National College of Ireland

Configuration Manual

Detecting Type and Severity of Speech Impairment using Deep-Learning Algorithms and Clustering

MSc. in Data Analytics

Ankit Chatterjee

Student ID: x21169993

School of Computing

National College of Ireland

Supervisor: Hicham Rifai

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Ankit Chatterjee
Student ID:	X21169993
Programme:	MSc in Data Analytics **Year:** 2023
Module:	Research Project
Lecturer:	Hicham Rifai
Submission Due Date:	14/08/2023
Project Title:	Detecting Type and Severity of Speech Impairment using Deep-Learning Algorithms and Clustering
Word Count:	1067 **Page Count: 11**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Ankit Chatterjee

Date: 14/08/2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□

Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Detecting Type and Severity of Speech Impairment using Deep-Learning Algorithms and Clustering

Ankit Chatterjee
21169993

1 Introduction

This project deals with the detection of the type of speech impediments using the Long Short-Term Memory Recursive Neural Network deep learning technique. The project makes use of raw audio files as dataset and splits the dataset into train and test validation sets. This document serves as a configuration manual and helps us to understand the configuration details of the entire project.

2. Environment configuration

To develop the code for the project, we have used python programming language. The version of python used for this project is 3.10.2.

```
0s  !python --version

    Python 3.10.12
```

2.1 Google Colab Pro

In this project, we have used Google Colab pro to run and execute the python code. Google Colab is a hosted Jupyter Notebook Service that provides access to the computing resources like GPUs and TPUs. The reason behind using the Colab service is its suitability for data mining, data science and machine learning projects. Google Colab provides a platform for the code to run in an environment with high specifications.

2.2 Python Libraries

The project code requires several python libraries for different steps. The following is the list of all the libraries that are required for importing, preprocessing and training of the data.

```
!pip list command
```

```
Package                           Version
--------------------------------- ---------------------
absl-py                           1.4.0
aiohttp                           3.8.5
aiosignal                         1.3.1
alabaster                         0.7.13
albumentations                    1.3.1
altair                            4.2.2
annotated-types                   0.5.0
anyio                             3.7.1
appdirs                           1.4.4
argon2-cffi                       21.3.0
argon2-cffi-bindings              21.2.0
array-record                      0.4.1
arviz                             0.15.1
astropy                           5.3.1
astunparse                        1.6.3
async-timeout                     4.0.2
attrs                             23.1.0
audioread                         3.0.0
autograd                          1.6.2
Babel                             2.12.1
backcall                          0.2.0
beautifulsoup4                    4.11.2
bleach                            6.0.0
blinker                           1.4
blis                              0.7.10
blosc2                            2.0.0
bokeh                             3.2.1
branca                            0.6.0
build                             0.10.0
CacheControl                      0.13.1
cachetools                        5.3.1
catalogue                         2.0.9
certifi                           2023.7.22
cffi                              1.15.1
chardet                           4.0.0
charset-normalizer                3.2.0
chex                              0.1.7
click                             8.1.6
click-plugins                     1.1.1
cligj                             0.7.2
cloudpickle                       2.2.1
cmake                             3.27.1
cmdstanpy                         1.1.0
colorcet                          3.0.1
colorlover                        0.3.0
community                         1.0.0b1
confection                        0.1.1
cons                              0.4.6
contextlib2                       21.6.0
contourpy                         1.1.0
convertdate                       2.4.0
cryptography                      41.0.3
cufflinks                         0.17.3
cvxopt                            1.3.2
cvxpy                             1.3.2
cycler                            0.11.0
```

The above snippet shows the different libraries required for different steps of the project.

3. Data Collection:

The data used for this project is in the form of audio data files that comprise of the voice recordings of patients suffering from stammering and dysarthria. The data for stammering is collected from the audio library of the University College London and all the data is in '.wav' format. The data for the dysarthria voice samples and control group is collected from Kaggle website.

4. Data Storage:

The data collected is stored in different directories with a label on each library in the google Colab platform. There are three directories for each of the labels i.e., 'stammering', 'dysarthria' and 'normal' speech. The below code snippet demonstrates the procedure for accessing the data in the Colab platform.

```
directory = '/content/'

# Iterate over subdirectories in the main directory
for subdir in os.listdir(directory):
    # Create full path to subdirectory
    subdir_path = os.path.join(directory, subdir)


    if not os.path.isdir(subdir_path):
        continue

    #Assigning Labels
    if 'stammer' in subdir:
        label = 0  # stammering
    elif 'normal' in subdir:
        label = 1  # normal speech
    elif 'dysarthria' in subdir:
        label = 2  # dysarthria speech
    else:
        continue  # if the folder does not match any of the above, skip

    # Iterate over files in subdirectory
    for filename in os.listdir(subdir_path):
        if filename.endswith(".wav"):
            # Create full path to audio file
            filepath = os.path.join(subdir_path, filename)
            audio_files.append(filepath)
            labels.append(label)
```

5. Stratified Sampling of the Data:

In the below code snippet, we use the 'train_test_split' function of the 'sklearn' package to split the entire dataset into test and train data. In this case, a stratified sampling technique is used to maintain the ratio of the test and train data sets.

```
train_files, test_files, train_labels, test_labels = train_test_split(audio_files, labels, test_size=0.2, stratify=labels, random_state=42)
```

6. DataLoader:

To ensure the effective use of memory, we use the PyTorch's DataLoader instance for this project. PyTorch is an opensource machine learning framework that makes use of the Python Programming language and the Torch library.

The DataLoader is used to load the audio data files in batches rather than loading the entire dataset at one instance.

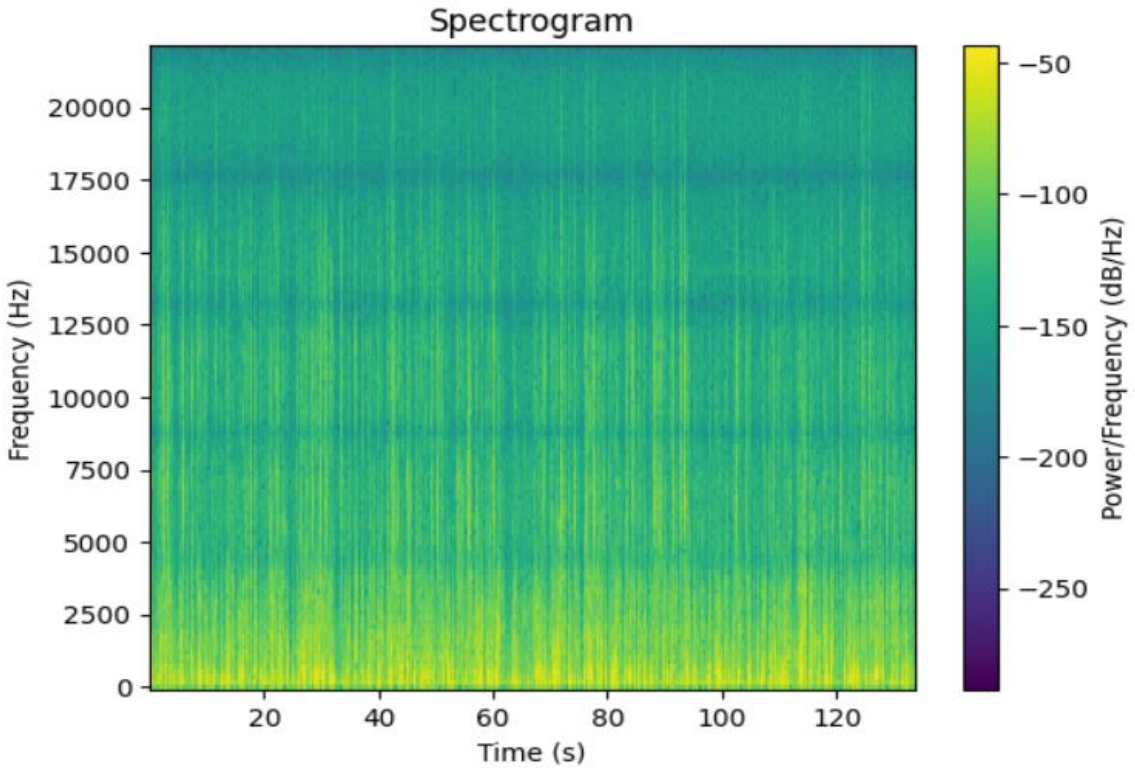
```
BATCH_SIZE = 64
LEARNING_RATE = 0.0001
EPOCHS = 50
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
train_dataloader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
test_dataloader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=True)
```

7. Plotting Spectrograms:

Spectrograms are widely used when examining data in the form of audio files. Spectrograms are like images of a photograph of a signal. It plots Frequencies in X-axis and Time in Y-axis. In the spectrograms used in this project, the plot conveys the strength of the waveform. The higher the energy of the waveform, brighter the color of the spectrogram.

The below snippet shows the spectrogram of smoothed audio waveform post of normal speech post the noise cancellation and the smoothing techniques.

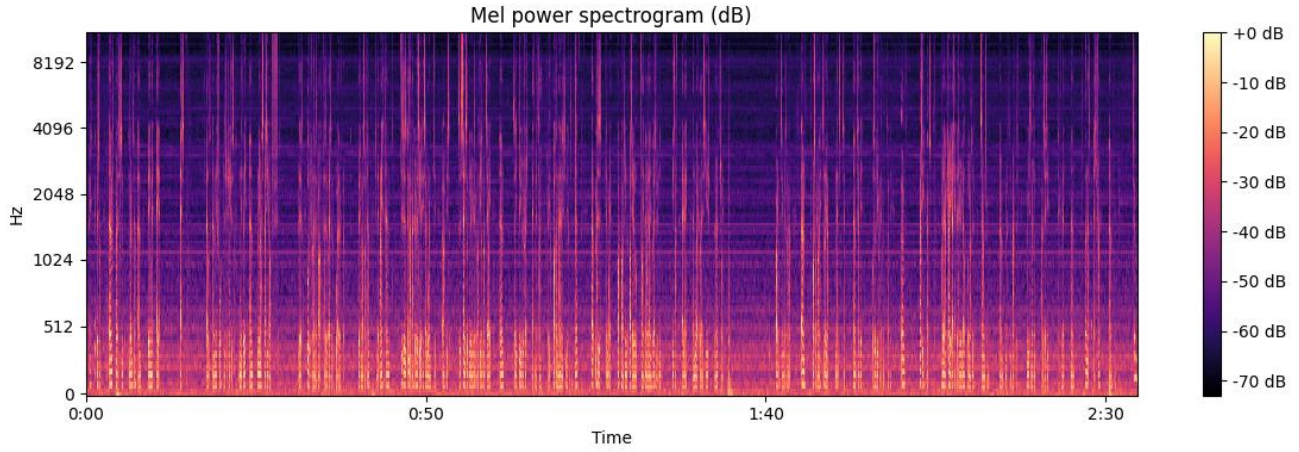


As part of the project, we use three datasets for analysis- Stammering, dysarthria and normal speech. Below are three Mel-Spectrograms for each type of speech.

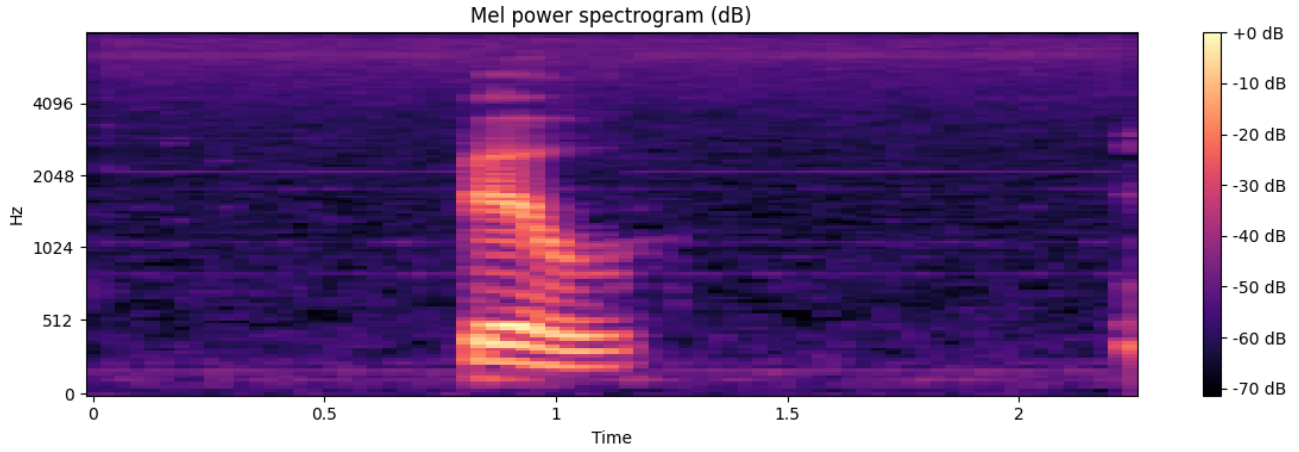
```
import numpy as np

def plot_sample_spectrogram(dataset):
    # Select the first sample of each class
    classes = [0, 1, 2]  # 0: stammering, 1: normal, 2: dysarthria
    samples = {c: next(file for (file, label) in zip(dataset.files, dataset.labels) if label == c) for c in classes}

    for c, file in samples.items():
        waveform, sr = torchaudio.load(file, normalize=True)
        waveform = waveform.squeeze().numpy()

        print(f"Class: {c}")

        # Compute the Mel spectrogram
        S = librosa.feature.melspectrogram(y=waveform, sr=sr, n_mels=128)

        # Convert to log scale (dB). We'll use the peak power (max) as reference.
        log_S = librosa.power_to_db(S, ref=np.max)

        # Plot the Mel spectrogram
        plt.figure(figsize=(12,4))
        librosa.display.specshow(log_S, sr=sr, x_axis='time', y_axis='mel')
        plt.title('Mel power spectrogram (dB)')
        plt.colorbar(format='%+02.0f dB')
        plt.tight_layout()
        plt.show()

# Call the function after creating the datasets
plot_sample_spectrogram(train_dataset)
```

The below Mel-Spectrograms are for the following three speech types respectively.

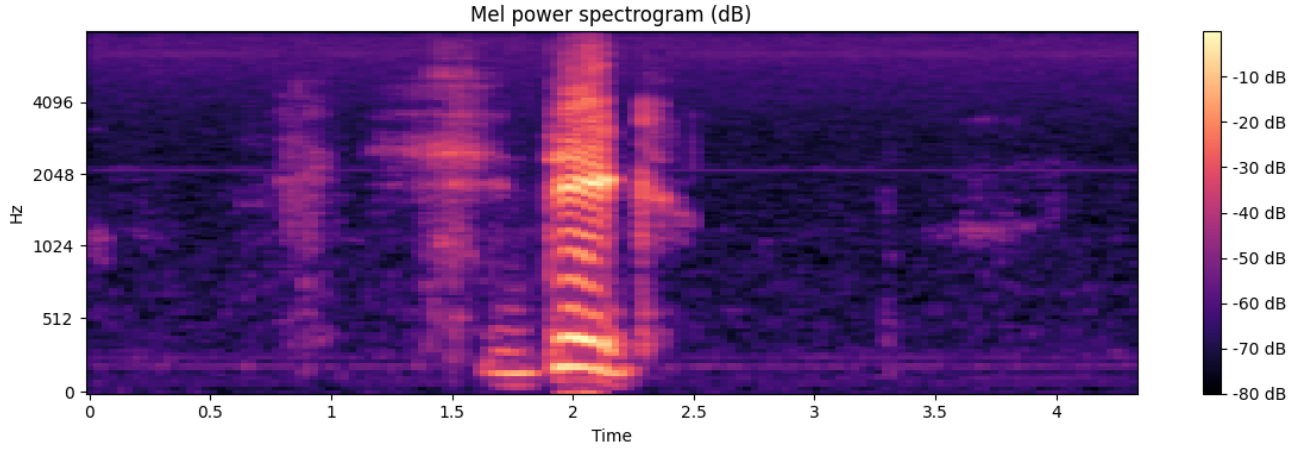
Stammering.



Normal **Speech**



Dysarthria



8. Defining and Training the LSTM-RNN Model:

The following are the libraries used for defining and training the LSTM model.

1. 'torch.nn': The layers of the LSTM model are defined with the PyTorch's neural network module.
2. torch.optim.Adam: This is used for implementing the Adam optimizer during the training of the model.
3. 'nn.CrossEntropyLoss': This is used for calculating the Cross Entropy Loss during the training of the model.

```
model = LSTMNetwork(num_mfcc).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
loss_fn = nn.CrossEntropyLoss()
```

4. 'train_single_epoch' This function is used for training one epoch so as to minimize the time taken during training of the model.

```
def train_single_epoch(model, data_loader, loss_fn, optimizer, device):
    for waveforms, labels in data_loader:
        waveforms, labels = waveforms.to(device), labels.to(device)
        outputs = model(waveforms)
        loss = loss_fn(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print(f"Loss: {loss.item()}")
```

5. 'train': This function is used to train the LSTM-RNN model on the entire dataset for the specified epochs.

```
def train(model, data_loader, loss_fn, optimizer, num_epochs, device):
    for epoch in range(num_epochs):
        print(f"Epoch {epoch+1}")
        train_single_epoch(model, data_loader, loss_fn, optimizer, device)
        print("---------------------------------------------")
    print("Training completed!!")
```

```
train(model, train_dataloader, loss_fn, optimizer, EPOCHS, device)
```

```
Epoch 1
Loss: 0.994907796382904
---------------------------------------------
Epoch 2
Loss: 1.0184102058410645
---------------------------------------------
Epoch 3
Loss: 1.001320242881775
---------------------------------------------
Epoch 4
Loss: 1.0130590200424194
---------------------------------------------
Epoch 5
Loss: 1.0657325983047485
---------------------------------------------
Epoch 6
Loss: 0.9718074798583984
---------------------------------------------
Epoch 7
Loss: 0.9121618866920471
```

9. Testing the Model:

The ‘test’ function is used for testing the model on the test data.

```
def test(model, data_loader, device):
    model.eval()
    correct_predictions = 0
    total_predictions = 0

    with torch.no_grad():
        for waveforms, labels in data_loader:
            waveforms, labels = waveforms.to(device), labels.to(device)
            outputs = model(waveforms)
            _, predicted = torch.max(outputs.data, 1)
            total_predictions += labels.size(0)
            correct_predictions += (predicted == labels).sum().item()

    print(f'Accuracy: {correct_predictions / total_predictions * 100:.2f}%')

test(model, test_dataloader, device)
```

10. Accuracy:

The accuracy of the model is calculated as 83.33%.

```
def test(model, data_loader, device):
    model.eval()
    correct_predictions = 0
    total_predictions = 0

    with torch.no_grad():
        for waveforms, labels in data_loader:
            waveforms, labels = waveforms.to(device), labels.to(device)
            outputs = model(waveforms)
            _, predicted = torch.max(outputs.data, 1)
            total_predictions += labels.size(0)
            correct_predictions += (predicted == labels).sum().item()

    print(f'Accuracy: {correct_predictions / total_predictions * 100:.2f}%')

test(model, test_dataloader, device)
```

```
Accuracy: 83.33%
```

11. Severity:

To detect the severity of the speech impediments, we use the K-Means Clustering method. We import the KMeans library from sklearn package. The code snippet for the plot is as follows:

```
from sklearn.cluster import KMeans
```

```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

# Initialize t-SNE
tsne = TSNE(n_components=2, random_state=0)

# Use only 1000 instances to speed up the process
smaller_mfcc_features = mfcc_features[:1000]
smaller_labels = severity_labels[:1000]

# Apply t-SNE to the data
tsne_results = tsne.fit_transform(smaller_mfcc_features)

# Create a scatter plot
plt.figure(figsize=(10,10))
for severity_class in [0, 1, 2]:
    # Select just the instances for this severity class
    selection = tsne_results[smaller_labels == severity_class]
    # Plot these instances
    plt.scatter(selection[:, 0], selection[:, 1], label=f"Class {severity_class}")

plt.legend()
plt.show()
```

In the below figure, the severity of the impediment is represented as follows:

Class 0 represents 'Mild Severity.'
Class 1 represents 'Moderate Severity.'
Class 2 represents 'Severe Severity.'