Abstract

In today's digital age, not most people understand the techniques and methods of inputting midi, so I thought of the most direct way to create rhythms through voice input, so that people who want to create rhythms but cannot play and record can use their own mouths to create the rhythms in their mind.

In this article, I will use python to train and recognize voice files. From starting sound judgment, feature extraction, training model, predicting classification, to final output of drum sound files. To implement each of these steps, I refer to the functions developed by librosa, soundfile, and sklearn. I used root-mean-square and onset strength for starting sound judgment, MFCC and zero crossing rate for features, SVM for modeling, and the soundfile library for drum output.

The user experience is not perfect because only a very small amount of data is used for training. If there are many drummers' or creators' voice input as training data, this idea may become one of the resident input plugins in DAW (Digital Audio Workstation) in the future.

摘要

在現今的數位時代，並不是大部分人都了解輸入 midi 的技巧和方法，因此我想到了透過語音輸入這種最直接的方法，讓想要創作節奏又沒辦法實際演奏錄音的人，可以用自己的嘴巴創作腦中的節奏。

本篇將藉由 python，實現語音檔的訓練及辨識。從起音判斷、特徵萃取、訓練模型、預測分類，到最後輸出鼓聲音檔，為了實現以上步驟，我引用了 librosa、soundfile 與 sklearn 開發的函式。起音判斷使用 RMS 與 onset strength；特徵使用 MFCC 與 zero crossing rate；模型使用 SVM；鼓聲輸出使用 soundfile 函式庫。

由於是用非常少量的資料進行訓練，使用者體驗還不是很完美，若是能有眾多鼓手或創作者的語音輸入當作訓練資料，本想法有機會成為未來 DAW(數位音樂製作軟體)中常駐的一種輸入方式插件。

一、 前言

在現今的數位時代，利用預先錄製好的音源及 midi 輸入創作音樂，已經變成音樂製作的一大部分。就算不會演奏爵士鼓，也可以透過 midi 一個個的輸入，創作出鼓的節奏及音樂。但並不是大部分人都了解輸入 midi 的技巧和方法，因此我想到了透過語音輸入這種最直接的方法，讓想要創作節奏又沒辦法實際演奏錄音的人，可以用自己的嘴巴創作腦中的節奏。

而要達成語音輸入其中很重要的一環就是語音辨識，語音辨識在近幾年已經發展的非常成熟，幾乎每個人的手機都有語音輸入的功能，且辨識效果及抗雜訊能力都很好。語音辨識的發展，其中一個很重要的聲音特徵就是 MFCC。MFCC 透過數個三角濾波器，將高維的頻譜降維後，再用導頻譜來代表整個頻譜的特徵。此算法不僅僅降低了資料的維度，還提供了比較兩頻譜相似度的判斷標準。MFCC 越接近的兩個聲音片段，聽起來的特色也真的會越像。

有了有效又有依據的聲音特徵之後，下一步要做的是語音辨識的分類任務。這些任務其實又可以分為很多種類，比如應用層面最廣，現在大多數人都在使用的連續語音辨識，在不知道資料長度、單詞斷點、語速等等重要資訊的情況下，完成連續語音辨識。還有每個訓練及測試音檔都已切割好，長度不一的辨認任務，這種任務通常在研究擷取什麼樣的特徵及辨認方法可以讓準確度提升。而為了讓實用性提升以及讓難度降低，本篇將處理未知聲音斷點、音檔固定長度的語音辨識任務。

為了找到一串音檔中的未知起音點，我混和了時域上的特徵 RMS(root mean square)以及頻率上的能量差距—onset strength—來做為起音點的判斷。為了固定每個片段的長度以方便分類，對於每個起音都只取固定的幾個 frames 作為測試音檔，這樣也可以固定每個聲音片段的特徵維度。

在分類方面，我使用 SVM(Support vector machine)。因為 SVM 的數學架構簡單，擅長處理類別少的分類問題，而且不需要很多的訓練資料及訓練時間就能達到其表現上限。在本篇，我會對音檔進行三類的分類，並讓其分別對應到鈸、小鼓以及大鼓的輸入。

最後，再利用抓到的起音點與分類結果，將鼓聲置入起音的位置輸出音檔，就可以聽到轉換後的鼓聲了。而現在碰到的困難就是轉換後的結果差強人意，雖然有一定的準確率，但只要幾個大鼓小鼓的分類錯誤，聽起來就會很不像想要創作的節奏。

本篇主要創新的地方有以下幾點

1. 利用 RMS 與頻譜能量差建立的起音判斷函式，並包含原 librosa 函式庫沒有的使用者參數設定，以符合使用者的需求。
2. 不判斷尾音時間，而使用固定的 frame 數量來做語音分類。
3. 語音輸入與音樂創作的結合，套用最簡單的語音辨識流程，來實現鼓聲的編輯創作的便利化，並聚焦在此目的，設計辨識及音節區分的演算法。
4. 使用者自訂語音輸入樣本，不須龐大的訓練資料就能有立即的辨識成果，並擁有抗環境穩定底噪的能力。

實作中許多部份直接引用 librosa 開發的套件進行聲音特徵處理，除此之外其他所有程式內容及設計都是自己實際實作的內容。

二、 實作假設

2.1 使用者錄音環境

此段將說明以下文中討論的使用者語音輸入環境假設。錄音檔的雜訊比無特別限制，只要能清楚聽到使用者錄的人聲就可以，但是雜訊只能有平穩、沒有突然音量起伏的聲音，比如白噪音、綿綿雨聲、空間混響等等，不特別處理其他會造成起音判斷的雜音。使用者可以使用任何設備，並以22050Hz 以上的取樣率進行錄音。大部分的音檔是使用44100Hz 的取樣率，以及直播主等級的動圈式麥克風錄製，錄製的音檔是單聲道，且會重新 resample 成22050Hz。

2.2 使用者音樂知識背景

使用者不需要懂得鼓譜、節拍、力度、口技等專業技能，只需要事先決定好三種聲音對應到小鼓、鈸、以及大鼓，並且以自己來的及的速度創作節奏即可。

2.3 軟體環境

直接在 python(.py)上執行程式，預設使用者有建立好包含所有使用到的函式庫，可以直接運行 python 程式檔。所有可調整參數都有設定成可以運行的數值。

2.4 引用函式

宣告外部函式包含 numpy、librosa、sklearn、soundfile、matplotlib.pyplot。

三、 實作方法

3.1 提取各種特徵

3.1.1 梅爾頻譜(Mel-spectrogram)

使用 librosa 函式計算音檔的 Mel-spectrogram，其中 n_fft(number of sample for FFT)、hop_length(frame hopping sample)、n_mels(number of bank)是可以在實驗中調整的參數，而這裡決定的 n_fft 與 hop_length 會套用到後面的其他特徵中。梅爾頻譜是 MFCC 的前置工作，也可以畫出梅爾頻譜，幫助視覺化聲音，觀察聲響的頻率分布。

3.1.2 onset strength

使用 librosa 函式計算聲音的起音強度。

計算出來就是一個長度與 frame 數相同且大於等於0的一維陣列，值越大代表那個 frame 與前一個 frame 的能量差距越大，用以判斷起音時間。

3.1.3 root-mean-square (RMS)

使用 librosa 函式計算聲音的 root-mean-square (RMS)，就是對應到聲音的能量。為了讓其數值與聽感符合，我將用其取 $\log_{10}$的數值。

計算出來是一個長度與 frame 數相同的一維陣列，值越大代表那個 frame 的能量越大，用以判斷起音時間。

3.1.4 zero cross rate

使用 librosa 函式計算 zero cross rate，當作聲音的特徵值之一。計算前要先將音檔的 DC bios(mean of input data)減掉，才能確保 zero cross rate 的數值是正確的。

計算出來是一個長度與 frame 數相同且介於0到1的一維陣列。

3.1.5 MFCC

使用 librosa 函式，計算已有 mel spectrogram 其特定幾個 frame 的 MFCC 參數，其中 n_mfcc 是實驗中可以調整的參數，決定 MFCC 回傳的特徵維度。

3.2 起音判斷

3.2.1 目標

起音判斷的目標是對於一長串錄音音檔的資料，可以找到其中每個聲音，也就是使用者輸入的語音節奏音符出現的位置。又因為此應用是語音輸入節奏，所以假設每個語音節奏音符都會是起音明顯，且長度不長的聲音，因此，只要抓到起音時間點，該點就是使用者想輸入節奏的時間點，同時也是該語音節奏音符開始的地方。找到準確的起因時間，可以幫助我們做聲音的分類，以及讓輸出的鼓點符合使用者想像的節奏。

3.2.2 判斷依據

用以判斷的特徵就是3.1.2及3.1.3的 onset strength 與 RMS 的相加。

在實驗時發現，如果只採用 onset strength 做判斷依據，會額外抓到很多小聲的聲響，比如嘴唇造成的雜音，尾音的雜音等等；又或者因為 onset strength 只考慮差距，不考慮原聲音的能量，反而忽略掉使用者使用氣音的地方。

而如果只採用 RMS，抓的時間點會是聲音中能量最大的地方，而不是聲音剛發生的起音點。除此之外，還更可能在一個語音節奏音符中抓到超過一個峰值，這不是我們預期的成果。

而綜合考量這兩個特徵，就可以解決單獨使用其中一個時會遇到的問題。雖然兩個的單位跟數量級並不一樣，但是因 onset strength 是介於0到1的數值，直接相加反而可以讓 RMS 成為主要決定的依據，而 onset strength 可以幫助凸顯起音的時間點。而 RMS 與 onset strength 相加後的新特徵，在本篇稱為 rms_oenv。

3.2.3 peak pick

有了供判斷起音的特徵後，接著就是要判斷峰值出現的地方。

Peak pick 是我自己寫的 function，其可傳遞參數包含 x：輸入訊號、pre_max：往前比較的個數、post_max：往後比較的個數、delta：peak picking threshold、frames_taken_min_space：兩個 peak 之間的最短距離。

簡單的說，x[n]要被選為 peak 必須滿足：

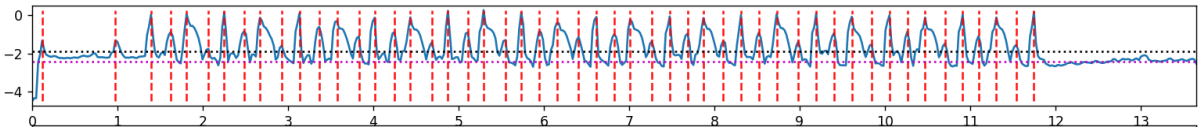
x[n] == max(x[n-pre_max : n+post_max+1]) and x[n] >= delta

而若有給定 frames_taken_min_space，則會在選完 peak 後再次檢查兩個 peak 彼此之間的距離，若小於 frames_taken_min_space，則會移除數值比較小的 peak，要設計這個是為了避免 peak picking 過度靈敏，抓到許多不正常的音符，可以根據使用者的速度調到合適的大小過濾錯誤的 peak。

在實作中，我給定：

```
pre_max = post_max = 2
delta = gate+(abs(sorted_rms_oenv.max()-gate)*peak_th), whose
gate =
mean(sorted_rms_oenv[np.size(sorted_rms_oenv)//10:np.size(sorted_rms_oenv)//4]),
peak_th = 0.2
```

意思是 peak 的定義是要超過 delta 且比前兩個跟後兩個都大的點。gate 是 rms_oenv 中的基礎門檻，約等於沒有輸入聲音時雜訊的大小，而 delta 就是 gate 到 rms_oenv 的最大值的距離乘上 peak_th。簡單的說，peak_th 越大，peak picking threshold 越大。當 peak_th = 0，所有高於雜訊的 peak 都會被選取；當 peak_th = 1，所有 peak 都不會被選取。



圖一、基底訊號是 rms_oenv，經過 peak pick 後，垂直的紅線代表抓到的起音點、紫色的點線代表 gate(預估的雜訊閥值)、黑色的代表 delta(peak picking threshold)

3.3 分類特徵

3.3.1 MFCC

在每個找到的起音點，取數個 frame 當作待分類的語音節奏音符，其中 pre_frame 與 frames_taken 是實驗中可以調整的參數。在每個找到的起音點，取其前面 pre_frame 個 frame，總共取 frames_taken 個 frame。在實驗中，這兩個參數的目標是要能涵蓋每個語音節奏音符的起音咬字及大部分聲音。經過測試，在給定3.1.1中 n_fft = 1024 與 hop_length = 512 時，發現 pre_frame = 4 and frames_taken = 6可以完整記錄大部分聲音(總涵蓋長度大約是0.2s)。

除了決定語音節奏音符的長度，還要決定 n_mfcc 與 n_mels 這兩個與 MFCC 有關的參數。我最後的決定方式是選取自己覺得用人耳也能清楚區分的音檔，讓其接受分類任務，設定 n_mfcc 與 n_mels 使其可以幾乎全對，最後決定 n_mfcc = 26 and n_mels = 64。

3.3.2 CMVN

將得到的 MFCC 經過 Cepstral Mean and Variance Normalization(CMVN)的計算後，作為聲音的主要分類特徵。所謂 CMVN，是指將每個語音節奏音符其 MFCC 的每個維度各自標準化，以縮小每個維度數值的差距，讓每個維度在之後分類器的重要程度差不多。

經過測試，CMVN 比一般的整體標準化更能凸顯每個聲音的特徵，讓每個聲音更好的被分類。

3.3.3 Final total feature

最後對於每個語音節奏音符，其包含6個 frame 約0.2s 的聲音，這6個 frame 每個都有26維、經過 CMVN 標準化過後的 MFCC，以及1維的 zero cross rate，並將其全部 resize 成一維陣列，總共有6*27=162維的 features。

為了避免起音點不同造成的些許位移誤差，每個起音點會被當作5個同類別語音節奏音符，每個位移一個 frame，取五組 features 進行後面訓練。

3.4 分類模型

3.4.1 SVM

本篇使用的分類模型是 SVM，因為 SVM 是很常用來處理 MFCC 等聲音特徵的分類模型。又本應用的 Ground truth data 非常少量，不適合使用 deep learning 等需要多資料訓練的方法。本篇使用的是 sklearn.svm 的 SVC function。而要分類的種類總共有三種，分別是使用者定義的、大鼓(label 0)、鈸(label 1)以及小鼓(label 2)。對於每個起音點，其必定會被歸類成這三類的其中一類。

3.4.2 訓練資料前處理

前處理分為兩種，一種是直接使用3.3.3的 features 進行訓練與分類，另一種是先將所有 features 透過 sklearn 的 preprocessing 進行標準化後，再進行訓練與分類。兩種的訓練是分開來的，所以可以看到兩者的結果。因為第二種的結果通常較好，所以選用第二種，所有展示的成果也將用標準化的作為前處理的方式。

3.4.3 kernel 與 model parameter

延續在數位訊號處理實驗中學到的經驗，分類以 MFCC 為特徵的聲音時以 rbf 為 kernel、model parameter 設 C = 1會有不錯的分類效果，單看訓練資料的話，其也確實能完美分隔三種不同類別。

3.5 鼓聲音檔輸出

每執行一次程式，就會依據抓到的起音時間，在對應的位置放入鼓聲，並利用 soundfile 輸出聲音檔。而要放入的鼓聲將會由 predict 出的 label 決定，放入小鼓、鈸以及大鼓的聲音。若是負責訓練的訓練音檔，一樣會照預設的順序輸出鼓聲，幫助使用者判斷自己的訓練音檔是不是沒有問題的。

四、 使用流程

4.1 錄音

4.1.1 錄音環境

由於沒有在 python 中實作錄音環境，現在還必須使用其他錄音設備及軟體錄好音檔後才能進行分析及處理，所以錄音步驟必須單獨完成，再把錄音檔放到指定目錄中。

4.1.2 ground truth 音檔

為了讓錄的音檔更符合到時候自由創作時可能會出現的聲音，我直接指定使用者要唱指定的節奏當作 ground truth 音檔，其節奏必須是[大鼓, 鈸, 小鼓, 鈸]的對應循環，比如動次打次動次打次動次打次…使用者可以自己定義任何聲音對應到大鼓、小鼓及鈸，但那些聲音最好符合有強烈的起音點，以及聲音不長的特性，效果會比較好。

4.1.3 自由創作音檔

自由創作顧名思義，可以用在 ground truth 定義的三種聲音自由創作節奏。錄音環境跟 ground truth 一樣更能減少誤差。

4.2 擷取 ground truth 聲音特徵

4.2.1 運行 Set ground truth.py

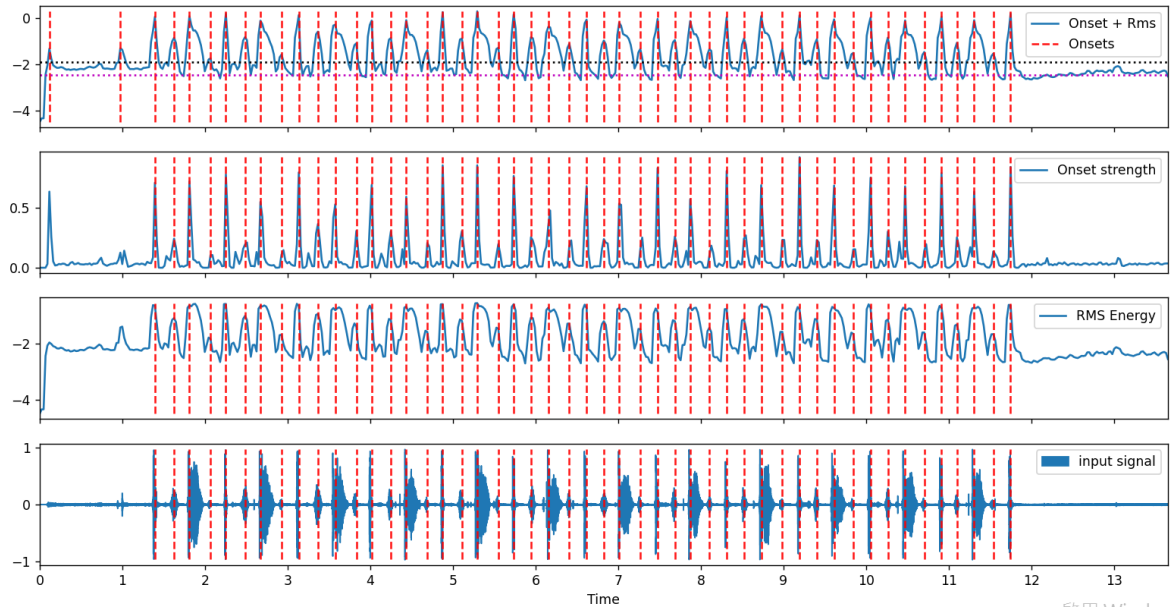
設定好錄好的音檔路徑並執行 Set ground truth.py，執行後會生成一張確認結果圖及 check_file.wav 音檔，供使用者檢查 ground truth 是否有正確對應到4.1.2所指定模式。結果圖包含音檔波形、RMS、onset strength、以及 rms_oenv 等欄位，並且包含所有找到的起音點，供使用者對照。

而 check_file 則會將每個起音點，按照[大鼓, 鈸, 小鼓, 鈸]的順序放入鼓聲，讓使用者聽聽看起音點是不是正確的。

4.2.2 修正起音點

若發現起音點有多或少，則需要調整 peak_th、frames_taken_min_space 等 peak pick 相關參數來調整結果，或是使用手動方式移除不想要的起音點。如圖中範例，一開始頭兩個是雜音造成的起音點，就可以手動移除這兩個點，讓第一個起音點確實對應到大鼓的聲音，且確保後面的順序都是[大鼓, 鈸, 小鼓, 鈸]。

因為此步驟是建立 ground truth，必須確保完全正確才能進行下一個步驟。



圖二、確認圖。從上到下分別是 rms_oenv、onset strength、RMS、聲音波形。
rms_oenv 圖中垂直的紅色 dash line 是找到的起音點。
其餘圖中垂直的紅色 dash 則是使用者刪除不要的點後的起音時間點。

4.3 將自由創作轉換鼓聲

運行 Main.py，依據自己創作的速度調整合適的 frames_taken_min_space，以避免抓到過多的錯誤起音點、設定 peak_th 調整 peak pick 的靈敏度，指定好要轉換的音檔路徑後，就會讀取 Set ground truth.py 產生的 features 訓練模型，並預測每個起音點應該是屬於哪一類，最後輸出鼓聲音檔。也可以根據自己需求，看需不需要顯示確認圖。

五、 成果

下面的網站中有範例的語音輸入及鼓聲輸出：

GitHub 網站：https://fish0510.github.io/Demo-result/

範例1是 Beatbox 的聲音，因為聲音符合起音強烈、區別性高、聲音又短的特徵，以及 Beatbox 的音不會有音調、語氣等情緒性的變動，辨別效果很好，輸出的鼓聲聽起來很正確。

範例2是一般鼓手在描述節奏時常用的聲音，因為輸入時會依據創作者想強調的節奏點或情緒，影響輸入語音的音調及發音方法，導致 Ground Truth 的聲音跟實際在創作時的聲音不盡相同，從輸出結果可以聽到一些分類錯誤的聲音。對於這種類型的聲音，可能會需要更多不同節奏的 Training data，來讓模型能辨別使用者對於同個聲響，不同的音調及表現方式。

範例3是請不會打鼓也不會輸入 midi 的同學試用的結果。輸出結果雖然聽起來正

確率很高，但其實在錄音階段遇到比較多的困難。在錄音時，他們在自由創作時的聲音，常會跟自己建立的 Ground Truth 有出入，這點從範例3-3也可以聽到。他定義的小鼓跟創作時的小鼓是不同聲音，雖然分類出來的最後結果依然是正確的。

以三個範例總結，本篇使用的方法只適合分類如 Beatbox 般固定、區別性高的聲音，當使用者依自己想像中的情緒描述一段節奏時，往往會產生與自己定義的 Ground Truth 不一樣，因此準確率會下降。

六、 結論

本篇使用非常基本的特徵 MFCC 與 zero cross rate，及構造簡單的模型 SVM 等方式，只需少量的訓練資料就成功的完成了使用語音輸入節奏的應用。並建立了一個以 RMS 與 onset strength 組合而成的起音判斷算法。對於起音強烈、區別性高、聲音又短的聲音來說，聆聽起來，辨別的效果很不錯。使用流程稍顯複雜，對於 Ground Truth 的建立流程還需要再簡化會比較實用。

心得

因為加入迴聲社擔任鼓手，才想到要用語音輸入來創作節奏，想藉此把生活中突然冒出的靈感記錄下來，並用語音辨識轉成 midi 或鼓聲進行進一步的創作。

這算是獨自一人從0開始的專題。一開始什麼都不會，在學長跟教授的幫助下才開始熟悉 python、知道 Mel Spectrogram 這個特殊的頻譜、瞭解 MFCC 這個語音辨識中重要的語音特徵。知道理論及架構後，慢慢地嘗試，將心中想達成的目標一步步實作出來。

過程中遇到無力的時候，跟學長姐及同學討論總能給我一些新的想法，讓我的準確率跟合理性越來越好。那些借來的 functions，也在嘗試中慢慢瞭解其中的邏輯跟原理。

我覺得這個不是能令人刮目相看、耳目一新的題目，但是透過它讓我自己成長了很多，學到了很多專業的知識及技能！