DSP Midterm

2011 Autumn

1. 要辨識L個獨立的字($w_1 \sim w_L$)，假設我們有1個observation，$\bar{o}$，這個$\bar{o}$可能是$w_1 \sim w_L$其中一個的語音，如果要辨識$\bar{o}$是哪一個字，就要分別去計算$P(\bar{o}|\lambda_k)$，$k=1\sim L$

如果不使用forward algorithm或Viterbi algorithm，那計算$P(\bar{o}|\lambda_k)$的時間複雜度是$O(state^{time})$，計算量太大，故使用以Dynamic programming為基礎的forward algorithm或Viterbi algorithm，以空間(儲存計算量)來換取時間，降低時間複雜度。

兩者的差異在於，forward algorithm是算出$\bar{o}$被λ_k這個model產生的機率，而Viterbi algorithm則是算出λ_k這個model產生$\bar{o}$的最可能路徑的機率。

也就是

Forward: $P(\bar{o}|\lambda_k)$ = P(λ_k產生$\bar{o}$的最可能路徑) + P(λ_k產生$\bar{o}$的其他路徑)

Viterbi: $P(\bar{o}, \bar{s}|\lambda_k)$ = P(λ_k產生$\bar{o}$的最可能路徑)

而 $\arg\max_k P(\bar{o}|\lambda_k) \sim \arg\max_k P(\bar{s}^*, \bar{o}|\lambda_k)$ 之所以成立是因為大部份有最高$P(\bar{s}, \bar{o}|\lambda_k)$的model也會有最高的$P(\bar{o}|\lambda_k)$

2. 若likelihood function (即$P(\bar{o}|\lambda)$)每次都會增加，就代表model的精確度不斷在上升，所以我認為只要iteration越多次，精確度就會上升。

3. (a) initial model

(b) $\sum_{O_t=walk} \gamma_t(i)$, i = Rainy or Sunny 代表在那段時間裡，state 是 rainy 或 sunny，而被拿到的 observation 是 walk 的次數。

例如 (shop, walk, clean, walk, walk, clean) 的 $\sum_{O_t=walk} \gamma_t(i) = 3$

(c) 把 (shop, walk) 代入 Alice 和 Bob 的 model，

Alice:

$\delta_1(rainy) = 0.6 \times 0.4 = 0.24$

$\delta_1(sunny) = 0.4 \times 0.4 = 0.16$

$\delta_2(rainy) = \max \begin{cases} \delta_1(rainy) \times 0.5 \\ \delta_1(sunny) \times 0.5 \end{cases} \times 0.2 = 0.12 \times 0.2 = 0.024$

$\delta_2(sunny) = \max \begin{cases} \delta_1(rainy) \times 0.5 \\ \delta_1(sunny) \times 0.5 \end{cases} \times 0.5 = 0.06$

$\Rightarrow P^* = \max\{\delta_2(rainy), \delta_2(sunny)\} = 0.06$

Bob:

$\delta_1(rainy) = 0.4 \times 0.4 = 0.16$

$\delta_1(sunny) = 0.6 \times 0.3 = 0.18$

$\delta_2(rainy) = \max \begin{cases} \delta_1(rainy) \times 0.5 \\ \delta_1(sunny) \times 0.5 \end{cases} \times 0.1 = 0.009$

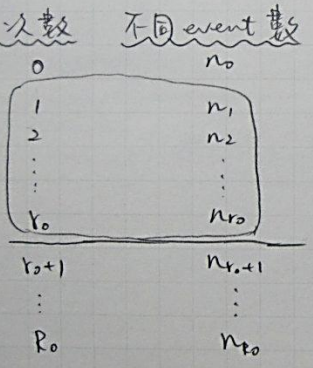
$\delta_2(sunny) = \max \begin{cases} \delta_1(rainy) \times 0.5 \\ \delta_1(sunny) \times 0.5 \end{cases} \times 0.5 = 0.045$

$P^* = \max\{\delta_2(rainy), \delta_2(sunny)\} = 0.045$

$\because P^*_{Alice} > P^*_{Bob}$

$\therefore$ (shop, walk) → Alice, (clean, clean) → Bob

4. Katz smoothing 主要是維持發生多次的事件次數不變，將發生不到多次事件的次數拿去給未發生的事件。但未發生事件的機率不盡然相同，所以未發生事件分配到的次數由這些事件的 next-lower-order model 來決定。以下例來說明分配次數給未發生事件的 procedure



① 決定 r_0，發生次數超過 r_0 的事件不受影響

② 用 $r^* = (r+1)\cdot\frac{n_{r+1}}{n_r}$ 計算出 d_r ($r \to d_r$)

③ 發生 1 次的總數由 $1\times n_1 \to 1\times d_1\times n_1$

⋮

發生 r_0 次的總數由 $r_0\times n_{r_0} \to r_0\times d_{r_0}\times n_{r_0}$

④ 由 1~r_0 次事件貢獻的 $\sum_{r=1}^{r_0} n_r(1-d_r)r = n_1$ 分配給 unseen events

⑤ 由 next-lower-order model 決定各 unseen event 分配到的次數

5. (a) 因為 Fourier transform 無法得到 local 的資訊，若把整段語音訊號拿來做 FT，整段訊號的性質都會混在一起。而做 windowing 可以把一小段時間的訊號取出來分析，這一小段時間裡的聲音可能是相似的，這樣的分析才能獲得有意義的資訊。

(b) Pre-emphasis 是將高頻訊號的振幅放大，這麼做的理由有二：

① 由於產生語音的系統特性，語音訊號的頻率每增加 10 倍，振幅就會下降 20 dB，故將高頻訊號振幅放大至與低頻訊號同 level

② 人耳對於 1kHz 以上的訊號較為敏感，為模擬此項特性故做 pre-emphasis

6.

(a) $H(x) = -\sum_{i=1}^{M} p_i[\log(p_i)]$ 是 random variable x 攜帶訊息的平均量，可由其公式 $H(x) = \sum_{i=1}^{M} p_i I(x_i)$ 看出，$I(x_i)$ 是 $X=x_i$ 攜帶的訊息量，p_i 是 $X=x_i$ 發生的機率，所以 $H(x)$ 就是 $I(x_i)$, $i=1\sim M$ 的 mean。

由另一觀點來看，$H(x)$ 同時表示了產生 X 的來源的不確定性，若這個來源產生 X 的值越不確定（即各種 x_i 的機率越一致），那麼每個 $I(x_i)$ 的值越高，$H(x)$ 也就越高，信息量跟不確定性的關係再度成立。

(b) 前面已敘述 $H(x)$ 代表不確定性，decision tree 的目的是要讓性質相近的 triphone 被分到同一群以做到 parameter sharing，而"性質相近"可以用 $H(x)$ 來衡量，以將 node n 分成 a, b 的例子來說明

首先定義 $\bar{H}_n = \{-\sum_i p(c_i|n)\log[p(c_i|n)]\}p(n)$；$p(c_i|n)$ 是 n裡的 sample 是 c_i 的機率，$p(n)$ 是 $\frac{\text{n裡的 sample}}{\text{total sample}}$

$\Rightarrow \bar{H}_n$ 表示了 n 這個 node 裡 sample 成份的不確定度，

$\bar{H}_n$ 越高表示 n 裡的組成越複雜（我們顯然希望 $\bar{H}_n$ 低一些）

而要產生更好的 decision tree 就是要讓 $\bar{H}_a$ 和 $\bar{H}_b$ 的和比 $\bar{H}_n$ 低，所以我們關注的量是 $\bar{H}_n - [\bar{H}_a + \bar{H}_b]$，也就是 $\Delta\bar{H}_n$，在產生 decision tree 時只要在每一次 split 都使 $\Delta\bar{H}_n$ 最大即可。

7. LBG algorithm 是 train VQ code book 的演算法，該算法步驟如下

Step 1. 先初始化一個 L=1, 1-vector 的 codebook

$$\bar{v} = \frac{1}{N}\sum_j \bar{x}_j$$ （即取所有點的平均）

Step 2. 將 L 變成 2 倍，先取好兩倍 $\bar{v}_k$, $k=1\sim L$

Step 3. 用 K-means algorithm 來 train L-vector codebook

Step 4. 結束，或回到 step 2

LBG algorithm 之所以比 K-means 好是因為 K-means 初始的 L 個 $\bar{v}_k$ 是亂選的，最後 train 出來的 codebook 可能是 local optimal 的 codebook，而 LBG 則是先從平均的點開始 train，得到結果也就不那麼受 initial condition 影響了。