二、距離與相似度

2 – 1. 歐幾里得距離

距離度量（distance measure）是所有辨識系統上最基本的一環，而不同的度量方式，亦會導致不同的辨識比對結果。在眾多的距離度量公式中，最傳統的度量法稱之為「歐幾里得距離」（Euclidean distance）。假設在 d 維空間中有兩點**a** = [a_1, a_2, …, a_d]，**b** = [b_1, b_2, …, b_d]，則其歐幾里得距離可表示為：

$$dist(\mathbf{a},\mathbf{b}) = \sqrt{\sum_{1}^{d}(a_i - b_i)^2}$$

2 – 2. L-p Norms

我們可以把上述的歐幾里得距離再加以延伸，便可得出他的一般通式：L-p norms。L-p norms 又稱 Minkowski distance，假設在 d 維空間中有兩點**a** = [a_1, a_2, …, a_d]，**b** = [b_1, b_2, …, b_d]，則其 Minkowski distance 可表示為：

$$dist(\mathbf{a},\mathbf{b}) = \left(\sum_{1}^{d}|a_i - b_i|^p\right)^{\frac{1}{p}}$$

當 p = 1，*dist*(**a**,**b**) 表示**a**、**b** 間各維度絕對值總和，又稱 city block distance 或 taxicab distance。

當 p = 2，*dist*(**a**,**b**) 即為歐幾里得距離。

當 p = ∞，*dist*(**a**,**b**) = max| a_i – b_i |，稱為 maximum distance。

範例：

```
a=1:5;b=5:-0.2:4.2;
v=a-b;
norm(v,1)   % L-1 norm
norm(v,2)   % L-2 norm
norm(v,inf) % L-inf norm
```

```
ans =
    9.6000
ans =
    5.2154
ans =
     4
```

2 – 3. 餘弦相似度

餘弦相似度（cosine similarity）乃是傳統文件分類中，最常被拿來度量文件間距離的基本度量方法，其以兩個 d 維向量間的角度差異來度量該向量間的距離，所得數據介於 0 ~ 1 之間，當兩向量角度越相近時，所求出的餘弦距離越接近 1；反之，則越接近 0。假設在 d 維空間中有兩點 $\mathbf{a} = [a_1, a_2, \ldots, a_d]$，$\mathbf{b} = [b_1, b_2, \ldots, b_d]$，則其餘弦相似度可表示為：

$$sim(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}||\mathbf{b}|}$$

範例：

```
a=[1 1 1]; b=[1 0 0];
cosineDistance = dot(a,b) / (norm(a)*norm(b))

cosineDistance =
    0.5774
```

2 – 4. 最長相同子字串

假如待測兩向量間的維度不相同，我們便無法計算他們之間的 L-p Norms，這時最常被用來度量 1-d 或 2-d 的字串間距離的方式便是最長相同子字串（longest common subsequence，簡稱 LCS）[24]。

在解釋 LCS 之前，我們要先來解釋何謂子自串（subsequence），假設有一個字串 $\mathbf{x} = [x_1, x_2, \ldots, x_m]$，有另一個字串 $\mathbf{z} = [z_1, z_2, \ldots, z_k]$代表 X 的子字串，那麼我們必定能夠找到一組遞增的索引（index）$[i_1, i_2, \ldots, i_k]$，使得：

$$x_{i_j} = z_j \quad , j = 1, 2, \ldots, k$$

舉例來說，**x** = [A, C, D, D, E, A, D] ，**z** = [C, E, A, D]，則 i = [2, 5, 6, 7]。

假設現在有 **x**, **y** 兩個字串，若存在一個字串 **z** 同時為 **x** 與 **y** 的子字串，那麼 **z** 便稱為 **x**, **y** 的相同子字串（common subsequence）。因此顧名思義，LCS 的目的則是找出在所有的 **z** 中，長度最長的字串。假設 **x** = [A, B, C, D, B, A, B]，**y** = [B, D, C, A, A, B]，則 [B, C, A]、[B, A, B]、[B, D, A, B]均可以稱為 **x**, **y** 的相同子字串，不過因為 [B, D, A, B]擁有最長的長度，因此 LCS（**x**, **y**）= [B, D, A, B]。

根據 LCS 的特性，我們可以發現 LCS 具有下面三個特點：

假設 $\mathbf{x}_m = [x_1, x_2, \ldots, x_m]$， $\mathbf{y}_n = [y_1, y_2, \ldots, y_n]$，最長相同子字串 $\mathbf{z}_k = [z_1, z_2, \ldots, z_k]$：

1. 假如 $x_m = y_n$，則 $z_k = x_m = y_n$ 且 LCS（$\mathbf{x}_{m-1}$, $\mathbf{y}_{n-1}$）= $\mathbf{z}_{k-1}$。
2. 假如 $x_m \neq y_n$ 且 $z_k \neq x_m$，則 LCS（$\mathbf{x}_{m-1}$, $\mathbf{y}_n$）= $\mathbf{z}_k$。
3. 假如 $x_m \neq y_n$ 且 $z_k \neq y_n$，則 LCS（$\mathbf{x}_m$, $\mathbf{y}_{n-1}$）= $\mathbf{z}_k$。

根據上面的特點，我們可以將 LCS（$\mathbf{x}_m$, $\mathbf{y}_n$）這個問題簡化成 LCS（$\mathbf{x}_{m-1}$, $\mathbf{y}_{n-1}$）、LCS（$\mathbf{x}_{m-1}$, $\mathbf{y}_n$）或 LCS（$\mathbf{x}_m$, $\mathbf{y}_{n-1}$），而每個子問題又可簡化成更小的 LCS 問題，因此我們可以採用一種遞迴式（recursive）的方法來解決 LCS 問題。

在這邊我們先定義一個變數 $c(i,j)$ 表示 LCS（$\mathbf{x}_i$, $\mathbf{y}_j$）的長度，則我們可以獲得下面的通式：

$$c(i,j) = \begin{cases} 0 & \text{假如} i=0 \text{或} j=0 \\ c(i-1, j-1)+1 & \text{假如} i,j>0 \text{且} x_i = y_j \\ \max(c(i,j-1), c(i-1,j-1)) & \text{假如} i,j>0 \text{且} x_i \neq y_j \end{cases} \qquad (2\text{-}4.1)$$

透過動態規劃（dynamic programming）的方式，我們可以在 O（mn）的時間內得出 LCS，我們將全部的 $c(i,j)$ 儲存於一個 c(0..m, 0..n) 的矩陣中，然後以另一個矩陣 parent（1..m, 1..n）紀錄通式（2-4.1）中的情形：

```
LCS_Length（x, y）
1. m = length（x）
2. n = length （y）
3. for i = 1: m
4.       c（i, 0）= 0
5. for j = 1: n
6.       c（0, j）= 0
```

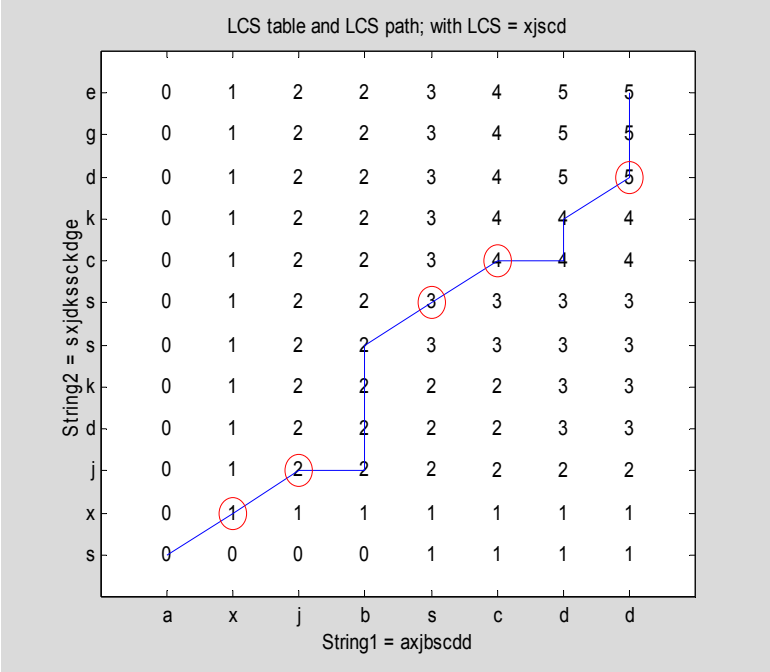
```
7. for i = 1: m
8.       for j = 1: n
9.          if xi = yj
10.            c (i, j) = c (i-1, j-1) + 1
11.         parent (i, j) = " ↖"
12.      elseif c (i-1, j) ≧ c (i, j-1)
13.         c (i, j) = c (i-1, j)
14.            parent (i, j) = " ↑ "
15.         else
16.            c (i, j) = c (i, j-1)
17.            parent (i, j) = " ← "
18. return c and parent
```

在得到 c 與 parent 這兩個矩陣後，我們只要從矩陣的最後端一路往前回溯，即可得到 LCS。

```
Print_LCS (parent, x, i, j)
1. if i =0 or j = 0
2.       return
3. if parent (i, j) = " ↖"
4.       Print_LCS (parent, x, i-1, j-1)
5.       print xi
6. elseif parent (i, j) = " ↑"
7.       Print_LCS (parent, x, i-1, j)
8. else
9.       Print_LCS (parent, x, i-1, j-1)
```

範例：

```
lcs;
```



2 – 5. 最長相同連續子字串

最長相同連續子字串（longest common consecutive subsequence）簡稱 LCCS，它的基本含意乃是由最長相同子字串（LCS）衍生而來，其最大的不同在於它的相同子字串（common subsequence）必須是連續的。

假設有一個字串 **x** = [x_1, x_2, …, x_m]，有另一個字串 **z** = [z_1, z_2, …, z_k]代表 **x** 的連續子字串（consecutive subsequence），那麼我們必定能夠找到一組連續遞增的索引 idx = [i, i+1, i+2, …, i+k-1]，使得：

$$x_{i+j-1} = z_j \quad , \quad j = 1, 2, ..., k$$

舉例來說，**x** = [A, C, D, D, E, A, D]，**z** = [C, D, D, E]，則 idx = [2, 3, 4, 5]。

倘若現在有 **x**, **y** 兩個字串，假如存在一個字串 **z** 同時為 **x** 與 **y** 的連續子字串，那麼 **z** 便稱為 **x**, **y** 的相同連續子字串（common consecutive subsequence）。而 LCCS 的目的便是找出長度最長的相同連續子字串。假設 **x** = [A, B, C, D, B, A, B]，**y** = [B, C, D, A, A, B]，則 [A, B]、[C, D]、[B, C, D]均可以稱為 **x**, **y** 的相同連續子字串，其中因為 [B, C, D]擁有最長的長度，因此 LCCS（**x**, **y**）= [B, C, D]。

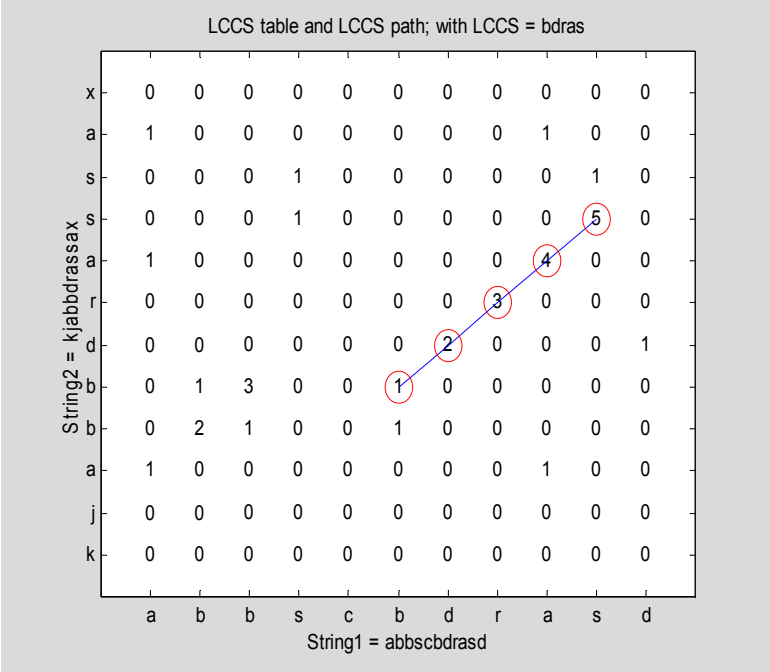
基於 LCCS 與 LCS 相當類似的定義，我們同樣可以採用類似的動態規劃的方式求出 LCCS：

```
LCCS_Length (x, y)
1.  m = length (x)
2.  n = length  (y)
3.  for i = 1: m
4.       c (i, 0) = 0
5.  for j = 1: n
6.       c (0, j) = 0
7.  for i = 1: m
8.       for j = 1: n
9.          if xi = yj
10.            c (i, j) = c (i-1, j-1) + 1
11.           else
12.              c (i, j) = 0
13.    return c
```

由於 LCCS 的條件限制比 LCS 還嚴格，因此 LCCS_Length（x, y）比 LCS_Length（x, y）更加簡單，再由 c 矩陣回溯求 LCCS 時，也僅需考慮 45°斜線上的路徑。

範例：

```
lccs;
```



2 – 6. 動態時間扭曲

動態時間扭曲（dynamic time warping）[14] 簡稱 DTW，是語音訊號處理上很常被使用的一種相似度估測方式，其主要的精神便是希望提供一種在時間軸上富有更大彈性的相似度比對方法，使測試資料（通常以字串的方式表現）能透過在時間軸上扭曲（即伸展或壓縮），找到與參考資料間最小誤差的非線性對應。

假設我們的測試資料長度為 I，參考資料長度為 J：

測試資料：$t(1), t(2), t(3), ..., t(I)$
參考資料：$r(1), r(2), r(3), ..., r(J)$

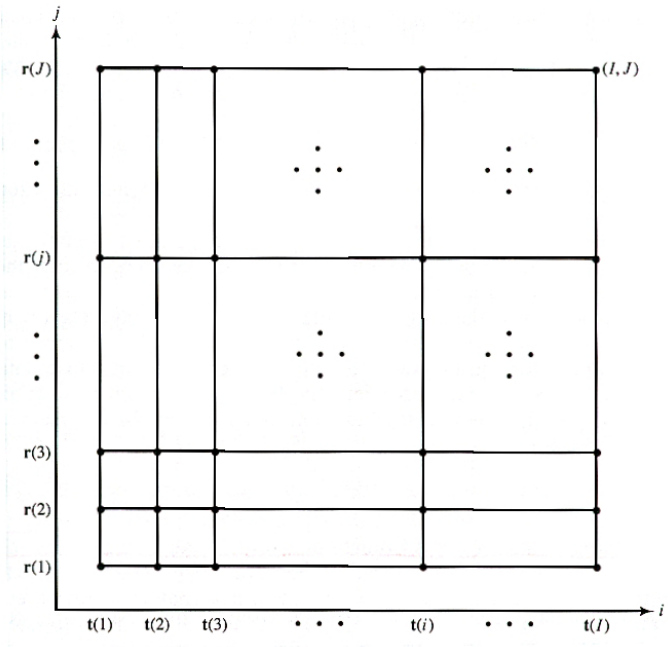


圖 2-6.a：動態時間扭曲比對示意圖 [14]

圖 2-6.a 是常見的動態時間扭曲比對示意圖。DTW 的目的便是在 **t**、**r** 構成的平面上找出一條最佳（測試資料與參考資料間的距離 D 為最小）的對應路徑 $path(i_k, j_k)$，使得 $t(i_k)$對應到 $r(j_k)$，其中，k = 1, 2, …, K，且 i_k 與 j_k 都必須遞增。

$$D = \sum_{k=1}^{K} d(i_k, j_k)$$

$$d(i_k, j_k) = dist(t(i_k), r(j_k))$$

其中 $d(i_k, j_k)$可以為任意一種距離測量方式，最常見的就是歐幾里得距離。

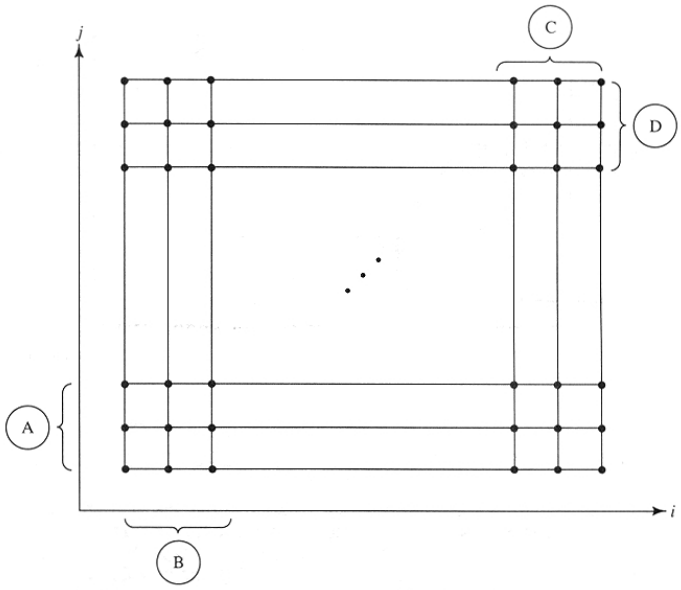


圖 2-6.b：DTW 彈性起始點與終點示意圖 [14]

在實際運算上，我們可以透過動態規劃的方式找出最佳的 path。首先我們先定義出可能的起始點與終點，如圖 2-6.b 所示，A、D 分別為參考資料可能的起點範圍與終點範圍；B、C 分別為測試資料可能的起點範圍與終點範圍。因此，我們可以定義出在起始點與終點的 D：

$$D(i_k, 0) = 0,\ i_k \in B$$
$$D(i_k, 0) = Inf,\ i_k \notin B$$
$$D(0, j_k) = 0,\ j_k \in A$$
$$D(0, j_k) = Inf,\ j_k \notin A$$

至於終點範圍的限制，可以在動態規劃最後回溯最佳路徑時予以限制。另外，基於局部最佳值能導致整體最佳值的概念，定義出每一點 $D(i_k, j_k)$ 可能的路徑來源 $parent(i_k, j_k)$：

$$D(i_k, j_k) = parent(i_k, j_k) + d(i_k, j_k)$$
$$parent(i_k, j_k) = \min(D(p,q))\ , p \le i_k, q \le j_k$$
$$d(i_k, j_k) = dist(t(i_k), r(j_k))$$

其中，p 與 q 的限制可以根據各種不同的比對資料及問題類型予以變化。圖 2-6.c 為幾種在語音辨識上較常見的 p、q 限制：

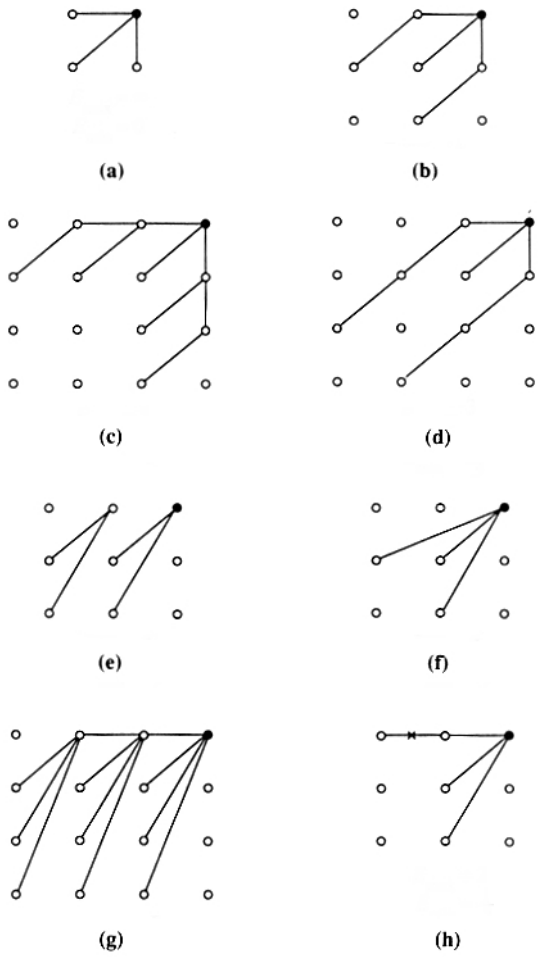
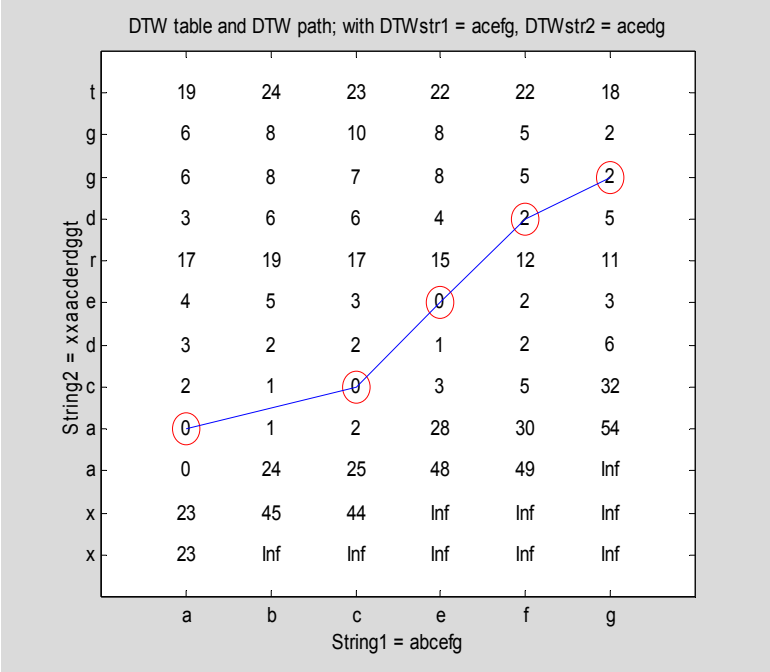


圖 2-6.c：常見的 DTW 限制條件 [14]

範例：

```
str1 = 'abcefg';
str2 = 'xxaacderdggt';
win1 = 2;
win2 = 2;
plot_opt = 1;
[minDistance, DTWpath, DTWtable] = dtw(str1,str2,win1,win2, plot_opt);
```


DTW table and DTW path; with DTWstr1 = acefg, DTWstr2 = acedg
t 19 24 23 22 22 18
g 6 8 10 8 5 2
g 6 8 7 8 5 2
d 3 6 6 4 2 5
r 17 19 17 15 12 11
e 4 5 3 0 2 3
d 3 2 2 1 2 6
c 2 1 0 3 5 32
a 0 1 2 28 30 54
a 0 24 25 48 49 Inf
x 23 45 44 Inf Inf Inf
x 23 Inf Inf Inf Inf Inf
String2 = xxaacderdggt
a b c e f g
String1 = abcefg