**DSP Final Project**

b04902092 張均銘

ㄧ、主題簡介

口琴real time將聲音轉成簡譜

二、大致作法

使用python pyaudio的部分錄音，自己吹口琴建立do ~ so的data和short pause的data，使用htlik的套件將錄好的wav檔轉成MFCC檔，並且用機器學習或深度學習的模型訓練，並比較效果的好壞。

三、使用套件

python3.6

pyaudio : record

keras

Tensorflow

wave

numpy

sklearn

os,sys

scipy

四、作法

中心想法在於嘗試不同的訓練方法，因此沒有使用老師教的HMM model的方式來train，想比較用其他方式train出來的model的效果的好壞。

主要想法是使用decision tree regressor直接把mfcc data丟進去讓他分類

或者是dnn也是把mfcc的numpy array丟進去model裡讓dnn分類(分成五類，do,re,mi,fa,so)，因此不論是用tree還是dnn都要事先把data label，那因為存的時候不同音的data存在不同資料夾，因此讀的時候可以依據讀的資料夾就順便做好label的動作。

1. 自己寫record.py、自己吹口琴來蒐集data，record.py特色是能夠連續吹很多音符產生很多個別的wav檔存在Traindata資料夾內。即執行一次可以露個25個音之類的。
2. complement.py是因為有時候音會吹錯，所以要用這個來把Traindata李吹錯的那個音在吹一個正確的音覆蓋掉。
3. 用mfcc.py把wav檔讀入依據do,re,mi,fa,so四個資料夾內個別所有data拿來產生mfcc的numpy array，即do內的80個wav檔產生一個(80,43,36)的numpy array，80代表有80筆資料，43代表一筆資料有43frame，36代表mfcc array。每個音都產生這樣的data
4. 嘗試各種model，用sklearn呼叫decision tree、random forrest、svm，其中svm效果最差，就不特別討論，tree呼叫decision tree\_regressor。
5. 嘗試keras dnn模型，模型架構可以看code，就是疊個幾層dense來測試，因為keras比較難裝，所以是用python3.5的版本來裝來跑，其他model都適用python3.6，所以dnn.py這個檔案才會是獨立的QQ。
6. demo.py自己吹一段音來當作testing data，主要是以C大調音階或者歡樂頌的歌，存在Testdata資料夾內部。
7. test.py讀入testdata呼叫model predict後輸出判斷成的音。

五、發現

1. 原本以為do是最容易判斷的，因為音調很沉相較其他音差異很大，但看起來mfcc並沒有把它弄到一個比較分開的維度，所以do其實判斷起來誤差蠻大的
2. 判斷最多正確的反而是mi fa so，原因可能是在data蒐集時這幾個音吹得比較好吧，因為畢竟是中間的音，吹的時候比較進入狀況，do剛跑程式自己太緊張之類的沒進入狀況，data可能比較亂。
3. decision tree效果跟random forrest差不多，很明顯因為準確率都不高，所以做ensemble意義也不大QQ。
4. DNN真的效果比較好，相較於其他model，但train到大概300個epochs後就開始overfit。我認為最大的敗筆還是在training data太少，validation set、dropout的效用就很小。
5. 準去率(未作autoencoder降維，沒時間QQ)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Tree | Random forrest(7顆) | Dnn |
| test.wav(12個音) | 33% | 50% | 33% |
| file.wav(4個音) | 25% | 0% | 25% |
| doremi.wav(9個音) | 33% | 33% | 33% |

六、困難

1. data蒐集

需要搜集每個音符的wav檔，另外再創一個蒐集noise、pause的model，因此口琴要吹好久.....。目前是一個音符一個資料夾，內部就是該音的所有wav檔。寫的程式是sequential取data(do~so重複吹，一次搜集很多程式才停下來)，萬一節奏沒對到，例如re的wav檔前半段蒐集到前一個do的後半段的音。因此再取data時其實是會把蒐集到的frame只取中間那段，前後段去掉，以免蒐集到上一個音的尾音或下一個音的開頭。實際上也只能每個音蒐集80筆data，確實數量不夠多很難train。

2.一秒內mfcc frame太多

signal一秒有4萬多個數值、一秒的mfcc有將近200個frame，這代表兩個問題，資料非常難train起來(因為一秒一個音，等於一個音有200\*39個參數)，二來一秒內實際上吹出聲音的時間可能只有400ms之類的，因此其他時間都是雜訊，結果勢必不好。

解法：

原本錄音得到signal為40000多維，取聲音最大的區段取前後10000維。

使用np.argsort，因此sort完得array後半段的index就是signal最強的地方，實際上就是吹奏的地方，因此就去argsort完後的最後10000筆的index取mean試圖取得發聲的中心地方。得到的index基本上就是聲音吹出來的地方，再將signal濃縮為mean的index向後取的10000筆資料的signal，因此signal從原本的4萬個值，但很多雜訊濃縮到剩10000個值，而且是主要發音的地方。

3. 雜訊太多：

這大概也是我們沒有train起來的原因之一，因為很難找到完全安靜的地方，能做的就是擷取真正有在吹奏的那段部分其餘部分砍掉，做法如同上述，把錄音得到的signal取聲音變化最大的區段前後10000筆，使signal從40000多為的shape變成10000維。

4. htlik的mfcc是特殊儲存格式

htlik利用HCopy得到的.MFC檔網路上沒有看到怎麼把它變成39維vector，因此就自己手刻MFCC(mfcc.py) 其實還蠻難的。

5.data維度太多：

一個音的data取mfcc後每個frame有36維， 每個音有43個frame，所以data有1677個值，但是這麼多值只分五類(do,re,mi,fa,so)，可以預見的就是數值太多準確率肯定不高。相信降維絕對能使準確率高很多，可惜沒有時間做QQ。

解法：

使用Autoencoder這個很炫的降維技術來降維，主要在考量PCA、TSNE、

Autoencoder要用哪個

1.根據ML經驗PCA需要太多記憶體

2.TSNE要跑比較久

3.想體驗看看使用DL解決問題的感覺

七、Future Work

1. 蒐集更多更廣泛data(目前只蒐集do~so，可以蒐集所有升記號)
2. 更好的處理不同速度的歌曲(如慢歌、超快歌)
3. 連音、輕音、之類特殊音的處理。
4. 把model train更好

八、Reference:

[multi-output decision tree] <http://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression_multioutput.html#sphx-glr-auto-examples-tree-plot-tree-regression-multioutput-py>

[keras]

<https://keras.io/getting-started/sequential-model-guide/>

[install pyaudio]

<https://github.com/conda/conda/issues/5629>

[low dimension dnn overfit]

<https://www.quora.com/Are-deep-neural-networks-useful-for-problems-with-low-dimensional-input>