Discrimination of Reflected Sound Signals using Multilayer Perceptron

M.Eng - Information Technology
Course - Machine Learning
Pof. Dr. Andreas Pech

Anik Biswas (1324853)
anik.biswas@stud.fra-u as.de

Lakshmi Mounika Kolisetty (1324400)
lakshmi.kolisetty@stud.fra-uas.de

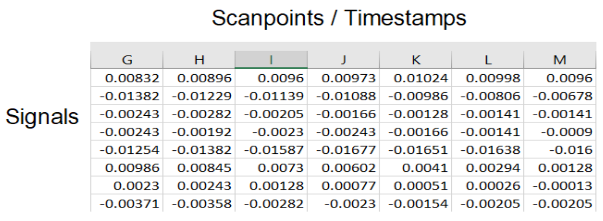
Bimal Kumar Sah (1327805)
bimal.sah@stud.fra-uas.de

Abstract—If an acoustic wave is incident on an object, certain sound signal is expected to be reflected from the surface of the object. The reflected signal is a convolution of the incident wave and the reflective properties of the surface. Measurement of this reflected signal, which is a time series, and analysis of the same can be used to determine the identification of the object. Classification is the process of categorizing different sets of data points into specific classes. Classification of time series data is a research problem for which several Machine Learning approaches can be used. As a prerequisite of this project, several measurements of reflected time signals in different timestamps have been taken from laboratory experiments and the dataset is provided. The dataset consists of reflected time signal measurement of three objects named Object #1, Object #2 and Object #3. This project uses Multilayer Perceptron based Machine Learning approach to build a classifier that would predict which of the three objects the signal is reflected from when a new set of time series data is provided.

Keywords – Acoustic Wave, Reflected Signal, Time Series, Classification, Machine Learning, Multilayer Perceptron

I. Introduction

Time Series Data consists of collection of repeated measurements over time. Usually, measurements are carried out from same source over a time interval. Measurement of reflected sound signal from an object over a number of timestamps is an example of time series data. Signals reflected from different object surface manifests different sets of patterns and characteristics because reflected signal can be considered as a convolution of incident wave and surface property. In this project, we were provided with datasets containing laboratory measurements of time signals reflected from three objects named Object 1, Object 2 and Object 3.


Scanpoints / Timestamps

Signals

G	H	I	J	K	L	M
0.00832	0.00896	0.0096	0.00973	0.01024	0.00998	0.0096
-0.01382	-0.01229	-0.01139	-0.01088	-0.00986	-0.00806	-0.00678
-0.00243	-0.00282	-0.00205	-0.00166	-0.00128	-0.00141	-0.00141
-0.00243	-0.00192	-0.0023	-0.00243	-0.00166	-0.00141	-0.0009
-0.01254	-0.01382	-0.01587	-0.01677	-0.01651	-0.01638	-0.016
0.00986	0.00845	0.0073	0.00602	0.0041	0.00294	0.00128
0.0023	0.00243	0.00128	0.00077	0.00051	0.00026	-0.00013
-0.00371	-0.00358	-0.00282	-0.0023	-0.00154	-0.00205	-0.00205

Figure 1 Dataset Sample of Measured Reflected Signal

The datasets are tabulated in excel format where each row represents different reflected signals and each column represents same instance of time (timestamp) which can also be considered as scan points. The task is to build a classification mechanism that would be able to classify these three types of reflecting signal emitting from respective object. This classifier would subsequently be able to make prediction about the object source of totally new but similar type of time signal with reasonable accuracy. There are a few methods dedicated for Time Series classification and prediction. One of the popular choices for this purpose is Artificial Neural Network (ANN) which are computing algorithm that loosely models the neurons in biological brain. Within ANN, a large range of different architecture is available and of them Multilayer Perceptron (MLP) remains the most popular for modelling and forecasting Time Series [1]. In this project we have taken up Multilayer Perceptron approach to build the Machine Learning (classifier) model. The following sections briefly discusses about theoretical background, project implementation and result obtained from the experiment.

II. Multilayer Perceptron and Time Series Forecasting

As discussed earlier, Multi Layer Perceptron (MLP) is a type of ANN where neurons are grouped in layers and only forward connections exist. A typical MLP consists of an input layer followed by one or several hidden layers and output layers, including neurons, weights and a transfer functions [2]. Signals are usually transmitted in one direction throughout the network: from input to output. There is no loop since each neuron's output has no effect on the neuron itself. This architecture is called feedforward [3].

Each neuron (noted i) transforms the weighted sum (weight w_{ij}, bias b_i) of inputs (x_j) into an output y_i using a transfer or activation function (f). The output can be described with the following equation [3]:

$$y_i = f(\sum_{j=1}^{n} x_j w_{ij} + b_i) \qquad (1)$$

We can combine output y_i from all the neurons together and get the resultant output y as

$$y = f(W.x + b) \qquad (2)$$

In the above model, if we have one single input layer and subsequently one output layer, it is called Perceptron. A Multi Layer Perceptron (MLP) is a composition of an input layer, at least one hidden layer and an output layer.

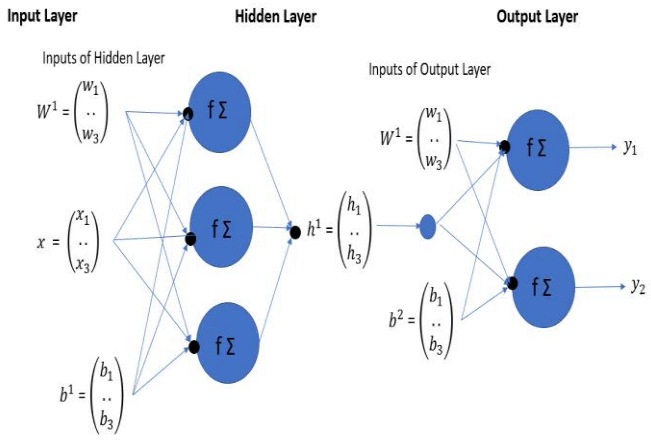

Input Layer
Hidden Layer
Output Layer
Inputs of Hidden Layer
Inputs of Output Layer
$W^1 = \begin{pmatrix} w_1 \\ .. \\ w_3 \end{pmatrix}$
$x = \begin{pmatrix} x_1 \\ .. \\ x_3 \end{pmatrix}$
$b^1 = \begin{pmatrix} b_1 \\ .. \\ b_3 \end{pmatrix}$
$h^1 = \begin{pmatrix} h_1 \\ .. \\ h_3 \end{pmatrix}$
$b^2 = \begin{pmatrix} b_1 \\ .. \\ b_3 \end{pmatrix}$
f Σ
y_1
y_2

Figure 2 MLP with one Hidden Layer

The calculations are the same as for a Perceptron, but there are now more hidden layers to integrate before the output y is reached.

$$h^1 = f(W^1.x + b^1)$$

$$y = f(W^2.h^1 + b^2) \tag{4}$$

As depicted in Figure 3, a perceptron with a single layer and one input generates classification regions in semi planes. By increasing the layers, each neuron acts as a standard perceptron for the outputs of the neurons in the preceding layer. This enables the output of the network to estimate further decision regions which are the result of the intersection of semi planes generated from previous neurons [3].

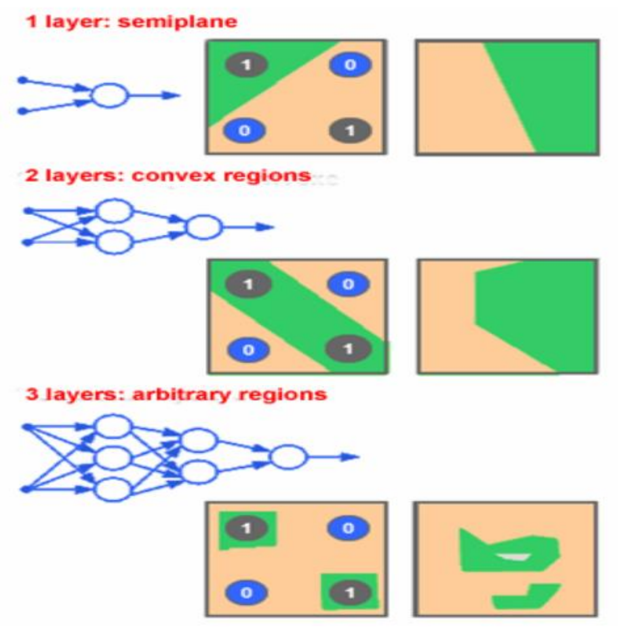


Figure 3 Decision Regions of multilayer perceptrons [3]

The Activation functions determine the output of a neural network i.e., if a neuron should be activated by calculating the weighted sum and adding bias with it. It is found that linear activation functions do not provide any improvement in computing power of multi layer network. Some of the popular choice of non-linear activation function used in multilayer perceptron have been sigmoid or logistic function, hyperbolic tangent function etc. More recently rectified linear unit (ReLU) activation function has been shown to yield impressive results.

The ReLU function can be mathematically defined as:

$$f(x) = \max(0, x) \tag{5}$$

As per the definition. It can be intuitively observed that ReLU function retains only the positive elements and discards all the negative elements by setting those to 0. For negative input the derivative of the ReLU function is zero and for positive entity it is 1. The main reason for using ReLU is that its gradient computation is very easy (either 0 or 1) and it helps in better optimization achievement [4].

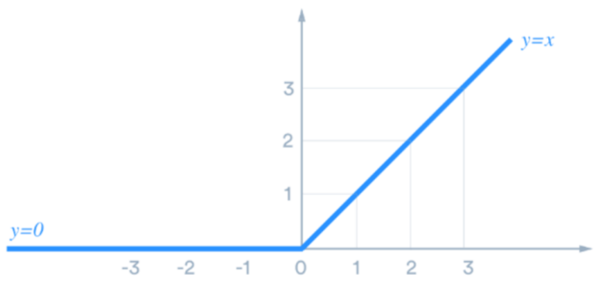


Figure 4 ReLU Activation Function

To optimize the trainable parameter W and b, backpropagation algorithm is implemented. The following error function has to be minimized

$$E = |y - \bar{y}|^2 \tag{6}$$

Here y is the target output of the output layer and $\bar{y}$ is the perceptron output. For minimization, sequential gradient descent is used. In this technique, the cost function is observed as a function of the trainable parameters w such as W and b. Furthermore, the newly optimized parameter can be initialized as

$$W^{\sim} = W - \eta \nabla_E(W) \tag{7}$$

which moves towards the global minimum. The parameter η is the learning rate of the ANN. [5]

The feature of ANN and subsequently MLP can be effectively used for time series forecasting. The design of the MLP algorithm depends on the nature of the Time Series problem imposed. Based on the dataset, two types of Time Series can be categorised which are Univarate Time series and Multivarate Time series. Univarate Time series implies dataset comprising single time-dependent variable.

Past values from the dataset will help extract a pattern and determine a prediction model. Multivariate time series has more than one time-dependent variable. Each variable depends on its past values and as well as on other variables. This dependency is used to predict future outcomes [6]. In our project we are provided with the dataset which contains only single variable time dependent observations which is an example of univariate time series. In the subsequent sections, detailed algorithm and implementation approach of this time series forecasting model is discussed.

III. Project Implementation

A basic Flow chart of the Project Implementation is prepared as below :

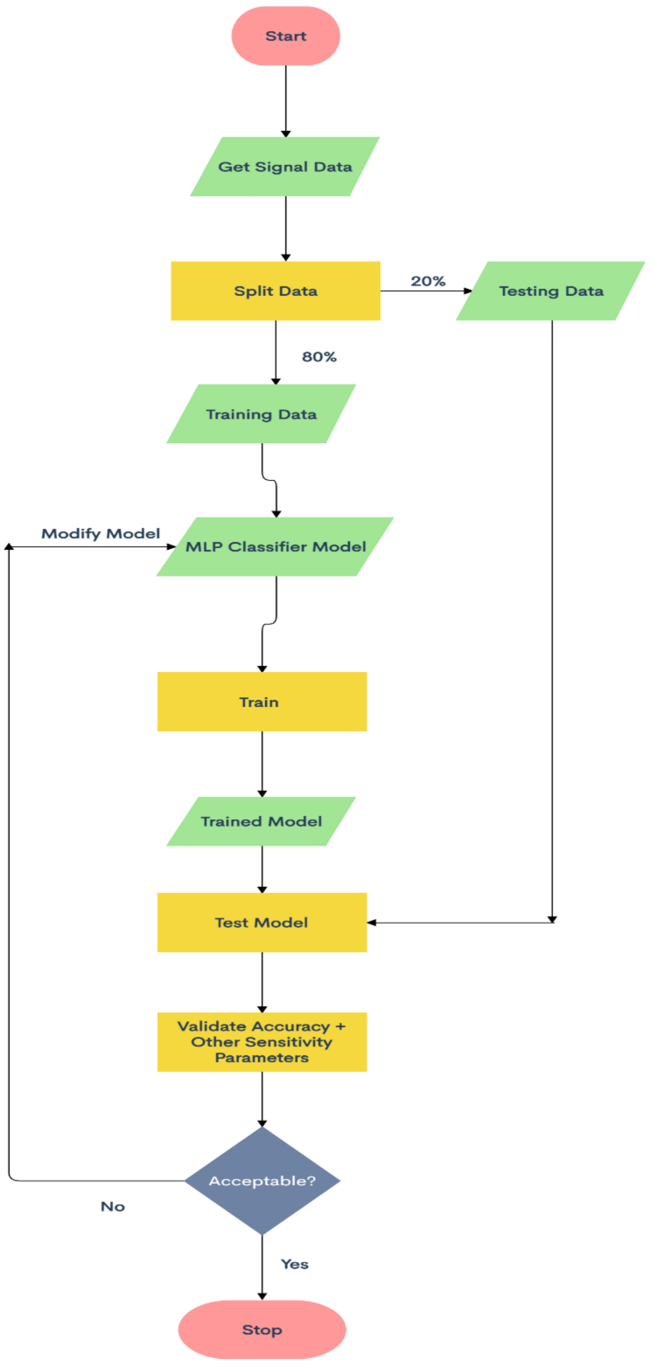


Figure 5 Classifier Implementation Flowchart

Environment Setup: For simplicity, consistency, flexibility, platform independence and access to great ML libraries we have implemented our project in Python language. Anaconda Installer for Python 3.8 has been used to install the necessary libraries. PyCharm 2020.3.4 has been used as an Integrated Development Environment (IDE).

Code Implementation: The overall implementation procedure of this specific MLP classifier model can be divided into the following sections:

- **A) Graphical User Interface**
- **B) Data Preparation**
- **C) MLP Classifier Training Model**
- **D) Prediction and Performance Validation**
- **E) Testing**

A) Graphical User Interface: The Graphical User Interface (GUI) is designed for the purpose of input and output handling. We have used Tkinter which is a Python binding of cross platform GUI toolkit Tk. The GUI has been designed to cater Input-Output (I/O) for both Training and Testing operation in a single window. The following modules are imported in the python source code:

```
import tkinter as tk

from tkinter import *

from tkinter import messagebox

from tkinter import filedialog
```

To start tkinter, we built a Tk root widget, which is a window with a title bar and other window manager-provided decoration. The root widget had to be generated before all other widgets, and only one root widget can exist.

```
root = tk.Tk()

root.title("Discrimination of Reflected
Sound Signal(MLP Classifier)")
```

A Tkinter Canvas grid geometry manager is used to put the widget in a 2D Table. This widget is split into a number of rows and columns as indicated in the code below. It helps us place message box, buttons etc in particular cells in the widget and achieve good geometrical alignment in the GUI display.

```
canvas = tk.Canvas(root, width = 1000,
height = 1000)
canvas.grid(rowspan = 14, columnspan =
15)
```

In several places in the grid-space, Label widgets have been used aesthetically to put display labels. An example is below:

```
tk.Label(root, text = "Training the
Model", font = 'customFont1', fg =
"black", bg = "sky blue", width =
15).grid(columnspan = 1, row = 0, column
= 0)
```

Two custom functions **browse_folder()** and **browse_file()** has been implemented to fetch specific folder and file in the host machine from GUI tool respectively. A global variable **folder_path** is declared in tkinter string variable premise. This variable would store the path of the folder where our input files would be selected from. Both the function uses filedialog module from tkinter to open directory and file source respectively.

```
def browse_folder():
    global folder_path
    filename= filedialog.askdirectory()
    folder_path.set(filename)

def browse_file():
  global file_path
  filename=filedialog.askopenfilename()
  folder_path.set(filename)
```

Specific buttons are created for Browsing Folder/Files and perform Training/Testing operation. In the respective button implementations, desired functions have been called.
For example, as shown in the below code snippet, the button "**Browse Folder**" calls the function **browse_folder()** so that specific source directory can be selected.

```
browseBtn = tk.Button(root, text = "Browse
Folder", command = lambda:browse_folder(),
font = 'customFont1', bg = "azure", fg =
"black", height = 2, width = 15)

browseBtn.grid(row = 1, column = 2)
```

Similarly, message boxes with tkinter for different input and output display features like Starting and Ending Column, Accuracy etc have been implemented. One of the example code snippets is as below [7]:

```
start_column = tk.IntVar()

tk.Label(root, text = "Signal Starts at
Column : ", font = 'customFont1', fg =
"black", width = 20).grid(columnspan =
1, row = 2, column = 0)

signal_start = tk.Entry(root, width = 5,
textvariable = start_column).grid(row =
2, column = 1, sticky = "W")
```

Similarly, GUI implementation with Tkinter has been implanted in subsequent section of our implantation to plot Accuracy and other assessment scores like F1 score, NPV etc. Also, in a way similar to the above implementation, GUI option for testing new dataset from excel files have been implemented in the subsequent Testing part of the implantation. Since the approach is same, the detailed explanation of those specific implementation of GUI is not included in this documentation.

B) Data Preparation:

A function **main**() is implemented inside which importing dataset, data pre-processing and preparation is done. For importing the dataset, python **pandas** library is used.
The following global variables are declared and their values are stored into a dataframe:

```
global training_set1, training_set2,
training_set3,validation_set1,validatio
n_set2, validation_set3
```

The below variables are declared globally at the beginning of the function. The use of the individual variables would be discussed in the relevant sections in this article:

```
global folder_path, start_column,
end_column, classifier, string_variable
training_set1 = pd.DataFrame()

global tp, tn, fp, fn, fdr, npv, tpr,
tnr, roc, f1
```

Similarly, variable **object1_data, object2_data** and **object3_data** variables are created and initialized with blank dataframes. These dataframes are for storing the values read from excel file.

```
object1_data = pd.DataFrame()
object2_data = pd.DataFrame()
object3_data = pd.DataFrame()
```

From the directory selected from GUI input, excel files are searched, read and dataset is stored into Dataframe. In the stored Dataframe, only specific range of column is used which is instructed from GUI with starting and ending column (signal length) parameter. The read datasets are stored in three sets object1_data, object2_data and object3_data respectively.

```
signal_width = range(start_column,
end_column)
………….

df=pd.read_excel(os.path.join(subdir1,f
), header = None, usecols =
signal_width )

df.insert(0, "Object Type", "Object 1")

object1_data=object1_data.append
(df,ignore_index=True)
……….
Object2_data=object1_data.append
(df,ignore_index=True)
……….
Object3_data=object1_data.append
(df,ignore_index=True)
```

Creating models that precisely map the given inputs to the given outputs is the goal of supervised machine learning.

Measuring the precision of the model depends on the problem and dataset. For the classification problem some of the indicators of this measurement are Accuracy, F1 Score etc. It is imperative to say that unbiased evaluation is needed to properly apply these measures to effect. This means, the model needs to be evaluated with unknown data which has not been fed to the model yet. For this reason, splitting the dataset is essential to evaluate unbiased prediction performance. For creating the model, initially we would split the dataset into two subsets namely the Training Set and the Validation Set [8]. We used scikit-learn or sklearn package to split the dataset and subsequently create the MLP Training model. Sklearn is a python library that provides various data processing features like classification, clustering and model selection. The below package is used for the data split operation:

```
from sklearn.model_selection import
train_test_split
```

The function `train_test_split()` splits the data arrays into two subsets: training data and testing data. This testing data is used as Validation set.

```
training_set1,validation_set1=
train_test_split(object1_data,
test_size = 0.2, random_state = 21)
```

As per the code snippet above, the dataset `object1_data` is divided into two training and testing set. The `test_size` parameter sets the size of the testing set. As per the specification above the test size would be 0.2 or 20% and hence the training size would be 0.8 or 80 %. The parameter `random_state` controls the randomization during splitting. The value 21 in this parameter is chosen randomly but this number helps to reproduce the same result if the function `train_test_split` is rerun. Similarly, the training and validation set are produced for the other two datasets object2_data and object3_data individually. [9]

Next all the training and validation datasets from individual object datasets are appended and stored in two cumulative datasets named training_set and validation_set.

```
#Appending training data from all the
object types

training_set = training_set.append
(training_set1,ignore_index = True)

training_set = training_set.append
(training_set2,ignore_index = True)

training_set= training_set.append
(training_set3,ignore_index = True)

#Appending validation data from all the
object types

validation_set= validation_set.append
(validation_set1,ignore_index = True)

validation_set= validation_set.append
(validation_set2,ignore_index = True)

validation_set= validation_set.append
(validation_set3,ignore_index = True)
```

The output of the consolidated training and validation test is segmented to 4 portions of data which are X_train, X_val, Y_train and y_val.

X_train : This consists of variables that will be used for training the model. Since test size is 0.2, this set will contain the remaining 80% of the observations.

X-val : The remaining 20% of the observations from the data which would not be used in training set but would be used for predictions to test the accuracy of the model.

Y_train : This is the set of labels to all the data in X_train.

Y_val : This is the set of labels to all the data in Y_train. It used to test the accuracy between actual and predicted categories.

The values and labels are set for the above parameters by slicing the arrays on training_set and validation_set data.

```
X_train =training_set.iloc[:,1:].values

Y_train =training_set.iloc[:,0].values

X_val =validation_set.iloc[:,1:].values

y_val = validation_set.iloc[:,0].values
```

C) MLP Classifier Training Model :

When we use some of the classification techniques other than Neural Networks for Time Series data, a key practice is to do feature extraction manually. For example, making Short-Time-Fourier-Transform and extracting features like MFCC (Mel Frequency Cepstral Coefficients). In our project with the Multi Layer Perceptron model, we are implementing Deep Neural Network approach through which we can do away with this fixed pre-processing of data (feature extraction as mentioned earlier) and incorporate feature learning within the training process by downsampling the data on subsequent layers (pooling) of the Neural Network. This improves the overall performance of the model [10].

The particular problem in our project warrants identification of three different objects. Hence the classification problem can be categorised to be a Multiclass Classification and the output labels are supposed to correspond to data belonging to three entities namely Object#1, Object#2 and Object#3. The classifier training parameters are set in scikit-learn in the following way:

```
 classifier=MLPClassifier(hidden_layer_
sizes=(100,50,50),max_iter=300,activati
on='relu',solver='adam',random_state=1)
```

`MLPClassifier` is a scikit function for training MLP models. The parameters passed within the function are deciding factor for the efficacy of the training performance.

hidden_layer_sizes : Two factors need to be considered while passing values in this parameter – how many hidden layers need be there in the Neural Network and how many neurons need to be there in the respective hidden layers. In this case we have used three hidden layers containing 100,50 and 50 neurons respectively. Although at first it is difficult to predict from the dataset that how many hidden layers is required, subsequent trial with different number of layers helped us determine that optimal numbers of hidden layer in this case would be 3. Intuitively, increased number of hidden layers as well as increased number of neurons per hidden layers are supposed to make the network capable of learning more complex classification function and thus perform more efficient classification. In practice it has been observed that increasing the above parameters gives rise to some problems which are Gradient Descent Problem (Vanishing and Exploding) and Overfitting. [11]
Basically, in Gradient-based learning methods, gradient tends to get smaller as we proceed through the chain of hidden layers which means layers in the early layers in the chain learn much slower than the latter part of the layers. The issue is that in some situations, the gradient will be so small that the weight will be unable to change its value and thus deteriorating the performance. This is the Vanishing Gradient Descent Problem. Opposite to the Vanishing Gradient Descent Problem, if an activation function is used whose derivatives can process larger values, the gradient sometimes gets much larger in early layers. As the magnitudes of the gradients scale up, an unstable network emerges, which can lead to bad prediction results. This is termed as exploding gradient problem. [11]
Increasing the number of neurons per layer i.e., making the Neural Network deeper, causes overfitting. Overfitting in a model suggests that model learns the training dataset too well, performs well on the training dataset but fails to perform well on a holdout sample. [12]
As a bias-variance trade-off and finetuning the parameters over several trials, the parameter values of hidden_layer_sizes are set as described in the code snippet above. A comparative analysis of the results over different parameters had been carried out and based on the outcome this specific values for hidden layer size have been finalized.

`activation`: As discussed in the earlier part of this article, `'relu'` is used as the Activation function through the layers of the Neural Network.

`solver` : Adam, an adaptive learning rate optimization is used for weight optimization. It can automatically calibrate the amount to update and finetune parameters based on adaptive estimates of lower-order moments. As per sclearn guideline, adam works quite well with large datasets for training and validation performance. [13]

`max_iter` : This denotes maximum number of iteration the solver would iterate. Typically, solver iterates until convergence.In our case, for 'adam' solver this determines the number of times each data point will be used.
`random_state` : This determines and controls the random number generation for weights and bias initialization as well as batch sampling. The value passed in this parameter will have an effect on the reproducibility of the function's output. As in our case, passing an integer would reproduce same result across multiple function calls.

Once the training model is declared, it is time for the model to learn from the training data. The classifier estimator instance namely 'classifier' in our case is fitted to the model by the scikit-learn function `fit`.

```
 classifier.fit(X_train, Y_train)
```

D) Prediction and Performance Validation:

As discussed earlier, the 20 % of the data set which was reserved for validation is now subsequently used to asses how efficiently the model is able to classify the respective objects. Inherently, our task here is to predict the label of the objects given the set of values or features. In another way, we can say that if a new, unknown observation is feed to the model it would lookup the reference database, compare the closest features and assign the predominant class to the specific observation.

```
y_pred = classifier.predict(X_val)
```

It is essential to validate the performance of the model by examining the accuracy of the model so that the parameters can be tuned to improve the performance. Sensitivity and Specificity are two evaluation metrics often used to determine the performance of the ML model. Sensitivity can be defined as the measurement of the proportion of actual positive cases that got predicted as positive (or True Positive). This also implies that there would be some other instances of actual positive cases which would be predicted incorrectly. This set is termed as False Negative. [14]
Mathematically, sensitivity can be defined as following

$$Sensitivity = \frac{True\ Positive}{True\ Positive + Flase\ Negative}$$

Similarly, Specificity measures the proportion of the actual negatives which got predicted as negative. Mathematically it can be defined as

$$Specificty = \frac{True\ Negative}{True\ Negative + Flase\ Positive}$$

Confusion Matrix is a good technique that uses the sensitivity and specificity parameters to create a visualization of the performance of the supervised learning algorithm. In the Confusion Matrix, each row represents the outcomes corresponding to the predicted class and each column represents that of the actual class (or vice e versa). [15]

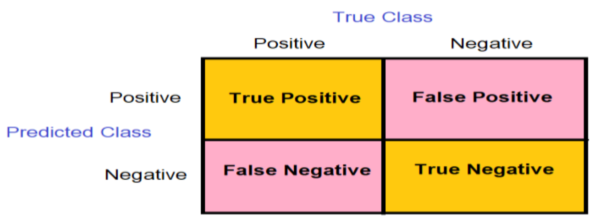


Figure 6 Sample Confusion Matrix for Binary Classification

Since our problem set belongs to a Multiclass Classification problem, the confusion matrix can be extended to the following form:

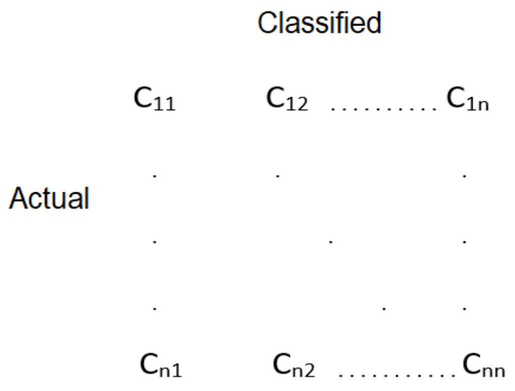


Figure 7 Multiclass Confusion Matrix

The confusion elements C_{ij} can be identified as results derived from mathematical operations on the Sensitivity and Specificity parameters. The sklearn confusion_matrix function computes the earlier prediction y_pred with actual observations y_val. Subsequently matrix is plotted for display. [16]

```
cm = confusion_matrix(y_pred, y_val)

disp=plot_confusion_matrix (classifier,
X_val, y_val,cmap=plt.cm.Blues)

plt.show()
```

Accuracy: This is a metric used for classification performance evaluation. By definition, accuracy is the fraction of correct predictions to the total number of predictions. The function `accuracy_score` in sklearn computes subset accuracy from set of labels predicted with respect to corresponding actual set of labels for multilabel classification problem. [17]

```
accuracy =accuracy_score(y_pred, y_val)
```

Although Classification Accuracy is a very simple and intuitive measure, this alone is not a good evaluation option especially for class imbalanced dataset i.e., distribution of examples across the classes is not equal

F1 Score: F Score in general is a measure of the test's accuracy. F1 score is the harmonic mean of two measures –precision (also known as positive predictive value (PPV)) and sensitivity (also known as recall). PPV can be defined mathematically as:

$$PPV = \frac{Number\ of\ \ True\ Positive}{Number\ of\ Positive\ Calls}$$

Hence, F1 score can be mathematically defined as :

$$F1\ score = \frac{2 * (precision * recall)}{precision + recall}$$

F1 score for all the sets are implemented in the code with the sklearn function f1_score. [18]

Here the score would be calculated for all the labels and displayed:

```
f1Score=f1_score(y_pred,y_val,average=
None)

f1Score = f1Score.tolist()

f1Score  =  ['{:.2f}'.format(elem)  for
elem in f1Score]

for i in range(3):
    f1[i].set(f1Score[i])
```

The sensitivity parameters like True Positive(TP), True Negative (TN), False Positive (FP) and False Negative(FN) are extracted from the confusion matrix and featured for all the class labels as per the following code snippet:

```
FP = cm.sum(axis=0) - np.diag(cm)

FN = cm.sum(axis=1) - np.diag(cm)

TP = np.diag(cm)

TN = cm.sum() - (FP + FN + TP)

FP = FP.astype(float)

FN = FN.astype(float)

TP = TP.astype(float)

TN = TN.astype(float)

Tp = TP.tolist()
for i in range(3):
     tp[i].set(Tp[i])

Tn = TN.tolist()
for i in range(3):
    tn[i].set(Tn[i])
```

```
Fp = FP.tolist()
for i in range(3):
    fp[i].set(Fp[i])

Fn = FN.tolist()
for i in range(3):
    fn[i].set(Fn[i])
```

In the similar way the measures False Discovery Rate(FDR) and Negative Predictive Value is computed for all the labels. Also, Sensitivity (also known as hit rate, recall or True Positive Rate) and Specificity (Also known as Selectivity or True negative Rate) is computed for all the labels.

```
# False discovery rate
FDR = FP/(TP+FP)

# Negative predictive value
NPV = TN/(TN+FN)

# Sensitivity, hit rate, recall, or true
positive rate
TPR = TP/(TP+FN)

# Specificity, selectivity or true
negative rate
TNR = TN/(TN+FP)
```

Receiver Operating Characteristics (ROC) :
ROC curve which is also known as AUC-ROC curve is an evaluation metrics to visualize the performance of Multi Class Classification problem. The AUC term stands for Area Under the Curve. The ROC curve is a probability curve that plots TPR vs FPR at different classification threshold. AUC represents the degree of separability between classes. In this section we combine both and represent as ROC curve. Since ROC features TPR in Y axis and FPR in X axis, the higher steepness and larger area under the curve is better for the model ideally. [19]

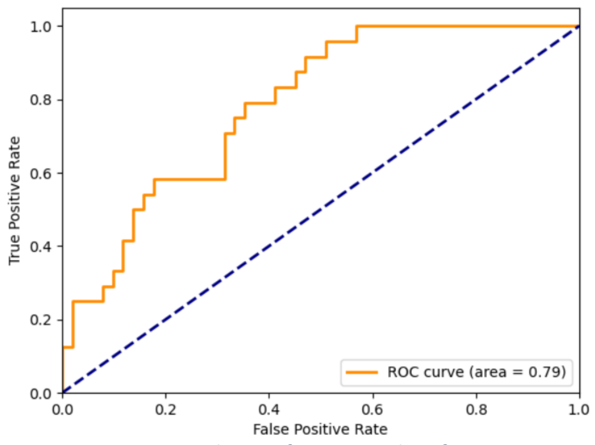


Figure 8 Sample ROC for Binary Classification

As an extension to multilabel classification, macro-averaging is used. This gives equal weight to the classification of each label. The sklearn roc_auc_score is used for Multiclass classification. Compute Area Under the Receiver Operating Characteristic Curve from prediction scores. For our classification problem, three labels are fitted in the curve and their relative separability is measured from the ROC computation. [20]

The code to implement the algorithm and display the result curve is implemented as below:

```
fpr = dict()
tpr = dict()
roc_auc = dict()
n_classes = 3

for i in range(n_classes):
    fpr[i],tpr[i], _= roc_curve
(np.array(pd.get_dummies(y_val))[:, i],
np.array(pd.get_dummies(y_pred))[:, i])
        roc_auc[i] = auc(fpr[i],
tpr[i])

all_fpr = np.unique ( np.concatenate
([fpr[i] for i in range(n_classes)]))

mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
        mean_tpr += np.interp(all_fpr,
fpr[i], tpr[i])

mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"],
tpr["macro"])

lw=2
plt.figure(figsize=(8,5))
plt.plot(fpr["macro"], tpr["macro"],
            label='macro-average ROC
curve (area = {0:0.2f})'

''.format(roc_auc["macro"]),
            color='green',
linestyle='dotted', linewidth=4)

    colors = cycle(['purple', 'sienna',
'cornflowerblue'])
    for i, color in
zip(range(n_classes), colors):
        plt.plot(fpr[i], tpr[i],
color=color, lw=lw,
                label='ROC curve of
object {0} (area = {1:0.2f})'
                ''.format(i + 1 ,
roc_auc[i]))
print("roc_auc_score of object " , i+1,
": ", roc_auc[i])

print("\n")
plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.annotate('Random Guess',(.5,.48))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver        Operating
Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()

Roc = list(roc_auc.values())
Roc = ['{:.2f}'.format(elem) for elem in
Roc]
for i in range(4):
     roc[i].set(Roc[i])
```

D) Testing:

The following algorithm has been separately implemented for the event of testing new dataset to classify the three objects.

- GUI implementation is done to receive a new Excel file containing dataset.
- The script reads the excel data and predict it from the trained MLP classification model.
- A new excel file is created where at the last column, corresponding to the particular row (signal) the predicted object name is added.

To carry out the above implementation, a new function named testing () is used. From the dataset that was provided for this project, we separated some signals from that data randomly which contains all the Objects (Object#1, Object #2 and Object#3) and compiled in a single file. Hence, our model has never seen this new dataset before. This dataset would be fed to our model to predict which objects the individual signals belong to. Since we as operators already have the knowledge about the signals and their corresponding object, we can validate the efficacy of our prediction by manually comparing the result.

The following global variables are used to read the input file and write the output with object name:

```
global file_path, file_output_path
```

We use the signal length same as it was used in Training. If the signal length (Number of columns) used in Testing is not of the exact number as used in Training then we receive column length mismatch error. Rectifying this issue can be considered as a future improvement prospect of this project.

```
signal width  =  range(start_column,
end_column)
```

The signal data is read from the input excel file is read and stored in the variable :

```
test_data  =  pd.read_excel(file_path,
header = None, usecols = signal_width)
```

The trained model now can predict the output from this newly read data

```
y_test       =       classifier.predict
(test_data.values)
```

Now, the predicted output is generated in an excel file. A new column is appended in the output file and the output corresponding to the particular signal (row) is added for identification.

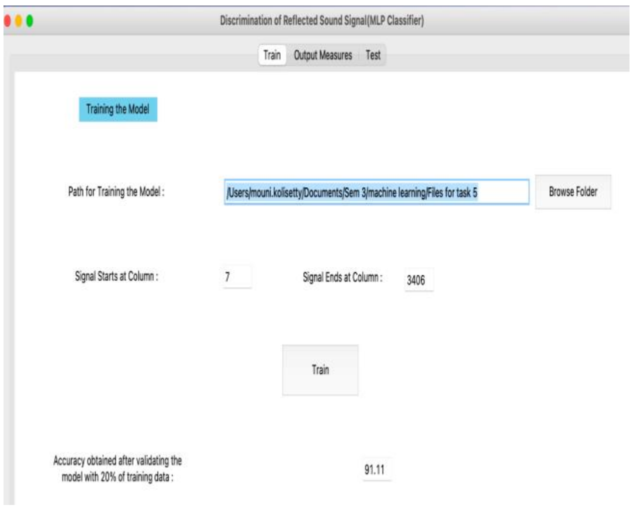
```
workbook=        openpyxl.load_workbook
(file_path)
worksheet = workbook.worksheets[0]
worksheet.insert_cols(3407)
y = 1
for x in range(len(y_test)):
    cell_to_write  =  worksheet.cell(row
= y, column = 3407)
    cell_to_write.value = y_test[x]
    y += 1

workbook.save(file_output_path)
```

Noteworthy to say, the trained classification model with the signal length 3400 is already saved in a pickle file. Hence, if we need to test with a new dataset with the same dimension then we can just load the model (do not need to train it again) and test the same way. A separate python script is written for the same which can be run for this purpose. The pickle save and load model feature is implemented in the following way:

```
#Saving the model

filename = 'trained_model.pkl'
pickle.dump(classifier,open(filename,
'wb'))

#Loading and testing the model

with open(filename, 'rb') as file:
    pickle_model = pickle.load(file)
y_test      =      pickle_model.predict
(test_data.values)
```

IV. Result and Discussions

We start by running the python script in our Conda environment with PyCharm IDE. It is imperative to mention that if the environment is different then there may be requirement to install python module and relevant libraries to run this script.



Path for Training the Model : /Users/mouni.kolisetty/Documents/Sem 3/machine learning/Files for task 5

Signal Starts at Column : 7

Signal Ends at Column : 3406

Accuracy obtained after validating the model with 20% of training data : 91.11

Figure 9 GUI for File Input and Train Accuracy Output

As depicted in Fig 9, the training file is input by specifying the containing directory, column dimensions are provided and the model is trained. We obtain an accuracy of 91.11% which is reasonably acceptable for medium to large scale Time Series classification. As discussed earlier, Accuracy not being the only reliable parameter for performance evaluation especially when we have an asymmetric database, we look for other assessment metric results subsequently.

The plotted Confusion Matrix is displayed as below which marks the TP, TN, FP and FN values:

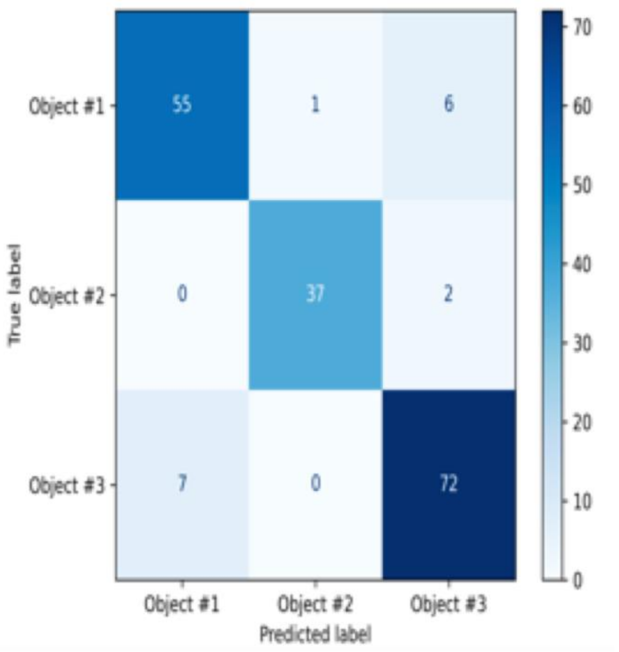
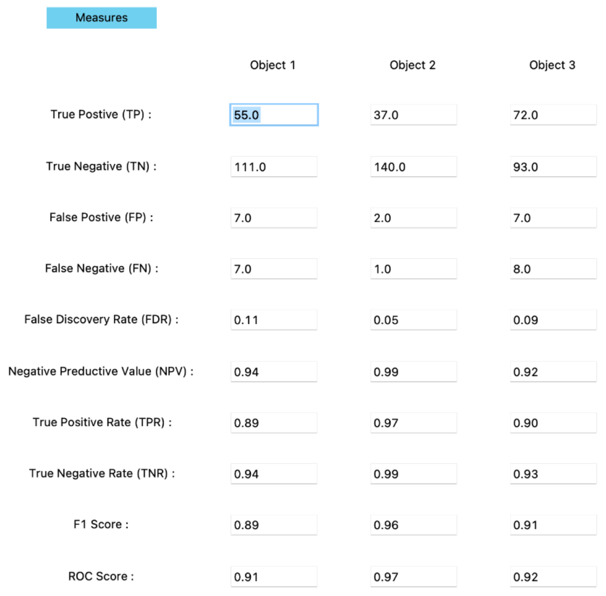


Figure 10 Generated Multiclass Confusion Matrix

The respective assessment measure metrics are simultaneously displayed in our GUI as following:



	Object 1	Object 2	Object 3
True Postive (TP) :	55.0	37.0	72.0
True Negative (TN) :	111.0	140.0	93.0
False Postive (FP) :	7.0	2.0	7.0
False Negative (FN) :	7.0	1.0	8.0
False Discovery Rate (FDR) :	0.11	0.05	0.09
Negative Preductive Value (NPV) :	0.94	0.99	0.92
True Positive Rate (TPR) :	0.89	0.97	0.90
True Negative Rate (TNR) :	0.94	0.99	0.93
F1 Score :	0.89	0.96	0.91
ROC Score :	0.91	0.97	0.92

Figure 11 MLP Trained Model Evaluation Metrics

We observe from the result that among all the objects, Object #2 demonstrates least number of False Positive and False Negative. Both this parameters predict the incorrectly true and false predictions for a particular label. Subsequently we see Object #2 also pertains to least False Discovery Rate, highest Negative Predictive Value, Sensitivity (TPR), Specificity (TNR), F1 Score and ROC score. The corresponding ROC plot further substantiate the outcomes.

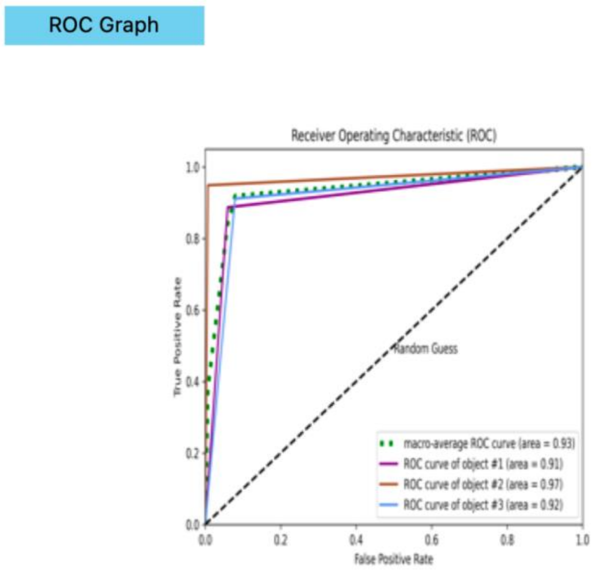
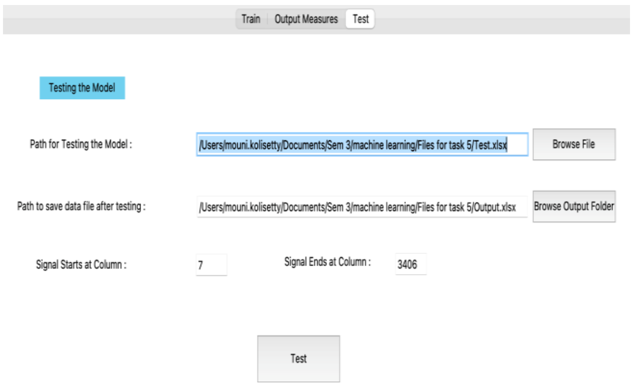


Figure 11 ROC Plot for Multiclass classification

As we observe in Fig 11, there is a steeper curve and thus there exists more area under the curve for Object #2 compared to other two objects. This suggests that classification for Object #2 is supposed to be more accurate than other two objects based on the assessment metrics. Although, it must be pointed out that overall model assessment metrics for other two objects (Object #1 and

Object #3) do manifests significantly well performance outcome in classification in terms of the respective F1 Score, Sensitivity, Specificity and ROC Score.

Now, as the model is trained and validated to be performing well, we can test the efficacy on a completely unknown set of data and identify the efficacy by examining the result.



Path for Testing the Model : /Users/mouni.kolisetty/Documents/Sem 3/machine learning/Files for task 5/Test.xlsx

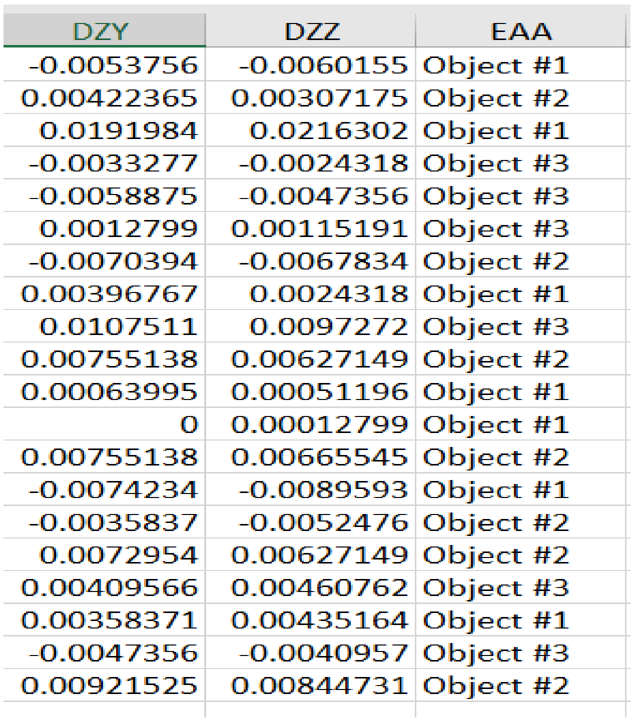
Path to save data file after testing : /Users/mouni.kolisetty/Documents/Sem 3/machine learning/Files for task 5/Output.xlsx

Signal Starts at Column : 7

Signal Ends at Column : 3406

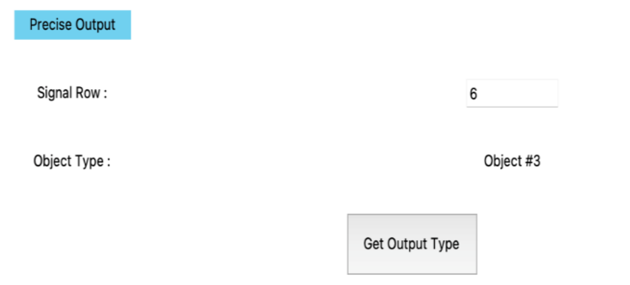
Figure 12 GUI to Input Test file

As depicted in the GUI in Fig 12, the file Test.xlsx is provided as input to the model. Signal length (Column duration) exactly matching with the one used in training the model is used. This file contains various combination of signals from the three Objects. The output is generated in a new excel file and new labels corresponding to the individual signal is created in the last column as as we can see in Fig 13.



DZY	DZZ	EAA
-0.0053756	-0.0060155	Object #1
0.00422365	0.00307175	Object #2
0.0191984	0.0216302	Object #1
-0.0033277	-0.0024318	Object #3
-0.0058875	-0.0047356	Object #3
0.0012799	0.00115191	Object #3
-0.0070394	-0.0067834	Object #2
0.00396767	0.0024318	Object #1
0.0107511	0.0097272	Object #3
0.00755138	0.00627149	Object #2
0.00063995	0.00051196	Object #1
0	0.00012799	Object #1
0.00755138	0.00665545	Object #2
-0.0074234	-0.0089593	Object #1
-0.0035837	-0.0052476	Object #2
0.0072954	0.00627149	Object #2
0.00409566	0.00460762	Object #3
0.00358371	0.00435164	Object #1
-0.0047356	-0.0040957	Object #3
0.00921525	0.00844731	Object #2

Figure 13 New Output Labels Generated in the Output file



Signal Row : 6

Object Type : Object #3

Figure 14 GUI for Output Object Display for specific Signal Number

We have validated from our knowledge of the database that all the predictions made by the model exactly matches with the desired result i.e., correct object classification. In the GUI, we have implanted an option in which if a particular signal needs to be inspected for which object it belongs to, we need to input the row number and the corresponding output Object type would be displayed as depicted in Fig 14.

This concludes the article with the inference that this Multilayer Perceptron based Classification model can be used for Multiclass classification and subsequently object identification based on similar Acoustic signal data. The outcome of the classification process is expected to be reasonably accurate.

References

[1] C. Voyant, M. Muselli, C. Paoli and M.-L. Nivet, "Numerical Weather Prediction (NWP) and hybrid ARMA/ANN model to," *Energy,* 2012.

[2] C. Voyant, W. Tamas, C. Paoli and G. Notton, "Time series modeling with pruned multi-layer perceptron and 2-stage damped least-squares method," *Journal of Physics Conference Series,* 2013.

[3] P. Marius, V. E. Balas, L. Perescu-Popescu and N. E Mastorakis, "Multilayer perceptron and neural networks," *WSEAS Transactions on Circuits and Systems,* no. 8, 2009.

[4] D. Stursa and P. Dolezel, "Comparison of ReLU and linear saturated activation functions in neural network for universal approximation," *International Conference on Process Control,* 2019.

[5] C. M. Bishop, Pattern Recognition and Machine Learning, Springer.

[6] M. I.A., Exchange Rate Forecasting: Techniques and Applications. Finance and Capital Markets Series, London: Palgrave Macmillan, 2000, pp. 62-113.

[7] "Python Tkinter Documentation," [Online]. Available: https://docs.python.org/3/library/tkinter.html.

[8] R. Medar, V. S. Rajpurohit and R. B. , "Impact of Training and Testing Data Splits on Accuracy of Time Series Forecasting in

Machine Learning," *nternational Conference on Computing, Communication, Control and Automation (ICCUBEA),*, pp. 1-6, 2017.

[9] scikit-learn, "sklearn.model_selection.train_test_split," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

[10] C. Jatturas, S. Chokkoedsakul, P. D. N. Avudhva, S. Pankaew, C. Sopavanit and W. Asdornwised, "Feature-based and Deep Learning-based Classification of Environmental Sound,," *IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia),* pp. 126-130, 2019.

[11] M. A. Nielson, Neural Networks and Deep Learning, Determination Press, 2015.

[12] X. Ying, "An Overview of Overfitting and its Solutions," *Journal of Physics: Conference Series,* no. 1168 022022, 2019.

[13] "sklearn.neural_network.MLPClassifier," https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier. [Online].

[14] M. Al-Mistarihi, P. Phukpattaranont and . E. S. Ebbini, "A two-step procedure for optimization of contrast sensitivity and specificity of post-beamforming Volterra filters," *IEEE Ultrasonics Symposium,* vol. 2, pp. 978-981, 2004.

[15] F. J. A.-L. . J. R.-A. and M. V. A.-F. , "Complete Control of an Observed Confusion Matrix," *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium,* pp. 1222-1225, 2018.

[16] "sklearn confusion matrix," https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html. [Online].

[17] "sklearn accuracy_score," scikit learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html.

[18] "sklearn.metrics.f1_score," scikit learn , [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html .

[19] S. Ho Park, J. Mo Goo and C.-H. Jo, "Receiver Operating Characteristic (ROC) Curve: Practical Review for Radiologists," *Korean Journal of Radiology,* 2004.

[20] "Receiver Operating Characteristic (ROC)," scikit-learn, [Online]. Available: https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html.

Task Number	Group Number
5	2
Name of Team Member	Tasks Performed
Anik Biswas	Code: 1. Implementing GUI for Reading Input File and Model Training 2. Performing Data pre-processing and Training Dataset Preparation 3. Saving and Loading Trained Model in and from pickle file
	Report: 1.Introduction,Multilayer Perceptron and Time Series Forecasting 2. Project Implementation - Graphical User Interface and Data preparation
Lakshmi Mounika Kolisetty	Code: 1.Implementing GUI for Training Output Measures 2. MLP Classifier Training Model 3. Model Performance Validation
	Report: 1.Project Implementation - MLP Classifier Training Model 2. Project Implementation – Prediction and Performance Validation
Bimal Kumar Sah	Code: 1.Implementing GUI for Testing 2. Testing Model with New Data 3. Generate Output Labels in Excel File and GUI
	Report: 1. Project Implementation- Testing 2. Result and Discussion