小组成员分工

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 乐谱录入 | 生成WAV | 解析WAV | 离散余弦变换 | 支持向量机 |
| 孙新昊 | √ |  | √ |  | √ |
| 曾浩 | √ | √ |  | √ |  |
| 朱锐东 | √ |  |  |  |  |

音乐生成部分

1. MuseScore生成MIDI

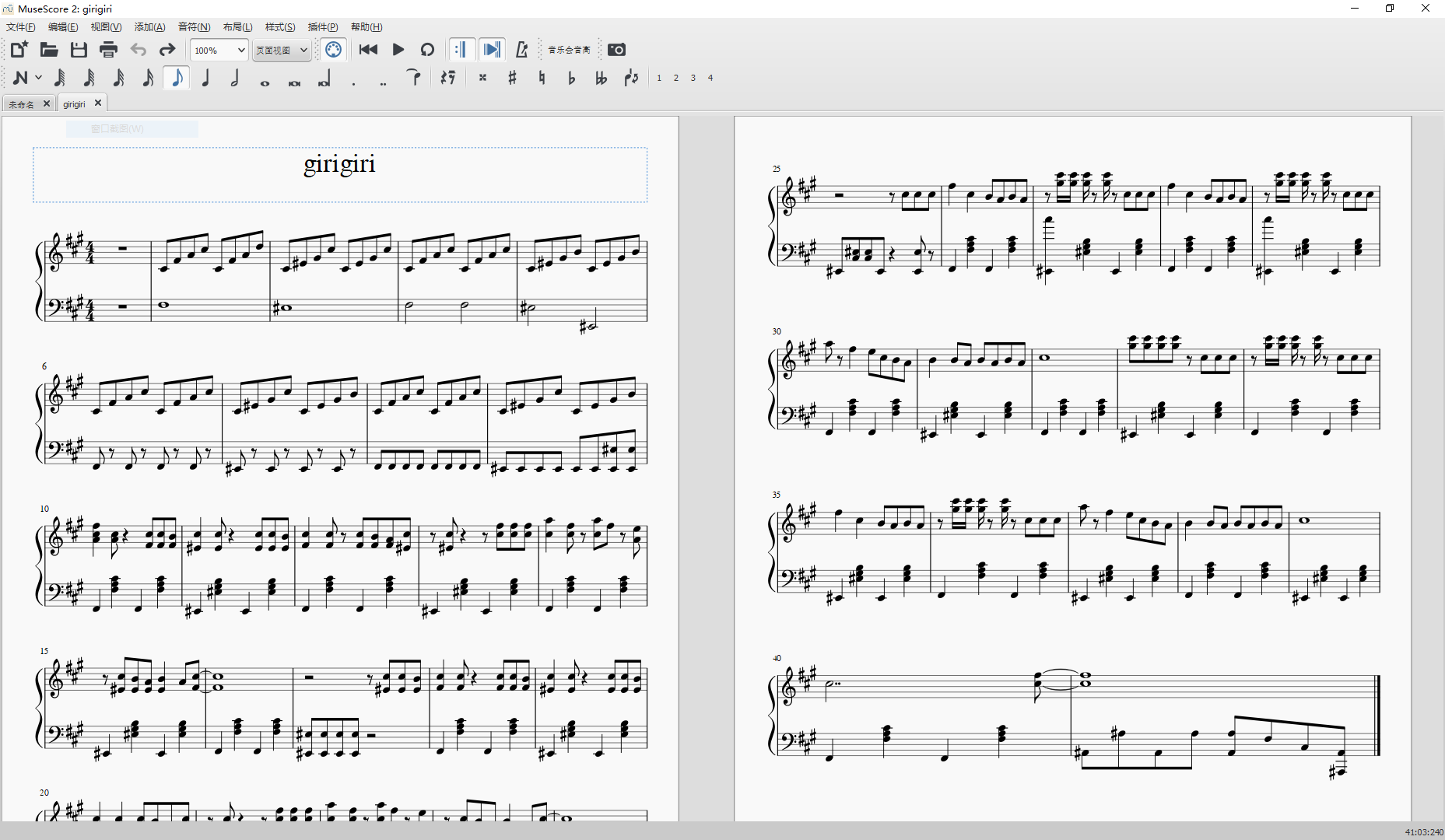
MuseScore缪斯乐谱是一个乐谱编辑器可以手动输入乐谱并导出MIDI文件。

图1在MuseScore中手动输入girigiri爱

2. 使用TiMidity++生成WAV

导出MIDI文件后，使用TiMidity++可以将MIDI转为WAV

脚本为generate-wave.sh

音乐分类部分

1.解析WAV

使用python的wave package

wave带有解析wav的功能，可以获得wav文件的帧率和帧数，并提供一个指针，可以通过指针来访问wav文件的具体内容。wav格式为一个记录了每个时间点的声音波形的值，左右声道交错存储。通过对wav中波形切分时间片并作离散余弦变换可以获得其频率分布。

2. 离散余弦变换

3.支持向量机

支持向量机Support Vector Machines简称为SVM，其基本原理为对于两组高位空间中的数据点，找到一个超平面，使得两组数据分别处于超平面的两侧，并且到超平面的距离尽可能远，错误的数据点尽可能少。如果直接对数据使用SVM，则要求数据集本身是线性可分的，对于非线性的数据集，可以通过核方法升维后变得线性可分。核方法通过定义一个核函数(kernel)从而将低维空间的数据点映射到高维空间。

举一个简单的例子，假设数据集在二维空间上关于一个圆分为圆内和圆外两部分，此数据集在二维空间上不是线性可分的，因为不存在一条直线能够划出圆的效果，但是其在三维空间是可分的，可以通过一个函数将圆映射为一个半球面，此时圆面所在的平面就在三维空间中分割了数据集。

python工具包sklearn中已经有svm模型了，可以直接使用。

直接选用默认的svm.SVC，然后输入标注好的训练集和训练集标签，让SVM拟合训练集数据。然后在测试集上测试。

其中SVM输入数据为输入的音乐片段进行离散余弦变换后再1200个音分上的直方图分布，为一个1200维浮点向量。

其中数据集由字母歌，girigiri爱，歌唱祖国，燕园情四首歌构成

每首歌均有三个律制，被且为五份，3份训练，2份测试。

成功率20/24

代码

<https://github.com/CyanoHao/Musical-Temperament-Identification>

from sklearn import svm

import math

import collections

import wave

import numpy as np

import math

import matplotlib.pyplot as plt

def ratio\_to\_cent(num):

    decpart = num - math.floor(num)

    cent = int(round(decpart \* 1200))

    if cent == 1200:

        cent = 0

    return cent

def read\_wave(filename, start = 0, end = 1):

    wavefile = wave.open(filename, 'r') # open for reading

    #读取wav文件的四种信息的函数。期中numframes表示一共读取了几个frames。

    nchannels = wavefile.getnchannels()

    sample\_width = wavefile.getsampwidth()

    framerate = wavefile.getframerate()

    numframes = wavefile.getnframes()

    bytes\_per\_frame = nchannels \* sample\_width

    # print("channel",nchannels)

    # print("sample\_width",sample\_width)

    # print("framerate",framerate)

    # print("numframes",numframes)

    # 每个 block 3 秒钟，频谱分辨率为 1/3 Hz

    frames\_per\_block = int(framerate \* 3)

    blocks = int(numframes / frames\_per\_block)

    freqs = np.fft.fftfreq(frames\_per\_block)

    spectum = list(0 for \_ in range(frames\_per\_block))

    spectum = np.array(freqs)

    # 对每个 block 做 FFT，将所得频谱叠加

    for \_ in range(int(blocks \* start), int(blocks \* end)):

        amps = list(0 for \_ in range(frames\_per\_block))

        buffer = wavefile.readframes(frames\_per\_block)

        for j in range(frames\_per\_block):

            ch0 = buffer[bytes\_per\_frame \* j:bytes\_per\_frame \* j + sample\_width]

            amps[j] = int.from\_bytes(ch0, byteorder = 'little', signed = True)

        data = np.array(amps)

        w = np.fft.fft(data)

        spectum += np.abs(w)

        # # Find the peak in the coefficients

        # idx = np.argmax(np.abs(w))

        # freq = freqs[idx]

        # freq\_in\_hertz = abs(freq \* framerate)

        # print(freq\_in\_hertz)

    cent\_hist = list(float(0) for \_ in range(1200))

    # 取 110 Hz 到 1760 Hz 转换成音分（A4 上下各 2 个八度）

    for i in range(frames\_per\_block):

        freq\_in\_hertz = freqs[i] \* framerate

        if freq\_in\_hertz >= 110 and freq\_in\_hertz <= 1760:

            cent = ratio\_to\_cent(freq\_in\_hertz / 440)

            cent\_hist[cent] += spectum[i]

    # normalize cent\_hist

    cent\_hist\_total = float(0)

    for i in range(len(cent\_hist)):

        cent\_hist\_total += cent\_hist[i]

    for i in range(len(cent\_hist)):

        cent\_hist[i] = cent\_hist[i]/cent\_hist\_total

    '''

    fig, ax = plt.subplots()

    ax.plot(list(range(1200)), cent\_hist)

    plt.show()

    print(cent\_hist\_total)

    '''

    return cent\_hist

#训练集文件名和测试集文件名

filename\_list = ['girigiri-十二平均律.wav', 'girigiri-五度相生律.wav', 'girigiri-纯律.wav', '歌唱祖国-十二平均律.wav', '歌唱祖国-五度相生律.wav', '歌唱祖国-纯律.wav', 'alphabet-十二平均律.wav', 'alphabet-五度相生律.wav', 'alphabet-纯律.wav','燕园情-十二平均律.wav', '燕园情-五度相生律.wav', '燕园情-纯律.wav']

filename\_list\_testing = ['girigiri-十二平均律.wav', 'girigiri-五度相生律.wav', 'girigiri-纯律.wav', '歌唱祖国-十二平均律.wav', '歌唱祖国-五度相生律.wav', '歌唱祖国-纯律.wav', 'alphabet-十二平均律.wav', 'alphabet-五度相生律.wav', 'alphabet-纯律.wav','燕园情-十二平均律.wav', '燕园情-五度相生律.wav', '燕园情-纯律.wav']

#分段个数

training\_segment = 3

testing\_segment = 2

segment = training\_segment + testing\_segment

training\_data = [0] \* len(filename\_list) \* training\_segment

training\_label = [0] \* len(filename\_list) \* training\_segment

for i in range(len(filename\_list)):

    for j in range(training\_segment):

        training\_data[i \* training\_segment + j] = read\_wave(filename\_list[i], float(j)/float(segment), float(j + 1)/float(segment))

        training\_label[i \* training\_segment + j] = i % 3

testing\_data = [0] \* len(filename\_list\_testing) \* testing\_segment

testing\_label = [0] \* len(filename\_list\_testing) \* testing\_segment

for i in range(len(filename\_list\_testing)):

    for j in range(testing\_segment):

        testing\_data[i \* testing\_segment + j] = read\_wave(filename\_list\_testing[i], float(training\_segment + j)/float(segment), float(training\_segment + j + 1)/float(segment))

        testing\_label[i \* testing\_segment + j] = i % 3

clf = svm.SVC(gamma='scale')

clf.fit(training\_data, training\_label)

correct\_count = 0

predict = clf.predict(testing\_data)

print (predict)

for i in range(len(filename\_list\_testing) \* 2):

    if(predict[i] == testing\_label[i]):

        print('correct')

        correct\_count += 1

print('correct count',correct\_count)