Configuration Manual

MSc Research Project
Data Analytics

Jacob Benny Packiaraj
Student ID: x22188801

School of Computing
National College of Ireland

Supervisor: Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Jacob Benny Packiaraj
Student ID:	X22188801
Programme:	Data Analytics
Year:	2023-2024
Module:	M Sc Research Project
Lecturer:	Vladimir Milosavljevic
Submission Due Date:	16/09/2024
Project Title:	Integrating audio and text data with deep learning to detect depression
Word Count:	1885
Page Count:	14

Signature:	Jacob Benny Packiaraj
Date:	12/08/2024

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):	

Configuration Manual

Jacob Benny Packiaraj
x22188801

1 Introduction

This configuration manual is related to the project 'Integrating audio and text data with deep learning to detect depression'. The report contains the information about the system configuration, the site to request access and download the dataset, the steps carried out for preprocessing and the implementation of deep learning models. The report also contains some high-level information of the dataset.

2 System Configuration

Jupyter notebook IDE is installed in the local machine is used for the development, preprocessing, model building and training, and evaluation for this project.

2.1 Hardware

A 64-bit windows 11 operating system machine is used in this project. It has 16 GB of RAM and requires minimum 100GB of storage to place the audio files of the participants. The processor in the machine is Intel i5-12500H with an integrated Intel Iris Xe graphics which is used to train the deep learning models.

2.2 Software

A local instance of 64-Bit Anaconda navigator 2.6.0 is installed to access Jupyter notebook for the development of the learning models.

Python 3.12.4 is the programming language used to connect with the dataset and to build the models.

2.3 Python packages used

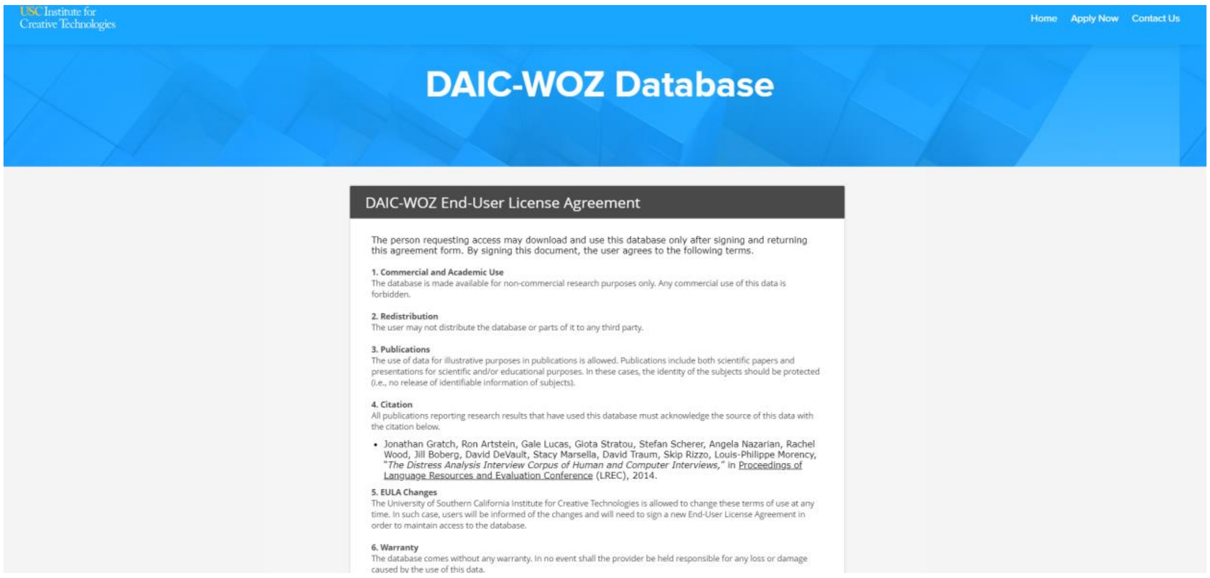
The modules and libraries used for the development of deep learning models are

Data Manipulation	: Numpy 1.26.4 and Pandas 2.2.2
Data Preprocessing and Evaluation	: Scikit- learn 1.4.2
Deep learning	: tensorflow 2.17.0 and pytorch
Modelling	: Keras 3.4.1, keras-tuner 1.4.7
Visualization	: Seaborn 0.13.2, Matplotlib 3.8.4
Storing large datasets	: h5py 3.11.0
Data Manipulation in transcript	: Spacy 3.7.5
Natural language processing	: nltk 3.8.1 (Natural language toolkit)
Data Imbalance	: imblearn 0.12.3 for performing SMOTE
Metrics	: Scikit- learn 1.4.2
Audio file manipulation	: pyAudioAnalysis 0.3.14
Audio file creation	: scipy 1.12.0

Reading and writing audio file	: wave 1.5.0
Audio Analysis	: librosa 0.10.2
Ensemble models	: Scikit- learn 1.4.2 for Gradient boosting

3 Dataset

The dataset used for in this research is Distress Analysis Interview Corpus (DAIC-WOZ) containing clinical interviews to support the diagnosis of psychological distress conditions such as anxiety, depression and post-traumatic stress disorder (PTSD). To gain access to the dataset, submit a non-disclosure agreement in the website from an academic e-mail address.



DAIC-WOZ End-User License Agreement

The person requesting access may download and use this database only after signing and returning this agreement form. By signing this document, the user agrees to the following terms.

1. Commercial and Academic Use
The database is made available for non-commercial research purposes only. Any commercial use of this data is forbidden.

2. Redistribution
The user may not distribute the database or parts of it to any third party.

3. Publications
The use of data for illustrative purposes in publications is allowed. Publications include both scientific papers and presentations for scientific and/or educational purposes. In these cases, the identity of the subjects should be protected (i.e., no release of identifiable information of subjects).

4. Citation
All publications reporting research results that have used this database must acknowledge the source of this data with the citation below.

- Jonathan Gratch, Ron Artstein, Gale Lucas, Giota Stratou, Stefan Scherer, Angela Nazarian, Rachel Wood, Jill Boberg, David DeVault, Stacy Marsella, David Traum, Skip Rizzo, Louis-Philippe Morency, "*The Distress Analysis Interview Corpus of Human and Computer Interviews,*" in Proceedings of Language Resources and Evaluation Conference (LREC), 2014.

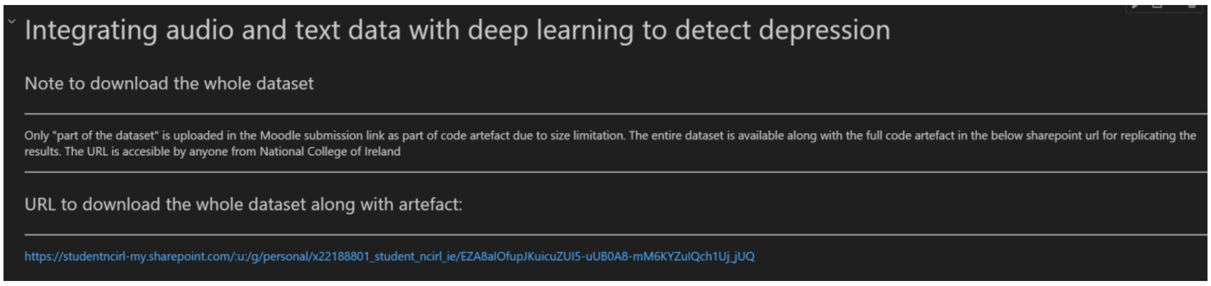
5. EULA Changes
The University of Southern California Institute for Creative Technologies is allowed to change these terms of use at any time. In such case, users will be informed of the changes and will need to sign a new End-User License Agreement in order to maintain access to the database.

6. Warranty
The database comes without any warranty. In no event shall the provider be held responsible for any loss or damage caused by the use of this data.

Figure 1: Dataset request form DAIC WOZ.

The dataset contains 189 recording of the participants in the clinical study and has the transcript file and audio and video features extracted from the participant's recording.

To download the complete dataset that is used for this research project,follow the steps outlined in the Jupyter notebook that is accessible at the URL. Due to space constraint in Moodle, the complete dataset and code artefact is available in this OneDrive URL and it is accessible by users from National College of Ireland.



Integrating audio and text data with deep learning to detect depression

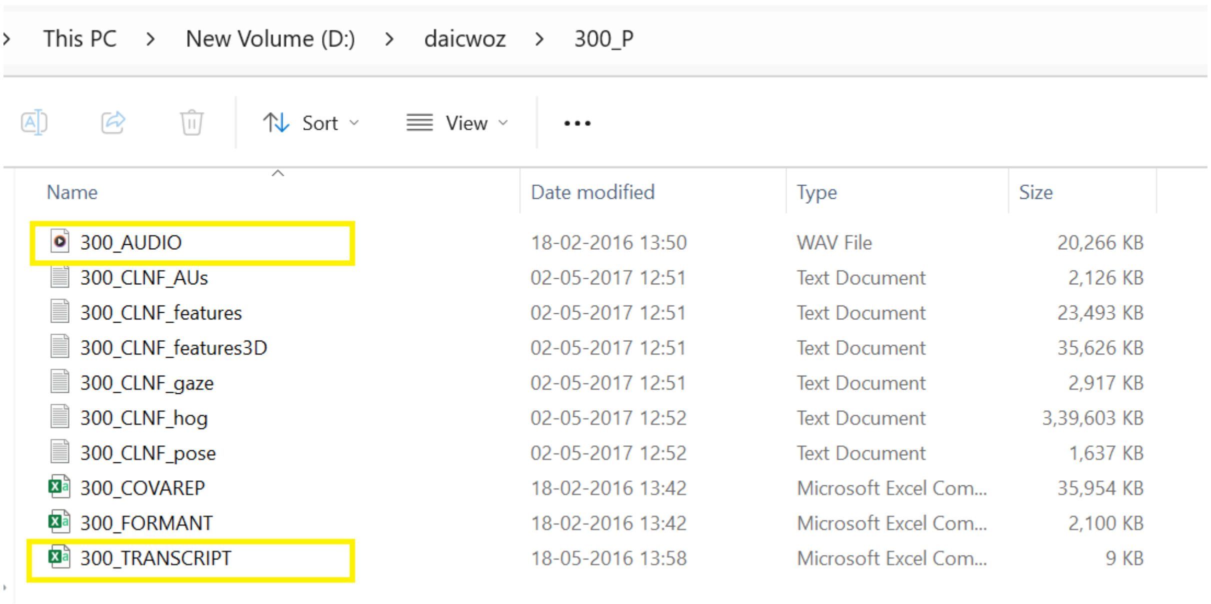
Note to download the whole dataset

Only "part of the dataset" is uploaded in the Moodle submission link as part of code artefact due to size limitation. The entire dataset is available along with the full code artefact in the below sharepoint url for replicating the results. The URL is accesible by anyone from National College of Ireland

URL to download the whole dataset along with artefact:

https://studentncirl-my.sharepoint.com/:u:/g/personal/x22188801_student_ncirl_ie/EZA8alOfupJKuicuZUI5-uUB0A8-mM6KYZuIQch1Uj_jUQ

Figure 2: Full Dataset available in the URL.

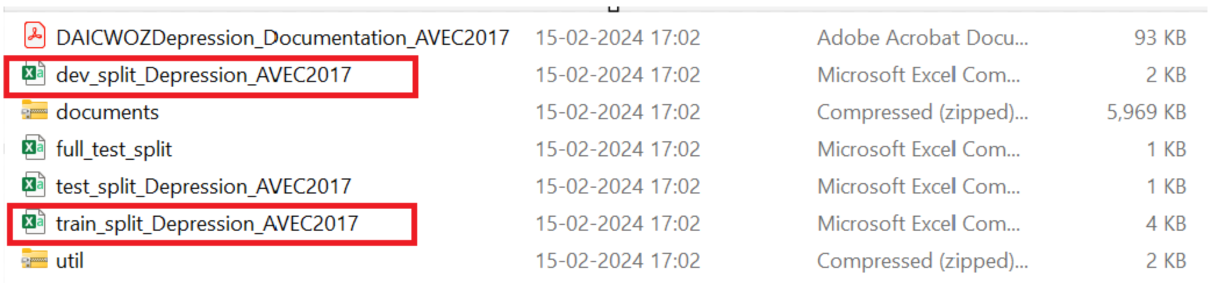


> This PC > New Volume (D:) > daicwoz > 300_P

Name	Date modified	Type	Size
300_AUDIO	18-02-2016 13:50	WAV File	20,266 KB
300_CLNF_AUs	02-05-2017 12:51	Text Document	2,126 KB
300_CLNF_features	02-05-2017 12:51	Text Document	23,493 KB
300_CLNF_features3D	02-05-2017 12:51	Text Document	35,626 KB
300_CLNF_gaze	02-05-2017 12:51	Text Document	2,917 KB
300_CLNF_hog	02-05-2017 12:52	Text Document	3,39,603 KB
300_CLNF_pose	02-05-2017 12:52	Text Document	1,637 KB
300_COVAREP	18-02-2016 13:42	Microsoft Excel Com...	35,954 KB
300_FORMANT	18-02-2016 13:42	Microsoft Excel Com...	2,100 KB
300_TRANSCRIPT	18-05-2016 13:58	Microsoft Excel Com...	9 KB

Figure 3: Files from the dataset used for the analysis.

For this research, the highlighted files xxx_AUDIO and xxx_TRANSCRIPT is picked from all the participant's recordings and features are extracted from these files for deep learning (Figure 3).

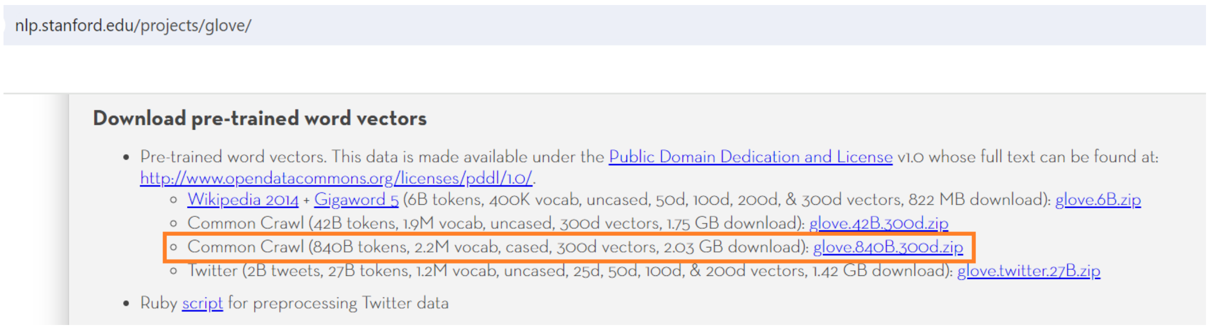


DAICWOZDepression_Documentation_AVEC2017	15-02-2024 17:02	Adobe Acrobat Docu...	93 KB
dev_split_Depression_AVEC2017	15-02-2024 17:02	Microsoft Excel Com...	2 KB
documents	15-02-2024 17:02	Compressed (zipped)...	5,969 KB
full_test_split	15-02-2024 17:02	Microsoft Excel Com...	1 KB
test_split_Depression_AVEC2017	15-02-2024 17:02	Microsoft Excel Com...	1 KB
train_split_Depression_AVEC2017	15-02-2024 17:02	Microsoft Excel Com...	4 KB
util	15-02-2024 17:02	Compressed (zipped)...	2 KB

Figure 4: Files for splitting the dataset into train and test.

The dataset is split into train and test based on the files train_split_Depression_AVEC2017 and dev_split_Depression_AVEC2017 respectively (Figure 4). The individual files contain 107 and 35 records respectively where 107 participant's features is used for the training of the model and 35 participant's features is used for the testing of the model.

GloVe – Global Vector for word representation is used for the natural language processing and the file "glove.840B.300d" is downloaded from the Glove website and placed in the root directory where the Jupyter notebook runs (Figure 5).



nlp.stanford.edu/projects/glove/

Download pre-trained word vectors

- Pre-trained word vectors. This data is made available under the Public Domain Dedication and License v1.0 whose full text can be found at: http://www.opendatacommons.org/licenses/pddl/1.0/.
  - Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): glove.6B.zip
  - Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip
  - Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip
  - Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download): glove.twitter.27B.zip
- Ruby script for preprocessing Twitter data

Figure 5: URL to download GloVe pre-trained word vectors.

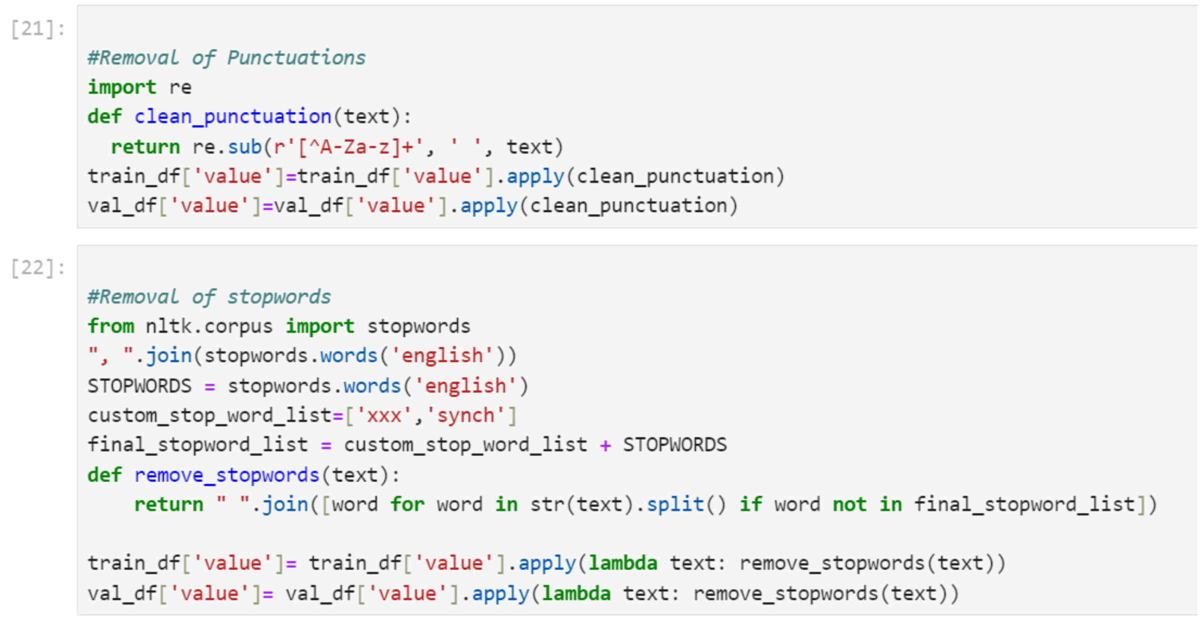
The audio and the transcript file are placed inside train and test folders respectively based on the train -test split file and these 2 folders are placed inside "wavFiles" and "Transcript" folders respectively in the root.

4 Preprocessing

The preprocessing steps contains the steps involved in extracting the features the raw file and transformed into Numpy arrays to feed into the deep learning models for binary classification of the depressive state. This section contains the highlights on the pre-processing followed in this research.

4.1 Transcript processing

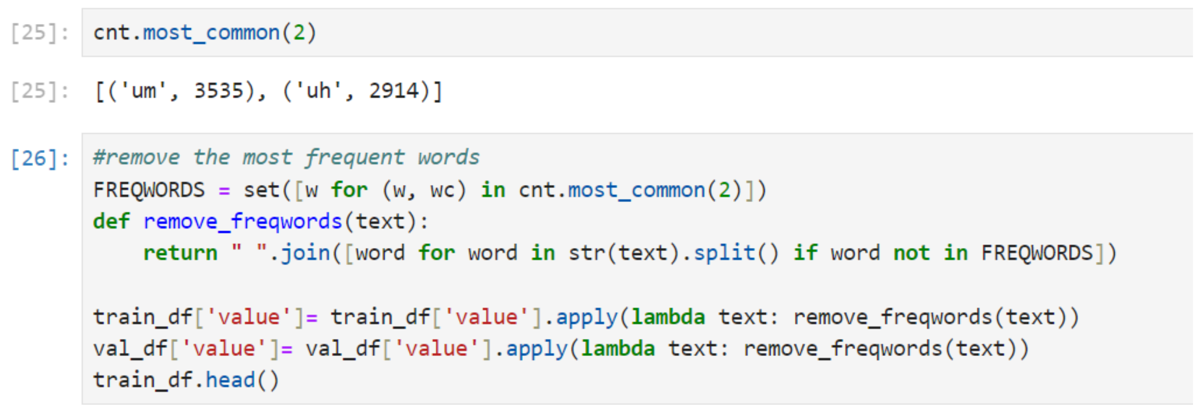
The preprocessing steps followed in the transcript file are the removal of punctuations, removal of stopwords, removal of most frequently used words. In this dataset "um" and "uh" are used predominantly by the participants and it is removed from the dataset.



```
[21]:
#Removal of Punctuations
import re
def clean_punctuation(text):
  return re.sub(r'[^A-Za-z]+', ' ', text)
train_df['value']=train_df['value'].apply(clean_punctuation)
val_df['value']=val_df['value'].apply(clean_punctuation)
```

```
[22]:
#Removal of stopwords
from nltk.corpus import stopwords
", ".join(stopwords.words('english'))
STOPWORDS = stopwords.words('english')
custom_stop_word_list=['xxx','synch']
final_stopword_list = custom_stop_word_list + STOPWORDS
def remove_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in final_stopword_list])

train_df['value']= train_df['value'].apply(lambda text: remove_stopwords(text))
val_df['value']= val_df['value'].apply(lambda text: remove_stopwords(text))
```

Figure 6: Pre-processing of transcript files-1.

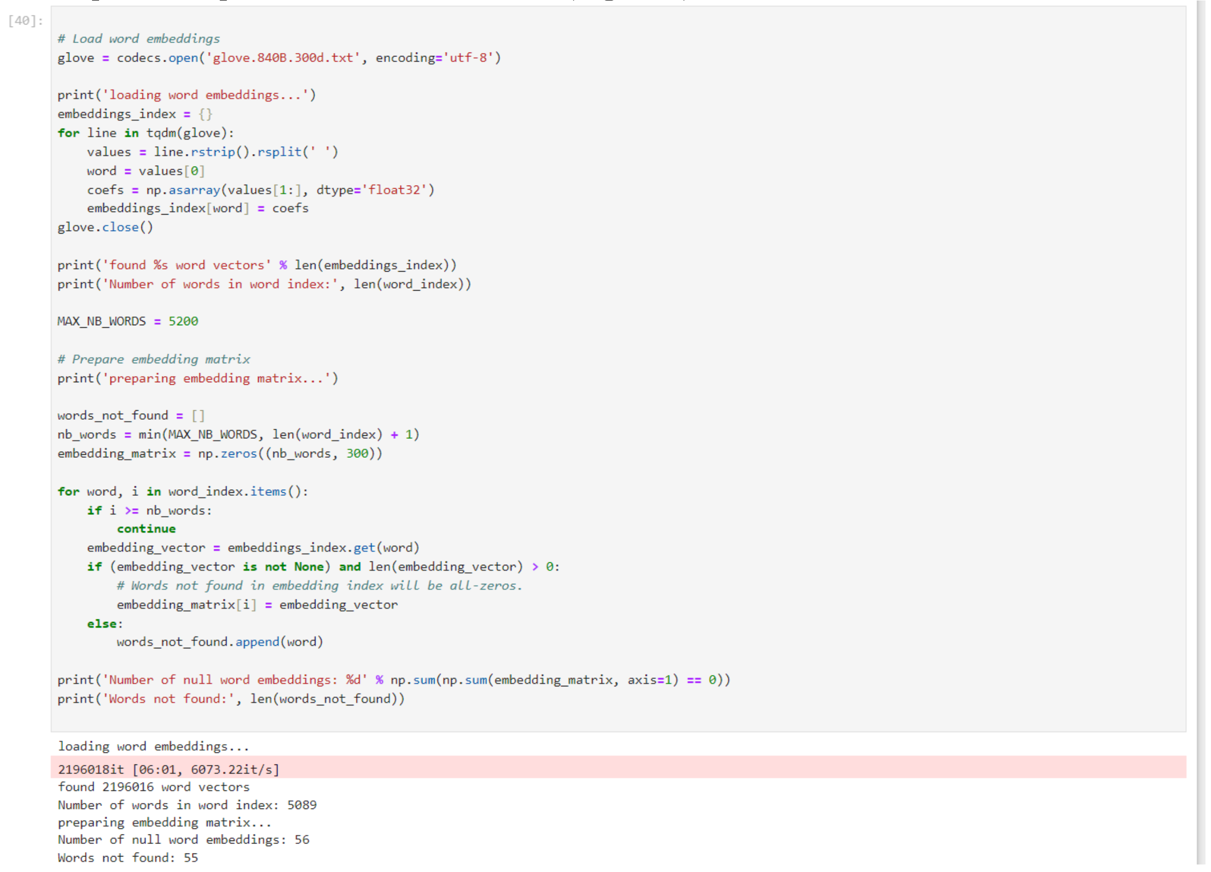


```
[25]: cnt.most_common(2)

[25]: [('um', 3535), ('uh', 2914)]

[26]: #remove the most frequent words
      FREQWORDS = set([w for (w, wc) in cnt.most_common(2)])
      def remove_freqwords(text):
          return " ".join([word for word in str(text).split() if word not in FREQWORDS])

      train_df['value']= train_df['value'].apply(lambda text: remove_freqwords(text))
      val_df['value']= val_df['value'].apply(lambda text: remove_freqwords(text))
      train_df.head()
```

Figure 7: Pre-processing of transcript files 2.

Lemmatization of the text data is done using the spacy library by loading the "en_core_web_sm" English language model that contains components for tokenization, part-of-speech tagging,named entity recognition, etc. Lemmas are extracted and the original dataset is updated.

The file from the GloVe website is loaded into the Python notebook file and is used to train the dataset with the pre-trained GloVe model and the embedded matrix generated at the final step is used directly in machine learning models providing meaningful word representations that improve the performance of NLP tasks (Figure 8).

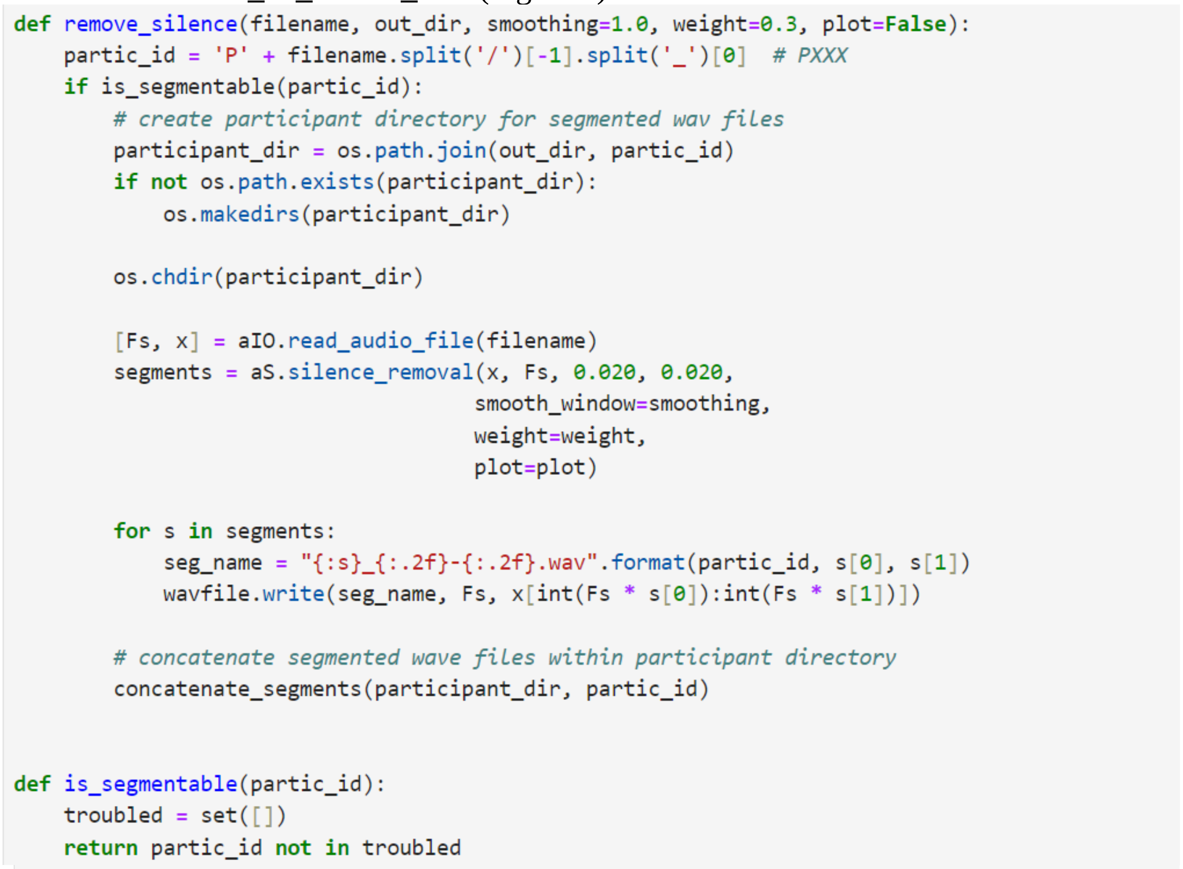


```
[40]:
# Load word embeddings
glove = codecs.open('glove.840B.300d.txt', encoding='utf-8')

print('loading word embeddings...')
embeddings_index = {}
for line in tqdm(glove):
    values = line.rstrip().rsplit(' ')
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
glove.close()

print('found %s word vectors' % len(embeddings_index))
print('Number of words in word index:', len(word_index))

MAX_NB_WORDS = 5200

# Prepare embedding matrix
print('preparing embedding matrix...')

words_not_found = []
nb_words = min(MAX_NB_WORDS, len(word_index) + 1)
embedding_matrix = np.zeros((nb_words, 300))

for word, i in word_index.items():
    if i >= nb_words:
        continue
    embedding_vector = embeddings_index.get(word)
    if (embedding_vector is not None) and len(embedding_vector) > 0:
        # Words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector
    else:
        words_not_found.append(word)

print('Number of null word embeddings: %d' % np.sum(np.sum(embedding_matrix, axis=1) == 0))
print('Words not found:', len(words_not_found))

loading word embeddings...
2196018it [06:01, 6073.22it/s]
found 2196016 word vectors
Number of words in word index: 5089
preparing embedding matrix...
Number of null word embeddings: 56
Words not found: 55
```

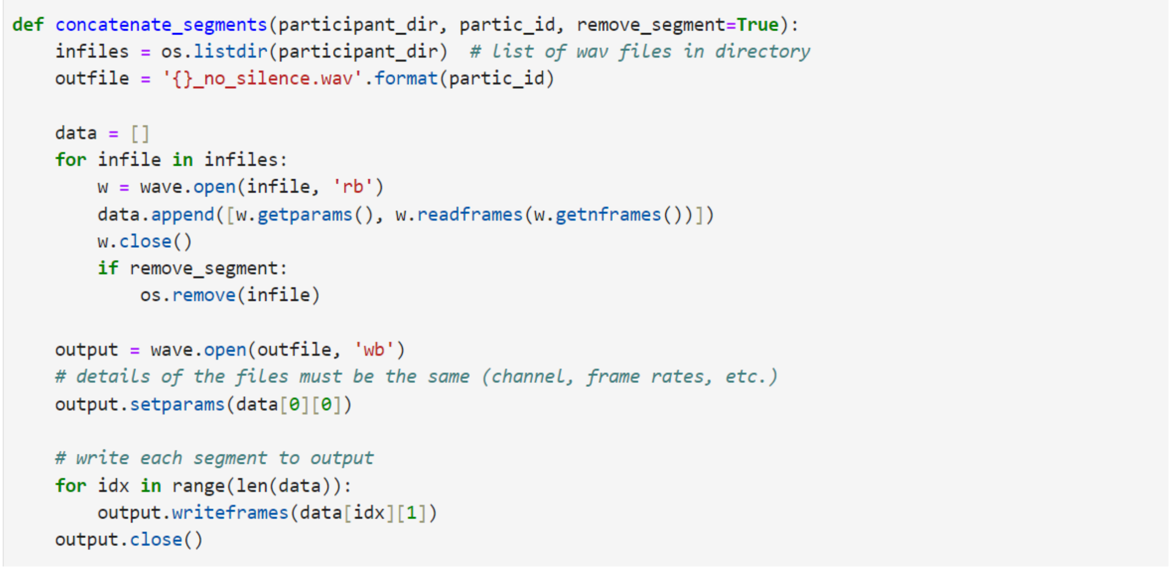
Figure 8: GloVe toolkit to generate embedding matrix.

4.2 Audio processing

The key pre-processing step performed is the removal of Silence from the raw audio file for effective features only extracted in the further steps of modelling. Here remove_silence, is_segmentable and concatenate_segments functions were defined to process individual files, create segments from a file, remove the silence part and concatenate the segments to a single file with the name xx_no_silence_wav (Figure 9).



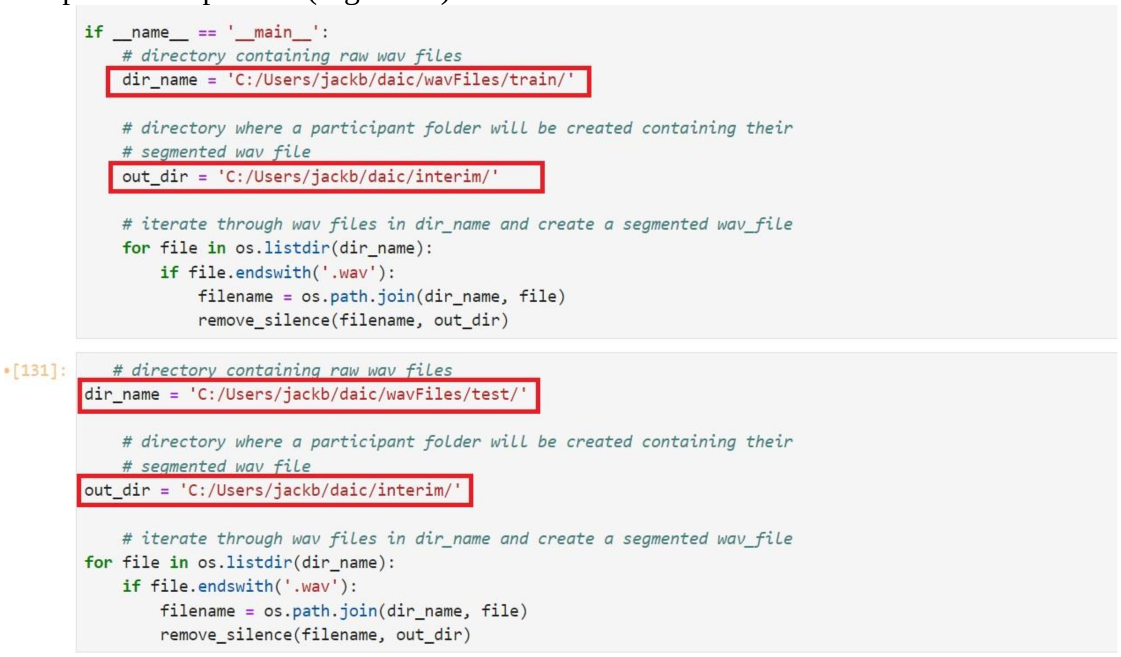
```python
def remove_silence(filename, out_dir, smoothing=1.0, weight=0.3, plot=False):
    partic_id = 'P' + filename.split('/')[-1].split('_')[0]  # PXXX
    if is_segmentable(partic_id):
        # create participant directory for segmented wav files
        participant_dir = os.path.join(out_dir, partic_id)
        if not os.path.exists(participant_dir):
            os.makedirs(participant_dir)

        os.chdir(participant_dir)

        [Fs, x] = aIO.read_audio_file(filename)
        segments = aS.silence_removal(x, Fs, 0.020, 0.020,
                                      smooth_window=smoothing,
                                      weight=weight,
                                      plot=plot)

        for s in segments:
            seg_name = "{:s}_{:.2f}-{:.2f}.wav".format(partic_id, s[0], s[1])
            wavfile.write(seg_name, Fs, x[int(Fs * s[0]):int(Fs * s[1])])

        # concatenate segmented wave files within participant directory
        concatenate_segments(participant_dir, partic_id)


def is_segmentable(partic_id):
    troubled = set([])
    return partic_id not in troubled
```



```python
def concatenate_segments(participant_dir, partic_id, remove_segment=True):
    infiles = os.listdir(participant_dir)  # list of wav files in directory
    outfile = '{}_no_silence.wav'.format(partic_id)

    data = []
    for infile in infiles:
        w = wave.open(infile, 'rb')
        data.append([w.getparams(), w.readframes(w.getnframes())])
        w.close()
        if remove_segment:
            os.remove(infile)

    output = wave.open(outfile, 'wb')
    # details of the files must be the same (channel, frame rates, etc.)
    output.setparams(data[0][0])

    # write each segment to output
    for idx in range(len(data)):
        output.writeframes(data[idx][1])
    output.close()
```

Figure 9: Functions defined for Pre-processing Audio files.

Note: While running the python notebook file, please replace the dir_name and out_dir to your corresponding root folder locations as this part is hardcoded and it needs to be updated to replicate the process (Figure 10).



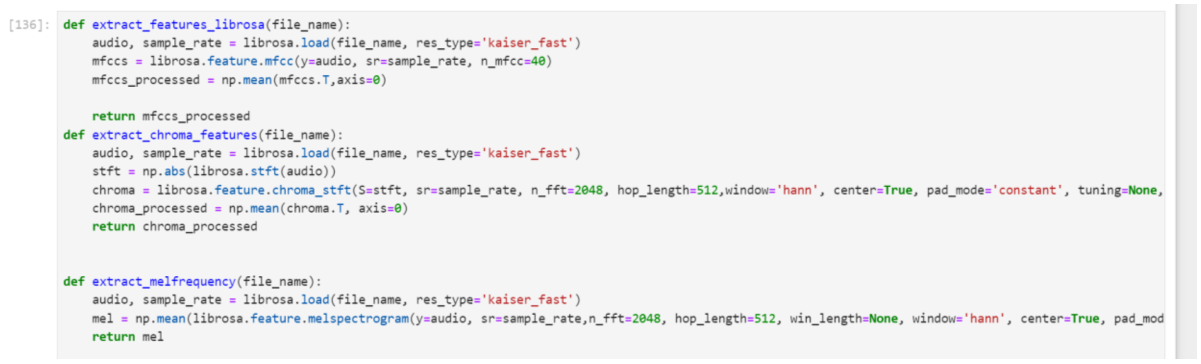
```
if __name__ == '__main__':
    # directory containing raw wav files
    dir_name = 'C:/Users/jackb/daic/wavFiles/train/'

    # directory where a participant folder will be created containing their
    # segmented wav file
    out_dir = 'C:/Users/jackb/daic/interim/'

    # iterate through wav files in dir_name and create a segmented wav_file
    for file in os.listdir(dir_name):
        if file.endswith('.wav'):
            filename = os.path.join(dir_name, file)
            remove_silence(filename, out_dir)
```

```
[131]:    # directory containing raw wav files
dir_name = 'C:/Users/jackb/daic/wavFiles/test/'

    # directory where a participant folder will be created containing their
    # segmented wav file
out_dir = 'C:/Users/jackb/daic/interim/'

    # iterate through wav files in dir_name and create a segmented wav_file
for file in os.listdir(dir_name):
    if file.endswith('.wav'):
        filename = os.path.join(dir_name, file)
        remove_silence(filename, out_dir)
```

Figure 10: Directory Rename for replicating the results.

5 Feature Extraction

This section covers the individual features extracted from the transcript and audio files. The transcript file is pre-processed and is ready to feed into deep learning models for training and validation. Here only the feature extraction from the audio file is discussed.



```
[136]: def extract_features_librosa(file_name):
           audio, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
           mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
           mfccs_processed = np.mean(mfccs.T,axis=0)

           return mfccs_processed
       def extract_chroma_features(file_name):
           audio, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
           stft = np.abs(librosa.stft(audio))
           chroma = librosa.feature.chroma_stft(S=stft, sr=sample_rate, n_fft=2048, hop_length=512,window='hann', center=True, pad_mode='constant', tuning=None,
           chroma_processed = np.mean(chroma.T, axis=0)
           return chroma_processed


       def extract_melfrequency(file_name):
           audio, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
           mel = np.mean(librosa.feature.melspectrogram(y=audio, sr=sample_rate,n_fft=2048, hop_length=512, win_length=None, window='hann', center=True, pad_mod
           return mel
```

Figure 11: Feature extraction for Audio files.

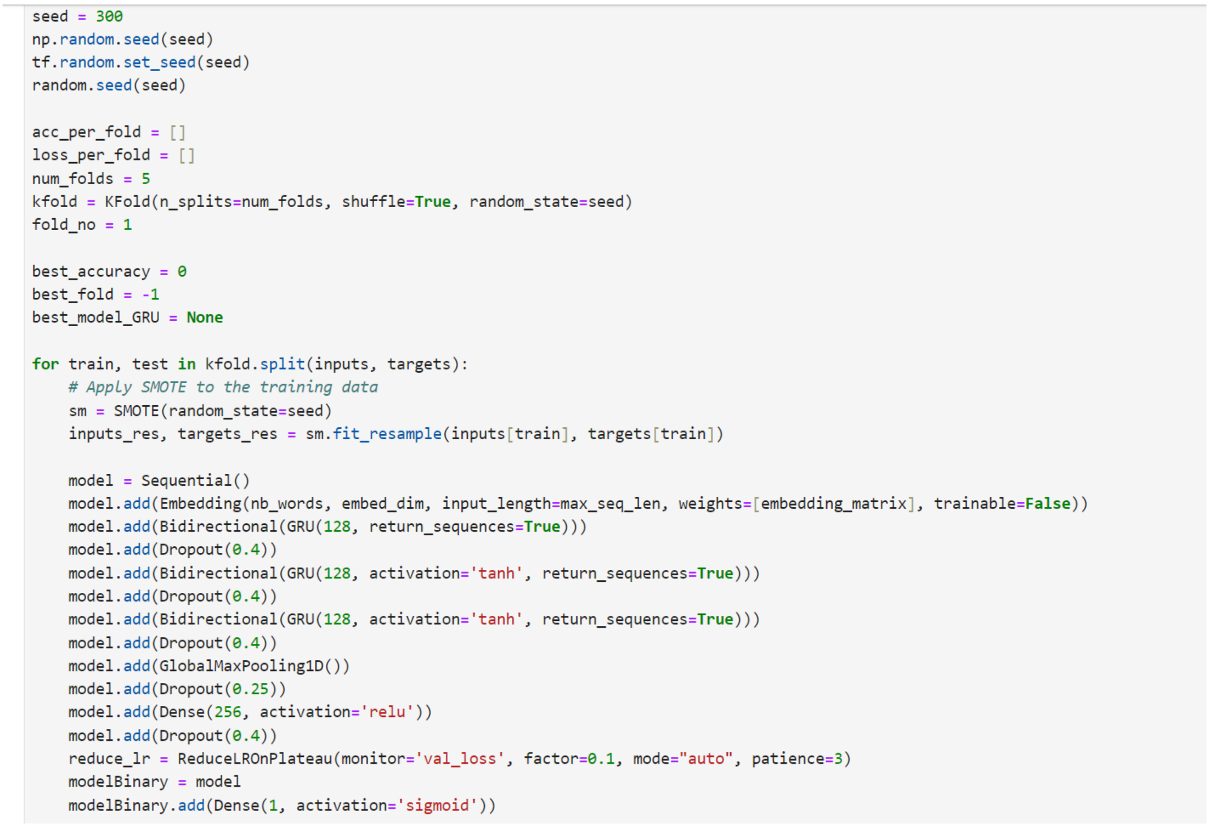
From the pre-processed audio file extract_features_librosa, extract_chroma_features, extract_melfrequency are defined to read the file by loading the wav file using librosa module and extracting the required features MFCCs, Chroma features and Mel-frequency from these files (Figure 11).

These features are extracted and stored into csv files named librosa_mfcc_feature_train and librosa_mfcc_feature_test

6 Deep learning models

This section covers the deep learning models identified for the training and evaluation of the text and audio modality.

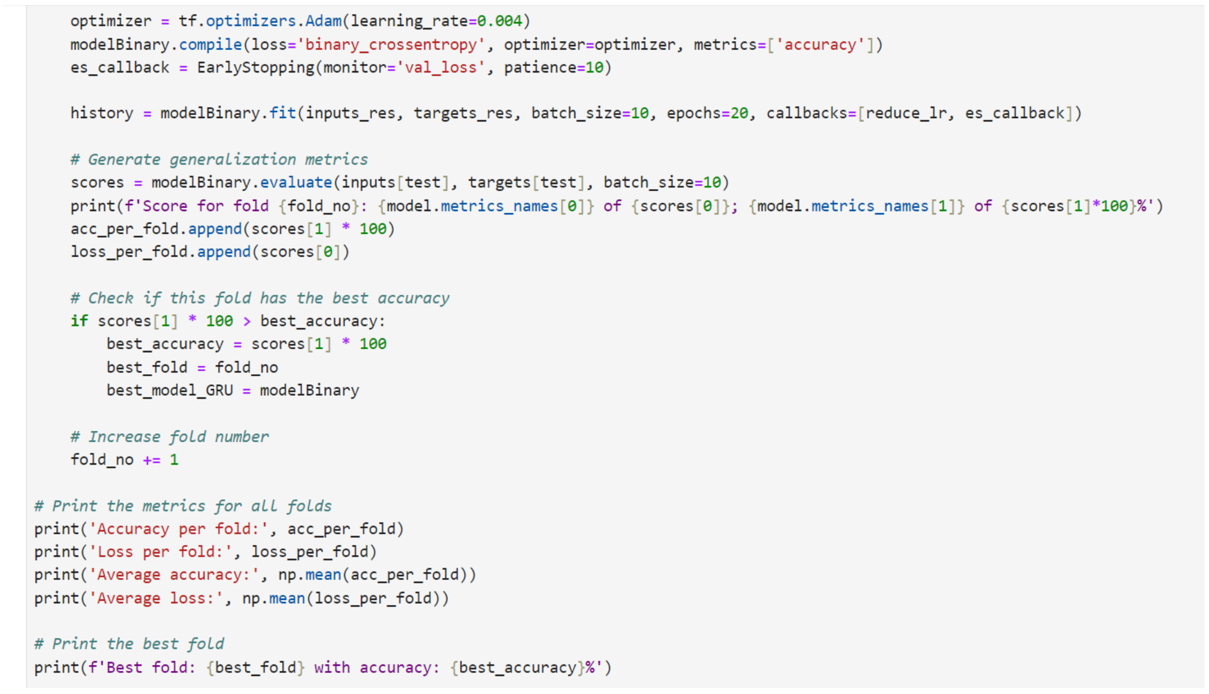
6.1 Bi-GRU Model for transcript



```
seed = 300
np.random.seed(seed)
tf.random.set_seed(seed)
random.seed(seed)

acc_per_fold = []
loss_per_fold = []
num_folds = 5
kfold = KFold(n_splits=num_folds, shuffle=True, random_state=seed)
fold_no = 1

best_accuracy = 0
best_fold = -1
best_model_GRU = None

for train, test in kfold.split(inputs, targets):
    # Apply SMOTE to the training data
    sm = SMOTE(random_state=seed)
    inputs_res, targets_res = sm.fit_resample(inputs[train], targets[train])

    model = Sequential()
    model.add(Embedding(nb_words, embed_dim, input_length=max_seq_len, weights=[embedding_matrix], trainable=False))
    model.add(Bidirectional(GRU(128, return_sequences=True)))
    model.add(Dropout(0.4))
    model.add(Bidirectional(GRU(128, activation='tanh', return_sequences=True)))
    model.add(Dropout(0.4))
    model.add(Bidirectional(GRU(128, activation='tanh', return_sequences=True)))
    model.add(Dropout(0.4))
    model.add(GlobalMaxPooling1D())
    model.add(Dropout(0.25))
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.4))
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, mode="auto", patience=3)
    modelBinary = model
    modelBinary.add(Dense(1, activation='sigmoid'))
```

Figure 12: Implementation of Bi-GRU.

The above image contains the implementation of Bidirectional Gated Recurrent Unit where the synthetic samples are generated for the training dataset using SMOTE and is fed into the model for training. 5-fold cross validation is used during the training and the model having highest accuracy during the training is used as the best model for further analysis.



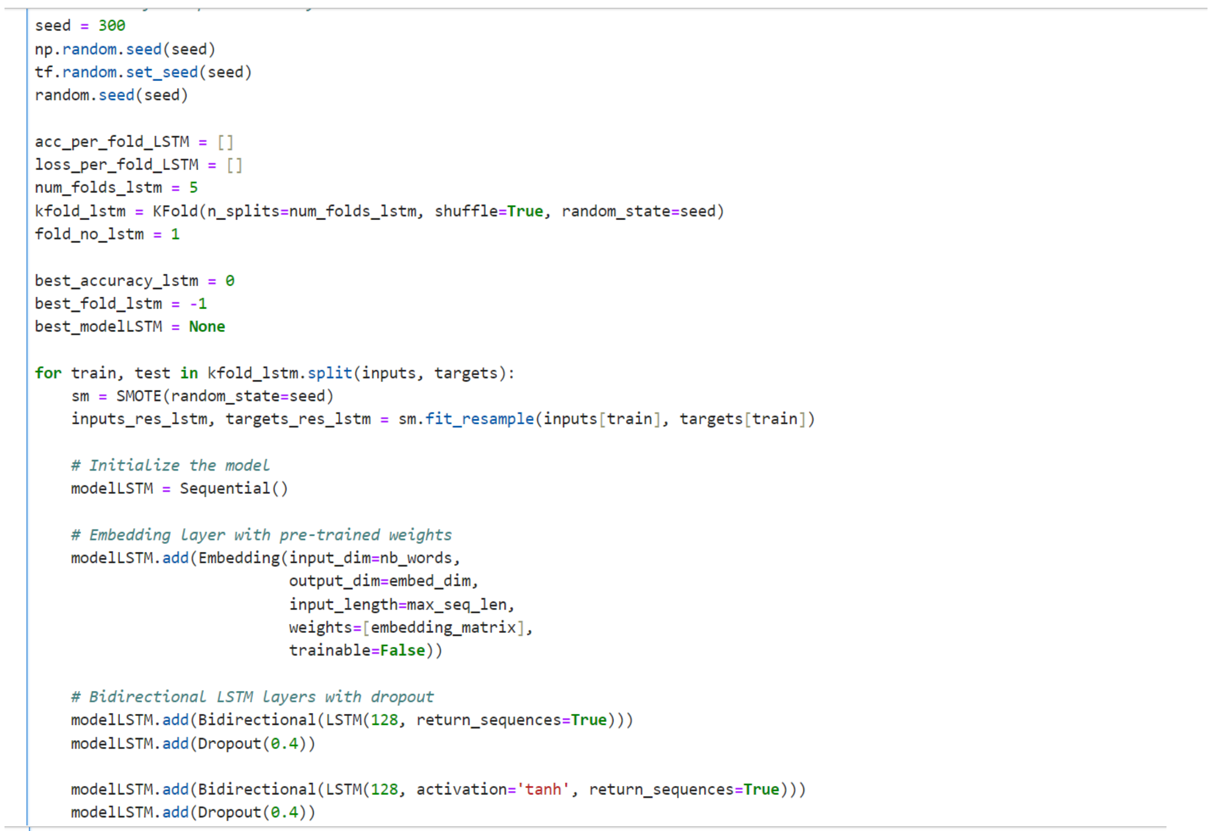
```
    optimizer = tf.optimizers.Adam(learning_rate=0.004)
    modelBinary.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    es_callback = EarlyStopping(monitor='val_loss', patience=10)

    history = modelBinary.fit(inputs_res, targets_res, batch_size=10, epochs=20, callbacks=[reduce_lr, es_callback])

    # Generate generalization metrics
    scores = modelBinary.evaluate(inputs[test], targets[test], batch_size=10)
    print(f'Score for fold {fold_no}: {model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%')
    acc_per_fold.append(scores[1] * 100)
    loss_per_fold.append(scores[0])

    # Check if this fold has the best accuracy
    if scores[1] * 100 > best_accuracy:
        best_accuracy = scores[1] * 100
        best_fold = fold_no
        best_model_GRU = modelBinary

    # Increase fold number
    fold_no += 1

# Print the metrics for all folds
print('Accuracy per fold:', acc_per_fold)
print('Loss per fold:', loss_per_fold)
print('Average accuracy:', np.mean(acc_per_fold))
print('Average loss:', np.mean(loss_per_fold))

# Print the best fold
print(f'Best fold: {best_fold} with accuracy: {best_accuracy}%')
```

Figure 13: Compilation of Bi-GRU and Metrics evaluation.

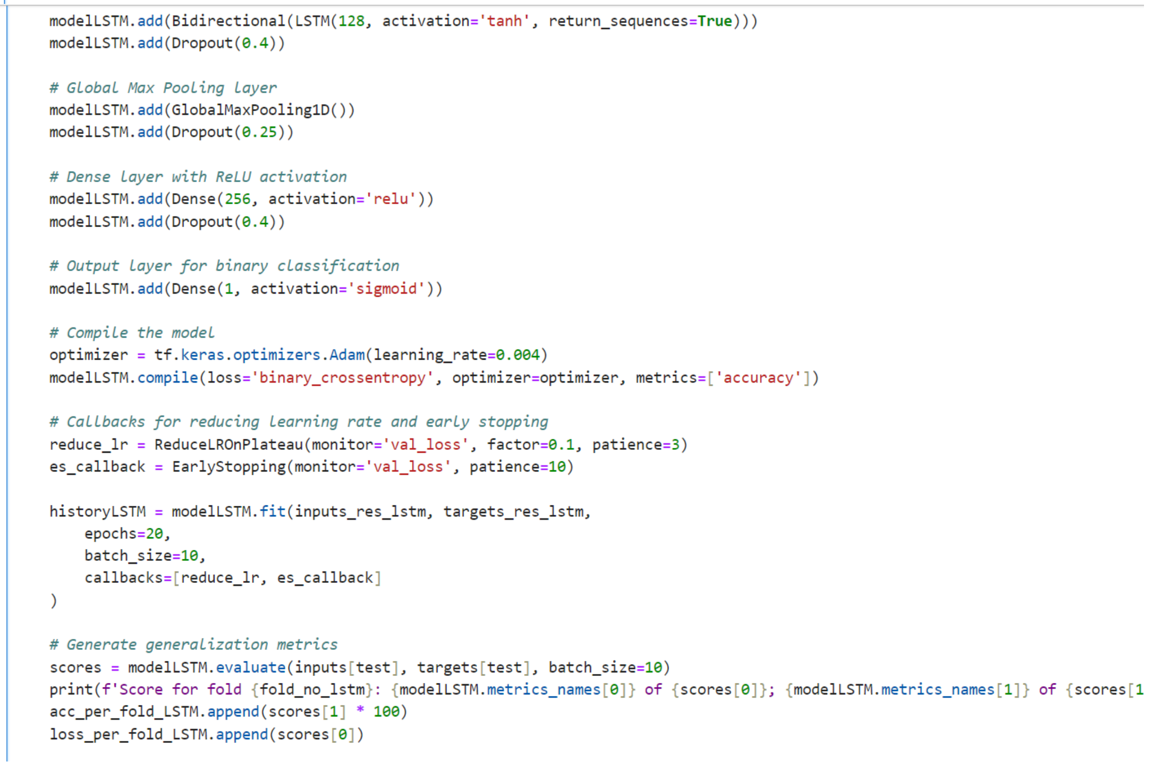
Adam optimizer is used during the compilation of the model and the model is trained with 20 epochs for each fold.

6.2 Bi-LSTM Model for transcript

A similar implementation of the bidirectional Long Short-term memory (Bi-LSTM) is present in the image for training the transcript file. 5-fold cross validation and SMOTE is similarly used to find the model with highest accuracy and to nullify the imbalance in the dataset respectively.



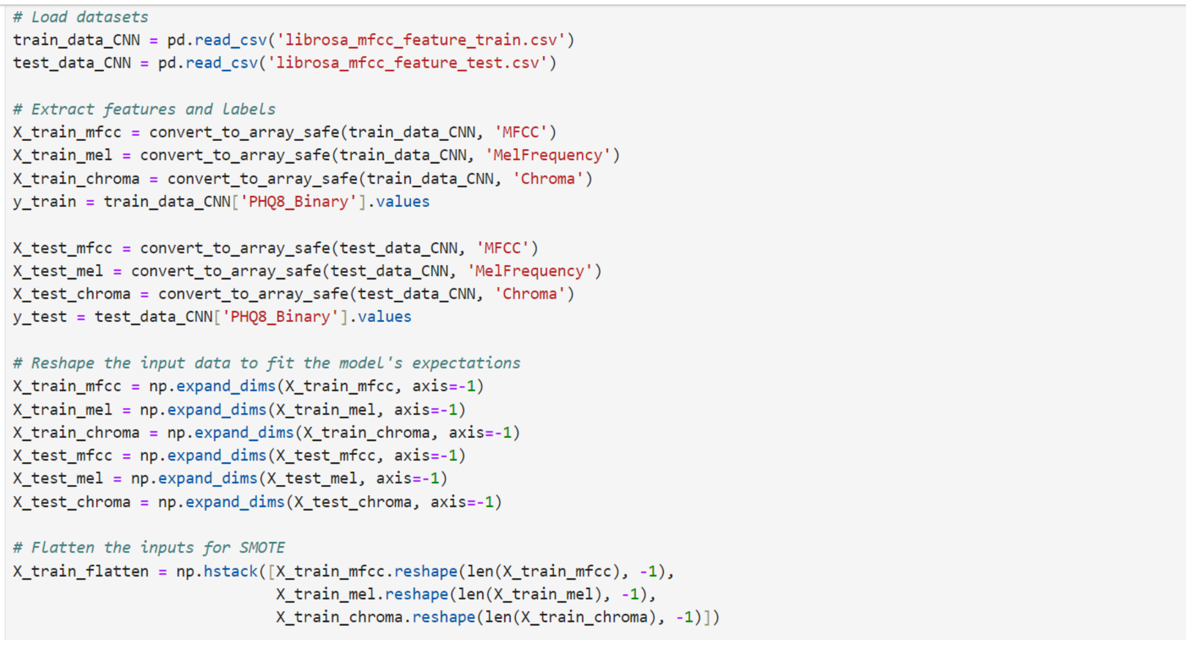
```
seed = 300
np.random.seed(seed)
tf.random.set_seed(seed)
random.seed(seed)

acc_per_fold_LSTM = []
loss_per_fold_LSTM = []
num_folds_lstm = 5
kfold_lstm = KFold(n_splits=num_folds_lstm, shuffle=True, random_state=seed)
fold_no_lstm = 1

best_accuracy_lstm = 0
best_fold_lstm = -1
best_modelLSTM = None

for train, test in kfold_lstm.split(inputs, targets):
    sm = SMOTE(random_state=seed)
    inputs_res_lstm, targets_res_lstm = sm.fit_resample(inputs[train], targets[train])

    # Initialize the model
    modelLSTM = Sequential()

    # Embedding layer with pre-trained weights
    modelLSTM.add(Embedding(input_dim=nb_words,
                            output_dim=embed_dim,
                            input_length=max_seq_len,
                            weights=[embedding_matrix],
                            trainable=False))

    # Bidirectional LSTM layers with dropout
    modelLSTM.add(Bidirectional(LSTM(128, return_sequences=True)))
    modelLSTM.add(Dropout(0.4))

    modelLSTM.add(Bidirectional(LSTM(128, activation='tanh', return_sequences=True)))
    modelLSTM.add(Dropout(0.4))
```



```
    modelLSTM.add(Bidirectional(LSTM(128, activation='tanh', return_sequences=True)))
    modelLSTM.add(Dropout(0.4))

    # Global Max Pooling layer
    modelLSTM.add(GlobalMaxPooling1D())
    modelLSTM.add(Dropout(0.25))

    # Dense layer with ReLU activation
    modelLSTM.add(Dense(256, activation='relu'))
    modelLSTM.add(Dropout(0.4))

    # Output layer for binary classification
    modelLSTM.add(Dense(1, activation='sigmoid'))

    # Compile the model
    optimizer = tf.keras.optimizers.Adam(learning_rate=0.004)
    modelLSTM.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])

    # Callbacks for reducing learning rate and early stopping
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=3)
    es_callback = EarlyStopping(monitor='val_loss', patience=10)

    historyLSTM = modelLSTM.fit(inputs_res_lstm, targets_res_lstm,
        epochs=20,
        batch_size=10,
        callbacks=[reduce_lr, es_callback]
    )

    # Generate generalization metrics
    scores = modelLSTM.evaluate(inputs[test], targets[test], batch_size=10)
    print(f'Score for fold {fold_no_lstm}: {modelLSTM.metrics_names[0]} of {scores[0]}; {modelLSTM.metrics_names[1]} of {scores[1
    acc_per_fold_LSTM.append(scores[1] * 100)
    loss_per_fold_LSTM.append(scores[0])
```

Figure 14: Implementation of Bi-LSTM.

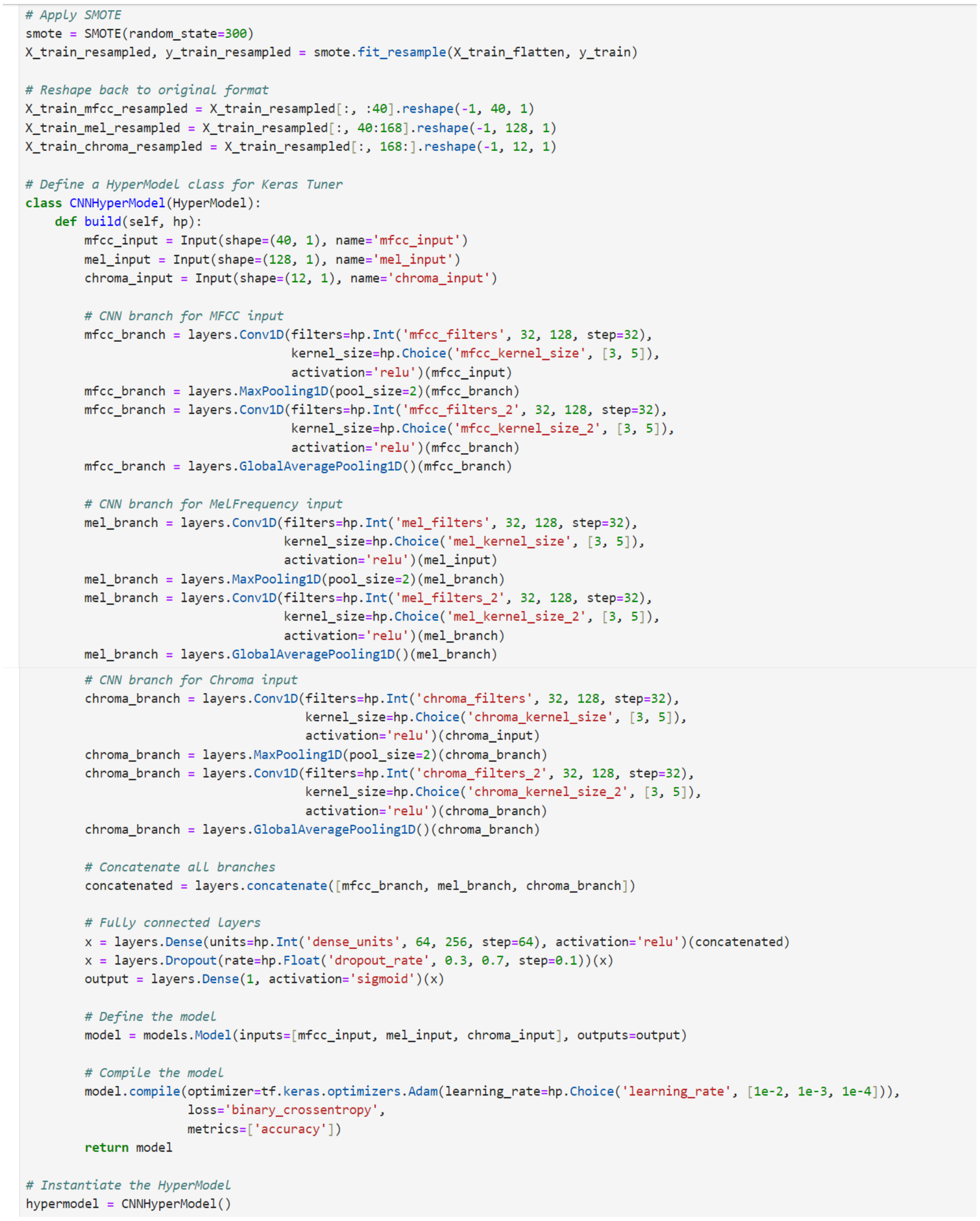
6.3 CNN Model for Audio



```
# Load datasets
train_data_CNN = pd.read_csv('librosa_mfcc_feature_train.csv')
test_data_CNN = pd.read_csv('librosa_mfcc_feature_test.csv')

# Extract features and labels
X_train_mfcc = convert_to_array_safe(train_data_CNN, 'MFCC')
X_train_mel = convert_to_array_safe(train_data_CNN, 'MelFrequency')
X_train_chroma = convert_to_array_safe(train_data_CNN, 'Chroma')
y_train = train_data_CNN['PHQ8_Binary'].values

X_test_mfcc = convert_to_array_safe(test_data_CNN, 'MFCC')
X_test_mel = convert_to_array_safe(test_data_CNN, 'MelFrequency')
X_test_chroma = convert_to_array_safe(test_data_CNN, 'Chroma')
y_test = test_data_CNN['PHQ8_Binary'].values

# Reshape the input data to fit the model's expectations
X_train_mfcc = np.expand_dims(X_train_mfcc, axis=-1)
X_train_mel = np.expand_dims(X_train_mel, axis=-1)
X_train_chroma = np.expand_dims(X_train_chroma, axis=-1)
X_test_mfcc = np.expand_dims(X_test_mfcc, axis=-1)
X_test_mel = np.expand_dims(X_test_mel, axis=-1)
X_test_chroma = np.expand_dims(X_test_chroma, axis=-1)

# Flatten the inputs for SMOTE
X_train_flatten = np.hstack([X_train_mfcc.reshape(len(X_train_mfcc), -1),
                             X_train_mel.reshape(len(X_train_mel), -1),
                             X_train_chroma.reshape(len(X_train_chroma), -1)])
```

Figure 15: Reshaping of features for CNN.

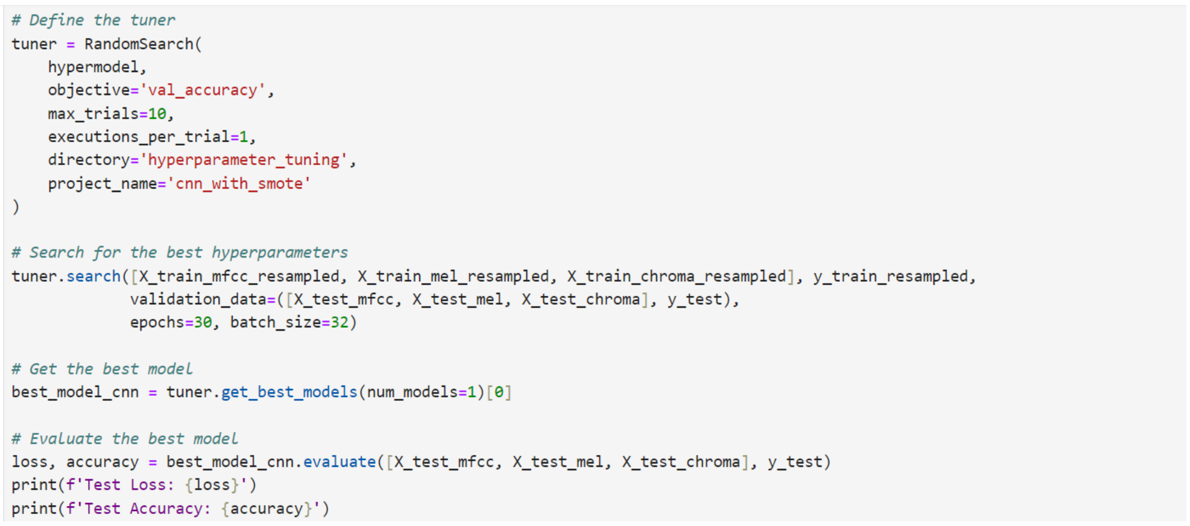
The Convolutional Neural Network model is identified for modelling and training of the audio features extracted from the raw wav files. The individual files librosa_mfcc_feature_train and librosa_mfcc_feature_test are loaded from the csv file and is loaded the string file to a numpy array for modelling. The indivual features are extracted from the indivdual array for each feature type and is reshaped to fit into the CNN model. The data is flattened to apply SMOTE and generate synthetic samples to balance the minority class in the dataset.



```python
# Apply SMOTE
smote = SMOTE(random_state=300)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_flatten, y_train)

# Reshape back to original format
X_train_mfcc_resampled = X_train_resampled[:, :40].reshape(-1, 40, 1)
X_train_mel_resampled = X_train_resampled[:, 40:168].reshape(-1, 128, 1)
X_train_chroma_resampled = X_train_resampled[:, 168:].reshape(-1, 12, 1)

# Define a HyperModel class for Keras Tuner
class CNNHyperModel(HyperModel):
    def build(self, hp):
        mfcc_input = Input(shape=(40, 1), name='mfcc_input')
        mel_input = Input(shape=(128, 1), name='mel_input')
        chroma_input = Input(shape=(12, 1), name='chroma_input')

        # CNN branch for MFCC input
        mfcc_branch = layers.Conv1D(filters=hp.Int('mfcc_filters', 32, 128, step=32),
                                    kernel_size=hp.Choice('mfcc_kernel_size', [3, 5]),
                                    activation='relu')(mfcc_input)
        mfcc_branch = layers.MaxPooling1D(pool_size=2)(mfcc_branch)
        mfcc_branch = layers.Conv1D(filters=hp.Int('mfcc_filters_2', 32, 128, step=32),
                                    kernel_size=hp.Choice('mfcc_kernel_size_2', [3, 5]),
                                    activation='relu')(mfcc_branch)
        mfcc_branch = layers.GlobalAveragePooling1D()(mfcc_branch)

        # CNN branch for MelFrequency input
        mel_branch = layers.Conv1D(filters=hp.Int('mel_filters', 32, 128, step=32),
                                   kernel_size=hp.Choice('mel_kernel_size', [3, 5]),
                                   activation='relu')(mel_input)
        mel_branch = layers.MaxPooling1D(pool_size=2)(mel_branch)
        mel_branch = layers.Conv1D(filters=hp.Int('mel_filters_2', 32, 128, step=32),
                                   kernel_size=hp.Choice('mel_kernel_size_2', [3, 5]),
                                   activation='relu')(mel_branch)
        mel_branch = layers.GlobalAveragePooling1D()(mel_branch)

        # CNN branch for Chroma input
        chroma_branch = layers.Conv1D(filters=hp.Int('chroma_filters', 32, 128, step=32),
                                      kernel_size=hp.Choice('chroma_kernel_size', [3, 5]),
                                      activation='relu')(chroma_input)
        chroma_branch = layers.MaxPooling1D(pool_size=2)(chroma_branch)
        chroma_branch = layers.Conv1D(filters=hp.Int('chroma_filters_2', 32, 128, step=32),
                                      kernel_size=hp.Choice('chroma_kernel_size_2', [3, 5]),
                                      activation='relu')(chroma_branch)
        chroma_branch = layers.GlobalAveragePooling1D()(chroma_branch)

        # Concatenate all branches
        concatenated = layers.concatenate([mfcc_branch, mel_branch, chroma_branch])

        # Fully connected layers
        x = layers.Dense(units=hp.Int('dense_units', 64, 256, step=64), activation='relu')(concatenated)
        x = layers.Dropout(rate=hp.Float('dropout_rate', 0.3, 0.7, step=0.1))(x)
        output = layers.Dense(1, activation='sigmoid')(x)

        # Define the model
        model = models.Model(inputs=[mfcc_input, mel_input, chroma_input], outputs=output)

        # Compile the model
        model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=hp.Choice('learning_rate', [1e-2, 1e-3, 1e-4])),
                      loss='binary_crossentropy',
                      metrics=['accuracy'])
        return model

# Instantiate the HyperModel
hypermodel = CNNHyperModel()
```

Figure 16: Implementation of CNN.

A custom class CNNHyperModel is defined for performing the hyperparameter tuning of the proposed CNN model using keras tuner. The shape of the input layer is defined and each of the features are modelled as a separate branch in the CNN and three branches for MFCC, Chroma and Mel-frequency is designed. The layers are then concatenated, and the model is compiled using Adam optimizer.

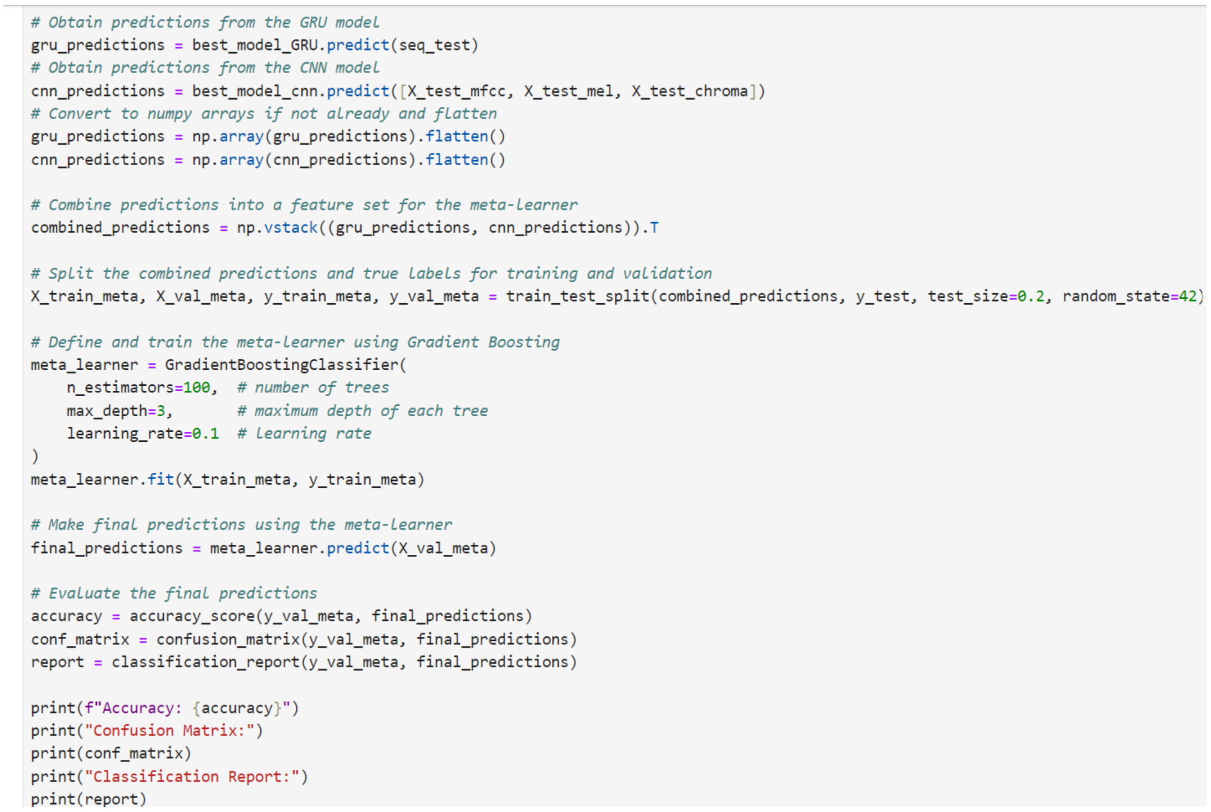


```python
# Define the tuner
tuner = RandomSearch(
    hypermodel,
    objective='val_accuracy',
    max_trials=10,
    executions_per_trial=1,
    directory='hyperparameter_tuning',
    project_name='cnn_with_smote'
)

# Search for the best hyperparameters
tuner.search([X_train_mfcc_resampled, X_train_mel_resampled, X_train_chroma_resampled], y_train_resampled,
             validation_data=([X_test_mfcc, X_test_mel, X_test_chroma], y_test),
             epochs=30, batch_size=32)

# Get the best model
best_model_cnn = tuner.get_best_models(num_models=1)[0]

# Evaluate the best model
loss, accuracy = best_model_cnn.evaluate([X_test_mfcc, X_test_mel, X_test_chroma], y_test)
print(f'Test Loss: {loss}')
print(f'Test Accuracy: {accuracy}')
```

Figure 17: Hyperparameter tuning for CNN.

Hyperparameter tuning using randomsearch is used to find the optimal hyparameters for this multi-input CNN model. A maximum of 10 trials is used and the criteria for the tuning is to have the maximum accuracy during training of the model. The best model is identified and is used further.

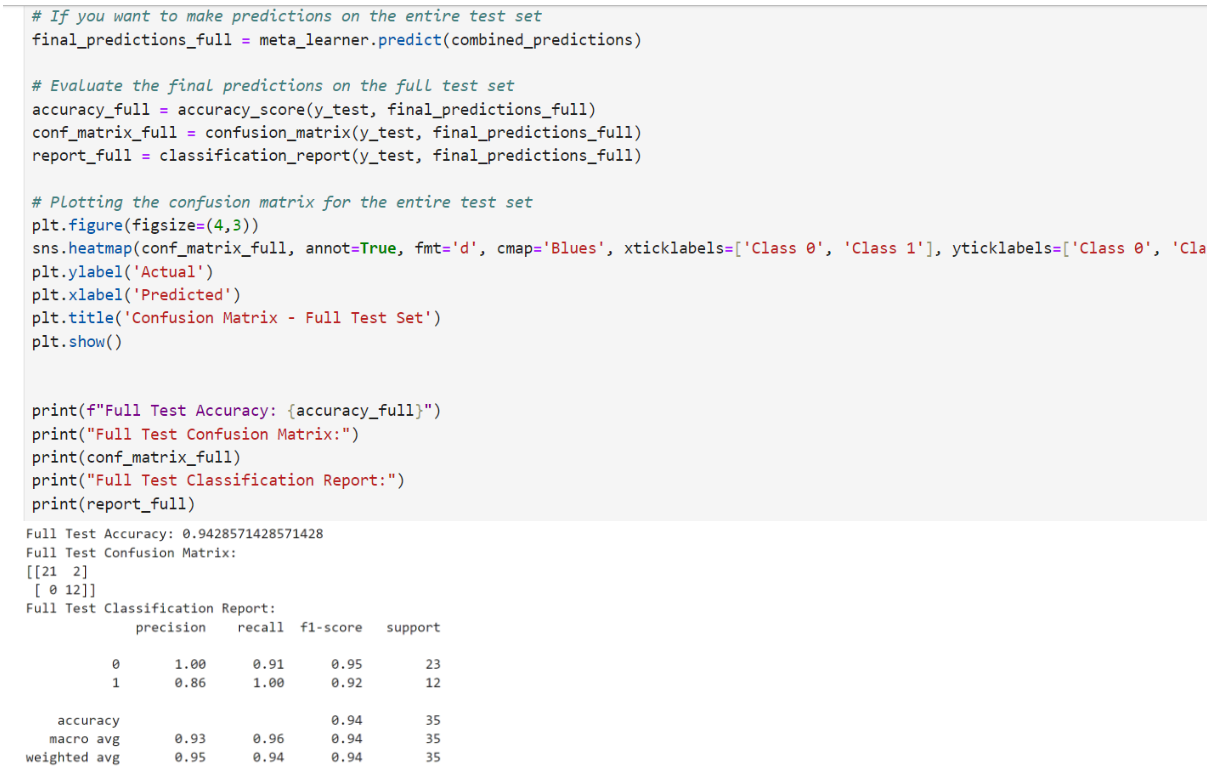
6.4 Meta-learner for a stacking ensemble



```python
# Obtain predictions from the GRU model
gru_predictions = best_model_GRU.predict(seq_test)
# Obtain predictions from the CNN model
cnn_predictions = best_model_cnn.predict([X_test_mfcc, X_test_mel, X_test_chroma])
# Convert to numpy arrays if not already and flatten
gru_predictions = np.array(gru_predictions).flatten()
cnn_predictions = np.array(cnn_predictions).flatten()

# Combine predictions into a feature set for the meta-learner
combined_predictions = np.vstack((gru_predictions, cnn_predictions)).T

# Split the combined predictions and true labels for training and validation
X_train_meta, X_val_meta, y_train_meta, y_val_meta = train_test_split(combined_predictions, y_test, test_size=0.2, random_state=42)

# Define and train the meta-learner using Gradient Boosting
meta_learner = GradientBoostingClassifier(
    n_estimators=100,  # number of trees
    max_depth=3,       # maximum depth of each tree
    learning_rate=0.1  # learning rate
)
meta_learner.fit(X_train_meta, y_train_meta)

# Make final predictions using the meta-learner
final_predictions = meta_learner.predict(X_val_meta)

# Evaluate the final predictions
accuracy = accuracy_score(y_val_meta, final_predictions)
conf_matrix = confusion_matrix(y_val_meta, final_predictions)
report = classification_report(y_val_meta, final_predictions)

print(f"Accuracy: {accuracy}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(report)
```

Figure 18: Stacking Ensemble using gradient Boosting.

The individual predictions from the model is obtained and the output is stacked as a feature for the meta-learner. A Gradient Boosting classifier is used for the meta-learner and the final prediction is performed for the dataset.



```
# If you want to make predictions on the entire test set
final_predictions_full = meta_learner.predict(combined_predictions)

# Evaluate the final predictions on the full test set
accuracy_full = accuracy_score(y_test, final_predictions_full)
conf_matrix_full = confusion_matrix(y_test, final_predictions_full)
report_full = classification_report(y_test, final_predictions_full)

# Plotting the confusion matrix for the entire test set
plt.figure(figsize=(4,3))
sns.heatmap(conf_matrix_full, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Cla
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix - Full Test Set')
plt.show()


print(f"Full Test Accuracy: {accuracy_full}")
print("Full Test Confusion Matrix:")
print(conf_matrix_full)
print("Full Test Classification Report:")
print(report_full)
```

```
Full Test Accuracy: 0.9428571428571428
Full Test Confusion Matrix:
[[21  2]
 [ 0 12]]
Full Test Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.91      0.95        23
           1       0.86      1.00      0.92        12

    accuracy                           0.94        35
   macro avg       0.93      0.96      0.94        35
weighted avg       0.95      0.94      0.94        35
```

Figure 19: Evaluation metrics for the ensemble classifier

Figure 19 contains the final prediction of the overall model with results including Accuracy, confusion matrix and classification report.

This approach improves the performance of the classification by leveraging the strengths of multiple model and making the final binary classification more robust.

References

University of Southern California. *Distress Analysis Interview Corpus (DAIC)*. Institute for Creative Technologies. Available at: https://dcapswoz.ict.usc.edu/ (Accessed: 11 August 2024)

Stanford University. *GloVe: Global Vectors for Word Representation*. Stanford NLP Group. Available at: https://nlp.stanford.edu/projects/glove/ (Accessed: 11 August 2024)