Maximum Loudness/Energy
def peak_t(spec, t):
    return t[np.argmax(abs(spec[0]))]
```

Figure 20 Function peak_t() to extract propagation delay from point of reflection to point of receiver

Like peak frequency the index of maximum spectral density is noted and at that index the time of propagation delay is stored.

6) *Total Energy:* This feature corresponds to the total energy of the signal waveform. The feature was used to take in account any energy loss at point of impact according to the surface of the object. At different surfaces signals will lose different amount of energies. This feature can successfully help to differentiate the object the signal impacts according to the energy lost as total energies of signals will be different which reflected on different objects.



```python
        sumenergy.append(np.sum(spectrogram))
```

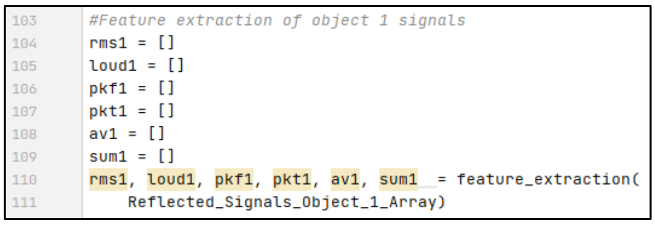
Figure 21 Total Energy from sum of spectral values

The summation is simply applied by using the inbuilt sum function from python library math and using the spectrum or sxx values from spectrogram as numpy array.

E. *Data Mining*

Data mining is the step when the final data frame to feed to the classifier is made. All the features that are discussed are run in a loop till the number of total signals. That is 240 for Object 1 signals and 330 for Object 2 signals. All these values are stored in array features by running a loop till the number of total signals.

The features discussed above are called in a single function that takes the object array file or test array file in its input argument. Fig 22 shows feature_extraction() called for object1 array.

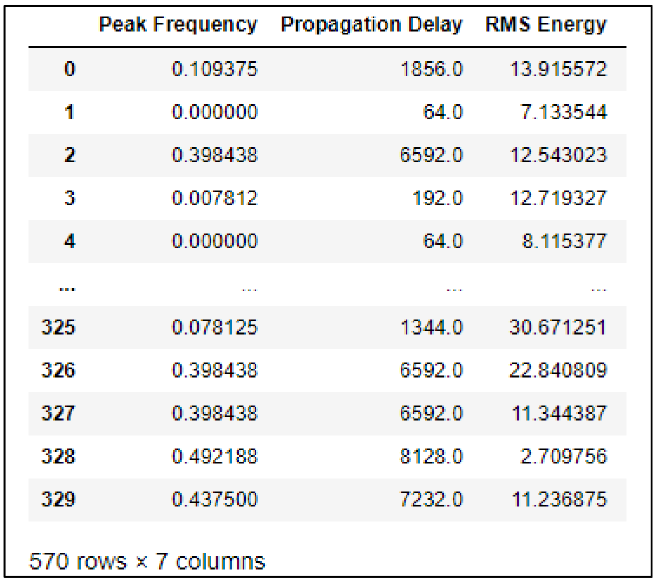


```python
#Feature extraction of object 1 signals
rms1 = []
loud1 = []
pkf1 = []
pkt1 = []
av1 = []
sum1 = []
rms1, loud1, pkf1, pkt1, av1, sum1 = feature_extraction(
    Reflected_Signals_Object_1_Array)
```

Figure 22 feature_extraction() to call all features functions

The corresponding values of features of each object are then concatenated and made into separate data frames. These data frames are then concatenated together to make a final data frame for our classifier to train on.

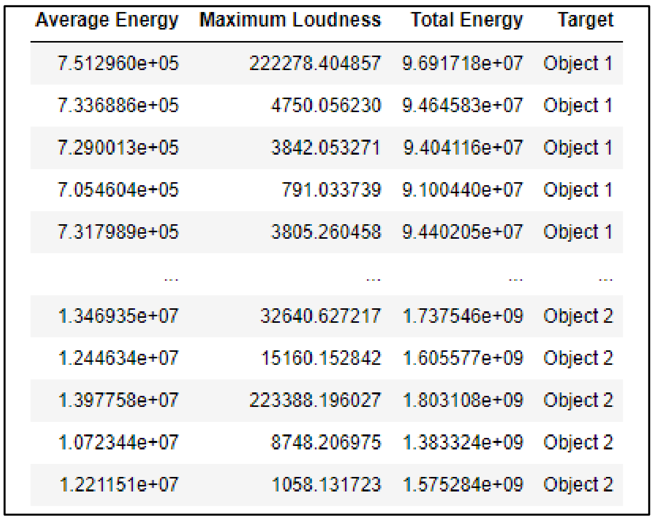
The following two images are one data frame split in two images.



	Peak Frequency	Propagation Delay	RMS Energy
0	0.109375	1856.0	13.915572
1	0.000000	64.0	7.133544
2	0.398438	6592.0	12.543023
3	0.007812	192.0	12.719327
4	0.000000	64.0	8.115377
...	...	...	...
325	0.078125	1344.0	30.671251
326	0.398438	6592.0	22.840809
327	0.398438	6592.0	11.344387
328	0.492188	8128.0	2.709756
329	0.437500	7232.0	11.236875

570 rows × 7 columns

Figure 23 Final half data frame 1

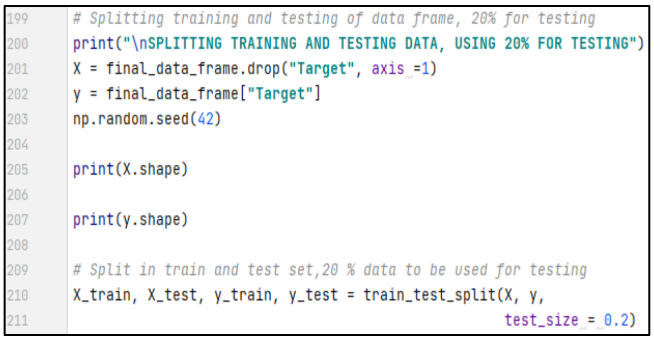


Average Energy	Maximum Loudness	Total Energy	Target
7.512960e+05	222278.404857	9.691718e+07	Object 1
7.336886e+05	4750.056230	9.464583e+07	Object 1
7.290013e+05	3842.053271	9.404116e+07	Object 1
7.054604e+05	791.033739	9.100440e+07	Object 1
7.317989e+05	3805.260458	9.440205e+07	Object 1
...	...	...	...
1.346935e+07	32640.627217	1.737546e+09	Object 2
1.244634e+07	15160.152842	1.605577e+09	Object 2
1.397758e+07	223388.196027	1.803108e+09	Object 2
1.072344e+07	8748.206975	1.383324e+09	Object 2
1.221151e+07	1058.131723	1.575284e+09	Object 2

Figure 24 Final half data frame 2 cont.

The model training is achieved by using scikit-learn module in python. The module consists of two functions called fit() and predict().

Before learning, the data frame is split into a training and a testing set. For the slitting of data another module called sklearn is used. It is also used further to retrieve our classifier modules. The Classifier algorithms which are loaded and used in this project are Decision Tree and Random Forest.



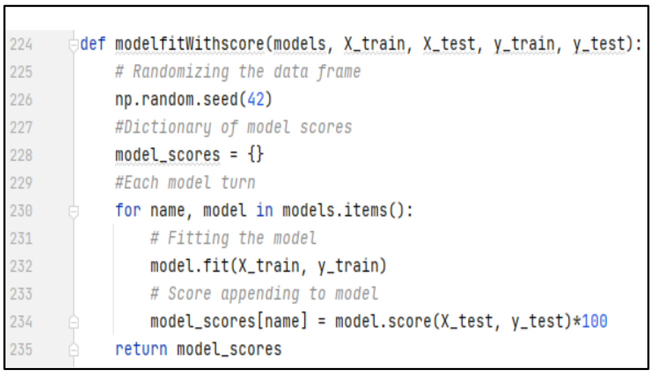
```
# Splitting training and testing of data frame, 20% for testing
print("\nSPLITTING TRAINING AND TESTING DATA, USING 20% FOR TESTING")
X = final_data_frame.drop("Target", axis =1)
y = final_data_frame["Target"]
np.random.seed(42)

print(X.shape)

print(y.shape)

# Split in train and test set,20 % data to be used for testing
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.2)
```

Figure 25 Splitting the training and testing data

The above line of code, imports the module sklearn.model and the train_test_split() function. It easily splits the data values X in test and training data. Similarly, it applies this to the data targets stored in y. A randomizer is used in between to pick random data points from the data frame for each of the training and testing data sets. In the above line of code 20% data is used for testing and 80% for training of the classifier.

Next the training of the classifier begins. The fit() function intakes the values referred to as 'X_train' or x data values and data targets as 'y_train' or y labels to fit the training instances against each other. The data provided to the fit() is the training data.

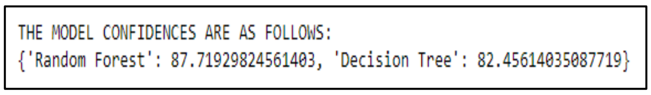
The following line of code in Fig. 26, the function retrieves the trained and fits the model or learns on the training data. Line 6 of the code above depicts that the learned model then receives testing data to apply on the models used. It saves the score of classification.



```
def modelfitWithscore(models, X_train, X_test, y_train, y_test):
    # Randomizing the data frame
    np.random.seed(42)
    #Dictionary of model scores
    model_scores = {}
    #Each model turn
    for name, model in models.items():
        # Fitting the model
        model.fit(X_train, y_train)
        # Score appending to model
        model_scores[name] = model.score(X_test, y_test)*100
    return model_scores
```

Figure 26 fit() function to learn the models according to the dataset and retrieving model scores

The following are the model scores, or accuracies received by the trained models on algorithms Random Forest and Decision.



```
THE MODEL CONFIDENCES ARE AS FOLLOWS:
{'Random Forest': 87.71929824561403, 'Decision Tree': 82.45614035087719}
```

Figure 27 Model Scores

VI. Steps to Rebuild the Solution

In order to recreate the provided solution, following steps must be performed:

- The solution is created using python version 3.7.3. So, the same version or higher is required. Also, pip is required to install the used packages
- All the packages required for the solution are specified in the *"requirement.txt"* file. To install these packages, run the pip command as: `$pip install -r requirements.txt`
- Firstly, to convert the time signal data files from .csv to merged excel format, *"dataset.py"* file needs to be run using `$python3 dataset.py.` This will provide the combined excel files for time signals of Object1 and Object2 respectively to be used in *app.py* and *main.py*. The code gives an option for creating a *test_obj.xlsx* file of any object as well which is mandatory for further *app.py* and *main.py* classifier assessment results. If the user wishes to not get their choice of *test_object* excel, *test_obj* of object 1 is already present in the directory.
- The *"main.py"* contains the code for processing the data, training, and testing of classifiers. It is a source code without GUI. And *"app.py"* is the source code with GUI. Either codes can be run accordingly using the command: `$python3 "file".py`

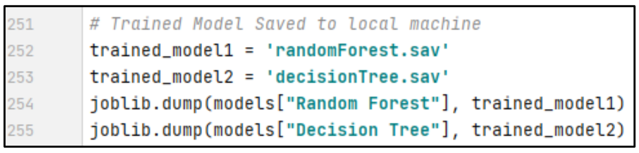
VII. Result Analysis

The following section is divided into two parts. The first section discusses the implementation and result of the classifier on a test signal. The second section discusses the classifier assessments in detail.

A. *Test signal classification*

In the following section we will test the classifier. The test (if the user did not change the file using dataset.py functionality) is carried out on object 1 time signal:

1. To start the testing of the classifier, the trained model is first saved on the local machine as a .sav file. In this project, module joblib is used for importing.

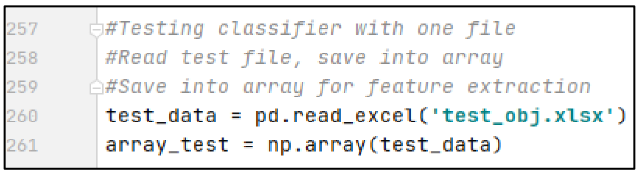


```
# Trained Model Saved to local machine
trained_model1 = 'randomForest.sav'
trained_model2 = 'decisionTree.sav'
joblib.dump(models["Random Forest"], trained_model1)
joblib.dump(models["Decision Tree"], trained_model2)
```

Figure 28 Saving of trained models using joblib module

2. The second step is to receive the signal to implement the test on. A simple csv read followed by excel write is used to storing the raw data to be classified. In our project all .csv files are converted to excels beforehand. This test

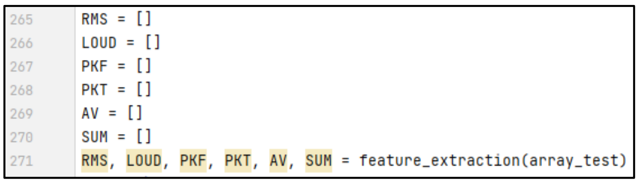
excel file is read into a numpy array to perform feature extraction for our classifier.



```
#Testing classifier with one file
#Read test file, save into array
#Save into array for feature extraction
test_data = pd.read_excel('test_obj.xlsx')
array_test = np.array(test_data)
```

Figure 29 Test data in array for feature extraction

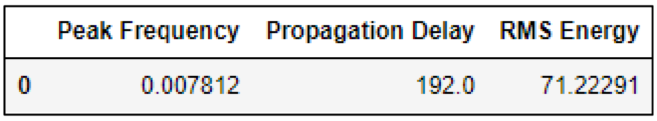
section V is taking place on the spectrogram of data. The following code shows the calculation of the features altogether using the function feature_extraction() on the spectrogram calculated.



```
RMS = []
LOUD = []
PKF = []
PKT = []
AV = []
SUM = []
RMS, LOUD, PKF, PKT, AV, SUM = feature_extraction(array_test)
```

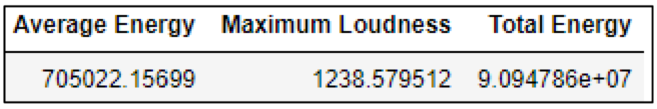
Figure 30 Spectrogram and feature extraction of test file

4. After the features are calculated they are compiled in a simple data frame like the data frame of the model to perform classification. Following is the resultant test signal data frame.



	Peak Frequency	Propagation Delay	RMS Energy
0	0.007812	192.0	71.22291

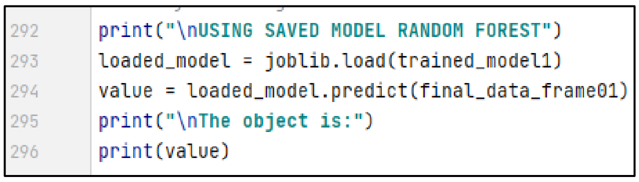
Figure 32 Test signal data frame



Average Energy	Maximum Loudness	Total Energy
705022.15699	1238.579512	9.094786e+07

Figure 33 Test signal data frame (contd.)

5. The next step includes putting the test signal data frame to test with the trained model. Using predict() function of scikit-learn, the data is retrieved and corresponding to the model label of the given data given are learnt. That is, predict function takes the new X values and predicts the y labels by using the trained model. The following is illustrated in the figure. The classifier in console dictates the result of 'Object 2' as shown below.

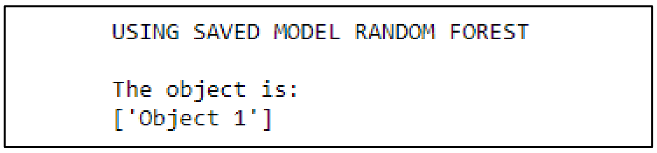


```
print("\nUSING SAVED MODEL RANDOM FOREST")
loaded_model = joblib.load(trained_model1)
value = loaded_model.predict(final_data_frame01)
print("\nThe object is:")
print(value)
```

Figure 35 Test signal prediction of object

B. *Classifier Assessment*

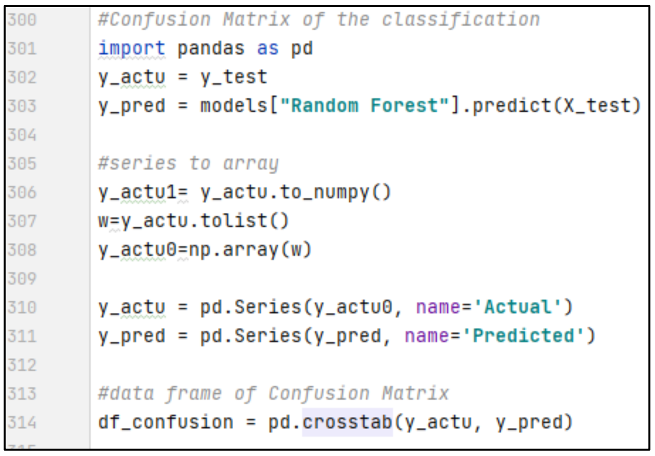
This section includes computing confusion matrix and other successor assessments important to gauge classifier performance.



```
USING SAVED MODEL RANDOM FOREST

The object is:
['Object 1']
```

Figure 33 Test Signal classification result

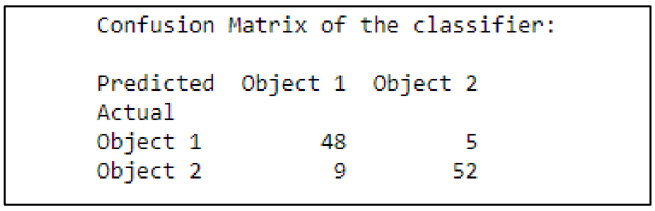
1. *Confusion Matrix:* Three confusion matrices are computed in this project using a data frame approach. The data frames are then plotted in a box diagram.
   a) *Normal Confusion matrix*



```
#Confusion Matrix of the classification
import pandas as pd
y_actu = y_test
y_pred = models["Random Forest"].predict(X_test)

#series to array
y_actu1= y_actu.to_numpy()
w=y_actu.tolist()
y_actu0=np.array(w)

y_actu = pd.Series(y_actu0, name='Actual')
y_pred = pd.Series(y_pred, name='Predicted')

#data frame of Confusion Matrix
df_confusion = pd.crosstab(y_actu, y_pred)
```

Figure 36 Confusion Matrix implementation

In Fig. 35 to make confusion matrix, the data labels y_pred which is the result by predicting is converted into a 1D array using numpy array. This aids in making a data frame for confusion matrix to print it and plot it in a color map as illustrated below.



```
Confusion Matrix of the classifier:

Predicted  Object 1  Object 2
Actual
Object 1         48         5
Object 2          9        52
```

Figure 31 Test Signal classification result

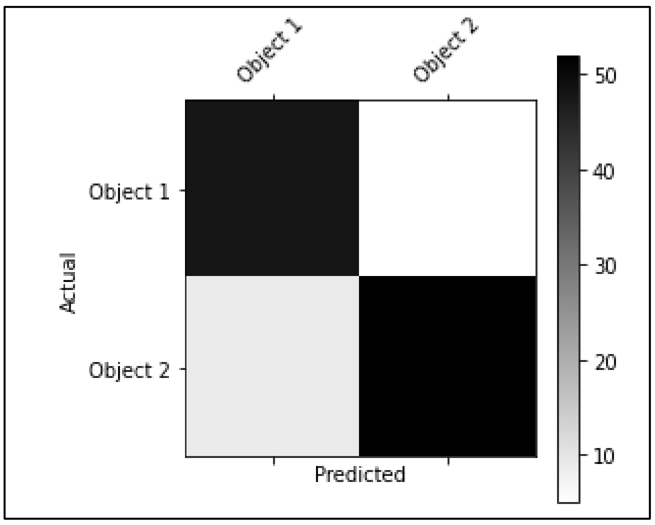
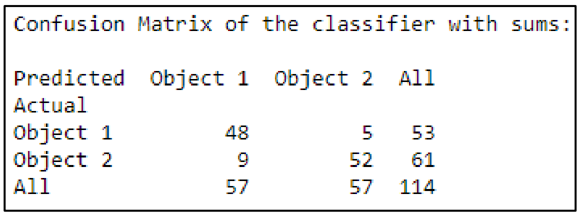


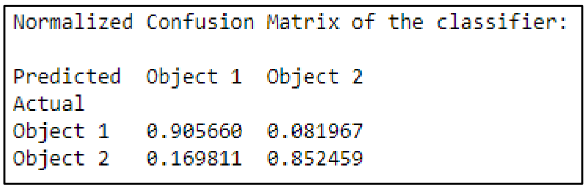
Figure 34 Confusion Matrix color map



```
Confusion Matrix of the classifier with sums:

Predicted  Object 1  Object 2  All
Actual
Object 1         48         5   53
Object 2          9        52   61
All              57        57  114
```

Figure 37 Confusion matrix with sums

c) *Normalized Confusion Matrix*



```
Normalized Confusion Matrix of the classifier:

Predicted  Object 1  Object 2
Actual
Object 1   0.905660  0.081967
Object 2   0.169811  0.852459
```

Figure 39 Normalized Confusion matrix

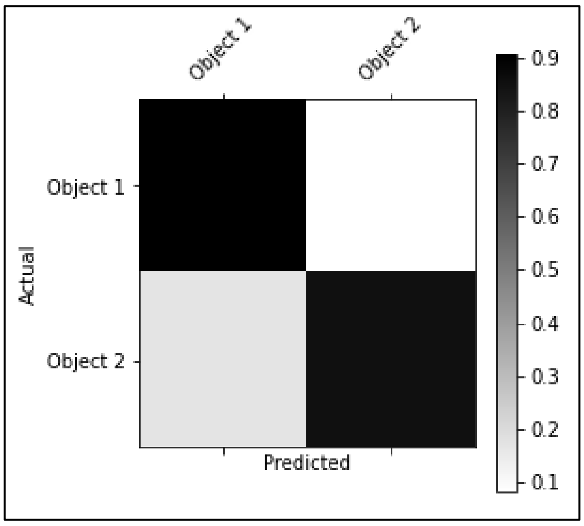
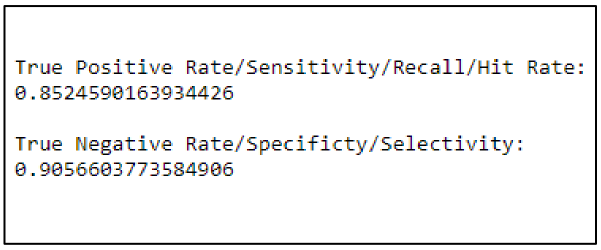


Figure 40 Normalized confusion matrix color map

2. *Sensitivity vs. Specificity:* As discussed in section IV, True Positive Rate (TPR) is the sensitivity of the classifier and True Negative Rate (TNR) is the specificity of the classifier. For a good classifier both values should be high.

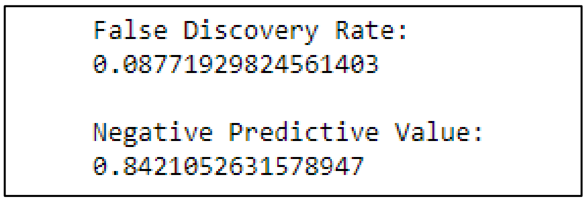
   Sensitivity shows how many positive results are genuinely positive. Specificity shows how many negative results are genuinely negative. For the implemented classifier both values are high.



```
True Positive Rate/Sensitivity/Recall/Hit Rate:
0.8524590163934426

True Negative Rate/Specificty/Selectivity:
0.9056603773584906
```

Figure 41 Sensitivity and Specificity of the classifier

3. *False Discovery Rate (FDR) and Negative Predictive Value (NPV):* The FDR is the proportion of results/discoveries that are false and NPV is the probability that the sample that returns a negative result really is negative. For a good classifier, NPV has to be high whereas an FDR has to be as low as possible.



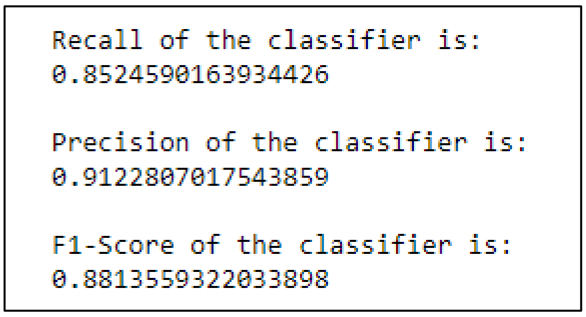
```
False Discovery Rate:
0.08771929824561403

Negative Predictive Value:
0.8421052631578947
```

Figure 38 False Discovery Rate and Negative Predictive Value of classifier

   Following are the computed values. The classifier shows a good classification result as FDR is very low and NPV is high enough.

4. *Recall, Precision and F1 score:* Recall of the classifier is the True Positive Rate. Recall shows out of all positive classes, how many predicted classes are correct. Precision shows out of all the positive predicted classes, how many are correct. These values should be high. But at times, for classifiers, according to the application these values differ and to assess a classifier a new value is used called the F score. It is the geometric mean of recall and precision and shows how far along is recall to precision.

   For the implemented classifier, recall and precision are high where precision is a bit higher. This shows a good classification result. With precision very high, that shows a classifier is more accurate. In the classifier implemented the recall and precision are a good trade off.



```
Recall of the classifier is:
0.8524590163934426

Precision of the classifier is:
0.9122807017543859

F1-Score of the classifier is:
0.8813559322033898
```

Figure 42 Recall, Precision and F1 score of the classifier

VIII. Conclusion

Figure 43 shows a comparison to the classifier assessment rates using bar chart.

The comparison depicts that false predictions are very low and true predictions are comparatively very high.

The ROC curve of the classifier (see fig 44) shows a good classification result, that is, the true positive probabilities are much higher than the false probabilities. The Area Under the Curve (AUC) shows the classifier separability in classes ability is 94%. That is 94% it can distinguish between the classes correctly.

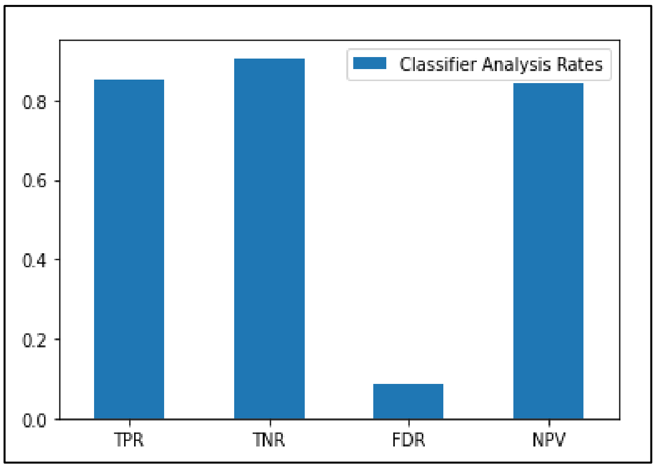


Figure 43 Classifier assessment rates comparison

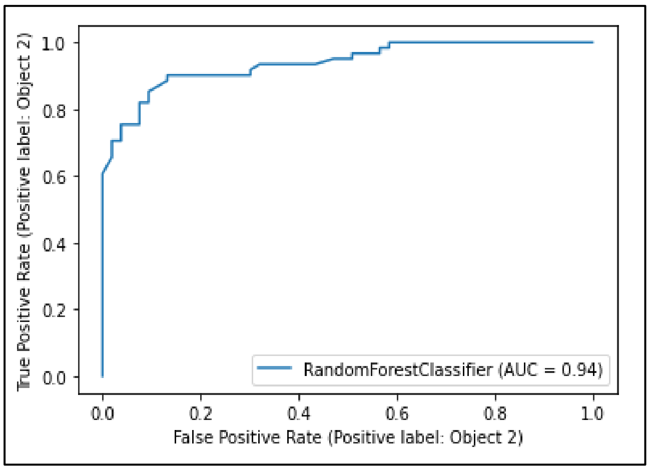


Figure 44 ROC Curve of classification

The following results depict the implementation of a good classifier that discriminates between the objects correctly at which the signal data is incident on.

Concluding, the paper put forward in detail; the significance of using Quadratic Time-Frequency Representations for acoustic signals and the approach of using Spectrograms, Binary Classification algorithms namely Decision Tree and Random Forest and classifier assessments. The classification algorithm discussed are then implemented, tested, assessed, and then integrated on a Graphical User Interface for user convenience. For future work on this topic, Wigner Ville distribution, Choi Williams Distribution, or Gabor Spectrogram of the signals are suggested to implement instead of spectrograms to see any probable convenient or faster feature extractions.

ACKNOWLEDGMENT

We would like to thank Prof. Dr. Andreas Pech for providing us the opportunity to work on this topic. This project helped us dive deep into Machine Learning formally, and we will take the learnings of this project ahead in the future. The credit for our findings goes to Professor Pech, and we extend our utmost gratitude towards him.

Table 1 Contribution

Section	***Title (document / program)***	***Contributor***	***Page No***
	MATLAB Wigner Ville Tests (prog.)	Vishakha	
I-II	Introduction /QTFR (doc.)	Vishakha	1
III	Binary Classification (doc.)	Jagdeep	2
IV	Classifier Assessment (doc.)	Jagdeep	4
	Initial GUI (tests and reruns)	Vishakha	
Va	Final GUI (doc. + prog.)	Jagdeep	5
Vb-Vf	Implementation (doc. + prog.)	Sidra	5
VII	Result Analysis (doc. + prog.)	Sidra	8
VIII	Conclusion (doc. + prog.)	Vishakha/ Sidra/ Jagdeep	10

REFERENCES

[1] M. J. Bianco, P. Gerstoft, J. Traer, E. Ozanich, M. A. Roch, S. Gannot and C.-A. Deledalle, "Machine learning in acoustics: a review," *Journal of the Acoustical Society of America*, vol. 146, no. 5, pp. 590-3628, 2019.

[2] R. Trifonov, D. Gotseva and V. Angelov, "Binary classification algorithms," *International Journal of Development Research*, vol. 7, no. 11, pp. 16873-16879, 2017.

[3] J. B. Manresa, *Time-frequency analysis, adaptive filtering and source separation*, Signals and Systems FI-UNER, Aalborg University, Aalborg, Den: Class Lecture, Feb 15, 2011.

[4] LabVIEW 2010 Advanced Signal Processing Toolkit Help, "Understanding Quadratic Time Frequency Analysis Methods (Advanced Signal Processing Toolkit)," 06 2010. [Online]. Available: https://zone.ni.com/reference/en-XX/help/371419D-01/lvasptconcepts/tfa_quadratic/. [Accessed 18 04 2021].

[5] R. Trifonov, D. Gotseva and V. Angelov, "BINARY CLASSIFICATION ALGORITHMS," *International Journal of Development Research*, vol. 07, no. 11, pp. 16873-16879, 2017.

[6] A. Navlani, "Decision Tree Classification in Python," 28 12 2018. [Online]. Available: https://www.datacamp.com/community/tutorials/decision-tree-classification-python. [Accessed 09 04 2021].

[7] Y.-y. Song and Y. Lu, "Decision tree methods: applications for classification and prediction," *Shanghai Arch Psychiatry*, vol. 27, no. 2, pp. 130-135, 2015.

[8] L. Breiman, "Random Forests," in *Machine Learning 45*, https://doi.org/10.1023/A:1010933404324, 2001, p. 5–32.

[9] C. Lui and K. Pareek, "Random Forest-based Voice Activity Detector," Unpublished, New York University, 2018.

[10] L. Ceriani and P. Verme, "The origins of the Gini index: extracts from Variabilità e Mutabilità (1912) by Corrado Gini. J Econ Inequal," vol. 10, p. 421–443, 2012.

[11] A. Sarica, A. Cerasa and A. Quattrone, "Random Forest Algorithm for the Classification of Neuroimaging Data in Alzheimer's Disease: A Systematic Review," 06 10 2017. [Online]. Available: https://doi.org/10.3389/fnagi.2017.00329. [Accessed 09 04 2021].

[12] Anurag, "Random Forest Analysis in ML and when to use it," 17 08 2018. [Online]. Available: https://www.newgenapps.com/blog/random-forest-analysis-in-ml-and-when-to-use-it/. [Accessed 09 04 2021].

[13] S. Narkhede, "Understanding Confusion Matrix," 09 05 2018. [Online]. Available:

https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62. [Accessed 04 10 2021].

[14] Google Developers, "Classification: ROC Curve and AUC," 10 02 2020. [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc. [Accessed 10 04 2021].

[15] W. Richardson and W. Lieurance, "Random Forests," 27 11 2019. [Online]. Available: https://kevintshoemaker.github.io/NRES-746/RandomForests.html. [Accessed 09 04 2021].

[16] A. Lineberry, "Exploring Random Forest Internal Knowledge and Converting the Model to Table Form," 20 09 2018. [Online]. Available: https://www.spotx.tv/resources/blog/developer-blog/exploring-random-forest-internal-knowledge-and-converting-the-model-to-table-form/. [Accessed 10 04 2021].