National College of Ireland

Configuration Manual

MSc Research Project
MSc Data Analytics

Kalyani Phursule
Student ID: 23229268

School of Computing
National College of Ireland

Supervisor: Prof. Furqan Rustam

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Kalyani Phursule
Student ID:	23229268
Programme:	MSc Data Analytics
Year:	2024-2025
Module:	MSc Research Project
Supervisor:	Prof. Furqan Rustam
Submission Due Date:	29/01/2025
Project Title:	Configuration Manual
Word Count:	767
Page Count:	9

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:	Kalyani Phursule
Date:	28th January 2025

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

Kalyani Phursule
23229268

1 Introduction 1

This research is about baby cry classification using machine learning models ranging from simple models like, SVM, KNN, RF,etc to advanced models like LSTM, RNN,etc including hybrid and ensemble approach for machine learning. The purpose of this document is to provide details regarding the pre-requisite such as software and hardware tools used in the developing the study, steps for collecting data and executing code.

2 Pre-requisite

The software and hardware requirements used are as follows:

2.1 Software Requirements

2.1.1 Python Environment

Python Version: 3.8 or higher. Pycharm or Anaconda Navigator as development environments.

2.1.2 Python Libraries

Python Lib	Version
numpy	1.24.3
pandas	2.0.3
matplotlib	3.7.5
seaborn	0.13.2
sklearn	1.3.0
tensorflow	2.13.0
keras	2.13.1
xgboost	2.1.1
scipy	1.10.1
torch	2.4.1

Table 1: Python libraries and their versions used in the project.

2.2 Hardware Requirements

Requirement	Specification
OS	Windows 11
CPU	Intel i7/i9 (10 cores or up)
RAM	8GB or higher
Disk Space	30 GB

Table 2: System requirements for running the project.

3 Data Collection

For this study open source data from Kaggel[1] data repository was used. It is set of labelled baby cries audio files with classes such as hungry, discomfort, tired, belly pain, and burping.

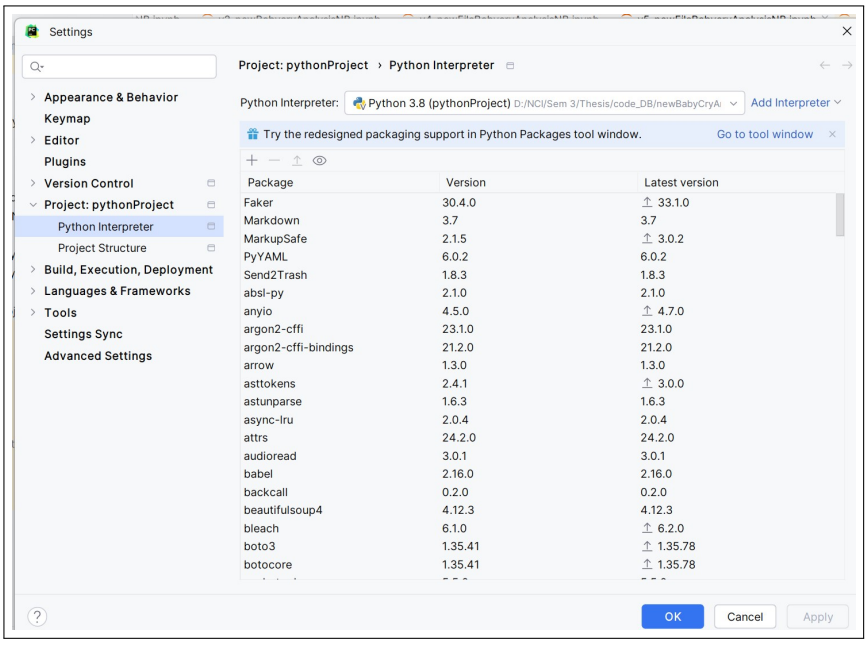
Download the dataset from the url and save it locally.

4 Environment Setup

1. Install PyCharm Community or Professional edition from JetBrains.

2. Open the PyCharm and set interpreter as 3.8

Goto –> File –> Setting –> Python interpreter –> Select version 3.8 or above

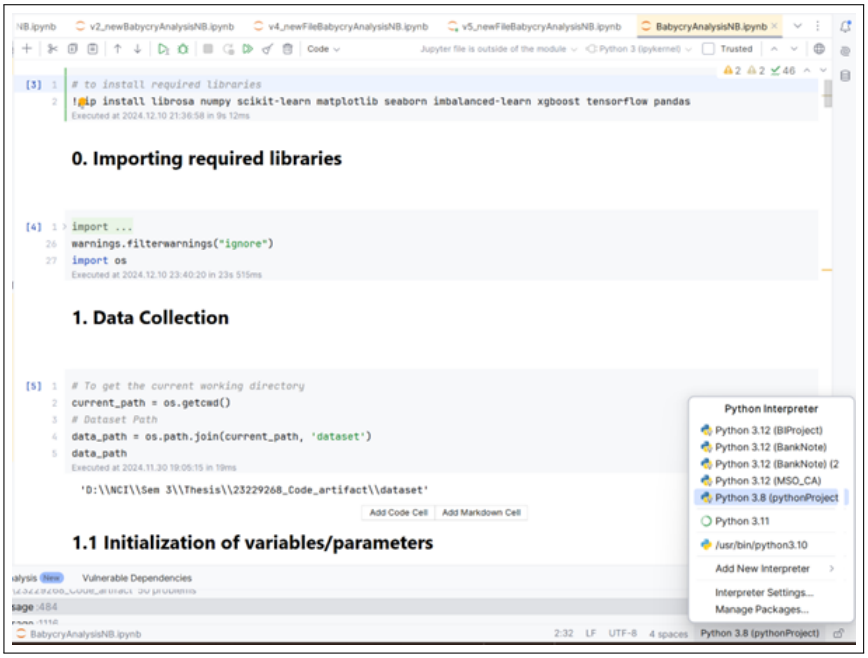


5 Steps to execute code

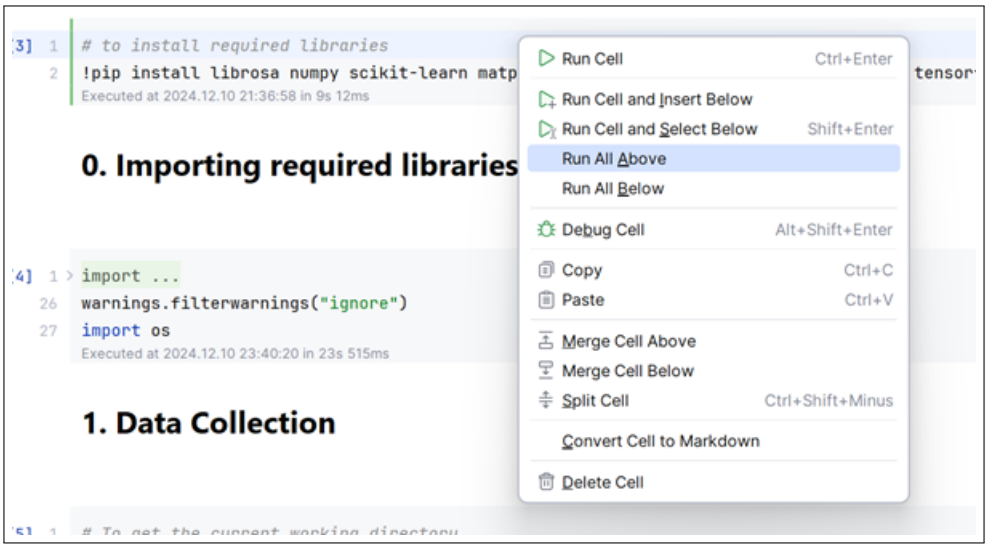
This section gives all required steps to execute the code and reproduce the results.

[1]data url: `https://www.kaggle.com/datasets/warcoder/infant-cry-audio-corpus`

1. Download Code Artifact Folder and Unzip it
2. Open the folder Code Artifact and locate 'dataset' folder
3. Unzip the downloaded data set from Section 2 into the 'dataset' folder
4. Right click on the script file BabycryAnalysisNB.ipynb then open with PyCharm
5. Make sure Python Interpreter is 3.8 or above



6. Right on the first cell and select Run All option.



6 Flow of the Code

6.1 Importing required libraries

All the necessary Python libraries are imported as first step.

```
import os
import librosa
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import SMOTE
from collections import Counter
import xgboost as xgb
from sklearn.neighbors import KNeighborsClassifier
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout,Flatten,SimpleRNN,Conv1D, MaxPooling1D
import pandas as pd
from tensorflow.keras.models import Model
from xgboost import XGBClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer,accuracy_score, f1_score, precision_score, recall_score, classification_report, confusion_matrix
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.preprocessing import label_binarize
from itertools import cycle
import warnings
warnings.filterwarnings("ignore")
import os
```
Executed at 2024.12.11 18:32:48 in 109ms

6.2 Importing required libraries

All the necessary Python libraries are imported as first step.

6.3 Data Path and Parameter initialization

Data path and located from local folder and needed parameters are initialized.

1. Data Collection

```
# To get the current working directory
current_path = os.getcwd()
# Dataset Path
data_path = os.path.join(current_path, 'dataset')
data_path
```
Executed at 2024.12.11 18:32:54 in 177ms

```
'D:\\NCI\\Sem 3\\Thesis\\code_DB\\newBabyCryAnalysis\\pythonProject\\dataset'
```

1.1 Initialization of variables/parameters

```
#classes for baby cires
class_labels = ['belly_pain', 'burping', 'discomfort', 'hungry', 'tired']

# Initialize lists to hold features and labels
X = []
y = []

# Parameters for feature extraction
n_mfcc = 13  # Number of MFCC features to extract
n_chroma = 12  # Number of chroma features
n_spectral_contrast = 7  # Number of spectral contrast features
```

6.4 Data Loading and MFCC feature extraction

Data is loaded and feature extraction is done in this step.

2. Data Loading and MFCC feature extraction

```
1
```

```
# Extract MFCC features from a given audio file.
def extract_mfcc(file_path):
    audio, sample_rate = librosa.load(file_path, res_type='kaiser_fast')
    mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=n_mfcc)
    # Padding to make the MFCCs to a fixed length
    pad_width = max_pad_len - mfccs.shape[1]
    if pad_width > 0:
        mfccs = np.pad(mfccs, pad_width=((0, 0), (0, pad_width)), mode='constant')
    else:
        mfccs = mfccs[:, :max_pad_len]
    return mfccs
```

Executed at 2024.12.11 18:32:55 in 31ms

```
# Loop through all class folder and extract MFCCs
for i, label in enumerate(class_labels):
    folder_path = os.path.join(data_path, label)
    for file_name in os.listdir(folder_path):
        if file_name.endswith('.wav'):
            file_path = os.path.join(folder_path, file_name)
            mfcc = extract_mfcc(file_path)
            X.append(mfcc)
```

6.5 Exploratory Data Analysis

Exploratory Data Analysis was done on audio data.

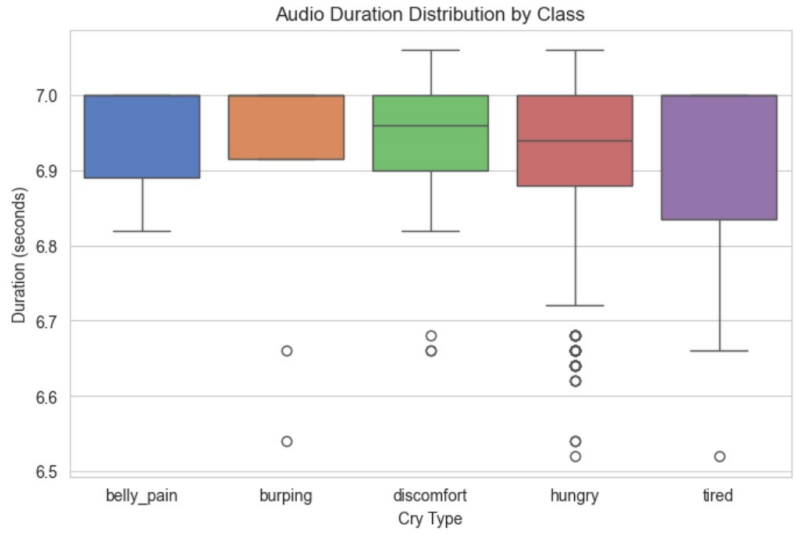
3.1 EDA: Checking Class Distribution

```
# 1. Class Distribution
class_counts = pd.Series(y1).value_counts()
plt.figure(figsize=(8, 5))
barplot = sns.barplot(x=[class_labels[i] for i in class_counts.index], y=class_counts.values, palette='muted')
plt.title("Class Distribution")
plt.xlabel("Cry Type")
plt.ylabel("Number of Samples")

# Adding labels to each bar
for bar in barplot.patches:
    plt.text(
        bar.get_x() + bar.get_width() / 2,  # X-coordinate
        bar.get_height() + 1,  # Y-coordinate
        f'{int(bar.get_height())}',  # Value to annotate
        ha='center',  # Horizontal alignment
        va='bottom',  # Vertical alignment
        fontsize=10  # Font size
    )

plt.show()
```

Executed at 2024.12.11 18:33:44 in 738ms

```
sns.boxplot(x=y1, y=durations, palette="muted")
plt.title("Audio Duration Distribution by Class")
plt.xlabel("Cry Type")
plt.ylabel("Duration (seconds)")
plt.xticks(ticks=range(len(class_labels)), labels=class_labels)
plt.show()
```

Executed at 2024.12.11 18:34:07 in 16s 686ms



6.6 Data Reshaping

Data reshaping was done for training the models.

4. Data Reshaping

```
[17] # Reshape X to be suitable for the machine learning model training
     X_mfcc = X_mfcc.reshape(X_mfcc.shape[0], -1)  # Flatten the MFCC array
     print(f'Extracted {X_mfcc.shape[0]} samples with shape {X_mfcc.shape}')
```

Executed at 2024.12.11 18:34:07 in 14ms

```
Extracted 457 samples with shape (457, 2262)
```

6.7 Model Training

Following machine learning model were trained with original data and later with augmented data using SMOTE.
Traditional ML: SVM, KNN, Random Forest, AdaBoost, XGBoost
Deep Learning : CNN, LSTM, RNN
Hybrid Model : CNN-XGBoost, CNN-LSTM
Ensemble Approach : Ensemble Model(CNN,RandomForest,XGBoost,SVM,KNN)

5.1.1 Traditional Model : SVM model Simple

```
# Initialize the SVM model
svm_model = SVC(kernel='linear', random_state=42)
```

Executed at 2024.12.11 18:34:07 in 30ms

```
# Define a list of scoring metrics
scoring_metrics = {
    'accuracy': 'accuracy',
    'f1_weighted': make_scorer(f1_score, average='weighted'),
    'precision_weighted': make_scorer(precision_score, average='weighted'),
    'recall_weighted': make_scorer(recall_score, average='weighted')
}


```

Executed at 2024.12.11 18:34:07 in 15ms

6.8 Imbalance Data Handling

To handle data imbalance SMOTE was applied.

5.2 Handling Data imbalance with SMOTE

[96]
```
# the number of samples per class before applying SMOTE
print(f"Original class distribution: {Counter(y_mfcc_train)}")

```

Executed at 2024.12.11 18:48:31 in 13ms

```
Original class distribution: Counter({3: 310, 2: 20, 4: 17, 0: 12, 1: 6})
```

[97]
```
# Applying SMOTE for oversampling
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_mfcc_train, y_mfcc_train)
```

Executed at 2024.12.11 18:48:31 in 294ms

[98]
```
# the number of samples per class after oversampling
print(f"Class distribution after SMOTE: {Counter(y_train_resampled)}")
```

Executed at 2024.12.11 18:48:31 in 16ms

```
Class distribution after SMOTE: Counter({1: 310, 3: 310, 2: 310, 4: 310, 0: 310})
```

6.9 Model Training with SMOTE

All the proposed models were again trained with balanced data.

5.3.11. Ensemble Model : (CNN,RF,SVM,XGBoost,KNN) with SMOTE

```
[176] 1 # Extracting features using already trained CNN model with SMOTE
      2 train_features = cnn_model_s.predict(X_train_resampled_cnn)
      3 test_features = cnn_model_s.predict(X_mfcc_test_cnn)
```

Executed at 2024.12.11 20:06:45 in 640ms

```
49/49 [==============================] - 0s 4ms/step
3/3 [==============================] - 0s 5ms/step
```

```
[177] 1 # Defining base classifiers and meta-classifier for stacking
      2 base_learners = [
      3     ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
      4     ('svm', SVC(kernel='linear', probability=True)),
      5     ('xgb',
      6      XGBClassifier(n_estimators=100, max_depth=5, random_state=42, use_label_encoder=False, eval_metric='mlogloss'))
      7 ]
```

Executed at 2024.12.11 20:06:45 in 23ms

```
[178] 1 # Meta-classifier (KNN)
      2 meta_learner = KNeighborsClassifier(n_neighbors=5)  # RandomForestClassifier(n_estimators=100, random_state=42)
```

Executed at 2024.12.11 20:06:45 in 25ms

```
[179] 1 #  Create a stacking classifier
      2 stacking_clf = StackingClassifier(estimators=base_learners, final_estimator=meta_learner, cv=5)
```

Executed at 2024.12.11 20:06:46 in 115ms

6.10 Model Evaluation

Evaluation of all the proposed models was performed using accuracy, F-1 Score, precision, recall and confusion matrix.

```
[ ] 1 # Evaluating the model
    2 accuracy = accuracy_score(y_mfcc_test, y_pred_stacking)
    3 f1 = f1_score(y_mfcc_test, y_pred_stacking, average='weighted')
    4 precision = precision_score(y_mfcc_test, y_pred_stacking, average='weighted')
    5 recall = recall_score(y_mfcc_test, y_pred_stacking, average='weighted')
    6
    7 print(f'Ensemble Accuracy: {accuracy * 100:.2f}%')
    8 print(f'Ensemble F1 Score: {f1:.2f}')
    9 print(f'Ensemble Precision: {precision:.2f}')
   10 print(f'Ensemble Recall: {recall:.2f}')
   11 new_row_cv = pd.DataFrame([["Ensemble Model(CNN,RandomForest, XGBoost, SVM,KNN) with SMOTE", accuracy, f1, precision, recall]],
   12                           columns=['Model', 'Accuracy', 'F1 Score', 'Precision', 'Recall'])
   13 results_df = pd.concat([results_df, new_row_cv], ignore_index=True)
   14 results_df
   15
```

Executed at 2024.12.11 20:06:56 in 118ms

```
Ensemble Accuracy: 71.74%
Ensemble F1 Score: 0.68
Ensemble Precision: 0.64
Ensemble Recall: 0.72
```

6. Evaluation : Plotting Performance Metrics

```
# Reshape the dataframe for easier plotting
n_results_df_melted = n_results_df.melt(
    id_vars="Model", var_name="Metric", value_name="Score"
)

# Set the plot style
plt.figure(figsize=(14, 8))
sns.set_theme(style="whitegrid")

# Create the barplot
sns.barplot(
    data=n_results_df_melted,
    x="Model",
    y="Score",
    hue="Metric",
    palette="Set2"
)

# Rotate x-axis labels for better visibility
plt.xticks(rotation=45, ha="right", fontsize=10)
plt.title("Model Performance Metrics with SMOTE", fontsize=16)
plt.xlabel("Model", fontsize=12)
plt.ylabel("Score", fontsize=12)
plt.legend(title="Metric", fontsize=10)
```