Source Identification using LSTM

Master of Engineerinng
Information Technology
Abhishek Sharma
1274291
asharma@stud.fra-uas.de

***Abstract*—This paper is a part of an individual project of Frankfurt university of applied science which describes the identification of two given sources of Time-series data using the LSTM (Long short-term Memory) Model. LSTM is a better version of RNN (Recurrent neural network) known to solve Long term sequence dependencies. In this project, sequence classification of time series data (sequence of data given in time) is implemented, and the category of the sequence is predicted via LSTM Model. Since there are only two sources of data therefore it is also called a binary classifier. The complete project is implemented in python using 'Spyder IDE'.**

Keywords—LSTM, RNN, Time-series data, sequence classification, Binary classification, Machine Learning, sequential data, Neural Network, python

I. INTRODUCTION

LSTM is a special version of RNN (Recurrent neural network) used to learn long-term patterns. For a long period, RNN dominates over LSTM for Time-series and sequential data but the success of RNN in real-world applications was limited because of the increase in time lags between relevant context or information which makes learning difficult. The main reason for this failure is a back-propagated error. The "Long short-term memory" algorithm solves this problem by enforcing constant error flow. Using gradient descent LSTM explicitly learns when to store information and when to access it [1].

Time-series data is the set of data points taken at a specified time usually at equal intervals. It is used to predict the future value by analyzing past values. In Time series data, X-axis is Time and Y-axis is the amplitude of data. Here, in this project, we have two sources of data and we are using the LSTM network since it is suitable for Time-series data [2].

In this paper, the sequence classification problem is addressed and implemented practically using python in Spyder IDE. Time-series data is already provided by the University professor therefore, only data preprocessing needs to be done for the Model. Then, LSTM Model is created.

First, we load all the available CSV data files of source A and source G. Data source A contain 150 rows,3406 columns and source G contains 200 rows, 3406 columns but Information in the first 6 columns of both source A and Source G are not actual data points, therefore, actual data of source A and source G contains 150 rows, 3400 columns & 200 rows, 3400 columns respectively. After that, vertical stacking is done by using NumPy and panda's library and total data becomes 350 rows x 3400 columns. Reshaping of data is also required because TensorFlow will take input in a particular dimension. Now, horizontal stacking is done for assigning labels '0' and '1'. For model training, data is split into "Training" and "Testing" data usually in the ratio of 80 % for training and 20 % for testing. Now after that, LSTM Model is defined, compile, evaluate and fit which is explained in detail in a subsequent section. Finally, the prediction result of test data is displayed. The library used for Model creation is TensorFlow, Keras, NumPy, pandas.

II. METHODOLOGY

In this section, a detailed step-by-step process of source classification using LSTM is explained. Firstly, data preprocessing of two sources are discussed, and then model creation. The overview of the project step is shown below.

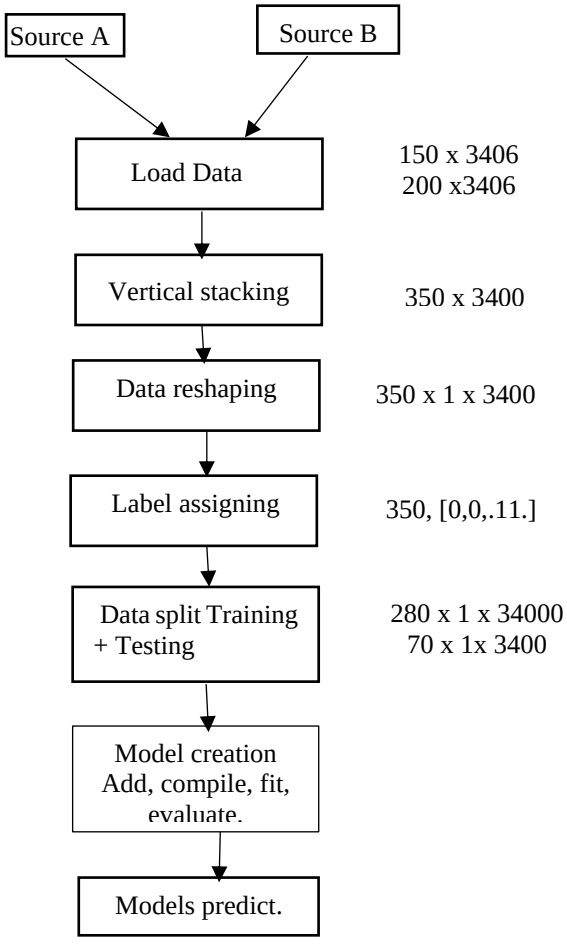

Fig. 1. **Flow diagram of Project Implementation**

All the name of elements of Array and is value is shown below table which is important to understand the implementation.

Name of Arrays	Size
dataFrame A	150 x 3406
dataFrame G	200 x 3406
X_A	150 x 3400
X_G	200 x 3400
X	350 x 3400
X_test	70 x 1 x 3400
X_train	280 x1 x3400
Y_test	70,
Y_train	280,
y	350,

Table showing Arrays and its values

A. Data set

The data is provided from two sources one from source A and the other one from source G. There is a total of 3 excel files recorded for source A.i.e. record_data_2020-10-29_10-30-40_A.xlsx, record_data_2020-10-29_10-31-22_A.xlsx, record_data_2020-10-29_10-34-8_A.xlsx, and similarly, there are a total of four excel files are recorded for source G. i.e data_2020-10-29_11-36-39_G.xlsx, record_data_2020-10-29_11-37-24_G.xlsx, record_data_2020-10-29_11-39-13_G.xlsx.

Each source A and source G contains 50 rows and 3406 columns respectively. A plot of one of the rows of source A and source G is drawn below Fig:3 and Fig: 4 respectively

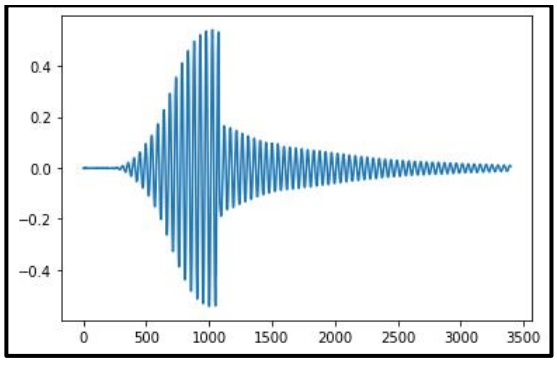

Fig. 2. DATA A (1x3400) record_data_2020-10-29_10-30-40_A.xlsx

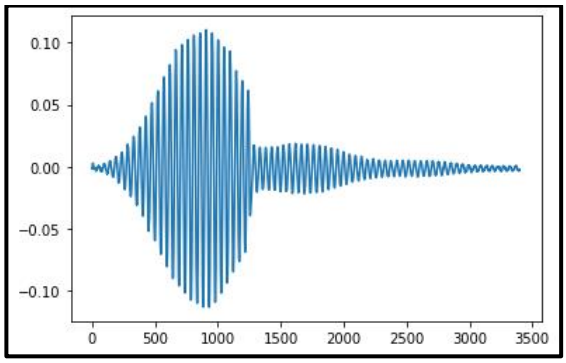

Fig. 3. DATA G (1x3400)record data_2020-10-29_11-36-39_G.xlsx

The first six columns of both data source A and G are some other information not considered as main data. Main data are valid from columns 7 to 3400.

The python library used to plot the above graph is Matplotlib. Matplotlib libraries lie under pyplot submodule and imported as plt alias as shown below [3].

```
import matplotlib.pyplot as plt
```

Fig. 4. Matplotlib Library

Now, we have in total 150 rows of data from source A and 200 rows of data from source G.

Again, plot of one of the rows of data of source A and Source G are shown below. It can be easily seen that source A data doesn't change much but source G data has some variations as shown below.

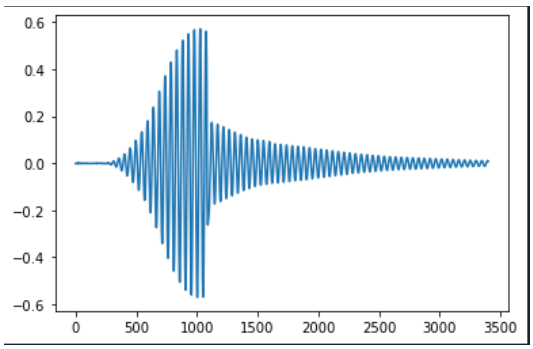

Fig. 5. DATA A (1x3400) record_data_2020-10-29_10-34-8_A.xlsx

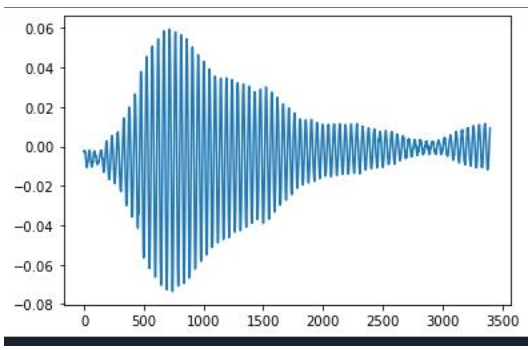

Fig. 6. DATA G (1x3400) record_data_2020-10-29_11-37-24_G.xlsx

B. *Data preprocessing for training*

1) Python libraries for Data preparation:

```
import pandas as pd
import numpy as np
```

Fig. 7. Panda as pd and numpy as np library

Numpy and pandas are the two highly used libraries in python for computation related to any kind of data. Numpy means numeric python used for high mathematical computation on multidimensional arrays and matrix. While padas library is used for computing data in columns. It provides an in-memory 2d table object called DataFrame [4].

2) Data Loading:

Data is a very important part of training any machine learning algorithm. More data leads to good training. Therefore, data preprocessing is required before data is fed into the machine learning model. Here, we have used pandas and NumPy libraries for handling the data. All the csv files of source A are merged into one csv file and all the csv files of source G is merged into one csv file of source G as shown below.

```
dataFrameA = pd.read_excel('data/train&test/A.xlsx', header=None)
dataFrameG = pd.read_excel('data/train&test/G.xlsx', header=None)
```

Fig. 8. Padas Data Frame

3) Data conversion and reshaping:

Now, total data is used by removing the first 6 columns (1 to 6) which do not contain useful information for our training as shown below:

```
x_A = np.array(dataFrameA.drop([0,1,2,3,4,5], axis=1))
x_G = np.array(dataFrameG.drop([0,1,2,3,4,5], axis=1))
```

Fig. 9. Dropping the first 6 columns from the original data using NumPy

After the first 6 column dropout, total data in source A and source G contains 150X3400 and 200X3400 rows & columns respectively as shown below:

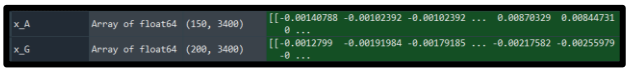

Fig. 10. Showing total source data in A (150x3400) and source B (200x3400)

After that, we have to append data vertically therefore, we used numpy function vstack () as shown below:

```
x = np.vstack((x_A, x_G))
```

Fig. 11. Vertical stacking using Numpy

Here, the total data of source G is appended below source A and stored in variable x which is called features in machine learning.

Now, we have to reshape our stacked data in a new dimensional array.

```
x = np.reshape(x, (x.shape[0], 1, x.shape[1]))
```

Fig. 12. Reshaping fuction of total source data

Reshaping is necessary because TensorFlow will take input in a particular dimension as shown below [5].

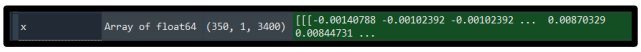

Fig. 13. Reshaped data

4) Data labeling:

Now, we do horizontal stacking for labeling. Assignment of the label is required to predict the output.

(x, y) = {features, label}

Here, '0' is assigned to Data A, and '1' is assigned to Data G as shown below.

```
y = np.hstack((np.zeros(x_A.shape[0]), np.ones(x_G.shape[0])))
```

Fig. 14. Horizontal stack showing labels 0 and 1 to total 350 rows

C. *Data splitting into Train and Test*

In machine learning, data splitting is a very important part as it is useful for prediction performance and better model creation [6].

Data can be split into three categories randomly are:

- Training set: This data set is required to train or fit the model.
- Validation set: This data is required for parameter tuning and to avoid overfitting.
- Test set: This data is used for final evaluation of model and should not be used for validation.

In this project, we have imported a **train_test_split ()** from sklearn as shown below.

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

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
```

Fig. 15. Showing train_test_split function for data splitting

As shown above, train_test_split () will take four arguments x, y, test_size, random_state and return four sequences describe below [6]:

- x_train: The training part of first sequence (x)
- x_test: The test part of first sequence (x)
- y_train: The training part of second sequence (y)
- y_test: The test part of second sequence (y)

In this project, out of 100% data, 80% is used for training and the remaining 20 % is used for testing(evaluating). Random_state is used here to randomize during split. It can take any int value but its default value is 0.

Now, we are ready with our training set (x_train, y_train) and test set (x_test, y_test) for evaluating the model.

D. *LSTM Model creation*

In this project, LSTM model is used for sequence classification of given Time-series data as shown below in a simple block diagram.

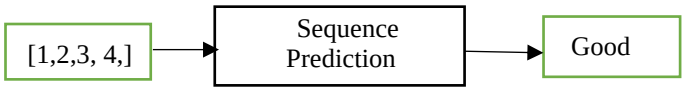

Fig. 16. Block diagram showing an overview of a classification model

1) Python libraries: First of all we have to import python libraries for implementing the model. Below libraries are used

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout,LSTM
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDis
```

Fig. 17. All python libraries for implementation of the model

Tensorflow is a widely used machine learning library or framework used for computing machine learning algorithms and training neural networks. Tensor flow is derived from the tensors which are nothing but a multidimensional array. Another important library used here is Keras. Keras is a high-level machine learning API running on top of TensorFlow and highly recommend for Time series analysis and LSTM. Sklearn or sci-kit learn is also a very useful open-source library that provides various tools for model fitting, data preprocessing, model selection, and evaluation [7] [8] [9].

2) Define the Model:

The first step in model creation is to define the network. Keras library provides a sequence of layers for neural network. The container for this layer is a sequential class. Now we can add the layers so that they can be connected. LSTM () is a recurrent layer comprised of the memory unit. Fully connected layer followed by Dense layer. The first hidden layer must define the shape of the input (number of input) [10].

Input is three-dimensional contains samples, timesteps, features. It is not necessary to show samples.

- Samples: Rows in data
- Time steps: past observation for feature
- Features: Columns in data

Here, in a project, we have used 32 neurons in a single layer of LSTM followed by 2 dense layers. Due to Dense Layer each neuron from first layer will be connected neuron in second layer of the model.

Dropout is also used here. Dropout is used to avoid overfitting in the model. Dropout is nothing but just dropping some interconnected neuron in a network.

```
model = Sequential()

model.add(LSTM(32,input_shape=(x_train.shape[1:]),activation='relu',return_sequences=True))
model.add(Dropout(0.2))
model.add(Dense(2,activation='softmax'))
```

Fig. 18. Showing Model creation

3) Activation Function used in Model:

a) ReLu: stands for Rectifier Linear unit. It is a non-linear activation function highly used in Machine Learning because it does not activate all the neurons at the same time *[10]*.

F(x)= max (0,1)

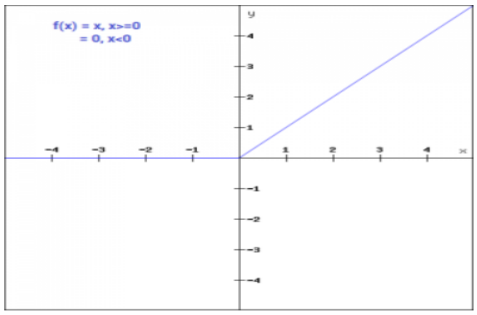

Fig. 19. ReLU function Plot

b) Softmax: It is used to calculate the probability of each target class over the possibility of all target classes. The calculated probability is in the range of 0 to 1 and the sum of all probability is equal to 1. Therefore, softmax is used for multiclass classification.

4) Compile Model:

The compilation is the next step after we define our model. Compilation requires some parameters specific mean for training. In this project, we have used three arguments for model compilation as shown below:

a) *Loss function*: sparse_categorical_crossentropy is used as a loss function. It is provided by tf. Keras library.
b) *Optimizer*: Adam optimizer algorithm is used
c) *Metrics*: Metric is set to 'accuracy'

```
model.compile(loss='sparse_categorical_crossentropy',
    optimizer=tf.keras.optimizers.Adam(lr=1e-3,decay=1e-5),
    metrics=['accuracy'])
```

Fig. 20. Compilation of model and its arguments

5) Fit Model:

Once the model is compiled, it is ready for fit. In this step, model will adapt the weights on the training data set. The network is trained using BPTT (Backpropagation through time) algorithm and optimize using adam optimizer and loss function during compilation.

```
model.fit(x_train,y_train,epochs=10,validation_split=0.2)
```

Fig. 21. Showing Model fit

Here, we have used epochs size 10 and the validation split is set to 0.2. we have total of 350 data out of that 20% is used for the test set and now 280 data is remaining. 20 % of 280 is used for validation therefore remaining data is 224 which is trained in 1 epoch so approximately 10 epochs are used to achieve good accuracy.

6) *Evaluate Model:*

Model evaluation is necessary to check the performance of the model. It is never trained on the training data set, separate 20% of original data is taken for evaluation. Verbose is defined to check the progress of the model here, it is set to 1 and it can also be set to 0.

```
model.evaluate(x_test, y_test, verbose= 1)
```

Fig. 22. Evaluate model

7) *Predict model:*

Three CSV test file is given for predicting the model are. Testfile1.xlsx, Testfile2.xlsx, Testfile3.xlsx. Now we will give x_pred1, x_pred2, x_pred3 as input to model. predict () function to calculate our prediction result y_pred1, y_pred2, y_pred3 as shown in below figure:

x_pred1	Array of float64	(50, 1, 3400)	[[[-0.00115191 -0.00255979 -0.00281577 ... 0.00332773 0.00319974
x_pred2	Array of float64	(50, 1, 3400)	[[[-0.00140788 -0.00102392 -0.00166386 ... -0.00422365 -0.0037117 - ...
x_pred3	Array of float64	(50, 1, 3400)	[[[-0.0037117 -0.00396767 -0.00435164 ... -0.0048636 -0.00435164

Fig. 23. Showing all the three Test files

```
Epoch 1/10
7/7 [==============================] - 1s 53ms/step - loss: 0.6021 - accuracy: 0.6987 - val_loss: 0.4922 - val_accuracy: 0.71
Epoch 2/10
7/7 [==============================] - 0s 14ms/step - loss: 0.4556 - accuracy: 0.7705 - val_loss: 0.4443 - val_accuracy: 0.75
Epoch 3/10
7/7 [==============================] - 0s 9ms/step - loss: 0.4163 - accuracy: 0.7533 - val_loss: 0.3735 - val_accuracy: 0.803
Epoch 4/10
7/7 [==============================] - 0s 8ms/step - loss: 0.3438 - accuracy: 0.8460 - val_loss: 0.3115 - val_accuracy: 0.946
Epoch 5/10
7/7 [==============================] - 0s 9ms/step - loss: 0.2604 - accuracy: 0.9515 - val_loss: 0.2639 - val_accuracy: 1.000
Epoch 6/10
7/7 [==============================] - 0s 8ms/step - loss: 0.2525 - accuracy: 0.9641 - val_loss: 0.2311 - val_accuracy: 1.000
Epoch 7/10
7/7 [==============================] - 0s 9ms/step - loss: 0.2050 - accuracy: 0.9871 - val_loss: 0.2054 - val_accuracy: 1.000
Epoch 8/10
7/7 [==============================] - 0s 11ms/step - loss: 0.1896 - accuracy: 0.9812 - val_loss: 0.1893 - val_accuracy: 1.00
Epoch 9/10
7/7 [==============================] - 0s 9ms/step - loss: 0.1580 - accuracy: 0.9856 - val_loss: 0.1787 - val_accuracy: 1.000
Epoch 10/10
7/7 [==============================] - 0s 9ms/step - loss: 0.1707 - accuracy: 0.9975 - val_loss: 0.1723 - val_accuracy: 1.000
3/3 [==============================] - 0s 2ms/step - loss: 0.1123 - accuracy: 1.0000
Result of Testfile_1=y_pred1
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0]
Result of Testfile_2=y_pred2
[0 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 0 0 0 1 1 0
 1 1 1 1 0 0 0 1 1 1 1 1 0]
Result of Testfile_3=y_pred3
[0 0 1 0 0 1 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 0 0 0 1 1 1 1 0 1 1
 1 1 1 1 1 1 1 0 1 1 1 0 1]
```

Fig. 24. Model Training and prediction results

As it is evident from the above figure, model accuracy is 100% in 10 epochs which is quite good, and the prediction result is displayed.

8) *Test Results:*

Prediction Result of Testfile_1:

```
Result of Testfile_1=y_pred1
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Fig. 25. Showing prediction result of Testfile_1.xlsx

Label 1= source G
Label 0 = source A

Prediction Result of Testfile_2:

```
Result of Testfile_2=y_pred2
[0 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 0 0 0 1 1 0
 1 1 1 1 0 0 0 1 1 1 1 1 0]
```

Fig. 26. Showing prediction result of Testfile_2.xlsx

Label 1= source G
Label 0 = source A

Prediction Result of Testfile_3:

```
Result of Testfile_3=y_pred3
[0 0 1 0 0 1 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 0 0 0 1 1 1 1 0 1 1
 1 1 1 1 1 1 1 0 1 1 1 0 1]
```

Fig. 27. Showing prediction result of Testfile_3.xlsx

Label 1= source G
Label 0 = source A

9) *Confusion Matrix:*

It is a table used to describe the performance of a classification model on a set of test data for which the true values are known. It is a table of two dimensions: Actual value and predicted value. A confusion matrix is also known as the error matrix [11].

It has four dimensions which are:

- True positives (TP): It is the case when both actual class and predicted class of data point is 1.
- True Negatives (TN): It is the case when both actual class and predicted class of data point is 0.
- False Positives (FP): It is the case when an actual class of data point is 0 and the predicted class of data point is 1.
- False Negatives (FN): It is the case when an actual class of data point is 1 and the predicted class of data point is 0.

```
y_act=y_test
y_predcm=model.predict(x_test)
y_predcm=np.argmax(y_predcm.reshape((y_predcm.shape[0],y_predcm.shape[2])), axis=1)
con_mat= confusion_matrix(y_act, y_predcm).ravel()
tn,fp,fn,tp=con_mat
display=ConfusionMatrixDisplay(confusion_matrix=con_mat.reshape(2,2))
display.plot()
tpr=tp/(tp + fn)
tnr=tp/(tn + fp)
fdr=tp/(fp + tp)
npv=tn/(tn + fn)
```

Fig. 28. Showing all parameters of the confusion matrix with formula

From the above figure, it is evident that we can calculate the value of tp,fp,fn,tn,tpr, tnr, fdr, npv. All the values are mentioned in fig:30 below

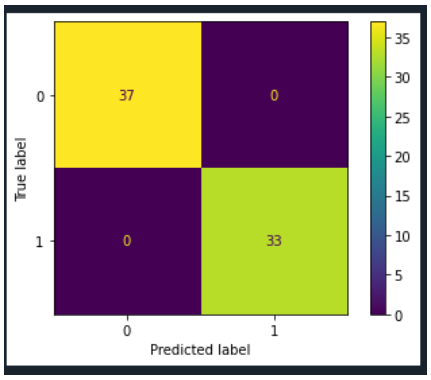

Fig. 29. Confusion Matrix showing True label vs predicted label

Name	Type	Size	Value
fdr	float64	1	1.0
fn	int64	1	0
fp	int64	1	0
npv	float64	1	1.0
tn	int64	1	37
tnr	float64	1	0.8918918918918919
tp	int64	1	33
tpr	float64	1	1.0

Fig. 30. Values of parameters of the confusion matrix

III. Conclusion and Result

In this paper, source identification is successfully achieved with a model accuracy of 100 %. Test results are depicted under section II (8) which validate the successful implementation of the project. Two sources. i.e source A and source G which contain Time series data of a signal is trained on the LSTM model and Model accuracy after 10 epochs is achieved 1. 100 % accuracy is very difficult to achieve in real-world applications with more data set but in our case data set is not too large and diverse. There is always a scope of improvement in model creation by tuning different hyperparameters which can be improved further. Along with the desired result, LSTM Network and neural network training are discussed in detail. We tried to bring all concepts of machine learning into this paper.

IV. Acknowledgment

I would like to thank Professor Dr. Andreas Pech for providing me this opportunity to work on this project. Your guidance, support, encouragement helped me to complete this project. Also, Thank you for providing me data sets.

V. References

[1] [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[2] [Online]. Available: https://blog.timescale.com/blog/what-the-heck-is-time-series-data-and-why-do-i-need-a-time-series-database-dcf3b1b18563/.

[3] [Online]. Available: https://www.w3schools.com/python/matplotlib_intro.asp.

[4] [Online]. Available: https://cloudxlab.com/blog/numpy-pandas-introduction/.

[5] [Online]. Available: https://www.w3schools.com/python/numpy_array_reshape.asp.

[6] [Online]. Available: https://realpython.com/train-test-split-python-data/.

[7] [Online]. Available: https://www.datacamp.com/community/tutorials/tensorflow-tutorial.

[8] [Online]. Available: https://medium.com/mlait/introduction-to-keras-deep-learning-library-2844b39f0496.

[9] [Online]. Available: https://scikit-learn.org/stable/getting_started.html.

[10] [Online]. Available: https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/

[11] [Online]. Available: https://www.geeksforgeeks.org/confusion-matrix-machine-learning/.

VI. SOURCE CODE (LSTM.PY)

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout,LSTM
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

#Loading the data
dataFrameA = pd.read_excel('data/train&test/A.xlsx', header=None)
dataFrameG = pd.read_excel('data/train&test/G.xlsx', header=None)

#Loading Testfiles
dataFramePred1 = pd.read_excel('data/pred/Testfile_1.xlsx', header=None)
dataFramePred2 = pd.read_excel('data/pred/Testfile_2.xlsx', header=None)
dataFramePred3 = pd.read_excel('data/pred/Testfile_3.xlsx', header=None)

#Dropping first 6 columns
x_A = np.array(dataFrameA.drop([0,1,2,3,4,5], axis=1))
x_G = np.array(dataFrameG.drop([0,1,2,3,4,5], axis=1))
x_pred1 = np.array(dataFramePred1.drop([0,1,2,3,4,5], axis=1))
x_pred2 = np.array(dataFramePred2.drop([0,1,2,3,4,5], axis=1))
x_pred3 = np.array(dataFramePred3.drop([0,1,2,3,4,5], axis=1))

#vertical stacking
x = np.vstack((x_A, x_G))

#Reshaping
x = np.reshape(x, (x.shape[0], 1, x.shape[1]))

x_pred1 = np.reshape(x_pred1, (x_pred1.shape[0], 1, x_pred1.shape[1]))
x_pred2 = np.reshape(x_pred2, (x_pred2.shape[0], 1, x_pred2.shape[1]))
x_pred3 = np.reshape(x_pred3, (x_pred3.shape[0], 1, x_pred3.shape[1]))

#Asigning Labels
y = np.hstack((np.zeros(x_A.shape[0]), np.ones(x_G.shape[0])))
```

```
#Data splitting
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)

#Model creation

model = Sequential()

# Add Model
model.add(LSTM(32,input_shape=(x_train.shape[1:]),activation='relu',return_sequences=True))
model.add(Dropout(0.2))

#Compile Model
model.add(Dense(2,activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy',
     optimizer=tf.keras.optimizers.Adam(lr=1e-3,decay=1e-5),
     metrics=['accuracy'])

model.summary()

#Fit model
model.fit(x_train,y_train,epochs=10,validation_split=0.2)

# evaluate model

model.evaluate(x_test, y_test, verbose= 1)

#confusion Matrix
y_act=y_test
y_predcm=model.predict(x_test)
y_predcm=np.argmax(y_predcm.reshape((y_predcm.shape[0],y_predcm.shape[2])), axis=1)
con_mat= confusion_matrix(y_act, y_predcm).ravel()
tn,fp,fn,tp=con_mat
display=ConfusionMatrixDisplay(confusion_matrix=con_mat.reshape(2,2))
display.plot()
tpr=tp/(tp + fn)
tnr=tp/(tn + fp)
fdr=tp/(fp + tp)
npv=tn/(tn + fn)
```

```
#model save and load
model.save("lstm.h5")

model = tf.keras.models.load_model("lstm.h5")

#predect model
y_pred1 = model.predict(x_pred1)
y_pred1 = np.reshape(y_pred1, (y_pred1.shape[0], y_pred1.shape[2]))
y_pred1 = np.argmax(y_pred1, axis= 1)
print('Result of Testfile_1=y_pred1')
print(y_pred1)



y_pred2 = model.predict(x_pred2)
y_pred2 = np.reshape(y_pred2, (y_pred2.shape[0], y_pred2.shape[2]))
y_pred2 = np.argmax(y_pred2, axis= 1)
print('Result of Testfile_2=y_pred2')
print(y_pred2)

y_pred3 = model.predict(x_pred3)
y_pred3 = np.reshape(y_pred3, (y_pred3.shape[0], y_pred3.shape[2]))
y_pred3 = np.argmax(y_pred3, axis= 1)
print('Result of Testfile_3=y_pred3')
print(y_pred3)
```

VII. Variable Explorer window

x_pred3	Array of float64	(50, 1, 3400)	[[[-0.0037117 -0.00396767 -0.00435164 ... -0.0048636 -0.00435164 ...
x_test	Array of float64	(70, 1, 3400)	[[[-0.00089593 -0.0012799 -0.00115191 ... 0.00831932 0.0072954 ...
x_train	Array of float64	(280, 1, 3400)	[[[-0.00319974 -0.00281577 -0.00230381 ... -0.0072954 -0.00883128 ...
y	Array of float64	(350,)	[0. 0. 0. ... 1. 1. 1.]
y_act	Array of float64	(70,)	[0. 0. 1. ... 0. 1. 0.]
y_pred1	Array of int64	(50,)	[0 0 0 ... 0 0 0]
y_pred2	Array of int64	(50,)	[1 1 1 ... 1 1 1]
y_pred3	Array of int64	(50,)	[1 1 1 ... 1 1 1]
y_predcm	Array of int64	(70,)	[0 0 1 ... 0 1 0]
y_test	Array of float64	(70,)	[0. 0. 1. ... 0. 1. 0.]
y_train	Array of float64	(280,)	[1. 1. 1. ... 0. 0. 1.]

Name	Type	Size	Value
a	Array of int64	(70,)	[0 0 1 ... 0 1 0]
con_mat	Array of int64	(4,)	[37 0 0 33]
dataFrameA	DataFrame	(150, 3406)	Column names: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 ...
dataFrameG	DataFrame	(200, 3406)	Column names: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 ...
dataFramePred	DataFrame	(50, 3406)	Column names: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 ...
dataFramePred1	DataFrame	(50, 3406)	Column names: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 ...
dataFramePred2	DataFrame	(50, 3406)	Column names: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 ...
dataFramePred3	DataFrame	(50, 3406)	Column names: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 ...
display	metrics._plot.co…	1	ConfusionMatrixDisplay object of sklearn.metrics._plot.confusion_matri ...
fdr	float64	1	1.0
fn	int64	1	0
fp	int64	1	0
fp	int64	1	0
npv	float64	1	1.0
tn	int64	1	37
tnr	float64	1	0.8918918918918919
tp	int64	1	33
tpr	float64	1	1.0
x	Array of float64	(350, 3400)	[[-0.00140788 -0.00102392 -0.00102392 ... 0.00870329 0.00844731 0 ...
x_A	Array of float64	(150, 3400)	[[-0.00140788 -0.00102392 -0.00102392 ... 0.00870329 0.00844731 0 ...
x_G	Array of float64	(200, 3400)	[[-0.0012799 -0.00191984 -0.00179185 ... -0.00217582 -0.00255979 -0 ...
x_pred	Array of float64	(50, 1, 3400)	[[[-0.00140788 -0.00102392 -0.00166386 ... -0.00422365 -0.0037117 - ...
x_pred1	Array of float64	(50, 1, 3400)	[[[-0.00115191 -0.00255979 -0.00281577 ... 0.00332773 0.00319974 ...
x_pred2	Array of float64	(50, 1, 3400)	[[[-0.00140788 -0.00102392 -0.00166386 ... -0.00422365 -0.0037117 - ...
x_pred3	Array of float64	(50, 1, 3400)	[[[-0.0037117 -0.00396767 -0.00435164 ... -0.0048636 -0.00435164 ...

x_pred3	Array of float64	(50, 1, 3400)	[[[-0.0037117 -0.00396767 -0.00435164 ... -0.0048636 -0.00435164 ...
x_test	Array of float64	(70, 1, 3400)	[[[-0.00089593 -0.0012799 -0.00115191 ... 0.00831932 0.0072954 ...
x_train	Array of float64	(280, 1, 3400)	[[[-0.00319974 -0.00281577 -0.00230381 ... -0.0072954 -0.00883128 ...
y	Array of float64	(350,)	[0. 0. 0. ... 1. 1. 1.]
y_act	Array of float64	(70,)	[0. 0. 1. ... 0. 1. 0.]
y_pred1	Array of int64	(50,)	[0 0 0 ... 0 0 0]
y_pred2	Array of int64	(50,)	[1 1 1 ... 1 1 1]
y_pred3	Array of int64	(50,)	[1 1 1 ... 1 1 1]
y_predcm	Array of int64	(70,)	[0 0 1 ... 0 1 0]
y_test	Array of float64	(70,)	[0. 0. 1. ... 0. 1. 0.]
y_train	Array of float64	(280,)	[1. 1. 1. ... 0. 0. 1.]