**２０２１　年度　　 卒業論文**

卒業論文題目

|  |
| --- |
| ラウンドロビン方式の負荷分散を導入した  Web楽曲分類サービスの設計と開発 |

　指導教員　　　　　　鷹野　孝典　教授

神奈川工科大学　情報工学科

　　　　　学籍番号　　　　1821144

　　　　　学生氏名　　　　吉井 智哉

　提出日　2021年　12月23日　指導教員　　印

　受理日　2021年　１2月23日　情報工学科長　　印

論文要旨

要旨を書く．

目次

[1. まえがき 1](#_Toc90889697)

[1.1 はじめに 1](#_Toc90889698)

[1.2 課題と提案内容 1](#_Toc90889699)

[1.3 本論文の構成 1](#_Toc90889700)

[**2.** 関連研究 2](#_Toc90889701)

[2.1 動画の印象評価データセット構築とその特性の調査 2](#_Toc90889702)

[2.2 2](#_Toc90889703)

[2.3 ニューラルネットワークを用いた音楽の自動ジャンル分類 2](#_Toc90889704)

[**3.** 提案システム 3](#_Toc90889705)

[3.1 提案システムの概要 3](#_Toc90889706)

[3.2 提案システムの実行手順 3](#_Toc90889707)

[**4.** 実装システム 4](#_Toc90889708)

[4.1 実装システムの概要 4](#_Toc90889709)

[4.2 実装システムの処理手順 4](#_Toc90889710)

[4.3 環境構築 4](#_Toc90889711)

[4.4 実装システムの構築 6](#_Toc90889712)

[4.4.1 numpyを利用した楽曲分析 6](#_Toc90889713)

[4.4.2 kerasを利用してジャンル推定モデルの作成と楽曲ジャンル推定 8](#_Toc90889714)

[4.5 分散処理の実装 11](#_Toc90889715)

[**5.** 実験 13](#_Toc90889716)

[5.1 実験目的 13](#_Toc90889717)

[5.2 実験環境の構築 13](#_Toc90889718)

[5.3 実験方法（予定） 13](#_Toc90889719)

[6. むすび 14](#_Toc90889724)

# まえがき

## はじめに

　インターネットの発展や，スマートフォンのようなコンピュータの普及によりSNS(Social Networking Service)が大きく発展した，これに伴って音楽投稿型のSNSも同様に発展を遂げた．そのため誰でも音楽をSNSなどに投稿できるようになっており，SNSなどに投稿される楽曲が莫大な量になってきている．

　また，音楽のジャンル分類は無意味であるといった意見が少なからずあるが，ジャンル分類にもメリットはある．ジャンル分類することでその楽曲の特徴をある程度つかむことができる．例えば，ロックといったジャンルであればテンポは速めでギターやベースなどの楽器が使われているという特徴がわかる．他にもクラッシックであればローテンポでピアノやバイオリンなどの楽器が使われているという特徴をつかむことができる．このようにジャンル分類することで楽曲の特徴をある程度指標として理解することができるので音楽ジャンル分類にも意味はある．他にも，SNSに投稿さる写真から動画コンテンツを作成するようなサービスがあるとき，作成された動画コンテンツとジャンル分けされた楽曲コンテンツとでジャンルマッチングを行うことで，より付加価値の高いコンテンツを作成できると考えた．

　本研究では楽曲SNSに投稿される大量の楽曲データから特徴を抽出し，その抽出した特徴から深層学習モデルを作成，その後深層学習モデルによるジャンル推定処理を行う．これらの処理には高負荷がかかると予測されるため，これらの処理を複数のサーバで分散処理することで処理速度の向上を実現する．

　また本研究での実装システムでは楽曲の周波数を視覚的に確認できるスペクトログラムを作成した．このスペクトログラムによって楽曲の特徴を視覚的にサービスのユーザに提示することができる．

## 課題と提案内容

　SNSに投稿される大量の楽曲データをジャンル推定するとき，1台のみのサーバで行うと処理速度に限界が生じることが課題としてあげられる．また，上記したような動画コンテンツと楽曲コンテンツのマッチングをするサービスの場合，更なる負荷が掛かると予測されるため．負荷分散を利用することで処理速度を向上することでサービスの向上につなげる．

本研究ではSNSに投稿される大量の楽曲データをラウンドロビン方式の負荷分散を導入したWeb楽曲サービスを実装する．

本研究の実験としては，1000件の楽曲データを用意し，その楽曲データを1台のRaspberry Piで楽曲推定処理を行い，その処理速度を計測する．その後4台のRaspberry Piで同処理を行い，その処理速度とRaspberry Pi 1台での処理速度との比較をし，評価する．

## 本論文の構成

本論文では全6章で構成されている．本章では研究の背景や目的，研究課題について述べる．第2章では関連研究について述べる．第3章では本研究の提案方式とその特徴である本システムの内容，第4章では提案方式を構築した本システムの設計及び実験システムの実装について述べる．第5章では本システムを利用して処理時間に着目した実験方法とその評価について述べる．第6章では謝辞および今後の展望を述べる

# 関連研究

本章では関連研究について述べる．

## 動画の印象評価データセット構築とその特性の調査

音楽に対する印象評価に関する研究は多数なされている中で大野直樹，中村聡史，山本岳洋，後藤真孝らはそうした研究を促進するためのデータセットも様々なものが構築されている一方で音楽と映像が同期して提示される音楽動画を対象とした印象評価に関する研究は、ほとんどなされていないという．この研究では500曲の音楽動画のサビ区間を対象とし，音楽のみ，動画のみ，音楽と映像の見合わせという3つのタイプの評価対象コンテンツを用意する．また，このコンテンツに対して8軸の印象評価を行ってもらうことで，メディアの及ぼす影響を明らかにする．さらに，これまで大野らが行ってきた，音楽動画全体に対する印象評価と，この研究で収集した音楽動画のさび区間に対する印象評価とを比較することで印象評価において注意すべき点についての考察を行った．

## 動画特徴量からの印象推定に基づく動画BGMの自動生成

　現在ではデジタルカメラやスマートフォンの普及により，写真や動画を撮影する機会が増え，またその撮影したものをSNSサイトに投稿することで，多くの人々と共有して楽しむようになっている．その際に，ただ撮影したものを投稿するのではなく、撮影映像にBGMを付与するなどの動画編集も行うようになってきている．しかし動画編集は一般的に，動画に合った音楽を自分で探したり、動画の長さに合うように音楽を調整したり，といった手間とスキルが必要になる．そこで清水，菅野，伊藤，嵯峨山，高塚らは，動画の印象に合った楽曲を自動付与することを目標として，動画特徴量からの印象推定結果に基づいた楽曲生成手法を提案している．この報告ではその改良についていくつかの技術要素を述べる．具体的には動画特徴量の抽出，印象評価のためのユーザインターフェースの回路湯，印象値の回帰手法の再考とメロディの音色選択，の核技術要素について述べた．

## ニューラルネットワークを用いた音楽の自動ジャンル分類

　スタンダードMIDIファイル形式の楽曲に対してその部分情報からその楽曲のジャンル推定をし，特徴をフィードバックするジャンル学習支援システムを構築することを最終目的とし．現在までに，4ジャンル計120曲をニューラルネットワークに学習させ，その学習させた局のジャンルを同定させることができている．さらに，評価実験として被験者にジャンル推定を行わせ，学習させたニューラルネットワークとの，推定率の比較を行い，本システムのほうが高い推定率を得たとしている．続いて学習させたニューラルネットワークに対して中間層の各ノードと連結しているリンクの重みを見比べることにより，専門家による意味解釈を行った．その結果，5つの中間層に対して，因子を抽出できた，これにより教育利用のためのジャンル学習支援システムの構築の可能性が示されたとしている．

# 提案システム

本章では本研究の提案方式とその特徴である本システムの内容について述べる．

## 提案システムの概要

本研究では，SNSなどに投稿される大量の楽曲を複数のサーバで並列に楽曲分類のための深層学習モデルを利用し楽曲のジャンル分類を行い，その楽曲と動画をマッチングするシステムを提案する．

本提案システムではジャンル分類に機械学習を利用するため，システムの処理に時間がかかる．加えてSNSなどに投稿される大量のデータを処理するため，なおさら時間がかかることが予測される．このような処理を1つのサーバで処理を行おうとすると処理時間が長くなってしまう．そこで処理するサーバを複数にし，並列処理でシステムの処理を行うこと処理速度を向上することができる．これによりシステム全体の処理時間を減らすことが可能である．

## 提案システムの実行手順

1. 分散処理するためのサーバを数台用意する
2. 用意したサーバでstep2~step6を処理
3. 楽曲SNSに投稿された楽曲を保存
4. 保存した楽曲の周波数を分析
5. 分析した周波数をスペクトログラムにして可視化
6. スペクトログラムを機械学習し楽曲ジャンルを推定

# 実装システム

本章では提案方式を構築した本システムの設計及び実験システムの実装について述べる

## 実装システムの概要

実装システムでは楽曲のジャンル推定処理を4台のRaspberry piに分散処理することで処理の負荷分散を実現している．

実装システムで分散処理に使用したサーバはRaspberry Pi 4B 4GBを4台，負荷分散機としてLet’sNote1台を使用する．また本システムのメインプログラムはプログラミング言語Pythonで記述している．

## 実装システムの処理手順

　実装システムの処理手順は下記に示す．

1. 楽曲の周波数を分析する
2. 分析した周波数を画像として可視化する
3. FMA\_SMALLデータセットを用いてジャンル推定モデルの作成
4. ジャンル推定モデルとstep2の画像を用いてジャンル推定
5. step1からstep4の処理をRaspberry Pi4台で分散処理

## 環境構築

提案システムのプロトタイプを実装した開発環境を表4.1に示す．提案システムを作成するために用いたパッケージを表4.2に示す．

表 4.1システム開発環境

|  |  |  |
| --- | --- | --- |
| 名称 | バージョン | 利用用途 |
| Windows10 | 20H2 | OS |
| Raspberry Pi | 4B 4GB | 分散処理に使用 |
| Raspberry Pi OS |  | Raspberry Pi のOS |
| Python | 3.7.3 | 開発言語 |
| FFmpeg | 4.4 | 楽曲を再生・変換する |
| Nginx |  | Webサーバ |
| Apache |  | Webサーバ |

表 4.2　システム構築に用いたパッケージ

|  |  |  |
| --- | --- | --- |
| ライブラリ名 | バージョン | 備考 |
| pydub | 0.25.1 | 音楽ファイルの読み込みに使用 |
| pandas | 1.3.4 | データ解析を容易にする |
| numpy | 1.21.3 | 機械学習の計算を早く効率的に行えるようにする |
| matplotlib | 3.4.3 | Pythonでグラフを描画する |
| seaborn | 0.11.2 | matplotlibの内部で動く．少ないコードで洗礼された図を描く |
| llvmlite | 0.32.1 | 後述するnumbaとlibrosaのインストールに必要 |
| numba | 0.49.0 | Pythonの関数を高速にするライブラリ |
| LibROSA | 0.8.1 | 音楽と音声の解析のためのPythonパッケージ．スペクトル解析，テンポの分析，画像出力など音楽の分析に必要な機能があらかじめ実装されている． |
| TensorFlow | 2.1.0 | Googleが開発した機械学習のソフトウェアライブラリ |
| keras | 2.3.1 | PythonでかかれたTensorFlow上で実行可能な高水準のニューラルネットワークライブラリ |

実装システムで利用したPythonパッケージのインストール方法を表4.3に示す

表 4.3　Pythonパッケージのインストール

|  |
| --- |
| sudo apt-get update  sudo apt-get upgrade  pip3 install pandas==1.3.4  pip3 install numpy==1.21.3  pip3 install matplotlib==3.4.3  pip3 install seaborn==0.11.2  sudo apt install llvm-7  LLVM\_CONFIG=/usr/bin/llvm-config-7  pip3 install llvmlite==0.31.0  pip3 install numba==0.48.0  pip3 install colorama==0.3.9  pip3 install librosa==0.6.3  pip3 install keras==2.3.1  pip3install <https://github.com/lhelontra/tensorflow-on-arm/releases/download/v2.1.0/tensorflow-2.1.0-cp37-none-linux_armv7l.whl>  pip3 install h5py==2.10.0 |

表 4.4FFmpegインストール

|  |
| --- |
| git clone git://source.ffmpeg.org/ffmpeg.git  wget <ftp://ftp.alsa-project.org/pub/lib/alsa-lib-1.1.6.tar.bz2>  tar xjvf alsa-lib-1.1.6.tar.bz2  cd alsa-lib-1.1.6  ./configure --prefix=/home/pi/ffmpeg  make  make install  cd /home/pi/ffmpeg  ./configure --enable-gpl --enable-nonfree --enable-mmal --enable-omx-rpi --enable-omx --extra-cflags="- I/home/pi/ffmpeg/include" --extra-ldflags="-L/home/pi/ffmpeg/lib" --extra-libs=-ldl |

## 実装システムの構築

　実装システムの構築手順を4.4.1からで説明する

### numpyを利用した楽曲分析

　機械学習を行うために楽曲の周波数を分析しスペクトログラムとして可視化する必要がある．スペクトログラムとは図4.1のような縦軸に周波数を，横軸に時間をとり色によってどの程度の周波数が多く出ているか表したものである．図4.1のスペクトログラムで示された楽曲では100Hz～900Hzの音が多く使われていることがわかる．

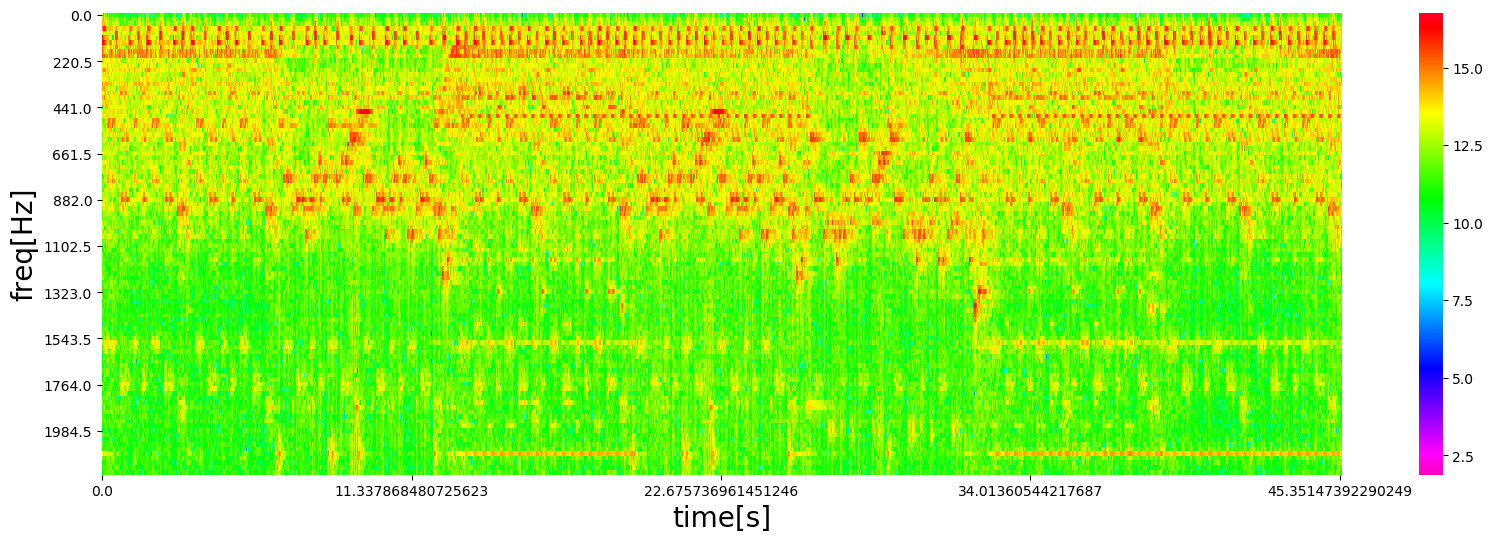


図 4.1スペクトログラム

本項では楽曲をスペクトログラムとして出力するプログラムheatmap.pyについて説明する．

まずは使用するPythonパッケージのインポートをする．

表 4.5パッケージのインポート

|  |
| --- |
| from pydub import AudioSegment  import os  import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  import seaborn as sns |

楽曲を分析するには楽曲を読み込む必要がある楽曲を読み込むプログラムは表4.4に示す．

表 4.6楽曲の読み込み

|  |
| --- |
| sound = AudioSegment.from\_file(‘filename’) |

この時filenameは解析したい楽曲のファイルの名前を入れる．

読み込んだ楽曲をnumpyで読み込みステレオ音声の片側の音声を抽出する。

表 4.7　numpyでステレオ音声の抽出

|  |
| --- |
| samples = np.array(sound.get\_array\_of\_samples())  sample = samples[::sound.channels] |

表 4.8フーリエ変換

|  |
| --- |
| spec = np.fft.fft(sample) |

表 4.9スペクトログラムのラベル表示

|  |
| --- |
| w = 2000　#窓幅  s = 500　#刻み |

表 4.10スペクトログラムの計算

|  |
| --- |
| #スペクトル格納用  ampList = []  #刻みずつずらしながら窓幅分のデータをフーリエ変換する  for i in range(int((sample.shape[0]- w) / s)):  data = sample[i\*s:i\*s+w]  spec = np.fft.fft(data)  spec = spec[:int(spec.shape[0]/2)]  spec[0] = spec[0] / 2  ampList.append(np.abs(spec)) |

これで楽曲に対する処理を終えたので，matplotlibとseabornを利用して実際にスペクトログラムとして可視化する．

表 4.11スペクトログラムとして可視化

|  |
| --- |
| df\_amp = pd.DataFrame(data=ampList, index=time, columns=freq)  a = plt.figure(figsize=(20, 6))  l = sns.heatmap(data=np.log(df\_amp.iloc[:, :100].T),  xticklabels=1000,  yticklabels=10,  cmap=plt.cm.gist\_rainbow\_r,  )  l.set\_ylabel("freq[Hz]",fontsize = 20)  l.set\_xlabel("time[s]",fontsize = 20)  a.savefig("%s\/%s.png"%(path, name),bbox\_inches="tight") |

これでスペクトログラムとして楽曲を可視化することができる．

### kerasを利用してジャンル推定モデルの作成と楽曲ジャンル推定

本項ではジャンル推定プログラムの作成をする．

このジャンル推定プログラムをpredict.pyとしプログラムの内容を表4.12から表20に示す．

ジャンル推定モデルの作成ではFMA\_SMALLデータセットを用いる．またジャンル推定におけるジャンルの種類はElectric，Folk，HipHop，International，Latin，Metal，Noise，Pop，Rock，Punkの10ジャンルをとする．

表 4.12使用したパッケージ

|  |
| --- |
| import librosa  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  import os  from PIL import Image  import pathlib  import csv  import time  # Preprocessing  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import LabelEncoder, StandardScaler  #Keras  import keras  from keras.models import Sequential,load\_model  from keras import models  from keras import layers  import warnings  warnings.filterwarnings('ignore') |

モデル作成に使用するデータセットをそれぞれのジャンルフォルダに格納した後，格納した楽曲から特徴を抽出しスペクトログラムに変換し，スペクトログラムにしたものをimg\_dateフォルダの各ジャンルフォルダに保存

表 4.13

|  |
| --- |
| cmap = plt.get\_cmap('inferno')  plt.figure(figsize=(10,10))  genres = 'Electric Folk HipHop International Latin Metal Noise Pop Rock Punk'.split()  for g in genres:  pathlib.Path(f'img\_data/{g}').mkdir(parents=True, exist\_ok=True)  for filename in os.listdir(f'{g}'):  songname = f'{g}/{filename}'  y, sr = librosa.load(songname, mono=True, duration=5)  plt.specgram(y, NFFT=2048, Fs=2, Fc=0, noverlap=128, cmap=cmap, sides='default', mode='default', scale='dB');  plt.axis('off');  plt.savefig(f'img\_data/{g}/{filename[:-3].replace(".", "")}.png')  plt.clf() |

楽曲からメル周波数ケプストラム係数(MFCC)，Spectral Centroid，，ZeroCrossingRate，Spectral Roll-offといった特徴を抽出し，抽出したものをcsvファイルとして保存．

表 4.14特徴を抽出したものをcsvに保存

|  |
| --- |
| header = 'filename chroma\_stft spectral\_centroid spectral\_bandwidth rolloff zero\_crossing\_rate'  for i in range(1, 21):  header += f' mfcc{i}'  header += ' label'  header = header.split()  file = open('data.csv', 'w', newline='')  with file:  writer = csv.writer(file)  writer.writerow(header)  genres = 'Electric Folk HipHop International Latin Metal Noise Pop Rock Punk'.split()  for g in genres:  for filename in os.listdir(f'{g}'):  songname = f'{g}/{filename}'  y, sr = librosa.load(songname, mono=True, duration=30)  chroma\_stft = librosa.feature.chroma\_stft(y=y, sr=sr)  spec\_cent = librosa.feature.spectral\_centroid(y=y, sr=sr)  spec\_bw = librosa.feature.spectral\_bandwidth(y=y, sr=sr)  rolloff = librosa.feature.spectral\_rolloff(y=y, sr=sr)  zcr = librosa.feature.zero\_crossing\_rate(y)  mfcc = librosa.feature.mfcc(y=y, sr=sr)  to\_append = f'{filename} {np.mean(chroma\_stft)} {np.mean(spec\_cent)} {np.mean(spec\_bw)} {np.mean(rolloff)} {np.mean(zcr)}'  for e in mfcc:  to\_append += f' {np.mean(e)}'  to\_append += f' {g}'  file = open('data.csv', 'a', newline='')  with file:  writer = csv.writer(file)  writer.writerow(to\_append.split()) |

表 4.15Pandasでデータを分析

|  |
| --- |
| data **=** pd**.**read\_csv('data.csv')  data**.**head()　#上から5件を表示 |

表 4.16不必要なコラムを削除

|  |
| --- |
| data **=** data**.**drop(['filename'],axis**=**1) |

表 4.17ラベルをエンコード

|  |
| --- |
| genre\_list **=** data**.**iloc[:, **-**1]  encoder **=** LabelEncoder()  y **=** encoder**.**fit\_transform(genre\_list)  scaler **=** StandardScaler()  X **=** scaler**.**fit\_transform(np**.**array(data**.**iloc[:, :**-**1], dtype **=** float) |

表 4.18トレーニングデータセットとテストデータセットに分ける

|  |
| --- |
| X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2) |

表 4.19kerasを利用したジャンル分類モデルの構築

|  |
| --- |
| model **=** models**.**Sequential()  model**.**add(layers**.**Dense(256, activation**=**'relu', input\_shape**=**(X\_train**.**shape[1],)))  model**.**add(layers**.**Dense(128, activation**=**'relu'))  model**.**add(layers**.**Dense(64, activation**=**'relu'))  model**.**add(layers**.**Dense(10, activation**=**'softmax'))  model**.**compile(optimizer**=**'adam',  loss**=**'sparse\_categorical\_crossentropy',  metrics**=**['accuracy']  history **=** model**.**fit(X\_train,  y\_train,  epochs**=**20,  batch\_size**=**128)  test\_loss, test\_acc **=** model**.**evaluate(X\_test,y\_test)  print('test\_acc: ',test\_acc) |

以上でモデルの作成ができた．ここからジャンル推定のプログラムを表4.20に示す．

表 4.20ジャンル推定

|  |
| --- |
| predictions = model.predict(X\_test)  predictions[0].shape  np.sum(predictions[0])  prediction = np.argmax(predictions[0])  print(genres[prediction]) |

これでジャンル推定することができた．

## 分散処理の実装

本節では4節で行ったジャンル推定処理を複数のサーバで処理を行う方法を書く．

本研究で分散処理に使用した機器を表4.21に示す．

表 4.21使用機器

|  |  |  |
| --- | --- | --- |
| 機器名 | 台数 | 使用用途 |
| Let’s Note | 1 | ロードバランサ機 |
| Raspberry Pi 4B 4GB | 4 | 分散処理サーバ |

　Let’s NoteにはWebサーバとしてNginx，Raspberry PiにはそれぞれにWebサーバとしてApacheを導入した．それぞれの導入方法を下記に示す．

### Let’s NoteにNginxを導入

1. <http://nginx.org/en/download.htmlからそれぞれのOS>に対応したものをダウンロード．
2. ダウンロードしたものを解凍する．
3. 解凍したものを利便上nginxとファイル名を変更しCドライブ直下に配置する．
4. 解凍したディレクトリでコマンドプロンプトを起動する
5. start nginxでnginxを起動する．
6. <http://localhost>にアクセスできれば完了．

### Raspberry PiにApacheを導入

1. sudo apt-get updateでパッケージリストの更新．
2. sudo apt-get upgradeでパッケージの更新．
3. sudo apt install apache2でapache2をインストール．
4. ブラウザを開きlocalホストにアクセスできるのかを確認．
5. 別端末ブラウザにてRaspberry PiのIPアドレスにアクセスできれば完了．

### 分散処理システムの導入

まず分散するRaspberry Pi4台のIPアドレスを取得する．

IPアドレスの取得はRaspberry Piのターミナルにてifconfigコマンドで確認できる．

今回では以下に表記する４つを使用する．

表 4.22分散処理サーバのIPアドレス

|  |  |
| --- | --- |
| 機器名 | IPアドレス |
| Raspberry Pi1 | 192.168.0.100 |
| Raspberry Pi2 | 192.168.5.2 |
| Raspberry Pi3 | 192.168.5.3 |
| Raspberry Pi4 | 192.168.5.4 |

その後Raspberry Piのホームディレクトリにcgi-binを作成．cgi-binの作成は以下のコマンド．

表 4.23　ディレクトリcgi-binの作成

|  |
| --- |
| mkdir cgi-bin |

この作成したcgi-binに4.4.2で作成したpredict.pyを配置しファイルに権限を持たせる．

ファイルに権限を持たせるコマンドは表4.24に示す．

表 4.24 predict.pyに権限を付与

|  |
| --- |
| chmod 755 predict.py |

これで権限を付与することができた．

そしたらホームに戻りWebサーバを立ち上げる．今回ではpython組込み式のWebサーバを起動する．python組込みwebサーバを起動するコマンドは表2.25に示す．

表 4.25 分散処理Webサーバ起動

|  |
| --- |
| python3 -m http.server --cgi 8000 |

これを4台それぞれで行う．

Webサーバを起動することができた．Nginxのconfigファイルを変更する．Nginxのconfigファイルはnginxをcドライブ直下に配置したのであれば，C:\nginx\confにあるnginx.confである．

それではnginx.confの設定を書き換えていく．まずnginx.confをテキストエディタで開く．そのnginx.conf内のhttp関数内に設定を追加していく．追加するものを表4.26に示す．

表 4.26 nginx.conf

|  |
| --- |
| http { 　　　　　　　　　　　　　　　　　　　　 //ここに追加  include mime.types;  default\_type application/octet-stream;  fastcgi\_read\_timeout 999999;  proxy\_read\_timeout 999999;  sendfile on;  keepalive\_timeout 65;  upstream backend1{  server 192.168.0.100:8000;  server 192.168.5.2:8000;  server 192.168.5.3:8000;  server 192.168.5.4:8000;  }  server {  listen 80;  server\_name localhost;  location / {  proxy\_pass http://backend1;  }  error\_page 500 502 503 504 /50x.html;  location = /50x.html {  root html;  }  }  } |

これによりnginxの設定が完了した．

設定が完了したらロードバランサ機でブラウザを立ち上げ．以下のURLにアクセスする．

<http://localhost/cgi-bin/predict.py>

このURLで4つのサーバのうちどれか1つにアクセスするため何度かブラウザをリロードし，ジャンル推定プログラムが動作していることが確認出来たら，分散処理システムの導入は完了である．

# 実験

本章では本システムを利用して処理時間に着目した実験方法とその評価について述べる．

## 実験目的

実験では4章にて実装した負荷分散システムを利用して．1000件の楽曲データを用いてRaspberry Pi 4台で分散処理を行うことで1台での処理速度との比較を行う。

## 実験環境の構築

　実験ではロードバランサ機としてLet’s Note1台を使用し，分散処理サーバとしてRaspberry Pi4B 4GBを4台使用する．

ロードバランサ機にはWebサーバにNginxを使用し，分散処理サーバのRaspberry PiにはApacheを使用する．

表 5.1実験使用機器

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 台数 | OS | Webサーバ |
| Let’s Note | 1 | Windows10 | Nginx |
| Raspberry Pi | 4 | Raspbian | Apache |

## 実験方法（予定）

　まず1000件の楽曲データを用意し，１台のRaspberry Piでのジャンル推定プログラムの処理時間を計る．その後4台のRaspberry Pi同じ処理を行い処理時間の比較する．



# むすび