Configuration Manual

MSc Research Project
Data Analytics

Rajesh Ramachandran Nair
Student ID: 20141289

School of Computing
National College of Ireland

Supervisor: Dr Catherine Mulwa

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Rajesh Ramachandran Nair
Student ID:	20141289
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Dr Catherine Mulwa
Submission Due Date:	16/08/2021
Project Title:	Configuration Manual
Word Count:	700
Page Count:	15

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:	
Date:	23rd September 2021

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

Rajesh Ramachandran Nair
20141289

1 Hardware Setup

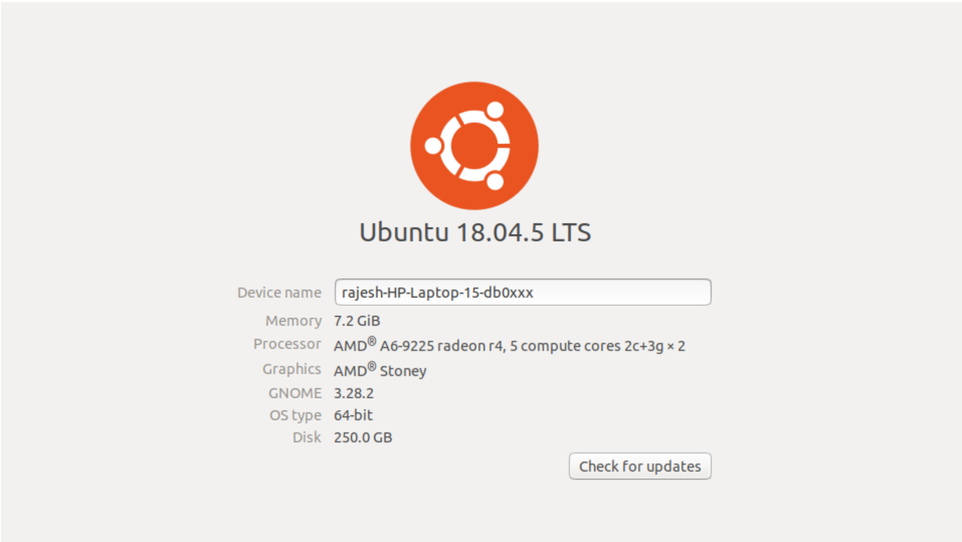

Figure 1: The flow indicating methods that is used to identify plagiarism in music

The specification shown in Figure 1 is the machine which was used in this research. It is having an installed RAM of 8 GB with a 64-bit Operating System and an installed Ubuntu 18.04.5LTS configuration. The Machine is equipped with AMD® A6-9225 Radeon r4, 5 compute cores 2c+3g $\times$ 2 processor, and AMD® Stoney graphics.

2 Package Requirements and Installation

Python is used to create this project. Because training on a GPU can take a long time, the IDE Google Colab was used to prepare the data and train the machine learning models. The following is a list of libraries that are necessary to execute this project. To ensure a smooth execution, make sure following libraries are installed on Python. A package called Music21 is to added to Access The functions to manipulate MIDI files.This Package was Developed by MIT.

- Numpy[1]

[1] `https://numpy.org`

- Pandas[2]
- NLTK[3]
- Sklearn[4]
- Matplotlib[5]
- Gensim[6]
- Music21[7]

3 Data Preparation

In Figure 2 the following libraries were imported to performed Data preparation and feature extraction

```
import numpy as np #for covering the text to vectors
import pandas as pd# for the dataframe
from music21 import converter, corpus, instrument, midi, note, chord, pitch,stream,roman
#for analysis of the MIDI files to understand the structure of the file
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
#to plot the analysis
import requests#for webscraping the data
from bs4 import BeautifulSoup #for webscraping the data
# from multiprocessing.dummy import Pool as ThreadPool # Use this when IO is the problem
from multiprocessing import Pool # Use this when CPU-intensive functions are the problem.
import gensim, logging
#use for the WORD2VEC model to find the similarity relationship between the similar chords
import pprint
```

Figure 2: Libraries required for the preprocess are imported

Next the Appropriate folders are created as shown in figure to store the data set and they are stored in them. To understand what an MIDI file is made of a sample file is download to understand them as shown in In Figure 3[8]

The sample MIDI file is opened to remove the drum track in order to extract the correct melody track from the file as shown in Figure 4

Next the notes are extracted from the MIDI file as shown in Figure 5

The extracted notes are visualized using the code in Figure 6.

Other properties like the Time signature,chords,key Signature is known from below code Figure 7.

[2] https://pandas.pydata.org
[3] https://www.nltk.org
[4] https://scikit-learn.org
[5] https://matplotlib.org
[6] https://radimrehurek.com
[7] https://web.mit.edu/music21/
[8] https://files.khinsider.com/midifiles/genesis/sonic-the-hedgehog/green-hill-zone.mid

```
[ ] # Defining some constants and creating a new folder for MIDIs.
    midi_path = "MIDIs"
    sonic_folder = "sonic"

    !rm -r $midi_path
    !mkdir $midi_path

    # Some helper methods.
    def concat_path(path, child):
        return path + "/" + child

    def download_midi(midi_url, path):
        !wget $midi_url --directory-prefix $path > download_midi.log

    # Downloading an example file.
    sonic_path = concat_path(midi_path, sonic_folder)
    download_midi(
        "https://files.khinsider.com/midifiles/genesis/sonic-the-hedgehog/green-hill-zone.mid",
        sonic_path)

    print(os.listdir(sonic_path))
```

Figure 3: Downloading a Single MIDI file and creating folder structure

After the structure of the MIDI file is understood next a collection of it is downloaded using beautiful soup a web scraping tool for python the data set includes 450 MIDI As in Figure 8 files[9] [10] [11] which will be stored in the directories created.Figure 8

After the downloaded files are loaded again to process it features of the MIDI especially the chords or the melody is extracted.Figure 9

A data frame is created to store the music name with their corresponding harmony after the harmonic reduction is done.Figure 10

Below image shows the vectorizing a harmony before feeding them to train for Word2Vec model. Figure 11

Word2Vec model is called to trin on the chords to perform chord substitution. Chord sustitution is nothing but substituting a chord progression with a similar chord which is used to perform harmonic Reduction.Figure 12

At last the MIDI files are converted to set of harmonic reduction values which will be used to feed the rest of the model.Figure 13.

[9] https://files.khinsider.com/midifiles/genesis/sonic-the-hedgehog
[10] https://files.khinsider.com/midifiles/genesis/sonic-the-hedgehog-2
[11] https://files.khinsider.com/midifiles/genesis/sonic-the-hedgehog-3

```
[ ] from music21 import converter, corpus, instrument, midi, note, chord, pitch

    def open_midi(midi_path, remove_drums):
        # There is an one-line method to read MIDIs
        # but to remove the drums we need to manipulate some
        # low level MIDI events.
        mf = midi.MidiFile()
        mf.open(midi_path)
        mf.read()
        mf.close()
        if (remove_drums):
            for i in range(len(mf.tracks)):
                mf.tracks[i].events = [ev for ev in mf.tracks[i].events if ev.channel != 10]

        return midi.translate.midiFileToStream(mf)

    base_midi = open_midi(concat_path(sonic_path, "green-hill-zone.mid"), True)
    base_midi

    <music21.stream.Score 0x7ffba930d750>
```

Figure 4: open close operation and removing the drum track from the file

```
def extract_notes(midi_part):
    parent_element = []
    ret = []
    for nt in midi_part.flat.notes:
        if isinstance(nt, note.Note):
            ret.append(max(0.0, nt.pitch.ps))
            parent_element.append(nt)
        elif isinstance(nt, chord.Chord):
            for pitch in nt.pitches:
                ret.append(max(0.0, pitch.ps))
                parent_element.append(nt)

    return ret, parent_element
```

Figure 5: Extraction of notes

```python
from music21 import stream

temp_midi_chords = open_midi(
    concat_path(sonic_path, "green-hill-zone.mid"),
    True).chordify()
temp_midi = stream.Score()
temp_midi.insert(0, temp_midi_chords)

# Printing merged tracks.
print_parts_countour(temp_midi)

# Dumping first measure notes
temp_midi_chords.measures(0, 1).show("text")
```

Figure 6: The extracted notes are visualized

```python
timeSignature = base_midi.getTimeSignatures()[0]
music_analysis = base_midi.analyze('key')
print("Music time signature: {0}/{1}".format(timeSignature.beatCount, timeSignature.denominator))
print("Expected music key: {0}".format(music_analysis))
print("Music key confidence: {0}".format(music_analysis.correlationCoefficient))
print("Other music key alternatives:")
for analysis in music_analysis.alternateInterpretations:
    if (analysis.correlationCoefficient > 0.5):
        print(analysis)
```

```
Music time signature: 4/4
Expected music key: a minor
Music key confidence: 0.8770275812674332
Other music key alternatives:
C major
F major
G major
d minor
```

Figure 7: Other properties of the Files are checked

```
[ ] def download_midi_files(url, output_path):
        site_request = requests.get(url)
        if (site_request.status_code != 200):
            raise Exception("Failed to access {0}".format(url))

        soup = BeautifulSoup(site_request.content, 'html.parser')
        link_urls = soup.find_all('a')

        for link in link_urls:
            href = link['href']
            if (href.endswith(".mid")):
                file_name = get_file_name(href)
                download_path = concat_path(output_path, file_name)
                midi_request = download_file(href, download_path)

    def start_midis_download(folder, url):
        !mkdir $folder # It is fine if this command fails when the directory already exists.
        download_midi_files(url, folder)

    target_games = dict()
    target_games["sonic1"] = "https://www.khinsider.com/midi/genesis/sonic-the-hedgehog"
    target_games["sonic2"] = "https://www.khinsider.com/midi/genesis/sonic-the-hedgehog-2"
    target_games["sonic3"] = "https://www.khinsider.com/midi/genesis/sonic-the-hedgehog-3"
    target_games["sonicAndKnuckles"] = "https://www.khinsider.com/midi/genesis/sonic-and-knuckles"

    for key, value in target_games.items():
        file_path = concat_path(sonic_path, key)
        start_midis_download(file_path, value)
```

Figure 8: A collection of 450 MIDI files are Webscraped

```python
def harmonic_reduction(midi_file):
    ret = []
    temp_midi = stream.Score()
    temp_midi_chords = midi_file.chordify()
    temp_midi.insert(0, temp_midi_chords)
    music_key = temp_midi.analyze('key')
    max_notes_per_chord = 4
    for m in temp_midi_chords.measures(0, None): # None = get all measures.
        if (type(m) != stream.Measure):
            continue

        # Here we count all notes length in each measure,
        # get the most frequent ones and try to create a chord with them.
        count_dict = dict()
        bass_note = note_count(m, count_dict)
        if (len(count_dict) < 1):
            ret.append("-") # Empty measure
            continue

        sorted_items = sorted(count_dict.items(), key=lambda x:x[1])
        sorted_notes = [item[0] for item in sorted_items[-max_notes_per_chord:]]
        measure_chord = chord.Chord(sorted_notes)

        # Convert the chord to the functional roman representation
        # to make its information independent of the music key.
        roman_numeral = roman.romanNumeralFromChord(measure_chord, music_key)
        ret.append(simplify_roman_name(roman_numeral))

    return ret
```

Figure 9: Harmonic reduction happens

```
]  def create_midi_dataframe(target_games):
       key_signature_column = []
       game_name_column = []
       harmonic_reduction_column = []
       midi_name_column = []
       pool = Pool(8)
       midi_params = []
       for key, value in target_games.items():
           folder_path = concat_path(sonic_path, key)
           for midi_name in os.listdir(folder_path):
               midi_params.append((key, concat_path(folder_path, midi_name)))

       results = pool.map(process_single_file, midi_params)
       for result in results:
           if (result is None):
               continue

           key_signature_column.append(result[0])
           game_name_column.append(result[1])
           harmonic_reduction_column.append(result[2])
           midi_name_column.append(result[3])

       d = {'midi_name': midi_name_column,
            'game_name': game_name_column,
            'key_signature' : key_signature_column,
            'harmonic_reduction': harmonic_reduction_column}
       return pd.DataFrame(data=d)

   sonic_df = create_midi_dataframe(target_games)
```

Figure 10: A new dataframe is created after the harmonic reduction

```
def vectorize_harmony(model, harmonic_reduction):
    # Gets the model vector values for each chord from the reduction.
    word_vecs = []
    for word in harmonic_reduction:
        try:
            vec = model[word]
            word_vecs.append(vec)
        except KeyError:
            # Ignore, if the word doesn't exist in the vocabulary
            pass

    # Assuming that document vector is the mean of all the word vectors.
    return np.mean(word_vecs, axis=0)

def cosine_similarity(vecA, vecB):
    # Find the similarity between two vectors based on the dot product.
    csim = np.dot(vecA, vecB) / (np.linalg.norm(vecA) * np.linalg.norm(vecB))
    if np.isnan(np.sum(csim)):
        return 0

    return csim
```

Figure 11: Vectorize the harmony for Word2vec

```
[ ] import pandas as pd
    import numpy as np

    sonic_df = pd.read_excel("/content/Sonic_MIDI.xlsx")
```

```
[ ] # import modules & set up logging
    import gensim, logging
    # logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)

    model = gensim.models.Word2Vec(sonic_df["harmonic_reduction"], min_count=2, window=4)
```

Figure 12: Word2Vec is performed for chord substitution

```
sonic_df = sonic_df.sample(frac=1).reset_index(drop=True)
```

```
riter = pd.ExcelWriter('/content/Sonic_similarity_shuffled.xlsx')
sonic_df.to_excel(writer)
writer.save()
```

Figure 13: The data is preprocessed to be fed to final models

4 Model Implementation and Evaluation

The following packages were imported to run the machine learning models As shown in Figure 14

```python
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
import re
from gensim import utils
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from gensim.models import Doc2Vec
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
```

Figure 14: imported packages

As shown in Figure 15 After the packages are imported the data In the Dataframe is being loaded

Then bag of words is being vectorized using CountVectorizer() and the dataset is split to 4:1 train test ratio for training,shown in Figure 16

After the train test split is done the train data is served to all the models one by one and the results are evaluated using the Confusion matrix and ROC Curve,shown in Figure 17,18,19,20

In this code the it checked at which value of K the KNN performs best and the respective graph is plotted.As shown in Figure 21,22 The KNN model is trained and evaluated.

This code will be used to check the accuracy of each model at different plagiarism values(0.8,0.7,0.6,05).As shown in Figure 23

This concludes with the Configuration Manual.

```
[ ] duplicate = []
    for i in sonic_df.score:
       if i > 0.5:#here the plagiarism values values can be adjusted to 0.8,0.7,0.6,0.5
          duplicate.append(1)
       else:
          duplicate.append(0)

    sonic_df['duplicate'] = duplicate
```

```
corpus = []#bag of words are created to be fed the models

for i in range(len(sonic_df)):
    source = re.sub("\[|\]|\,|\'|,","",sonic_df['sourceChord'][i])
    target = re.sub("\[|\]|\,|\'|,","",sonic_df['targetChord'][i])
    source = source.lower()
    target = target.lower()
    source = source.split()
    target = target.split()
    source = ' '.join(source)
    target = ' '.join(target)
    sourceandtarget = source + target
    corpus.append(sourceandtarget)#bag of words to be vectorized
```

Figure 15: plagiarism threshold can be adjusted and next cell is converting the chord to bag of words

```
from sklearn.feature_extraction.text import CountVectorizer #bag of words are created to vectors
cv = CountVectorizer(max_features = 577)
X = cv.fit_transform(corpus).toarray()
y = sonic_df['duplicate']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

Figure 16: Bag of words are vectorized and data is split to 4:1 ratio

```
#Naive Bayes Model
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
#evaluation and results
print(classification_report(y_test,y_pred))
#ROC curve
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
```

Figure 17: Naive Bayes model trained and evaluated

```python
#Logistic Regression
from sklearn.linear_model import LogisticRegression
classifier1 = LogisticRegression(random_state = 0)
classifier1.fit(X_train, y_train)
y_pred = classifier1.predict(X_test)
#evaluation and results
print(classification_report(y_test,y_pred))
#Roc curve
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
```

Figure 18: Logistic Regression model trained and evaluated

```python
#Decision tree Classifier
from sklearn.tree import DecisionTreeClassifier
classifier2 = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier2.fit(X_train, y_train)
y_pred = classifier2.predict(X_test)
#evaluation and results
print(classification_report(y_test,y_pred))
#Roc curve
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
```

Figure 19: Decision tree classifier model trained and evaluated

```python
#Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
classifier3 = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
classifier3.fit(X_train, y_train)
y_pred = classifier3.predict(X_test)
#evaluation and results
print(classification_report(y_test,y_pred))
#Roc curve
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
```

Figure 20: Random forest model trained and evaluated

```python
#KNN
from sklearn.neighbors import KNeighborsClassifier

#Setup arrays to store training and test accuracies
neighbors = np.arange(1,3)
train_accuracy =np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i,k in enumerate(neighbors):
    #Setup a knn classifier with k neighbors
    knn = KNeighborsClassifier(n_neighbors=k)

    #Fit the model
    knn.fit(X_train, y_train)

    #Compute accuracy on the training set
    train_accuracy[i] = knn.score(X_train, y_train)

    #Compute accuracy on the test set
    test_accuracy[i] = knn.score(X_test, y_test)

    print(k)
```

Figure 21: K number is determined

```
[ ] #K number plot
    plt.title('k-NN Varying number of neighbors')
    plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
    plt.plot(neighbors, train_accuracy, label='Training accuracy')
    plt.legend()
    plt.xlabel('Number of neighbors')
    plt.ylabel('Accuracy')
    plt.show()
```

```
#KNN model called
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(X_train,y_train)
y_pred = knn.predict(X_test)
#Evaluation and results
print(classification_report(y_test,y_pred))
#Roc Curve
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
```

Figure 22: KNN model trained and evaluated

```
#bar plot of accuracy of each model at different plagiararism threshold value
height = [95,92, 89, 86]
bars = ['0.5', '0.6', '0.7', '0.8']

plt.bar(bars, height)
plt.ylabel('Accuracy')
plt.xlabel('Different Plagiarism threshold')
plt.title('Accuracy at different threshold of KNN Model')
```

Figure 23: Accuracy graph for each model is plotted with different plagiarism threshold

References