

第一章 緒論

1.1 研究題目

本論文報告研究的題目是隱馬可夫模型與語音辨識的應用，說明隱馬可夫模型是如何運用觀測值與狀態的轉換機率來推斷狀態序列，且使用不同的演算法解決隱馬可夫模型的三個典型問題，並以 Java 程式執行實際範例。另外在語音辨識上，隱馬可夫模型如何處理與實作語音辨識程式。

1.2 相關文獻與研究

Kuo and Wang 提出一種針對聲音，分割音節的方法，該方法試圖模仿人類聲音音節分割的過程。使用最小邊界誤差（MBE）準則來執行隱馬可夫模型（HMM）訓練。也使用支撐向量機（SVM）方法來對 HMM 自動分段所得的時間位置做修正。而此研究也在 TIMIT 英語資料庫和 MATBN 漢語資料庫得到驗證[5]。

Chiu 提出一個以 HMM 為基礎的旋律抽取法，針對含有歌唱成分的複音音樂進行歌唱音高抽取，配合不同的方式進行特徵擷取，挑選出更趨穩定的峰值特徵，以提高音高曲線的準確性。首先，我們使用快速傅立葉轉換，將合理的人聲音高頻率範圍切割為數個頻段，將其視為隱藏式馬可夫模型中的不同狀態。使用不同的方法在頻譜上找出強度明顯的峰值作為特徵，藉由不同的峰值選取方法以篩選出更趨穩定的峰值特徵，期望能增進歌唱音高片段的辨識效果，取出更為準確的歌唱音高資訊。最後，以維特比解碼(Viterbi Decoding)得出連續歌唱音高曲線[1]。

第二章 隱馬可夫模型

本章主要參考 Dan Jurafsky 和 James H. Martin 作者的 *Speech and Language Processing* 來說明 HMM[4]。

2.1 隱馬可夫模型基本介紹

隱馬可夫模型(Hidden Markov Model, HMM)可以用來做時間序列資料的分析，目前應用上多為語音辨識、中文斷詞或光學字元辨識。HMM 是利用觀測值(observations)來推斷狀態(states)的一個算法，而狀態被稱為隱藏狀態，是因為我們看不到狀態，只看得到觀測值，所以實際上我們對狀態是不了解的，但我們知道狀態之間的轉移機率(Transition)，還有狀態對應到觀測值得發射機率(Emission)[3]。

2.2 隱馬可夫模型的三個主要問題

首先給定模型參數 $\lambda = (\pi, A, B)$ ，其中 A 為狀態轉換的機率矩陣， B 為特定狀態下的觀察值分佈機率矩陣， π 則為初始狀態的機率矩陣。當我們有觀察序列 $O = (O_1, O_2, \dots, O_t)$ 就能得出最有可能的隱藏狀態序列。

HMM 的三個主要問題，分別是：

1. 機率計算(Evaluation)：根據觀察序列 O 及模型參數 λ ，計算出此序列發生的機率 $P = (O | \lambda)$ ，使用 Forward Algorithm 或 Backward Algorithm 演算法來達成計算。
2. 最佳狀態序列(Decoding)：根據觀察序列 O 及模型參數 λ ，找出最有可能的狀態序列，使用的方法是 Viterbi Algorithm，從所有可能的狀態序列中，找出最有可能的一個。
3. 調整模型參數(Learning)：根據觀察序列 O ，調整模型參數 $\lambda = (\pi, A, B)$ ，使 $P = (O | \lambda)$ 值最大化，訓練 HMM 模型。

2.2.1 Forward Algorithm

Forward Algorithm 是透過變數 $\alpha_t(j)$ 由前往後，推算觀察值序列 O 在參數 λ 下的發生機率。

$\alpha_t(j)$ 定義為給定參數 λ ，在時間 t 時，狀態為 S_j ，且在時間 t 時之前的觀察值序列為 O_1, O_2, \dots, O_t 的機率，表示式如下：

$$\alpha_t(j) = P(O_1, O_2, \dots, O_t, q_t = S_j | \lambda) \quad (2.1)$$

而計算方式如下：

1. Initialization :

$$\alpha_1(j) = \pi_j b_j(O_1); \quad 1 \leq j \leq N \quad (2.2)$$

2. Recursion :

$$\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(O_t); \quad 1 \leq j \leq N, 1 < t \leq T \quad (2.3)$$

3. Termination :

$$P(O|\lambda) = \sum_{i=1}^N \alpha_T(i) \quad (2.4)$$

在式(2.2)中，先計算所有狀態在時間 $t=1$ 時的機率 $\alpha_t(j) = P(O_1, O_2, \dots, O_t, q_t = S_j | \lambda)$ ，就是 $\alpha_1(j) = \pi_j b_j(O_1)$ ，而在式(2.3)則是以時間 $t-1$ 的 $\alpha_{t-1}(i)$ ，來計算下一刻 t 的 $\alpha_t(j)$ 且要考慮所有的狀態，如圖 2-1 所示。而在式(2.4)是將所有的 $\alpha_t(j)$ 計算出來後，對於所有的可能計算出 $P(O|\lambda)$ 的機率。

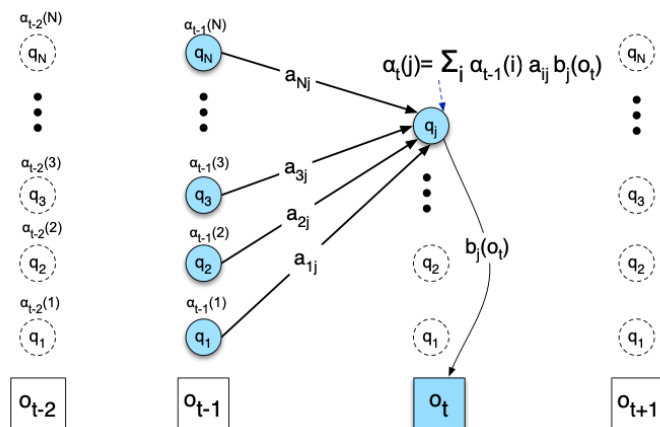


圖 2-1 Forward Algorithm 示意圖

2.2.2 Viterbi Algorithm

在 HMM 找最佳狀態序列的問題上，使用的方法是 Viterbi Algorithm，是利用 Dynamic Programing 的方式，在所有可能的狀態路徑中，找出最有可能的一個。

$$v_t(j) = \max_{q_1 \dots q_{t-1}} P(q_1, q_2, \dots, q_{t-1}, o_1, o_2, \dots, o_t, q_t = S_j | \lambda) \quad (2.5)$$

在此(2.5)式中，定義變數 $v_t(j)$ 代表考慮所有的狀態路徑 q_1, q_2, \dots, q_{t-1} ，在給定參數 λ 的情況下，時間 $t+1$ 之前的觀察值序列為 o_1, o_2, \dots, o_t 、時間 t 的狀態為 S_j 時，其最大的機率值。

實際計算方式如下：

1. Initialization :

$$\begin{aligned} v_1(j) &= \pi_j b_j(o_1); \quad 1 \leq j \leq N \\ bt_1(j) &= 0; \quad 1 \leq j \leq N \end{aligned} \quad (2.6)$$

2. Recursion :

$$\begin{aligned} v_t(j) &= \max_{1 \leq i \leq N} v_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T \\ bt_t(j) &= \operatorname{argmax}_{1 \leq i \leq N} v_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T \end{aligned} \quad (2.7)$$

3. Termination :

$$\begin{aligned} \text{The best score:} \quad P^* &= \max_{1 \leq i \leq N} v_T(i) \\ \text{The start of backtrace:} \quad qT^* &= \operatorname{argmax}_{1 \leq i \leq N} v_T(i) \end{aligned} \quad (2.8)$$

2.2.3 調整模型參數

訓練 HMM 模型中，最重要與最困難的部分就是調整模型參數 λ ，讓 $P = (O | \lambda)$ 的值最大，也稱為訓練模型。這裡會使用 Forward-backward Algorithm(也稱 EM Algorithm)。

Forward-backward Algorithm 的步驟如下：

1. 給定初值

2. E-step(expectation step)

估計目前的參數和訓練資料的可能性 $P(O | \lambda)$

3. M-Step(modification step)

調整目前的參數，使 $P(O | \lambda)$ 更大

4. 重複 E-Step 和 M-Step，直到 $P(O | \lambda)$ 不再變大

流程圖如下：

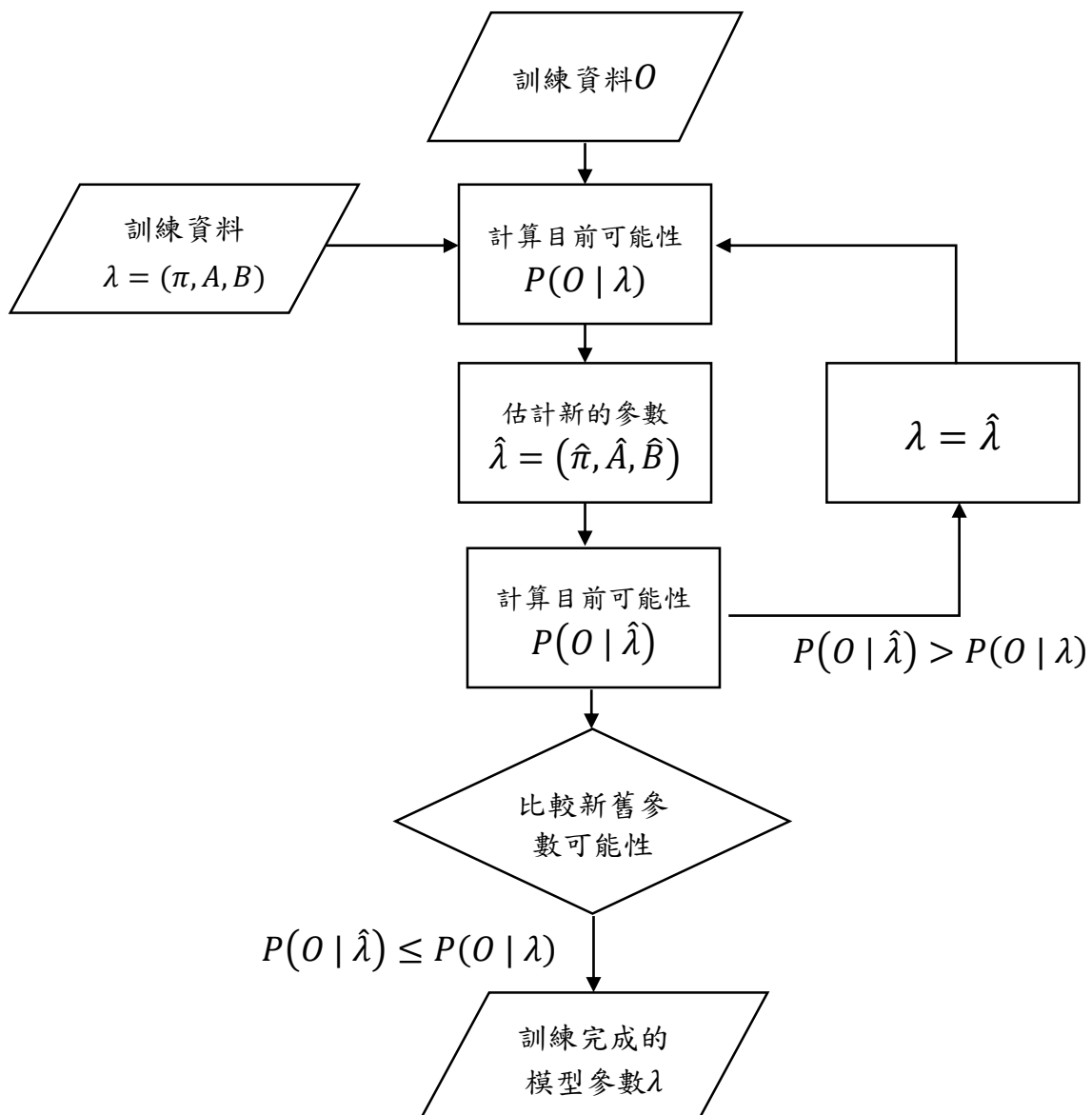


圖 2-2 Forward-backward Algorithm 流程圖

2.3 實作隱馬可夫模型

本節以 Java 程式執行實際範例，假設有三個狀態「Rainy」、「Sunny」和「Cloudy」，但我們無法直接觀察它們，它們對於我們是隱藏的，而我們只知道有一定的機率進行下列活動：「clean」和「walk」，就是觀察資料。下面程式執行將假設觀察序列 $O = (\text{clean} \rightarrow \text{walk} \rightarrow \text{walk} \rightarrow \text{clean})$ 及給定模型參數 $\lambda = (\pi, A, B)$ ，其中表 2-1 Table A 為狀態轉換的機率矩陣，表 2-2 Table B 為特定狀態下的觀察值分佈機率矩陣，初始狀態的機率矩陣 $\pi = (0.4, 0.3, 0.3)$ ，所有狀態轉換機率以圖 2-3 表示。

表 2-1 狀態轉換的機率矩陣 Table A

| | Rainy | Sunny | Cloudy |
|--------|-------|-------|--------|
| Rainy | 0.4 | 0.3 | 0.3 |
| Sunny | 0.6 | 0.3 | 0.1 |
| Cloudy | 0.2 | 0.5 | 0.3 |

表 2-2 特定狀態下的觀察值分佈機率矩陣 Table B

| | clean | walk |
|--------|-------|------|
| Rainy | 0.7 | 0.3 |
| Sunny | 0.3 | 0.7 |
| Cloudy | 0.5 | 0.5 |

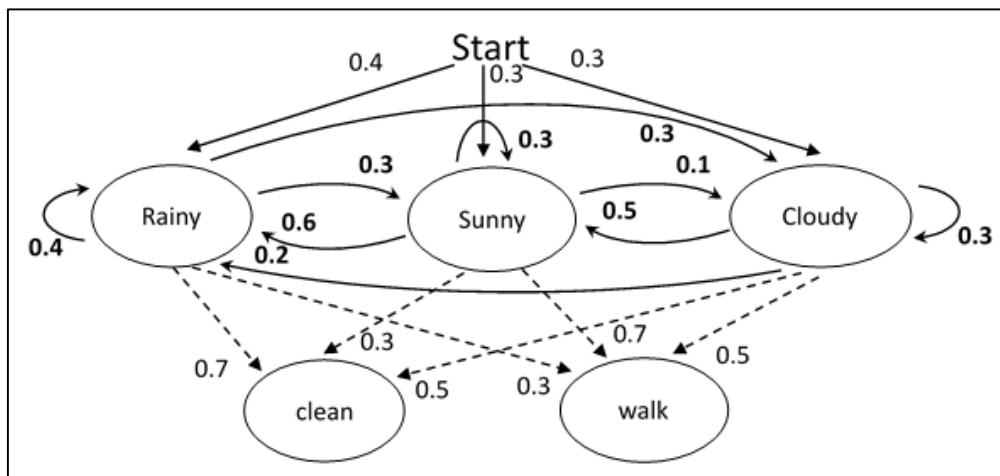


圖 2-3 所有狀態轉換機率

首先使用 Forward Algorithm 或 Backward Algorithm 演算法來計算 $P(O | \lambda)$ ，圖 2-4 為示意圖，圖 2-5 為程式執行結果，其中 fwd 宣告為二維陣列，儲存在目前觀察序列上，隱藏狀態的機率值加總，例如 fwd[1][0]代表在第二個觀察序列「walk」上，前一個分別不同的隱藏狀態對於目前隱藏狀態「Rain」的機率值加總。而由程式執行結果可以得出此序列的 $P(O | \lambda)$ 約為 0.065。

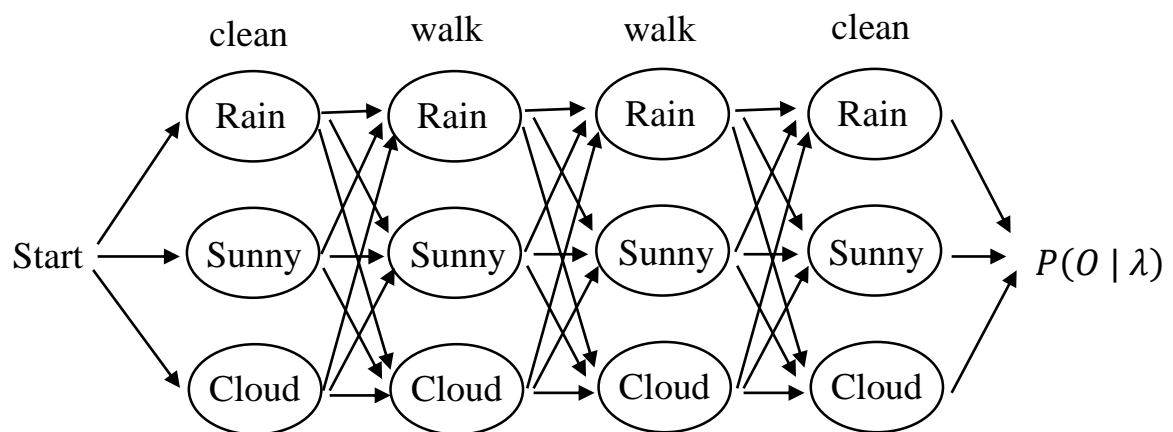


圖 2-4 計算 $P(O | \lambda)$ 示意圖

```

BlueJ: Terminal Window - HMM_java
Options
fwd[0][0]=0.27999999999999997
fwd[0][1]=0.09
fwd[0][2]=0.15
fwd[1][0]=0.0588
fwd[1][1]=0.13019999999999998
fwd[1][2]=0.06899999999999999
fwd[2][0]=0.034631999999999996
fwd[2][1]=0.06383999999999998
fwd[2][2]=0.025679999999999998
fwd[3][0]=0.04010495999999999
fwd[3][1]=0.012714479999999997
fwd[3][2]=0.012238799999999998
=====此序列機率=====
P(O|λ)= 0.065058239999999998

Can only enter input while your programming is running

```

圖 2-5 Forward Algorithm 程式執行結果

接下來找出最有可能的隱藏狀態序列，使用 Viterbi Algorithm 方法，也就是尋找當下狀態最大的機率值，圖 2-6 為示意圖，在計算結果上，最大機率值的狀態以黑色字與黑色箭頭表示。而圖 2-7 為程式執行結果，從假設的觀察序列 $O = (\text{clean} \rightarrow \text{walk} \rightarrow \text{walk} \rightarrow \text{clean})$ 及給定模型參數 $\lambda = (\pi, A, B)$ ，預測出的隱藏狀態序列為 $\text{Rainy} \rightarrow \text{Cloudy} \rightarrow \text{Sunny} \rightarrow \text{Rainy}$ 。

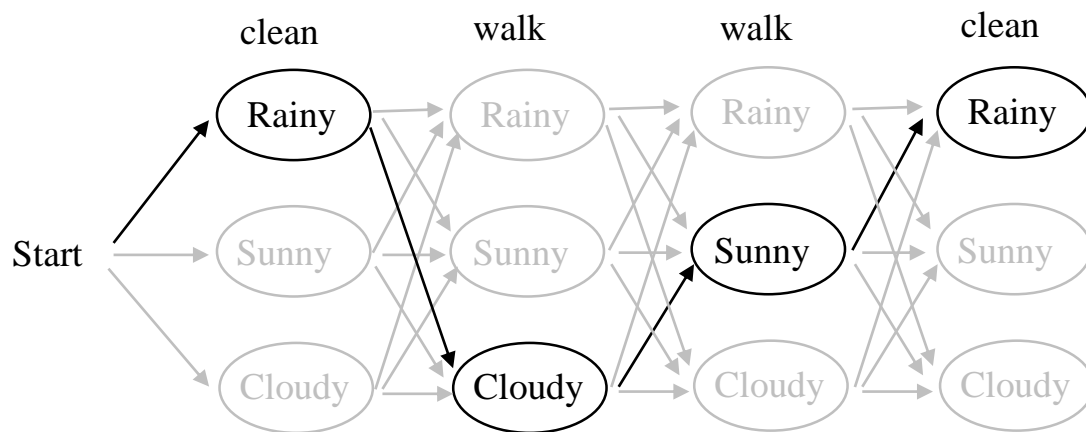


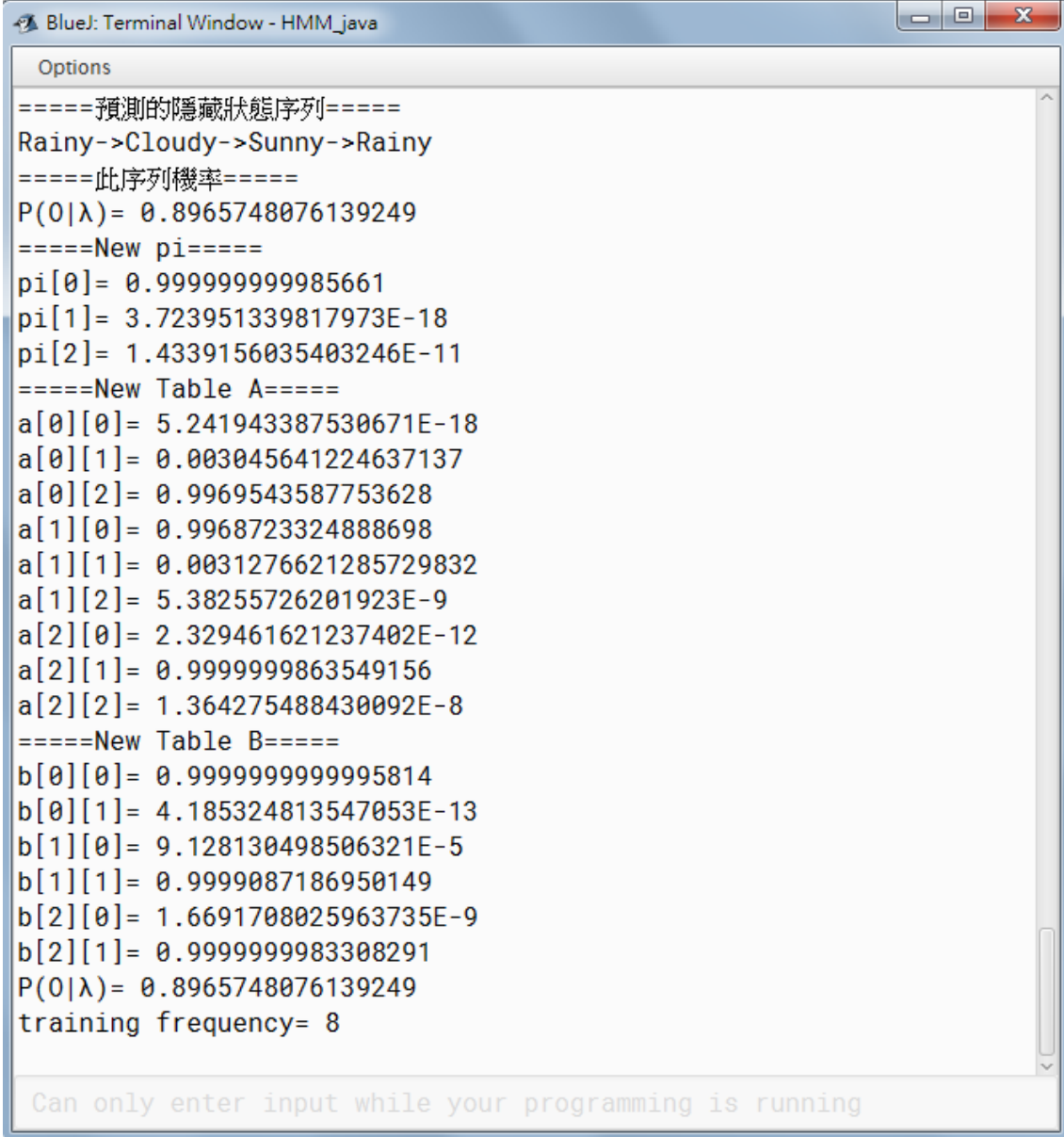
圖 2-6 使用 Viterbi Algorithm 找出隱藏狀態序列示意圖

```
BlueJ: Terminal Window - HMM_java
Options
=====預測的隱藏狀態序列=====
Rainy->Cloudy->Sunny->Rainy
=====此序列機率=====
P(O|λ)= 0.06505823999999998

Can only enter input while your programming is running
```

圖 2-7 預測隱藏狀態序列程式執行結果

最後是調整模型參數 $\lambda=(\pi, A, B)$ ，訓練 HMM 模型，讓 $P(O|\lambda)$ 的值最大且不再改變，使用 Forward-backward Algorithm。圖 2-8 為程式執行結果，可以看出當調整及訓練八次後的 $P(O|\lambda)$ 的值約為 0.89，且模型參數 Table A 與 Table B 機率矩陣內的值都趨近 1 與 0，初始狀態的機率矩陣則調整為 $\pi = (0.99, 3.7 \times 10^{-18}, 1.4 \times 10^{-11})$ 。



```
BlueJ: Terminal Window - HMM_java
Options
=====預測的隱藏狀態序列=====
Rainy->Cloudy->Sunny->Rainy
=====此序列機率=====
P(O|λ)= 0.8965748076139249
=====New pi=====
pi[0]= 0.9999999999985661
pi[1]= 3.723951339817973E-18
pi[2]= 1.4339156035403246E-11
=====New Table A=====
a[0][0]= 5.241943387530671E-18
a[0][1]= 0.003045641224637137
a[0][2]= 0.9969543587753628
a[1][0]= 0.9968723324888698
a[1][1]= 0.0031276621285729832
a[1][2]= 5.38255726201923E-9
a[2][0]= 2.329461621237402E-12
a[2][1]= 0.9999999863549156
a[2][2]= 1.364275488430092E-8
=====New Table B=====
b[0][0]= 0.9999999999995814
b[0][1]= 4.185324813547053E-13
b[1][0]= 9.128130498506321E-5
b[1][1]= 0.9999087186950149
b[2][0]= 1.6691708025963735E-9
b[2][1]= 0.999999983308291
P(O|λ)= 0.8965748076139249
training frequency= 8

Can only enter input while your programming is running
```

圖 2-8 調整模型參數程式執行結果

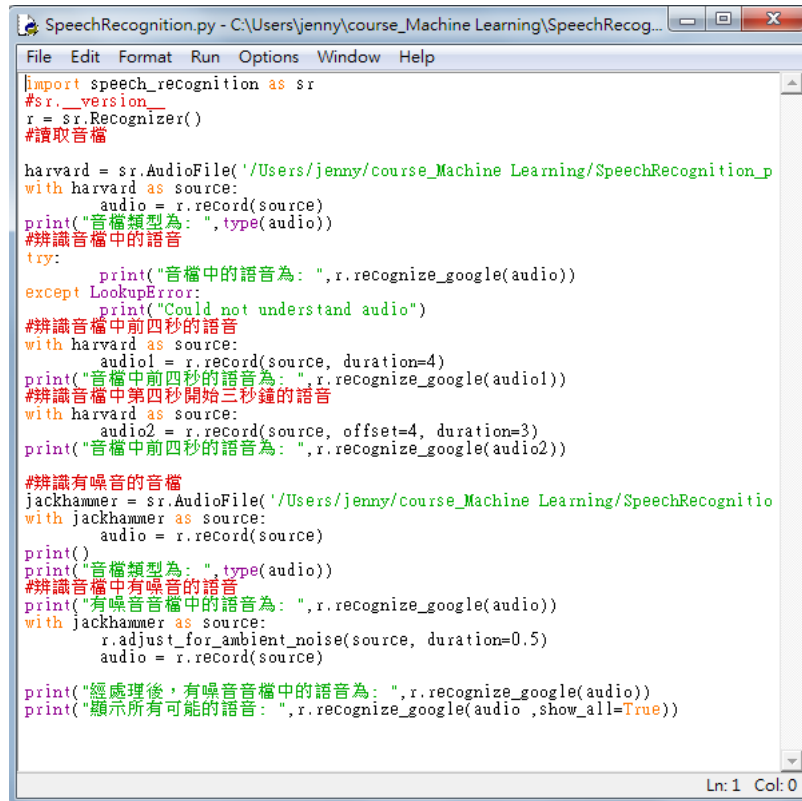
第三章 語音辨識

在語言學上，我們可以把人說話發出的聲音分成各種音節（syllable）。所以理論上，如果有一段錄音檔案，只要能分辨每一個音節發的音是哪些母音與子音，就能夠把這個語音檔辨識成文字。但這種「音節對應」的工作看似容易，但是實際上會遇到很多複雜的問題，以中文為例，兩個三聲的字連著念，前面的會讀成二聲，這會增加分辨過程的難度。

那 HMM 模型對語音辨識系統而言，語音的音節序列是看得到的訊號，而系統想要做的是推測出與其相對應的，看不到的「文字序列」，所以是 HMM 可以模擬的問題。隱馬可夫模型在語音辨識的應用，始於 1970 年 IBM 計畫(Jelinek)，而現今我們生活中可以看到的各種語音辨識系統，例如 Apple 的 Siri，Google 的 Voice search，背後都是以 HMM 作為基礎技術[2]。

3.1 實作語音辨識

本節是利用 Python 程式語言，並透過 Google 的語音辨識 API 做語音訊息轉文字。首先安裝 SpeechRecognition 套件，指令為(`pip3 install SpeechRecognition`)，若需要使用麥克風當作音源輸入，則必須安裝 PyAudio 模組，指令為(`pip3 install PyAudio`)。下圖 3-1 為讀入 wav 語音檔範例程式，其中不僅將語音轉為文字，還能透過參數「duration」與「offset」去辨識音檔中設定時間的語音，而也可以對有噪音的音檔做處理，或是顯示所有辨識可能的語音。圖 3-2 為程式執行結果。



```
SpeechRecognition.py - C:\Users\jenny\course_Machine Learning\SpeechRecog...
File Edit Format Run Options Window Help

import speech_recognition as sr
#sr.__version__
r = sr.Recognizer()
#讀取音檔

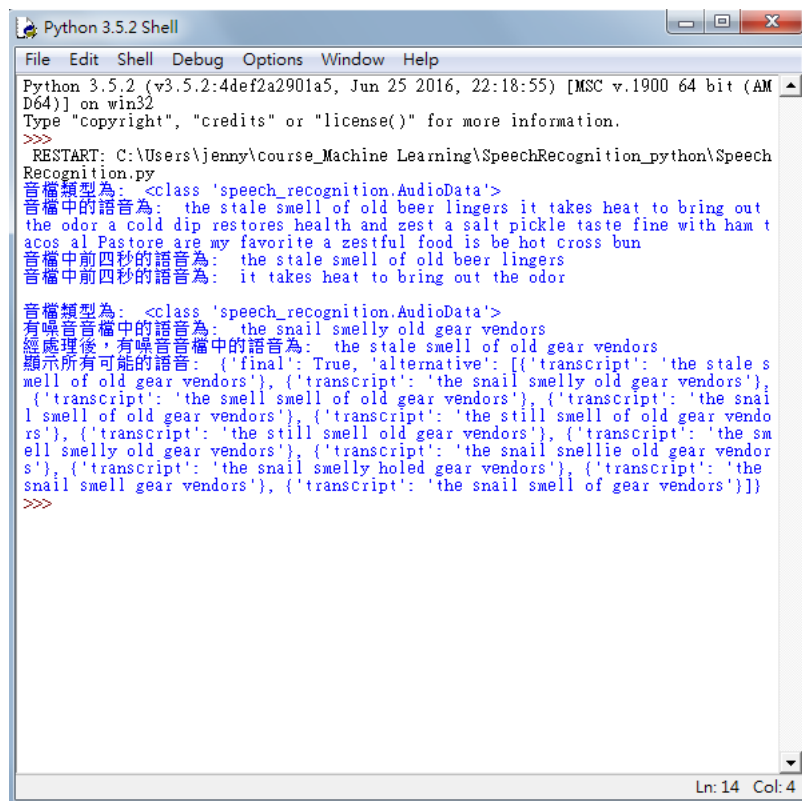
harvard = sr.AudioFile('/Users/jenny/course_Machine Learning/SpeechRecognition_p
with harvard as source:
    audio = r.record(source)
print("音檔類型為: ", type(audio))
#辨識音檔中的語音
try:
    print("音檔中的語音為: ", r.recognize_google(audio))
except LookupError:
    print("Could not understand audio")
#辨識音檔中前四秒的語音
with harvard as source:
    audiol = r.record(source, duration=4)
print("音檔中前四秒的語音為: ", r.recognize_google(audiol))
#辨識音檔中第四秒開始三秒鐘的語音
with harvard as source:
    audio2 = r.record(source, offset=4, duration=3)
print("音檔中前四秒的語音為: ", r.recognize_google(audio2))

#辨識有噪音的音檔
jackhammer = sr.AudioFile('/Users/jenny/course_Machine Learning/SpeechRecognitio
with jackhammer as source:
    audio = r.record(source)
print()
print("音檔類型為: ", type(audio))
#辨識音檔中有噪音的語音
print("有噪音音檔中的語音為: ", r.recognize_google(audio))
with jackhammer as source:
    r.adjust_for_ambient_noise(source, duration=0.5)
    audio = r.record(source)

print("經處理後，有噪音音檔中的語音為: ", r.recognize_google(audio))
print("顯示所有可能的語音: ", r.recognize_google(audio, show_all=True))

Ln: 1 Col: 0
```

圖 3-1 讀入 wav 語音檔範例程式



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help

Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:\Users\jenny\course_Machine Learning\SpeechRecognition_python\Speech
Recognition.py
音檔類型為: <class 'speech_recognition.AudioData'>
音檔中的語音為: the stale smell of old beer lingers it takes heat to bring out
the odor a cold dip restores health and zest a salt pickle taste fine with ham t
acos al Pastore are my favorite a zestful food is be hot cross bun
音檔中前四秒的語音為: the stale smell of old beer lingers
音檔中前四秒的語音為: it takes heat to bring out the odor

音檔類型為: <class 'speech_recognition.AudioData'>
有噪音音檔中的語音為: the snail smelly old gear vendors
經處理後，有噪音音檔中的語音為: the stale smell of old gear vendors
顯示所有可能的語音: ({'final': True, 'alternative': [{'transcript': 'the stale s
mell of old gear vendors'}, {'transcript': 'the snail smelly old gear vendors'},
{'transcript': 'the smell of old gear vendors'}, {'transcript': 'the snai
l smell of old gear vendors'}, {'transcript': 'the still smell of old gear vendo
rs'}, {'transcript': 'the still smell old gear vendors'}, {'transcript': 'the sm
ell smelly old gear vendors'}, {'transcript': 'the snail snellie old gear vendor
s'}, {'transcript': 'the snail smelly holed gear vendors'}, {'transcript': 'the
snail smell gear vendors'}, {'transcript': 'the snail smell of gear vendors'}])
>>>

Ln: 14 Col: 4
```

圖 3-2 語音辨識程式執行結果

第四章 結論

本論文報告是了解隱馬可夫模型的基本理論，與使用不同的演算法去解決隱馬可夫的三個主要問題，並假設簡單的觀察序列，去實作程式，預測隱藏狀態的序列，之後，訓練模型、調整參數。而因為實作的例子較於簡單，在訓練模型調整參數的次數大約都在十次上下，所以希望未來能嘗試去執行較複雜的例子。另外在語音辨識的應用上，是使用 Python 實作 Google 的語音辨識，將語音訊息轉文字，這部分主要是使用套件去執行，希望之後也能嘗試更複雜的研究，例如歌曲風格的辨識、模仿人類聲音音節分割等。

本報告讓我學習到了隱馬可夫模型的理論及如何有效解決問題與訓練模型，在學習的過程中了解許多相關的應用，並利用 Java 與 Python 語言做為程式執行工具。

參考文獻

- [1] 邱莉婷，適用於複音音樂之 HMM 音高追蹤器的改良，國立清華大學碩士論文，2012 年
- [2] 隱馬可夫模型：探索看不到的世界的數學工具。檢自 url:
<https://pansci.asia/archives/43586>
- [3] 顧正偉，利用多觀察值型隱馬可夫模型進行人體動作辨識，國立交通大學碩士論文，2005 年
- [4] Dan Jurafsky & James H. Martin.(2008), Speech and Language Processing.
- [5] J.-W. Kuo, H.-Y. Lo and H.-M. Wang, “Improved HMM/SVM methods for automatic phoneme segmentation”, in Proc. Interspeech, Antwerp, Belgium, pp. 2057-2060, 2007.