MIR HW Report

106062122 丁憶涵

Overview

整份作業分成了三個 task 五個 question，但我只將程式碼分成了兩個檔案，Q1~Q4 的內容都寫在 example_q1.py 中，而 Q5 則是在 example_q5.py 中。把 Q1~Q4 的內容都放在一起主要是因為方便修改，由於我的實作方法是把內容包成一個個的 function，不同 question 只需要 call 所需的 function 就好，為避免在更改時不小心少更新哪一個檔案中的內容，或是版本亂掉，因此乾脆將所有都放在一起，同樣的東西只會出現一次，看起來也更清楚乾淨。
在這份 report 中，我會先討論每個 question 的執行結果，再簡單敘述我的實作方法與過程。

TASK I

Q1

- Result discussion

```
GTZAN dataset
Task 1
***** Q1 *****
Genre           accuracy
blues              7.14%
country           32.32%
disco             31.63%
hiphop            13.58%
jazz              16.46%
metal             24.73%
pop               41.49%
reggae            32.99%
rock              34.69%
----------
Overall accuracy:       26.52%
```

在 GTZAN 這個 dataset 中，以 pop 類型的音樂 accuracy 最高，大概是因為我們的 key finding 根據的分類規則，是從以前到現在讓人覺得聽起來最容易喜歡或是接受的音樂表現方式，也最符合 pop music 的特色，在一瞬間吸引人的目光，讓人深刻記住且容易傳唱。而 blues 則是其中 accuracy 最低的，因為 blues 最一開始的出現也是最沒有特定音樂結構的音樂類型。blues 是從在工地中的黑人工作時隨意發出的吼聲、歡呼聲、哼唱等等綜合後慢慢演變而來，由於一開始只是個人表演因此都是自由發揮，並沒有特定的音樂結構，也沒有特定的規範，直到後來才漸漸有「12 小節藍調」的出現，但也只是告訴樂手大概的架構，並沒有詳細規範其中的內容，可能隨著演唱者的心情修改歌詞與旋律，旋律也不會遵照常見的 12 大調/小調，因此當我們用最常見的 12 大小調的分類方式進行偵測時，

準確率就會大大降低了。

```
GiantSteps dataset
Task 1
***** Q1 *****
----------
Overall accuracy:       22.35%
```

GTZAN dataset 中的 overall accuracy 是 26.52%，而 GiantSteps dataset 則只有 22.35%，由於我 Find Key 的方式是一樣的，因此我想或許是因為在 GiantSteps 中的音檔有比較多的電子音效，或是節奏感較強旋律感較弱的 Music piece，因此在 Find Key 的時候會比較無法準確抓到，因此 accuracy 較低。

- Implement

由於最後將 Q1~Q4 的功能都寫在一起，因此名為 Q1 的 function 就新增了許多不同的參數傳入。

```
#%% main
print("GTZAN dataset")
print("Task 1")
print("***** Q1 *****")
Q1(DB , 0 , FILES , False , False , False , False)
```

我在最下方命名了一個新的 cell 叫 main，功能就像在 c code 中的 main function 一樣，將所有的 call function 內容都放在這裡，感覺看起來比較整潔乾淨，也容易對照。

```
def Q1(DB , gamma , FILES , Q3 , Q4 , Q4_cqt , Q4_cens):
```

這是上面的 Q1 function 的參數對照。

DB 是目前的 dataset；gamma 則是在 Q2 中改變的 γ 值；FILES 是目前 dataset 中的所有檔案，會在進入 main 之前將該 dataset 中所有 FILE 儲存完畢，也會在需要用到該 dataset 的所有內容都執行完後再進行下一個 dataset 的內容，避免重複讀取太多次；Q3 是個 bool character，True 代表計算該次的 accuracy 時需要考慮 relative、fifth 和 parallel；Q4 也是一個 bool character，若 Q4 == True，代表這次 find key 時要使用 Krumhansl Schumuckler’s method；Q4_cqt 和 Q4_cens 同樣是 bool character，是要用於比較使用 chroma_stft、chroma_cens 和 chroma_cqt 會有什麼不同結果，因此若是 Q4_cqt == True 代表這次使用的是 chroma_cqt 得到的結果。

其他的程式碼都是打在 Q1 function 中。

```
if (DB == "GTZAN"):
    content = utils.read_keyfile(f,'*.lerch.txt')
    if (int(content)<0): continue # skip saving if key not found
    gen = f.split('/')[2]
    label[gen].append(utils.LABEL[int(content)])
    gens.append(gen)
elif (DB == "GiantSteps"):
    content = utils.read_keyfile(f,'*.key')
    content = utils.generalize_key(content)
    content = utils.str_to_lerch(content)
    if (int(content)<0): continue # skip saving if key not found
    label.append(utils.LABEL[content])
else:
    content = utils.read_keyfile(f,'*.lerch.txt')
    if (int(content)<0): continue # skip saving if key not found
    label.append(utils.LABEL[content])
```

因為觀察到 GiantSteps 中的 key file 都是.key 檔，因此我就新增了 elif 確定現在的是否是 GiantSteps dataset，並保留了原本的 else。

```
if(gamma == 0):
    if(Q4_cqt == True):
        cxx = chroma_cqt(y=y , sr=sr)
    elif(Q4_cens == True):
        cxx = chroma_cens(y=y , sr=sr)
    else:
        cxx = chroma_stft(y=y , sr=sr)
else:
    if(Q4_cqt == True):
        cxx = np.log10(1 + gamma * np.abs(chroma_cqt(y=y, sr=sr)))
    elif(Q4_cens == True):
        cxx = np.log10(1 + gamma * np.abs(chroma_cens(y=y, sr=sr)))
    else:
        cxx = np.log10(1 + gamma * np.abs(chroma_stft(y=y, sr=sr)))
chromagram.append(cxx)
```

cxx 的取的也多了很多判斷，讓我能夠用一個檔案實作所有內容，細節前面有提過了，因此就跳過。

```
chroma_vector = np.sum(cxx , axis=1)
key_ind = np.max(chroma_vector)
count =0
for i in chroma_vector:
    if(i == key_ind):
        key_ind = count%12
        break
    count += 1
chroma_vector = utils.rotate(chroma_vector.tolist(), 12-key_ind)
if(Q4 == True):
    p_major = pearsonr(chroma_vector , utils.KS["major"])
    p_minor = pearsonr(chroma_vector , utils.KS["minor"])
else:
    p_major = pearsonr(chroma_vector , utils.MODE["major"])
    p_minor = pearsonr(chroma_vector , utils.MODE["minor"])
if(p_major[0] > p_minor[0]):
    key_ind = (key_ind+3)%12
else:
    key_ind = (key_ind+3)%12+12
if DB=='GTZAN':
    pred[gen].append(utils.lerch_to_str(key_ind))
else:
    pred.append(utils.lerch_to_str(key_ind)) # you may ignore this when starting with GTZAN dataset
```

chroma_vector 是將 chromagram 中的內容每一列相加，讓他變成一個長度為 12 的 array，對應到每一個 key，用 rotate()將 tonic 移到 vector 的開頭位置，這樣和 MODE 進行比較的時候才會是正確的順序，pearsonr 則是確定兩個陣列的相似度，因此和 MODE[“major”]相似度較大的話就是 Major，又因為 chroma_vector 和 key_ind 在計算時是以’C’當開始音，和 annotation 要求的不相符，因此在存入 pred 前須將整個數值往後加三位，才能對應到正確的 annotation。

```
if DB=='GTZAN':
    label_list, pred_list = list(), list()
    print("Genre    \taccuracy")
    for g in GENRE:
        correct = 0
        fifth = 0
        relative = 0
        parallel = 0
        for acc_n in range(len(label[g])):
            l_tmp = utils.str_to_lerch(label[g][acc_n])
            p_tmp = utils.str_to_lerch(pred[g][acc_n])
            if (label[g][acc_n] == pred[g][acc_n]):
                correct += 1
            elif (l_tmp - p_tmp == 12 or l_tmp - p_tmp == -12):
                parallel += 1
            elif (l_tmp<12 and (l_tmp+7)%12 == p_tmp):
                fifth += 1
            elif (l_tmp>=12 and (l_tmp+7)%12 == p_tmp-12):
                fifth += 1
            elif ((l_tmp+9)%24 == p_tmp):
                relative += 1
        if(Q3 == True):
            try:
                acc = ((correct + 0.5*fifth + 0.3*relative + 0.2*parallel) / len(label[g]))*100
            except ZeroDivisionError:
                acc = 0
        else:
            try:
                acc = (correct / len(label[g]))*100
            except ZeroDivisionError:
                acc = 0
        ##########
        print("{:9s}\t{:8.2f}%".format(g,acc))
        label_list += label[g]
        pred_list  += pred[g]
else:
    label_list = label
    pred_list  = pred
```

因為只有 GTZAN 有不同的 genre 需要分開 print，因此其他都是直接將 label 存入 label_list，之後算總 accuracy 就好，而總 accuracy 的算法和個別的 accuracy 一樣只是範圍變大了而已。

為了避免每一行程式碼太長，我先將 label 和對應的 pred 的 key 換算成數值後再進行運算，parallel 會剛好相差 12，而 relative 則會相差 9（以 24 當一個循環的話），而 fifth 則是差七個音，但是會同為大調（或同為小調），因此我是將大調和小調分開，這樣只須確定 tonic 相同就好。

Q2

- Result discussion

```
r = 1
Genre           accuracy
blues             7.14%
country          34.34%
disco            32.65%
hiphop           13.58%
jazz             16.46%
metal            21.51%
pop              40.43%
reggae           31.96%
rock             33.67%
----------
Overall accuracy:        26.16%
```

```
r = 10
Genre           accuracy
blues             4.08%
country          31.31%
disco            28.57%
hiphop           14.81%
jazz             11.39%
metal            19.35%
pop              39.36%
reggae           28.87%
rock             30.61%
----------
Overall accuracy:        23.54%
```

```
r = 100
Genre           accuracy
blues             4.08%
country          31.31%
disco            28.57%
hiphop            9.88%
jazz             13.92%
metal            19.35%
pop              34.04%
reggae           24.74%
rock             27.55%
----------
Overall accuracy:        21.86%
```

```
r = 1000
Genre           accuracy
blues             5.10%
country          31.31%
disco            27.55%
hiphop            8.64%
jazz             13.92%
metal            19.35%
pop              32.98%
reggae           25.77%
rock             28.57%
----------
Overall accuracy:        21.86%
```

GTZAN dataset

先用不同的 γ 值進行 log 運算後再找 key 的話，可以發現當 γ 當 γ 越大，總 accuracy 越來越小，但並不是所有種類的音樂都越來越小，blues 在 γ=1000 的時候 accuracy 反而比 γ=100 的時候還要大，hiphop 也是在 γ=10 會有最大值，若是再和未進行 logarithmic compression 的 Q1 的執行結果進行比較，發現不同的音樂類型要得到最大的 accuracy，所需的條件都不太一樣，可能跟每一種音樂類型的特色有關，有些種類有比較多的樂器疊加在一起聲音互相受到的影響比較大，而有些音樂比較重節拍旋律來說反而比較簡單等等。

```
GTZAN dataset
Task 1
***** Q1 *****
Genre           accuracy
blues             7.14%
country          32.32%
disco            31.63%
hiphop           13.58%
jazz             16.46%
metal            24.73%
pop              41.49%
reggae           32.99%
rock             34.69%
----------
Overall accuracy:        26.52%
```

```
r = 1
----------
Overall accuracy:        21.69%
r = 10
----------
Overall accuracy:        20.70%
r = 100
----------
Overall accuracy:        20.03%
r = 1000
----------
Overall accuracy:        19.87%
```

GiantSteps dataset 中，同樣也是 γ 值越大，overall accuracy 越小。

因此可以得到的結論就是當 logarithmic compression 的 γ 值越大，會普遍降低 Key finding 的準確度。

- Implement

  除了設定了不同的 γ 值之外，其餘部分皆和 Q1 的 Implement 相同。

Q3

- Result discussion

```
***** Q1 *****
Genre           accuracy
blues              7.14%
country           32.32%
disco             31.63%
hiphop            13.58%
jazz              16.46%
metal             24.73%
pop               41.49%
reggae            32.99%
rock              34.69%
----------
Overall accuracy:        26.52%
```

```
***** Q3 *****
Genre           accuracy
blues             18.37%
country           54.85%
disco             49.80%
hiphop            20.62%
jazz              31.52%
metal             34.41%
pop               54.04%
reggae            46.29%
rock              47.76%
----------
Overall accuracy:        39.99%
```

```
GiantSteps dataset
Task 1
***** Q1 *****
----------
Overall accuracy:        22.35%
```

```
***** Q3 *****
----------
Overall accuracy:        32.33%
```

觀察兩個 dataset 的結果，若是用 Q3 中計算 accuracy 的方法得到的 accuracy，在 GTZAN 中 overall accuracy 增長了(39.99-26.52)/26.52 = 50.79%，而 GiantSteps 增長了(32.33-22.35)/22.35 = 44.65%

可以發現的是，其實受到泛音、以及和弦音彈奏時的輕重差異等等因素，導致聲音在偵測時的誤差非常嚴重，若是將這些都等比例的考慮進去，能有效的提高 accuracy 約 50%左右。

- Implement

  實作方法在 Q1 的 Implement 有詳細解釋過了，因此這部分就先跳過。

TASK I I

Q4

- Result discussion

```
Task 2
***** Q4_1 *****
Genre            accuracy
blues               17.35%
country             29.29%
disco               29.59%
hiphop              14.81%
jazz                16.46%
metal               26.88%
pop                 42.55%
reggae              36.08%
rock                34.69%
----------
Overall accuracy:       27.96%
use chroma_cqt
Genre            accuracy
blues               21.43%
country             48.48%
disco               28.57%
hiphop              22.22%
jazz                13.92%
metal               41.94%
pop                 40.43%
reggae              37.11%
rock                42.86%
----------
Overall accuracy:       33.57%
use chroma_cens
Genre            accuracy
blues               21.43%
country             41.41%
disco               25.51%
hiphop              23.46%
jazz                 8.86%
metal               35.48%
pop                 30.85%
reggae              30.93%
rock                35.71%
----------
Overall accuracy:       28.67%
```

```
GTZAN dataset
Task 1
***** Q1 *****
Genre            accuracy
blues                7.14%
country             32.32%
disco               31.63%
hiphop              13.58%
jazz                16.46%
metal               24.73%
pop                 41.49%
reggae              32.99%
rock                34.69%
----------
Overall accuracy:       26.52%
```

對 Jazz 和 rock 兩種類型來說，key 的判斷是不是 binary 並沒有任何影響，但對 reggae 和 metal 來說，accuracy 會降低不少，同時，blues 類型的音樂卻在改變了 key MODE 的判斷方式後大大的提升了 accuracy。這個結果可能和音樂的結構有關係，藍調音樂對於音樂的結構比較沒有強制的規範，都是比較隨興的為主，因此經過音或是非該調性音階上的音會比較常出現，因此當不是只有 1 和 0 兩種判斷時，能夠更加準確地找到他的 key 是什麼。

但我無法理解為什麼 reggae 和 metal 會在改用 Krumhansl-Schumuckler’s method 後 accuracy 反而降低了，可能跟他們的重拍特色有關？但若是要用 Krumhansl-Schumuckler’s method 進行 Key Finding 時，可能要避開這個兩類型的音樂，可以得到更好的結果。

chroma_stft 在抓他的 chroma vector 的時候，主要是抓最強烈的音，或是說是最大聲的音？其他較弱的音就比較被忽視，但 chroma_cqt 除了同樣會抓最強烈的音之外，對於中強度的音的紀錄也比較明顯，而

chroma_cens 則是有點像 chroma_stft 和 chroma_cqt 居中的狀態，但因為每個音樂類型也有不同的音樂結構特色，因此 overall accuracy 雖然是 chroma_cqt > chroma_cens > chroma_stft，但在個別上就有很大的差異，比如說 pop music 就是用 chroma_stft 會是最高的，而 country music 則是在 chroma_cqt 的使用下可以得到將近 50%的 accuracy。雖然我有實作了 chroma_cens 和 chroma_cqt，但造成補同結果的真實原因其實並不是非常清楚跟確定，只能透過結果以及查相關資料進行反推。

```
GiantSteps dataset
Task 1
***** Q1 *****
----------
Overall accuracy:        22.35%
```

```
Task 2
***** Q4_1 *****
----------
Overall accuracy:        27.81%
use chroma_cqt
----------
Overall accuracy:        37.75%
use chroma_cens
----------
Overall accuracy:        34.44%
```

GiantSteps 只有 overall accuracy，但若是將 GTZAN dataset 中得到的結論套用至此 dataset 中，依然可以得到相同的結果。

```
r = 1
Genre           accuracy
blues             17.35%
country           31.31%
disco             31.63%
hiphop            12.35%
jazz              15.19%
metal             25.81%
pop               42.55%
reggae            35.05%
rock              33.67%
----------
Overall accuracy:        27.72%
use chroma_cqt
Genre           accuracy
blues             22.45%
country           45.45%
disco             27.55%
hiphop            20.99%
jazz              11.39%
metal             39.78%
pop               38.30%
reggae            35.05%
rock              40.82%
----------
Overall accuracy:        31.90%
use chroma_cens
Genre           accuracy
blues             21.43%
country           41.41%
disco             23.47%
hiphop            20.99%
jazz               8.86%
metal             34.41%
pop               28.72%
reggae            29.90%
rock              34.69%
----------
Overall accuracy:        27.60%
```

```
r = 10
Genre           accuracy
blues             17.35%
country           28.28%
disco             26.53%
hiphop            14.81%
jazz              13.92%
metal             24.73%
pop               40.43%
reggae            30.93%
rock              32.65%
----------
Overall accuracy:        25.93%
use chroma_cqt
Genre           accuracy
blues             22.45%
country           41.41%
disco             26.53%
hiphop            23.46%
jazz              10.13%
metal             37.63%
pop               34.04%
reggae            34.02%
rock              38.78%
----------
Overall accuracy:        30.35%
use chroma_cens
Genre           accuracy
blues             21.43%
country           34.34%
disco             22.45%
hiphop            19.75%
jazz               7.59%
metal             34.41%
pop               24.47%
reggae            28.87%
rock              30.61%
----------
Overall accuracy:        25.33%
```

```
r = 100
Genre           accuracy
blues             20.41%
country           28.28%
disco             26.53%
hiphop            12.35%
jazz              15.19%
metal             24.73%
pop               37.23%
reggae            27.84%
rock              31.63%
----------
Overall accuracy:        25.33%
use chroma_cqt
Genre           accuracy
blues             25.51%
country           37.37%
disco             24.49%
hiphop            22.22%
jazz               7.59%
metal             35.48%
pop               28.72%
reggae            32.99%
rock              36.73%
----------
Overall accuracy:        28.43%
use chroma_cens
Genre           accuracy
blues             22.45%
country           27.27%
disco             22.45%
hiphop            19.75%
jazz               8.86%
metal             31.18%
pop               21.28%
reggae            28.87%
rock              27.55%
----------
Overall accuracy:        23.66%
```

```
r = 1000
Genre           accuracy
blues             21.43%
country           29.29%
disco             25.51%
hiphop            11.11%
jazz              15.19%
metal             24.73%
pop               37.23%
reggae            29.90%
rock              31.63%
----------
Overall accuracy:        25.57%
use chroma_cqt
Genre           accuracy
blues             25.51%
country           37.37%
disco             24.49%
hiphop            22.22%
jazz               7.59%
metal             35.48%
pop               28.72%
reggae            32.99%
rock              36.73%
----------
Overall accuracy:        28.43%
use chroma_cens
Genre           accuracy
blues             22.45%
country           23.23%
disco             22.45%
hiphop            19.75%
jazz              11.39%
metal             31.18%
pop               20.21%
reggae            27.84%
rock              26.53%
----------
Overall accuracy:        23.06%
```

當使用 Krumhansl-Schumuckler’s method 並進行 logarithmic compression 之後，不會再像之前一樣γ值越大，overall accuracy 越小，而是到γ=1000 時開始回升，有點像是一個曲線的波動的感覺，且經過 logarithmic compression，不再像 Q4_1 一樣 chroma_cqt > chroma_cens > chroma_stft，而是 chroma_cqt > chroma_stft > chroma_cens，而且γ值越大造成的差異越大，大概是因為經過 logarithmic compression 之後，除了音量的比例大小會跟人耳聽到的較為類似之外，也會將微小的差異進行放大，因此才會得到這個結果。

```
***** Q4_3 *****
Genre           accuracy
blues              29.08%
country            51.92%
disco              46.63%
hiphop             22.22%
jazz               32.15%
metal              37.74%
pop                54.89%
reggae             48.97%
rock               47.24%
----------
Overall accuracy:       41.34%
use chroma_cqt
Genre           accuracy
blues              30.00%
country            64.65%
disco              37.76%
hiphop             28.27%
jazz               28.10%
metal              46.99%
pop                46.91%
reggae             46.29%
rock               51.53%
----------
Overall accuracy:       42.56%
use chroma_cens
Genre           accuracy
blues              27.76%
country            56.36%
disco              33.47%
hiphop             29.51%
jazz               22.66%
metal              39.35%
pop                36.28%
reggae             39.90%
rock               43.47%
----------
Overall accuracy:       36.75%
```

將 Q4 應用到 Q3 中之後，原本 Q3 的 overall accuracy 是 39.99%也增加到了 41.34%，同時和 Q4_2 中一樣出現 chroma_cqt > chroma_stft > chroma_cens 的現象，雖然我不太確定原因。

```
Task 2
***** Q4_1 *****
----------
Overall accuracy:       27.81%
use chroma_cqt
----------
Overall accuracy:       37.75%
use chroma_cens
----------
Overall accuracy:       34.44%
```

然後是 GiantSteps dataset，不論是 Q4_1 還是 Q4_2、Q4_3，結果都和將 GTZAN dataset 在 Q4 的條件下應用在 Q1 時的 overall accuracy 的結論一樣，可能因為沒有多種的音樂類型對 overall accuracy 的值進行干擾，因此結果較為統一。

```
***** Q4_2 *****
r = 1
----------
Overall accuracy:        27.48%
use chroma_cqt
----------
Overall accuracy:        36.26%
use chroma_cens
----------
Overall accuracy:        33.61%
r = 10
----------
Overall accuracy:        27.48%
use chroma_cqt
----------
Overall accuracy:        35.26%
use chroma_cens
----------
Overall accuracy:        31.95%
r = 100
----------
Overall accuracy:        26.66%
use chroma_cqt
----------
Overall accuracy:        35.10%
use chroma_cens
----------
Overall accuracy:        29.47%
r = 1000
----------
Overall accuracy:        26.49%
use chroma_cqt
----------
Overall accuracy:        34.93%
use chroma_cens
----------
Overall accuracy:        28.15%
```

```
***** Q4_3 *****
----------
Overall accuracy:        37.15%
use chroma_cqt
----------
Overall accuracy:        43.92%
use chroma_cens
----------
Overall accuracy:        39.92%
```

- Implement

除了將對照的 Key 換成了 Krumhansl-Schumuckler’s method 的 Key，以及新增了 chroma_cens 和 chroma_cqt 之外，其餘部分皆和 Q1 的 Implement 相同。

TASK III

Q5

- Result discussion

```
BPS-FH
correct and parallel relative detection
Overall accuracy:       38.62%
correct detection
Overall accuracy:       24.26%
----------
```

```
A-MAPS
correct and parallel relative detection
Overall accuracy:       36.53%
correct detection
Overall accuracy:       21.89%
```

在 Q5 中，在計算 accuracy 時加入了 parallel 等的特性之後進行等比例的計算，增長的比例比前面的 Task 還要高，BPS-FH 有 59.19%而 A-MAPS 甚至有 66.88%。

- Implement

  BFS-FH

```
DB = 'BPS-FH'
last_label_num = 0
chromagram, label, index, pred = list(), list(), list(), list()
for f in glob(DB+'/wav/*.wav'):
    f = f.replace('\\','/')
    print(f)
    key = utils.parse_key([line.split('\t')[1] for line in utils.read_keyfile(f,'*.txt').split('\n')])
    label.extend(key)
    sr, y = utils.read_wav(f)
    count =0
    while count < len(label)-last_label_num:
        Q5_detect(count , sr , y , pred)
        count += 1
    last_label_num = len(label)
```

由於每秒要 find 一次 local key，但所有 file 的 label 又都存在同一個 list 中，因此我多開了一個 int 叫 last_label_num 用於存到前一個 file 執行結束後，有多少內容存在 label，便可算出該 file 有多少秒需要進行計算。

```
def Q5_detect(num , sr , y , pred):
    if(num < 16):
        tmp_y = y[:sr*(num+16)]
    elif(num < int(len(y)/sr)-16):
        tmp_y = y[(num-15)*sr:sr*(num+16)]
    else:
        tmp_y = y[(num-15)*sr:]
    cxx = librosa.feature.chroma_cqt(y=tmp_y , sr=sr)
    chroma_vector = np.sum(cxx , axis=1)
    key_ind = np.max(chroma_vector)
    count =0
    for i in chroma_vector:
        if(i == key_ind):
            key_ind = count%12
            break
        count += 1
    chroma_vector = utils.rotate(chroma_vector.tolist(), 12-key_ind)
    p_major = pearsonr(chroma_vector , utils.MODE["major"])
    p_minor = pearsonr(chroma_vector , utils.MODE["minor"])
    if(p_major[0] > p_minor[0]):
        key_ind = (key_ind+3)%12
    else:
        key_ind = (key_ind+3)%12+12
    print(num , "\t" , utils.lerch_to_str(key_ind))
    pred.extend([utils.lerch_to_str(key_ind)])
    return
```

在 Q5_detect function 中，先限定該次是要找哪一秒的 key，需要存取哪段 chromagram 的資料，我讓每一秒計算 local key 時都取前後 15 秒的資料一同計算，以增加 find key 的準確性。將選定的 chromagram 段落存進 tmp_y 後，其餘的執行方式皆和 Q1 相同。

A-MAPS

```
DB = 'A-MAPS'
label, pred = list(), list()
for f in glob(DB+'/*.mid'):
    f = f.replace('\\','/')
    midi_data = pretty_midi.PrettyMIDI(f)
    #get label
    key = midi_data.key_signature_changes
    for i in range(len(key)):
        key_num = utils.parse_key_number(key[i].key_number)
        label.append([int(key[i].time) , key_num])
```

由於 A-MAPS 這個 dataset 皆為 midi 檔，因此用的是 pretty_midi.PrettyMIDI 為每一個新的 midi file 建立一個新的 class PrettyMIDI 資料，並用 key_signature_change 取的該 midi file 的 label key，並存入 label 中。

```
chroma_vector = midi_data.get_chroma()
for count_time in range(int(midi_data.get_end_time())):
    tmp_vector = list()
    for x in range(12):
        if(count_time < 16):
            tmp_vector.append(chroma_vector[x][:100*(count_time+16)])
        elif(count_time < int(midi_data.get_end_time())-16):
            tmp_vector.append(chroma_vector[x][(count_time-15)*100:100*(count_time+16)])
        else:
            tmp_vector.append(chroma_vector[x][(count_time-15)*100:])
    tmp_vector = np.sum(tmp_vector , axis=1)
    key_ind = np.max(tmp_vector)
    count =0
    for i in tmp_vector:
        if(i == key_ind):
            key_ind = count%12
            break
        count += 1
    tmp_vector = utils.rotate(tmp_vector.tolist(), 12-key_ind)
    p_major = pearsonr(tmp_vector , utils.MODE["major"])
    p_minor = pearsonr(tmp_vector , utils.MODE["minor"])
    if(p_major[0] > p_minor[0]):
        key_ind = (key_ind+3)%12
    else:
        key_ind = (key_ind+3)%12+12
    pred.extend([utils.lerch_to_str(key_ind)])
    del tmp_vector
label.append([int(midi_data.get_end_time())])
```

用 PrettyMIDI 中的 get_end_time 取得該 midi 檔的時間長度，確定我需要算多少次 local key，而且由於 get_chroma 是取的 chroma vector，因此所有 data 是分成 12 個 list 存在 chroma_vector 中，因此用和 BFS-FH 相同的概念取得該次我計算 local key 所需要的 chromagram 並存入 tmp_vector 中，其餘部分皆和 Q1 相同。

```
correct = 0
fifth = 0
relative = 0
parallel = 0
count_len =0
for label_num in range(len(label)):
    if(len(label[label_num]) == 1): continue
    for acc_n in range(label[label_num+1][0] - label[label_num][0]):
        l_tmp = utils.str_to_lerch(label[label_num][1])
        p_tmp = utils.str_to_lerch(pred[acc_n+count_len])
        if (label[label_num][1] == pred[acc_n+count_len]):
            correct += 1
        elif (l_tmp - p_tmp == 12 or l_tmp - p_tmp == -12):
            parallel += 1
        elif (l_tmp<12 and (l_tmp+7)%12 == p_tmp):
            fifth += 1
        elif (l_tmp>12 and (l_tmp+7)%12 == p_tmp-12):
            fifth += 1
        elif ((l_tmp+9)%24 == p_tmp):
            relative += 1
    count_len += label[label_num+1][0] - label[label_num][0]
```

在計算 accuracy 和 Q1 不同的是，由於若是 local key 沒變，label 中只會有一個值，但 pred 中有多少秒就會有多少個，因此我在存進 label 時就把幾秒 local key 換成什麼和總共有幾秒的內容都一併存在 label 中，這樣我只要透過存取 label 中的前後 list 的值就可以知道目前的 label 是要對應到多少個 pred 資料。