# Configuration Manual

MSc Research Project
Data Analytics

Saviour Nickolas Derel Joseph Fernandez
Student ID: 21127051

School of Computing
National College of Ireland

Supervisor: Mr Vladimir Milosavljevic

**National College of Ireland**
**Project Submission Sheet**
**School of Computing**

| | |
|---|---|
| **Student Name:** | Saviour Nickolas Derel Joseph Fernandez |
| **Student ID:** | 21127051 |
| **Programme:** | Data Analytics |
| **Year:** | 2023 |
| **Module:** | MSc Research Project |
| **Supervisor:** | Mr Vladimir Milosavljevic |
| **Submission Due Date:** | 01/02/2023 |
| **Project Title:** | Configuration Manual |
| **Word Count:** | 553 |
| **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:** | Saviour Nickolas Derel |
| **Date:** | 30th January 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): | |

# Configuration Manual

Saviour Nickolas Derel Joseph Fernandez
21127051

## 1 Introduction

We will see all the used implemented techniques and the hardware specification used for the project "Comparison of Deep Learning and Machine Learning in Music Genre Categorization" in this configuration manual.

## 2 System & Software Specification

This research is carried out with the following system and software specifications, the system configuration is shown in Figure 1.

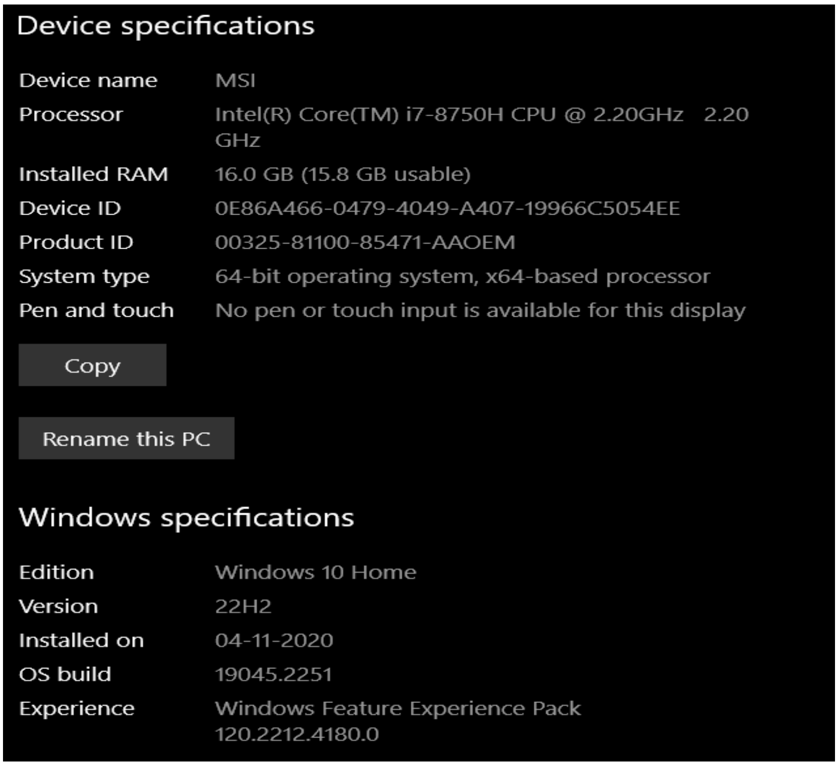


Figure 1: System Configuration

### 2.1 Softwares & Hardwares

- GPU: NVIDIA GeForce GTX 1050

- MS Office 365: The metadata is used in the form of Comma Separated Values (CSV) file.
- Anaconda Navigator: Python version is 3.9.7, Jupyter Notebook version is 6.4.5

# 3 Packages & Libraries

In order to perform data analysis on the data, necessary packages and libraries need to be imported. Figure 2 shows the list of libraries used for this project.

```
import os
import numpy as np
from models import cnn, cnn_lstm
from utils import f1_m, precision_m, recall_m, plot_graph
import matplotlib.pyplot as plt
import random
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras import optimizers
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.utils import plot_model, to_categorical
from tqdm import tqdm
import pandas as pd
random.seed(1)

# Usual Libraries
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import sklearn

import IPython.display as ipd
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import SGDClassifier, LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import f1_score
from sklearn.neural_network import MLPClassifier
from xgboost import XGBClassifier, XGBRFClassifier
from xgboost import plot_tree, plot_importance

from sklearn.metrics import confusion_matrix, accuracy_score, roc_auc_score, roc_curve
from sklearn import preprocessing
from sklearn.feature_selection import RFE
import warnings
warnings.filterwarnings('ignore')
```

Figure 2: Libraries Used for this Project

# 4 Dataset

For this project, a public data-set called Free Music Archive (FMA) data-set is used. The data-set can be accessed from `https://github.com/mdeff/fma`, and for this research a subset of 8000 mp3 audio tracks are used for computational purposes.

## 4.1 EDA

After the data is imported into the python environment, basic EDA is done on the 'fma_small' data which contains 8000 tracks. The Figure 3 shows the classification across various genres.

The below Figure 4 shows the code to generate the Mel-spectrogram and Figure 5 shows the Mel-spectrogram folk genres.

### Get Distribution of all Genres

```python
from collections import Counter
genre_count = Counter(df[('track', 'genre_top')].values)
songs_summary = pd.DataFrame({"label":list(genre_count.keys()), "length":list(genre_count.values())})
songs_summary
```

| | label | length |
|---|---|---|
| 0 | Hip-Hop | 1000 |
| 1 | Pop | 1000 |
| 2 | Folk | 1000 |
| 3 | Experimental | 1000 |
| 4 | Rock | 1000 |
| 5 | International | 1000 |
| 6 | Electronic | 1000 |
| 7 | Instrumental | 1000 |

```python
classes = list(np.unique(songs_summary.label))
class_dist = songs_summary.groupby(['label'])['length'].mean()
```

```python
fig, ax = plt.subplots()
ax.set_title('Class distribution', y=1.00)
ax.pie(class_dist, labels = class_dist.index, autopct='%1.1f%%', shadow = False)#, startangle = 90,textpr
ax.axis('equal') #for circle
plt.show()
```

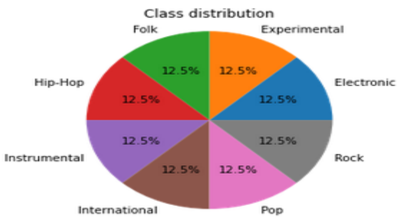


Figure 3: Genres Classification

```python
def generate_spectrogram(trackid, genre):
    filename = fetch_audio_path(audio_dir, trackid)
    y, sr = librosa.load(filename)
    print(len(y),sr)
    spectro = librosa.feature.melspectrogram(y = y, sr = sr, n_fft = 2048, hop_length = 512)
    spectro = librosa.power_to_db(spectro, ref = np.max)
    print(spectro.shape, genre)
    plt.figure(figsize = (10, 4))
    librosa.display.specshow(spectro, y_axis = 'mel', fmax = 8000, x_axis = 'time')
    plt.colorbar(format = '%+2.0f dB')
    plt.title(str(genre))
    plt.show()
```

Figure 4: Mel-spectrograms Function

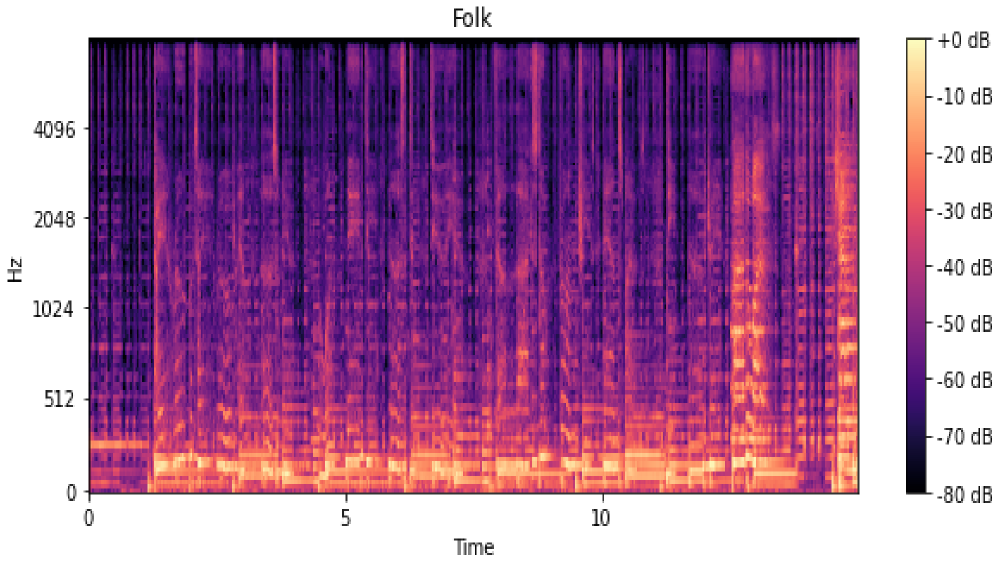


Figure 5: Mel-spectrograms of folk Genres

# 5 Data Pre-processing

Figure 6 shows the data pre-processing by feature extraction through Mel-spectrogram.

```python
def mp3_to_n_slice_np(file_path):
    """
    Function to convert .mp3 file to 10 slice mel spectrogram
    """
    y, sr = librosa.load(file_path)

    #Load mp3 file
    melspectrogram_array = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=128, fmax=8000)

    #raise to db values
    mel = librosa.power_to_db(melspectrogram_array)  #Convert a power spectrogram to decibel (dB) units

    #Covert mel values to image and slice into 10 parts
    plt.rcParams['figure.dpi'] = 100
    fig_size = plt.rcParams["figure.figsize"]
    fig_size[0] = float(mel.shape[1]) / float(100)
    fig_size[1] = float(mel.shape[0]) / float(100)
    plt.rcParams["figure.figsize"] = fig_size
    plt.axis('off')
    plt.axes([0., 0., 1., 1.0], frameon=False, xticks=[], yticks=[])
    librosa.display.specshow(mel, cmap='gray_r')
    buf = io.BytesIO()
    plt.savefig(buf, bbox_inches=None, pad_inches=0, format="jpg")
    plt.clf()
    plt.close()

    buf.seek(0)
    img_array = np.asarray(bytearray(buf.read()))
    img_col = cv2.imdecode(img_array, cv2.IMREAD_UNCHANGED) #Read image data from a memory cache and convert
    img_gray = cv2.cvtColor(img_col, cv2.COLOR_BGR2GRAY) #Converts an image from one color space to another
    buf.flush()
    buf.close()

    num_slices = int(img_gray.shape[1]/128)
    img_gray = img_gray[:,:(num_slices*128)]
    split_spectrum = np.hsplit(img_gray, num_slices)

    return split_spectrum, num_slices
```

Figure 6: Data Pre-processing

The below Figure 7 shows how each audio track is processed through Mel-spectrogram, finally, the data is stored in '.npy' format which will be used during the model building process.

**Processing of each song to feed into model**

```
X = []
y = []
name = []

#Iterate through each folder
for d in tqdm(directories):
    label_directory = os.path.join(folder_sample, d)

    #Get path of all .mp3 files for each folder
    file_names = [os.path.join(label_directory, f) for f in os.listdir(label_directory) if f.endswith(".mp3")]

    # Convert .mp3 files into mel-Spectograms
    for f in file_names:
        logger.info(f"Currently on {os.path.split(f)[1]}")
        try:
            split_output, n = mp3_to_n_slice_np(f)
            logger.info(f"The mp3 has been split into {n} slices")
            X.extend(split_output)

            track_id = int(os.path.split(f)[1].split(".")[0])
            track_index = list(tracks_id_array).index(track_id)
            output1 = [tracks_genre_array[track_index]]*n
            y.extend(output1)
            logger.info(f"Genre is {output1[0]}")

            output2 = [tracks_name_array[track_index]]*n
            name.extend(output2)
            logger.info(f"Song name is {output2[0]}")
        except Exception as e:
            logger.error(f"Could not load file due to {str(e)}")
print("Conversion Complete")
```

```
100%|██████████| 156/156 [2:33:30<00:00, 59.04s/it]
Conversion Complete
```

**Saving Train Data**

```
if not os.path.exists("./train"):
    os.mkdir("./train")
np.save("./train/features.npy", X)
np.save("./train/classes.npy", y)
np.save("./train/names.npy", name)
```

Figure 7: Processing of Audio Tracks

# 6 Classification Models

We will see the constructed architecture of CNN and CNN-LSTM models, along with the machine learning models.

## 6.1 CNN Architecture

The CNN model's construction process is shown in Figure 8 and its model architecture is shown in Figure 9, respectively.

```
### CNN ########
cnn = Sequential(name="CNN")
cnn.add(Conv2D(filters=64, kernel_size=[7, 7], kernel_initializer=initializers.he_normal(
    seed=1), activation="relu"))  # Dim = (122x122x64)
cnn.add(BatchNormalization())
cnn.add(AveragePooling2D(pool_size=[2, 2], strides=2))  # Dim = (61x61x64)

cnn.add(Conv2D(filters=128, kernel_size=[7, 7], strides=2, kernel_initializer=initializers.he_normal(
    seed=1), activation="relu"))  # Dim = (28x28x128)
cnn.add(BatchNormalization())
cnn.add(AveragePooling2D(pool_size=[2, 2], strides=2))  # Dim = (14x14x128)

cnn.add(Conv2D(filters=256, kernel_size=[3, 3], kernel_initializer=initializers.he_normal(
    seed=1), activation="relu"))  # Dim = (12x12x256)
cnn.add(BatchNormalization())
cnn.add(AveragePooling2D(pool_size=[2, 2], strides=2))  # Dim = (6x6x256)

cnn.add(Conv2D(filters=512, kernel_size=[3, 3], kernel_initializer=initializers.he_normal(
    seed=1), activation="relu"))  # Dim = (4x4x512)
cnn.add(BatchNormalization())
cnn.add(AveragePooling2D(pool_size=[2, 2], strides=2))  # Dim = (2x2x512)

cnn.add(BatchNormalization())
cnn.add(Flatten())  # Dim = (2048)
```

```
cnn.add(BatchNormalization())
cnn.add(Dropout(0.6))

cnn.add(Dense(1024, activation="relu",
              kernel_initializer=initializers.he_normal(seed=1)))  # Dim = (1024)
cnn.add(Dropout(0.5))

cnn.add(Dense(256, activation="relu",
              kernel_initializer=initializers.he_normal(seed=1)))  # Dim = (256)
cnn.add(Dropout(0.2))
cnn.add(Dense(64, activation="relu",
              kernel_initializer=initializers.he_normal(seed=1)))  # Dim = (64)
#cnn.add(Dropout(0.25))
cnn.add(Dense(32, activation="relu",
              kernel_initializer=initializers.he_normal(seed=1)))  # Dim = (32))
#cnn.add(Dropout(0.1))
cnn.add(Dense(8, activation="softmax",
              kernel_initializer=initializers.he_normal(seed=1)))
```

Figure 8: CNN Model

```
Training CNN...
Model: "CNN"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (32, 122, 122, 64)        3200

 batch_normalization (BatchN  (32, 122, 122, 64)       256
 ormalization)

 average_pooling2d (AverageP  (32, 61, 61, 64)         0
 ooling2D)

 conv2d_1 (Conv2D)           (32, 28, 28, 128)         401536

 batch_normalization_1 (Batc  (32, 28, 28, 128)        512
 hNormalization)

 average_pooling2d_1 (Averag  (32, 14, 14, 128)        0
 ePooling2D)

 conv2d_2 (Conv2D)           (32, 12, 12, 256)         295168

 batch_normalization_2 (Batc  (32, 12, 12, 256)        1024
 hNormalization)

 average_pooling2d_2 (Averag  (32, 6, 6, 256)          0
 ePooling2D)

 conv2d_3 (Conv2D)           (32, 4, 4, 512)           1180160

 batch_normalization_3 (Batc  (32, 4, 4, 512)          2048
 hNormalization)

 average_pooling2d_3 (Averag  (32, 2, 2, 512)          0
 ePooling2D)

 batch_normalization_4 (Batc  (32, 2, 2, 512)          2048
 hNormalization)

 flatten (Flatten)           (32, 2048)                0

 batch_normalization_5 (Batc  (32, 2048)               8192
 hNormalization)

 dropout (Dropout)           (32, 2048)                0

 dense (Dense)               (32, 1024)                2098176

 dropout_1 (Dropout)         (32, 1024)                0

 dense_1 (Dense)             (32, 256)                 262400

 dropout_2 (Dropout)         (32, 256)                 0

 dense_2 (Dense)             (32, 64)                  16448

 dense_3 (Dense)             (32, 32)                  2080

 dense_4 (Dense)             (32, 8)                   264
```

Figure 9: CNN Architecture Output

## 6.2 CNN-LSTM Architecture

Although reshape and permute layers are included in addition to the LSTM layers, the CNN-LSTM is constructed in this case fairly similarly to CNN. The CNN-LSTM model's construction process is shown in Figure 10 along with a code sample, and the model architecture is shown in Figure 11.

```
### CNN-LSTM ######
cnn_lstm = Sequential(name="CNNLSTM")
cnn_lstm.add(Conv2D(filters=64, kernel_size=[7, 7], kernel_initializer=initializers.he_normal(
    seed=1), activation="relu"))  # Dim = (122x122x64)
cnn_lstm.add(BatchNormalization())
cnn_lstm.add(AveragePooling2D(pool_size=[2, 2], strides=2))  # Dim = (61x61x64)

cnn_lstm.add(Conv2D(filters=128, kernel_size=[7, 7], strides=2, kernel_initializer=initializers.he_normal(
    seed=1), activation="relu"))  # Dim = (28x28x128)
cnn_lstm.add(BatchNormalization())
# Dim = (14x14x128)
cnn_lstm.add(AveragePooling2D(pool_size=[2, 2], strides=2))

cnn_lstm.add(Conv2D(filters=256, kernel_size=[3, 3], kernel_initializer=initializers.he_normal(
    seed=1), activation="relu"))  # Dim = (12x12x256)
cnn_lstm.add(BatchNormalization())
cnn_lstm.add(AveragePooling2D(pool_size=[2, 2], strides=2))  # Dim = (6x6x256)

cnn_lstm.add(Conv2D(filters=512, kernel_size=[3, 3], kernel_initializer=initializers.he_normal(
    seed=1), activation="relu"))  # Dim = (4x4x512)
cnn_lstm.add(BatchNormalization())
cnn_lstm.add(AveragePooling2D(pool_size=[2, 2], strides=2))  # Dim = (2x2x512)

cnn_lstm.add(BatchNormalization())
cnn_lstm.add(Dropout(0.6))

cnn_lstm.add(Reshape((512, -1)))
cnn_lstm.add(Permute((2, 1)))

cnn_lstm.add(LSTM(128, return_sequences=True, input_shape=(128, 128, 1)))
cnn_lstm.add(LSTM(128, input_shape=(128, 128, 1)))

cnn_lstm.add(Dense(1024, activation="relu",
                   kernel_initializer=initializers.he_normal(seed=1)))  # Dim = (1024)
cnn_lstm.add(Dropout(0.5))

cnn_lstm.add(Dense(256, activation="relu",
                   kernel_initializer=initializers.he_normal(seed=1)))  # Dim = (256)
cnn_lstm.add(Dropout(0.2))

cnn_lstm.add(Dense(64, activation="relu",
                   kernel_initializer=initializers.he_normal(seed=1)))  # Dim = (64)
#cnn.add(Dropout(0.25))
cnn_lstm.add(Dense(32, activation="relu",
                   kernel_initializer=initializers.he_normal(seed=1)))  # Dim = (32)
#cnn.add(Dropout(0.1))
cnn_lstm.add(Dense(8, activation="softmax",
                   kernel_initializer=initializers.he_normal(seed=1)))
```

Figure 10: CNN-LSTM Model

```
Training CNNLSTM...
Model: "CNNLSTM"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (32, 122, 122, 64)        3200

 batch_normalization_6 (Batc  (32, 122, 122, 64)       256
 hNormalization)

 average_pooling2d_4 (Averag  (32, 61, 61, 64)         0
 ePooling2D)

 conv2d_5 (Conv2D)           (32, 28, 28, 128)         401536

 batch_normalization_7 (Batc  (32, 28, 28, 128)        512
 hNormalization)

 average_pooling2d_5 (Averag  (32, 14, 14, 128)        0
 ePooling2D)

 conv2d_6 (Conv2D)           (32, 12, 12, 256)         295168

 batch_normalization_8 (Batc  (32, 12, 12, 256)        1024
 hNormalization)

 average_pooling2d_6 (Averag  (32, 6, 6, 256)          0
 ePooling2D)

 conv2d_7 (Conv2D)           (32, 4, 4, 512)           1180160

 batch_normalization_9 (Batc  (32, 4, 4, 512)          2048
 hNormalization)

 average_pooling2d_7 (Averag  (32, 2, 2, 512)          0
 ePooling2D)

 batch_normalization_10 (Bat  (32, 2, 2, 512)          2048
 chNormalization)

 dropout_3 (Dropout)         (32, 2, 2, 512)           0

 reshape (Reshape)           (32, 512, 4)              0

 permute (Permute)           (32, 4, 512)              0

 lstm (LSTM)                 (32, 4, 128)              328192

 lstm_1 (LSTM)               (32, 128)                 131584

 dense_5 (Dense)             (32, 1024)                132096

 dropout_4 (Dropout)         (32, 1024)                0

 dense_6 (Dense)             (32, 256)                 262400

 dropout_5 (Dropout)         (32, 256)                 0

 dense_7 (Dense)             (32, 64)                  16448

 dense_8 (Dense)             (32, 32)                  2080

 dense_9 (Dense)             (32, 8)                   264
```

Figure 11: CNN-LSTM Architecture Output

## 6.3 Machine Learning

The machine-learning models' construction process is shown in a snippet of code in Figure 12. For this research, eight alternative machine learning models have been developed.

**Machine Learning Models**

```
# Function to assess the accuracy of a model
def model_assess(model, title = "Default"):
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    #print(confusion_matrix(y_test, preds))
    print('Accuracy', title, ':', round(accuracy_score(y_test, preds), 5), '\n')

# Naive Bayes
nb = GaussianNB()
model_assess(nb, "Naive Bayes")

# Stochastic Gradient Descent
sgd = SGDClassifier(max_iter=5000, random_state=0)
model_assess(sgd, "Stochastic Gradient Descent")

# KNN
knn = KNeighborsClassifier(n_neighbors=19)
model_assess(knn, "KNN")

# Decission trees
tree = DecisionTreeClassifier()
model_assess(tree, "Decission trees")

# Random Forest
rforest = RandomForestClassifier(n_estimators=1000, max_depth=10, random_state=0)
model_assess(rforest, "Random Forest")

# Support Vector Machine
svm = SVC(decision_function_shape="ovo")
model_assess(svm, "Support Vector Machine")

# Logistic Regression
lg = LogisticRegression(random_state=0, solver='lbfgs', multi_class='multinomial')
model_assess(lg, "Logistic Regression")

# Cross Gradient Booster
xgb = XGBClassifier(n_estimators=1000, learning_rate=0.05)
model_assess(xgb, "Cross Gradient Booster")
preds = xgb.predict(X_test)
```

Figure 12: Machine Learning Model

# 7 Implementation of Code

- Download FMA data-set from `https://github.com/mdeff/fma`
- Download 'Final_Thesis_Project.zip', unzip it, and create a folder called 'FMA'.
- Unzip the downloaded data-set into the newly created 'FMA' folder.
- Run 'FMA-EDA.ipynb' and 'FMA-Preprocessing.ipynb' scripts to get the npy files that will be used during the model building process.
- Run 'FMA-Train.ipynb' script, When the trained model is done, the script prompts for epoch number to be fed for the test data validation, then the python script asks to save the model with 'Y' or 'N'.
- The previous step is repeated again for the CNN-LSTM model also. Finally, the machine-learning model is also completed.