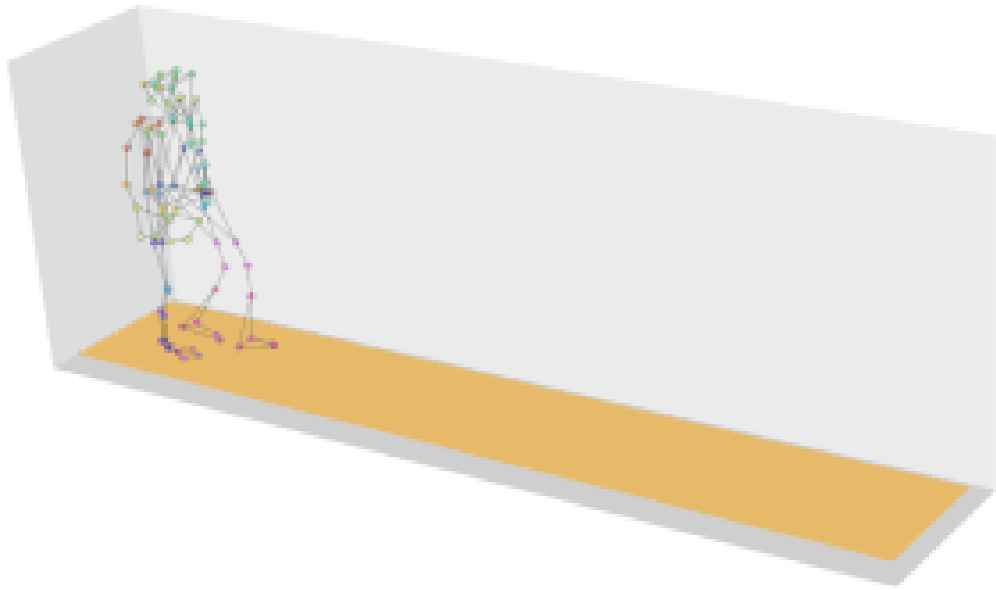


Algoritmo di Time Warping dinamico (DTW) nelle serie storiche

Aggiornato: 25 giugno 2022

L'articolo contiene informazioni sull'algoritmo Dynamic Time Warping (DTW).



Due ripetizioni di una sequenza di camminata sono state registrate utilizzando un sistema di motion capture. Mentre ci sono differenze nella velocità di camminata tra le ripetizioni, i percorsi spaziali degli arti rimangono molto simili. [Crediti](#)

introduzione

La frase "distorsione dinamica del tempo", a prima vista, potrebbe evocare immagini di Marty McFly alla guida della sua DeLorean a 88 MPH nella serie *Ritorno al futuro*. Purtroppo, la distorsione temporale dinamica non comporta il viaggio nel tempo; invece, è una tecnica utilizzata per confrontare dinamicamente i dati delle serie temporali quando gli indici temporali tra i punti dati di confronto non si sincronizzano perfettamente.

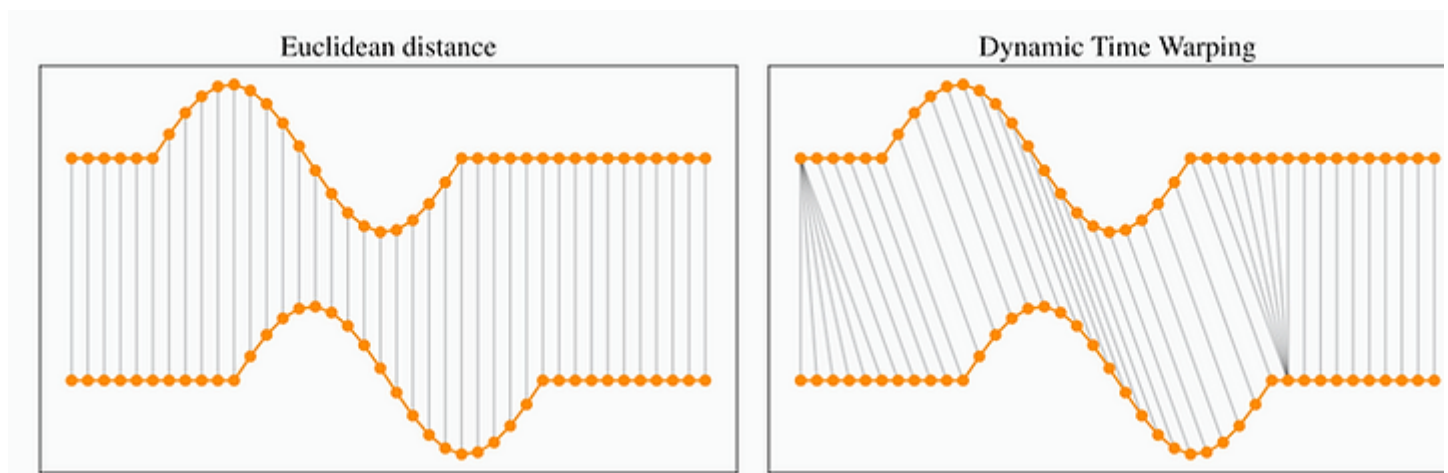


Come esploreremo di seguito, uno degli usi più salienti del time warping dinamico è nel riconoscimento vocale, ovvero determinare se una frase corrisponde a un'altra, anche se la frase viene pronunciata più velocemente o più lentamente del suo confronto. Puoi immaginare che questo sia utile per identificare le "parole di risveglio" utilizzate per attivare il tuo dispositivo Google Home o Amazon Alexa, anche se il tuo discorso è lento perché non hai ancora bevuto la tua tazza di caffè quotidiana.

Distorsione temporale dinamica (DTW)

Nell'analisi delle serie temporali, Dynamic Time Warping (DTW) è uno degli algoritmi per misurare la somiglianza tra due sequenze temporali di serie temporali, che possono variare in velocità. L'idea principale di DTW è calcolare la distanza dalla corrispondenza di elementi simili tra serie temporali. Utilizza la tecnica di programmazione dinamica per trovare la corrispondenza temporale ottimale tra elementi di due serie temporali.

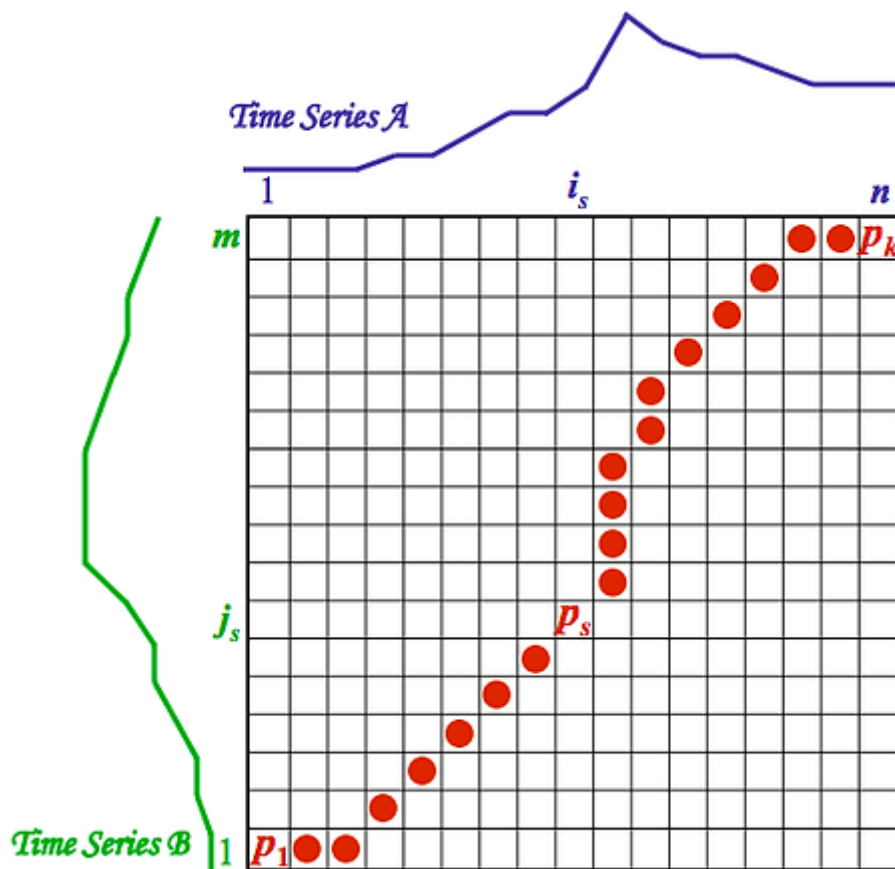
Ad esempio, le somiglianze nella deambulazione potrebbero essere rilevate utilizzando DTW, anche se una persona camminava più velocemente dell'altra o se durante un'osservazione si verificavano accelerazioni e decelerazioni. DTW è stato applicato a sequenze temporali di dati video, audio e grafici: in effetti, qualsiasi dato che può essere trasformato in una sequenza lineare può essere analizzato con DTW. Un'applicazione ben nota è stata il riconoscimento vocale automatico, per far fronte a diverse velocità di conversazione. Altre applicazioni includono il riconoscimento del parlante e il riconoscimento della firma online. Può essere utilizzato anche in applicazioni di abbinamento di forme parziali.



L'obiettivo dei metodi di confronto delle serie temporali è produrre una metrica di distanza tra due serie temporali di input. La somiglianza o la dissomiglianza di due serie temporali viene in genere calcolata convertendo i dati in vettori e calcolando la distanza euclidea tra quei punti nello spazio vettoriale.

In generale, DTW è un metodo che calcola una corrispondenza ottimale tra due date sequenze (es. serie temporali) con determinate restrizioni e regole:

1. Ogni indice della prima sequenza deve essere abbinato a uno o più indici dell'altra sequenza e viceversa
2. Il primo indice della prima sequenza deve essere abbinato al primo indice dell'altra sequenza (ma non deve essere la sua unica corrispondenza)
3. L'ultimo indice della prima sequenza deve essere abbinato all'ultimo indice dell'altra sequenza (ma non deve essere la sua unica corrispondenza)
4. La mappatura degli indici dalla prima sequenza agli indici dell'altra sequenza deve essere monotonicamente crescente, e viceversa, cioè se $j > i$ sono indici della prima sequenza, allora non devono esserci due indici $l > k$ nell'altra sequenza, in modo tale che l'indice i sia abbinato all'indice l e l'indice j sia abbinato all'indice k e viceversa.



Tempo normalizzato distanza tra Serie A e Serie B. Crediti .

Comprendi matematicamente DTW

Supponiamo di avere due sequenze come la seguente:

$$X = x_1, x_2, \dots, x_i, \dots, x_n$$

$$Y = y_1, y_2, \dots, y_j, \dots, y_m$$

Le sequenze X e Y possono essere disposte in modo da formare una griglia n per m, dove ogni punto (i, j) è l'allineamento tra x_i e y_j .

Un percorso di deformazione W mappa gli elementi di X e Y per minimizzare la *distanza* tra di loro. W è una sequenza di punti della griglia (i,j). Vedremo un esempio del percorso di deformazione più avanti.

Warping Path e distanza DTW

Il percorso ottimale per (i_k, j_k) può essere calcolato da:

$$D_{min}(i_k, j_k) = \min_{i_{k-1}, j_{k-1}} D_{min}(i_{k-1}, j_{k-1}) + d(i_k, j_k | i_{k-1}, j_{k-1})$$

dove d è la distanza euclidea. Quindi, il costo complessivo del percorso può essere calcolato come:

$$D = \sum_k d(i_k, j_k)$$

Restrizioni sulla funzione Warping

Il percorso di deformazione viene trovato utilizzando un approccio di programmazione dinamica per allineare due sequenze. Attraversare tutti i percorsi possibili è "combinatoriamente esplosivo". Pertanto, ai fini dell'efficienza, è importante limitare il numero di possibili percorsi di deformazione, e quindi vengono delineati i seguenti vincoli:

- **Condizione al contorno** : questo vincolo garantisce che il percorso di curvatura inizi con i punti iniziali di entrambi i segnali e termini con i loro punti finali.

$$i_1 = 1, i_k = n \quad \text{and} \quad j_1 = 1, j_k = m$$

- **Condizione di monotonicità** : questo vincolo preserva l'ordine temporale dei punti (non tornando indietro nel tempo).

$$i_{t-1} \leq i_t \quad \text{and} \quad j_{t-1} \leq j_t$$

- **Condizione di continuità (dimensione del passo)** : questo vincolo limita le transizioni del percorso a punti adiacenti nel tempo (non salti nel tempo).

$$i_t - i_{t-1} \leq 1 \quad \text{and} \quad j_t - j_{t-1} \leq 1$$

Oltre ai tre vincoli di cui sopra, esistono altre condizioni meno frequenti per un percorso di deformazione ammissibile:

- **Condizione della finestra di deformazione** : i punti consentiti possono essere limitati in modo che rientrino in una data finestra di deformazione di larghezza ω (un numero intero positivo).

$$|i_t - j_t| \leq \omega$$

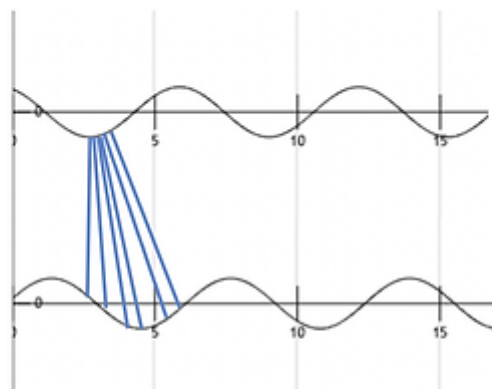
- **Condizione di pendenza** : il percorso di deformazione può essere vincolato limitando la pendenza e di conseguenza evitando movimenti estremi in una direzione.

Un percorso di deformazione accettabile ha combinazioni di mosse **del re degli scacchi** che sono:

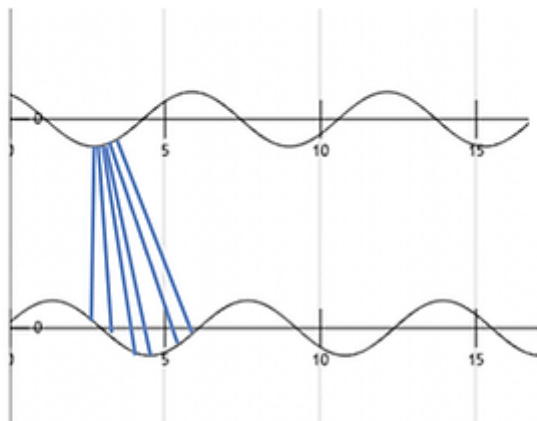
- Spostamenti orizzontali: $(i, j) \rightarrow (i, j+1)$
- Movimenti verticali: $(i, j) \rightarrow (i+1, j)$
- Mosse diagonali: $(i, j) \rightarrow (i+1, j+1)$

Perché abbiamo bisogno di DTW?

Qualsiasi serie di due tempi può essere confrontata utilizzando la distanza euclidea o altre distanze simili su base uno a uno sull'asse del tempo. L'ampiezza della prima serie temporale all'istante T verrà confrontata con l'ampiezza della seconda serie temporale all'istante T. Ciò si tradurrà in un punteggio di confronto e somiglianza molto scarso anche se le due serie temporali sono molto simili nella forma ma fuori di fase nel tempo.



DTW confronta l'ampiezza del primo segnale all'istante T con l'ampiezza del secondo segnale all'istante $T+1$ e $T-1$ o $T+2$ e $T-2$. Ciò garantisce che non dia un punteggio di somiglianza basso per segnali con forme simili e fasi diverse.

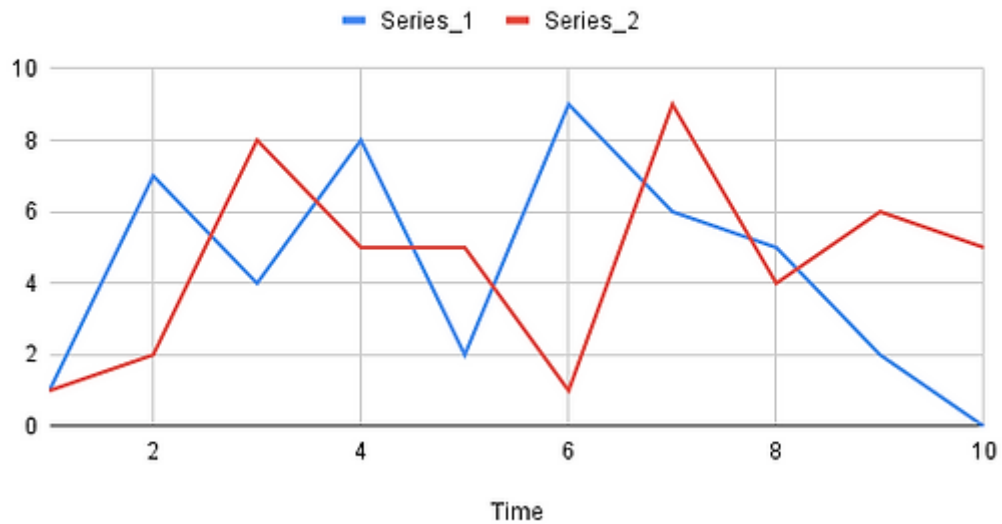


Come funziona l'algoritmo DTW?

Prendiamo due segnali di serie temporali A e B:

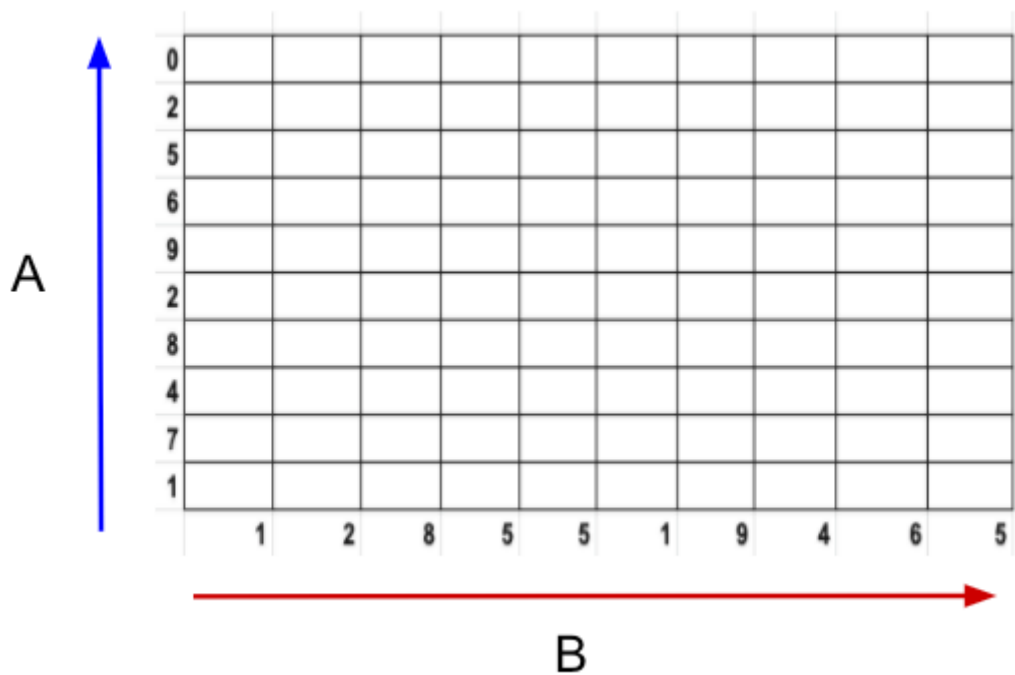
Time	Series_1	Series_2
1	1	1
2	7	2
3	4	8
4	8	5
5	2	5
6	9	1
7	6	9
8	5	4
9	2	6
10	0	5

Series_1 and Series_2



Passaggio 1: creazione della matrice dei costi vuota

Creare una matrice di costo vuota M con etichette x e y come ampiezze delle due serie da confrontare.



Passaggio 2: calcolo dei costi

Compila la matrice dei costi utilizzando la formula indicata di seguito a partire dagli angoli sinistro e inferiore.

$$M(i, j) = |A(i) - B(j)| + \min (M(i-1, j-1), M(i, j-1), M(i-1, j))$$

Dove,

M è la matrice

i è l'iteratore per la serie A
j è l'iteratore per la serie B

Quindi nella tabella sottostante abbiamo calcolato il costo di alcuni valori.

A

0	36	29								
2	35	27								
5	34	27								
6	30	24								
9	25	20								
2	17	13								
8	16	13	6	6						
4	9	7	6	3						
7	6	5	2	4	6	12	14	17	18	20
1	0	1	8	12	16	16	24	27	32	36
	1	2	8	5	5	1	9	4	6	5

B

Prendiamo alcuni esempi dalla tabella sopra (6, 9 e 12) per illustrare il calcolo come evidenziato nella tabella sottostante

Per 6,

A

0	36	29								
2	35	27								
5	34	27								
6	30	24								
9	25	20								
2	17	13								
8	16	13	6	6						
4	9	7	6	3						
7	6	5	2	4	6	12	14	17	18	20
1	0	1	8	12	16	16	24	27	32	36
	1	2	8	5	5	1	9	4	6	5

B

$$|8 - 5| + \min(6, 6, 3) = 6$$

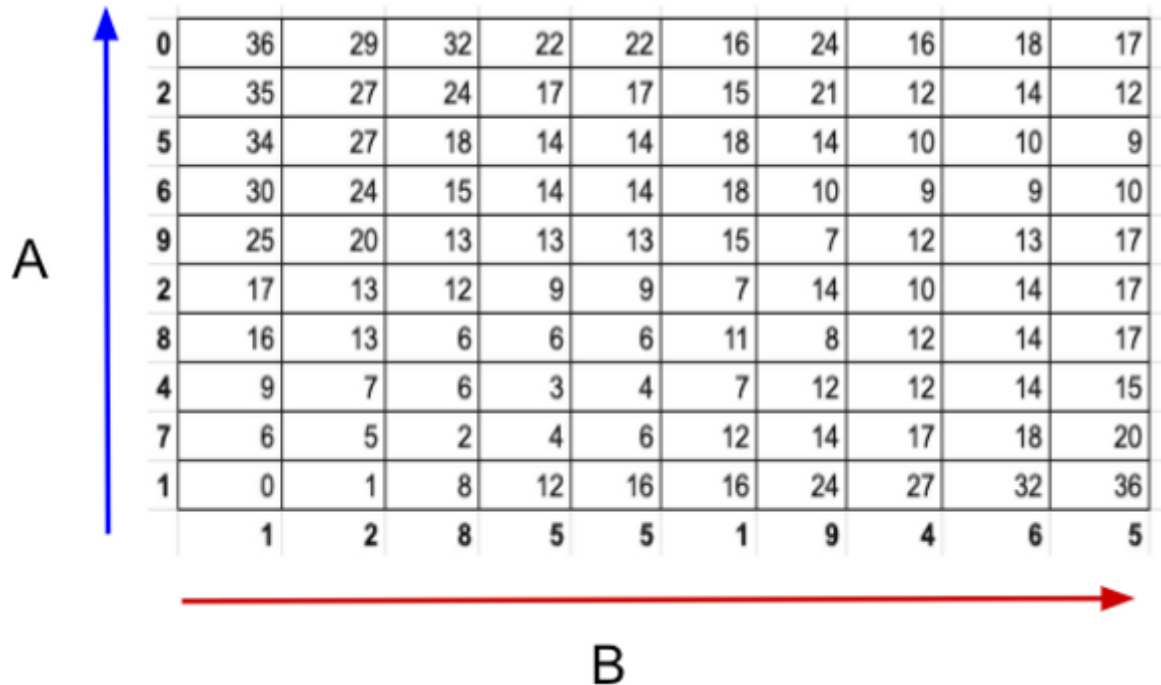
Allo stesso modo per 9,

$$|4 - 1| + \min(6) = 9$$

E lo stesso per 12,

$$|1 - 5| + \min(8) = 12$$

La tabella completa sarà simile a questa:

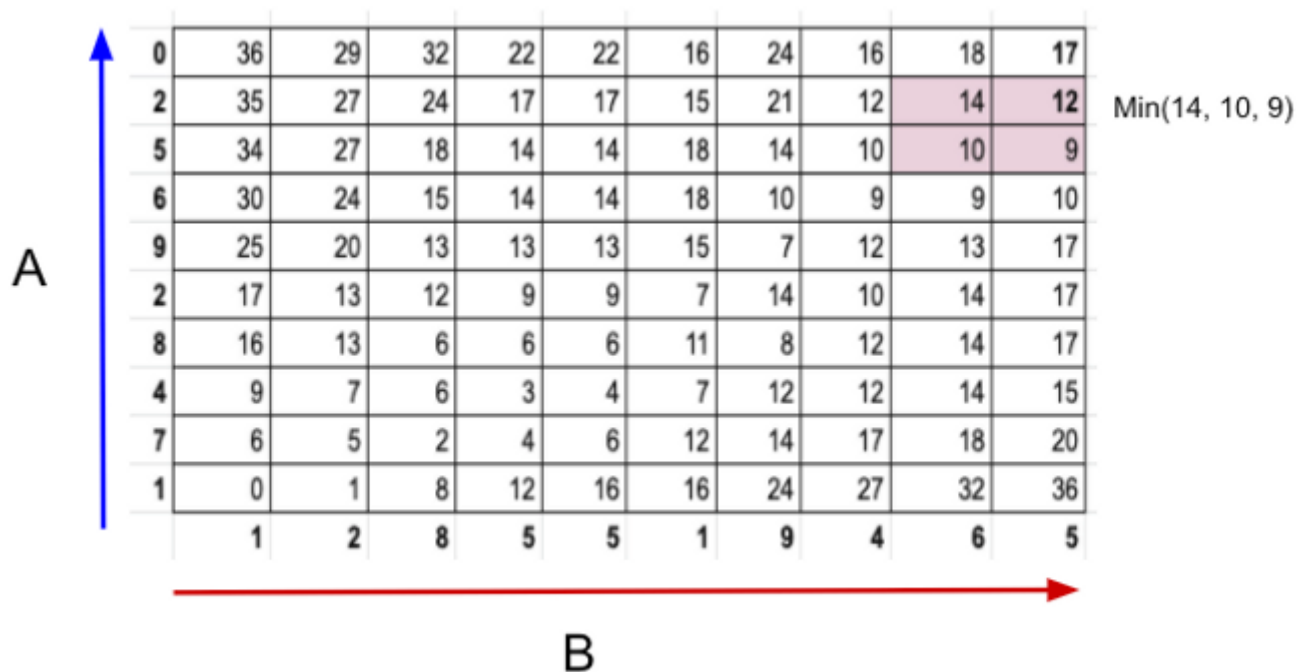
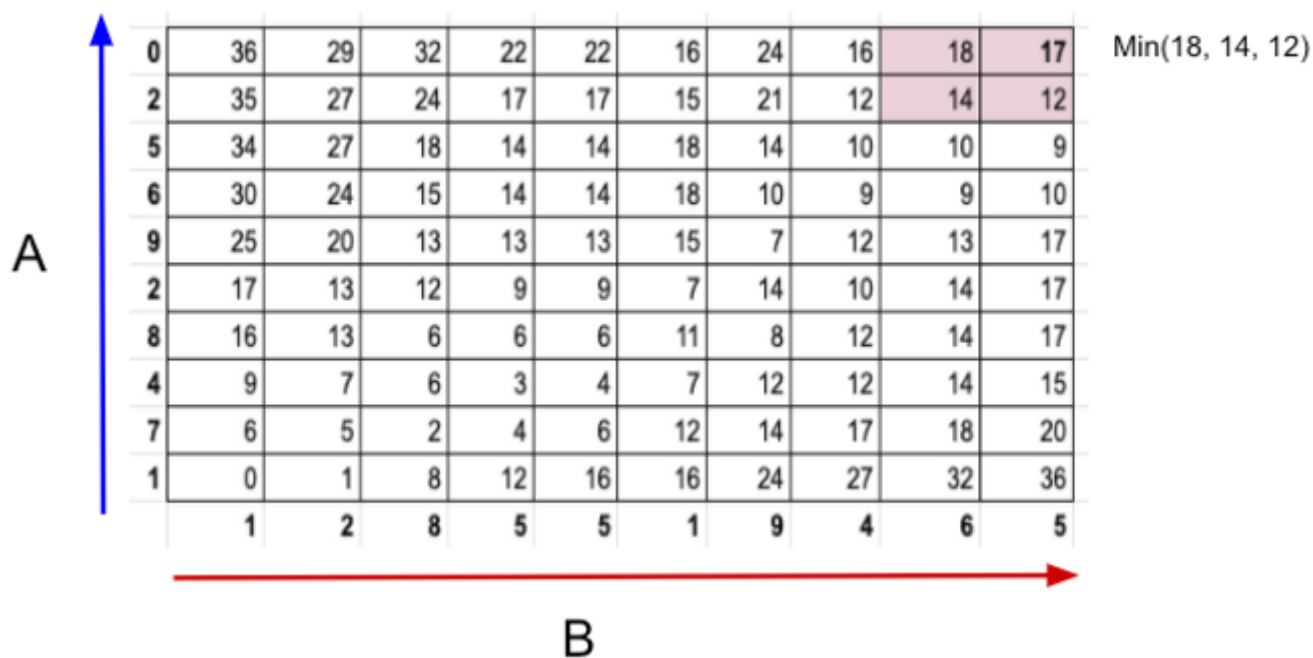


0	36	29	32	22	22	16	24	16	18	17
2	35	27	24	17	17	15	21	12	14	12
5	34	27	18	14	14	18	14	10	10	9
6	30	24	15	14	14	18	10	9	9	10
9	25	20	13	13	13	15	7	12	13	17
2	17	13	12	9	9	7	14	10	14	17
8	16	13	6	6	6	11	8	12	14	17
4	9	7	6	3	4	7	12	12	14	15
7	6	5	2	4	6	12	14	17	18	20
1	0	1	8	12	16	16	24	27	32	36
	1	2	8	5	5	1	9	4	6	5

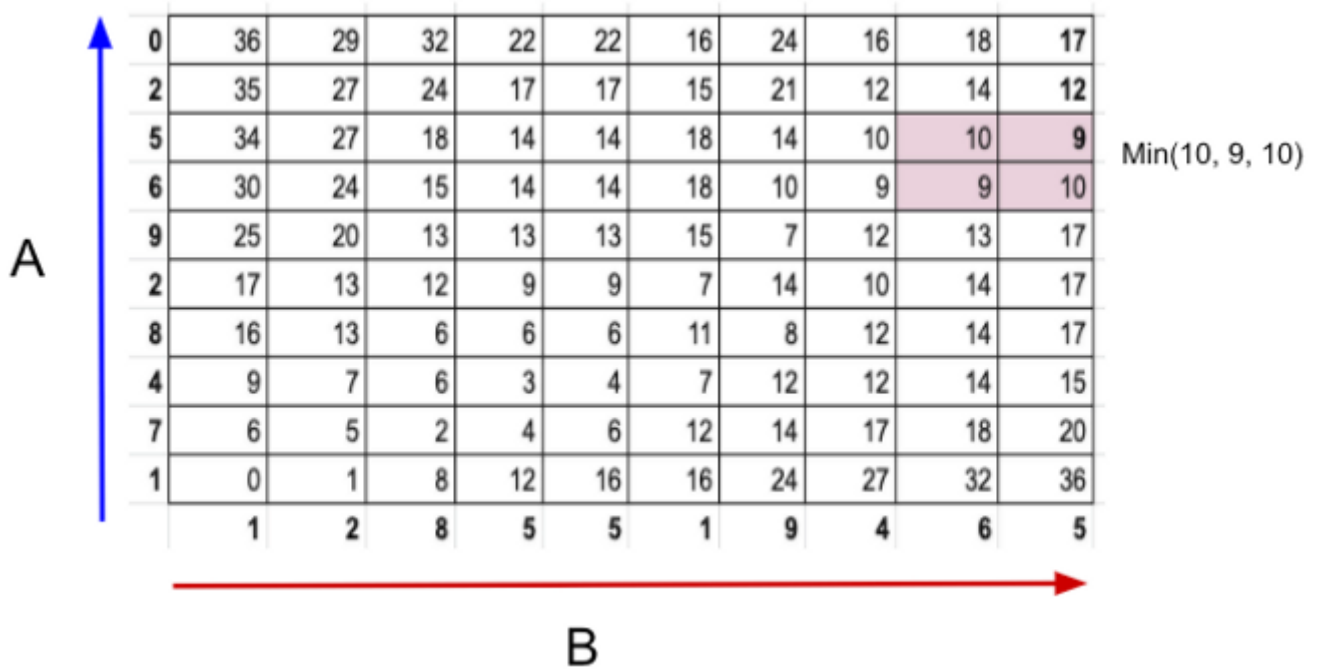
Passaggio 3: identificazione del percorso di deformazione

Identificare il percorso di deformazione partendo dall'angolo in alto a destra della matrice e attraversando in basso a sinistra. Il percorso di attraversamento viene identificato in base al vicino con un valore minimo.

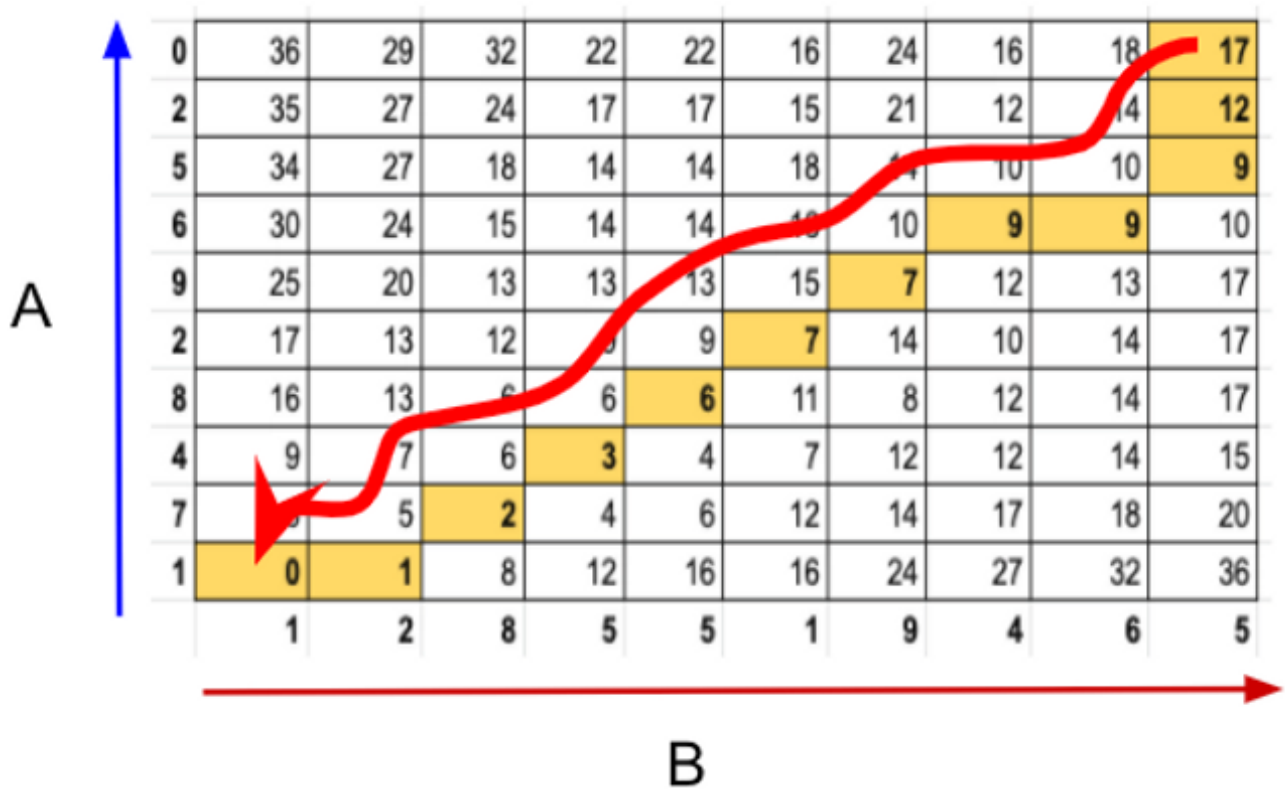
Nel nostro esempio, inizia con 17 e cerca il valore minimo, cioè 12, tra i suoi vicini 18, 14 e 12.



Il numero successivo nel percorso di curvatura trasversale è 9. Questo processo continua finché non raggiungiamo la parte inferiore dell'asse sinistro del tavolo.



Il percorso finale sarà simile a questo:



Lascia che questa serie di percorsi di deformazione sia chiamata come d.

$d = [17, 12, 9, 9, 9, 7, 7, 6, 3, 2, 1, 0]$

Passaggio 4: calcolo della distanza finale

Distanza normalizzata nel tempo, D

$$D = \frac{\sum_{i=1}^k d(i)}{\sum_{i=1}^k k}$$

dove k è la lunghezza della serie d.

k = 12 nel nostro caso.

D = (17 + 12 + 9 + 9 + 9 + 7 + 7 + 6 + 3 + 2 + 1 + 0) / 12

D = 6,8333

Implementazione DTW in Python

Per l'implementazione, useremo la libreria python [fastdtw](#).

FastDTW è un algoritmo approssimativo di Dynamic Time Warping (DTW) che fornisce allineamenti ottimali o quasi ottimali con un tempo $O(N)$ e una complessità di memoria , in contrasto con il requisito $O(N^2)$ per l'algoritmo DTW standard.

Installa la libreria:

```
pip install fastdtw
```

Importa tutte le librerie richieste

```
importa panda come pd
importa numpy come np
```

```
# Pacchetti di stampa
importa matplotlib . pyplot come plt
import seaborn come sbn
```

```
importa matplotlib come mpl
mpl . rcParams [ 'figure.dpi' ] = 150
savefig_options = dict ( format = "png" , dpi = 150 ,
bbox_inches = "tight" )
```

```
# Pacchetti di calcolo
da scipy . spaziale . distanza import euclidea
da fastdtw import fastdtw
```

Definiamo un metodo per calcolare la matrice dei costi accumulati D per il percorso di curvatura. La matrice dei costi utilizza la distanza euclidea per calcolare la distanza tra ogni due punti. I metodi per calcolare la matrice delle distanze euclidee e la matrice dei costi accumulati sono definiti di seguito:

```
def compute_euclidean_distance_matrix ( x , y ) - > np . array :
    """ "Calcola la matrice delle distanze
    Questo metodo calcola la distanza euclidea a coppie tra due
    sequenze .
    Le sequenze possono avere lunghezze diverse .
    """
    dist = np . zeri ( ( len ( y ) , len ( x ) ) )
    per i nell'intervallo ( len ( y ) ) : per j nell'intervallo
( len ( x ) ) :
        dist [ i , j ] = ( x [ j ] - y _ _ [ i ] )
        ) ** 2
    ritorno dist
```

```
def compute_accumulated_cost_matrix ( x , y ) - > np . array :
    """ "Calcola la matrice dei costi accumulati per il percorso
    di curvatura utilizzando la distanza euclidea
    """
    distanze = compute_euclidean_distance_matrix ( x , y )

    # Inizializzazione
    costo = np . zeri ( ( len ( y ) , len ( x ) ) )
    costo [ 0 , 0 ] = distanze [ 0 , 0 ]

    per i nell'intervallo ( 1 , len ( y ) ) :
        costo [ i , 0 ] = distanze [ i , 0 ] + costo [ i - 1 , 0 ]
    ] _

    per j nell'intervallo ( 1 , len ( x ) ) :
        costo [ 0 , j ] = distanze [ 0 , j ] + costo [ 0 , j - 1 ]
    ] _
```

```

# Costo del percorso di curvatura accumulato
for i nell'intervallo ( 1 , len ( y ) ) : for j
nell'intervallo ( 1 , len ( x ) ) :
    cost [ i , j ] = min (
        cost [ i - 1 , j ] , #inserimento _ _
        costo [ i , j - 1 ] ,      # cancellazione
        costo [ i - 1 , j - 1 ]    # match
    ) + distanze [ i , j ]

costo di restituzione

```

Ora, crea due sequenze.

```

x = [ 7 , 1 , 2 , 5 , 9 ]
y = [ 1 , 8 , 0 , 4 , 4 , 2 , 0 ]

```

Non possiamo calcolare la distanza tra x e y poiché non hanno lunghezze uguali.

```

fig , ax = plt . sottotrame ( figsize = ( 6 , 4 ) )

# Rimuovi il bordo e le zecche degli assi
fig . topa . set_visible ( falso )
ascia . asse ( 'off' )

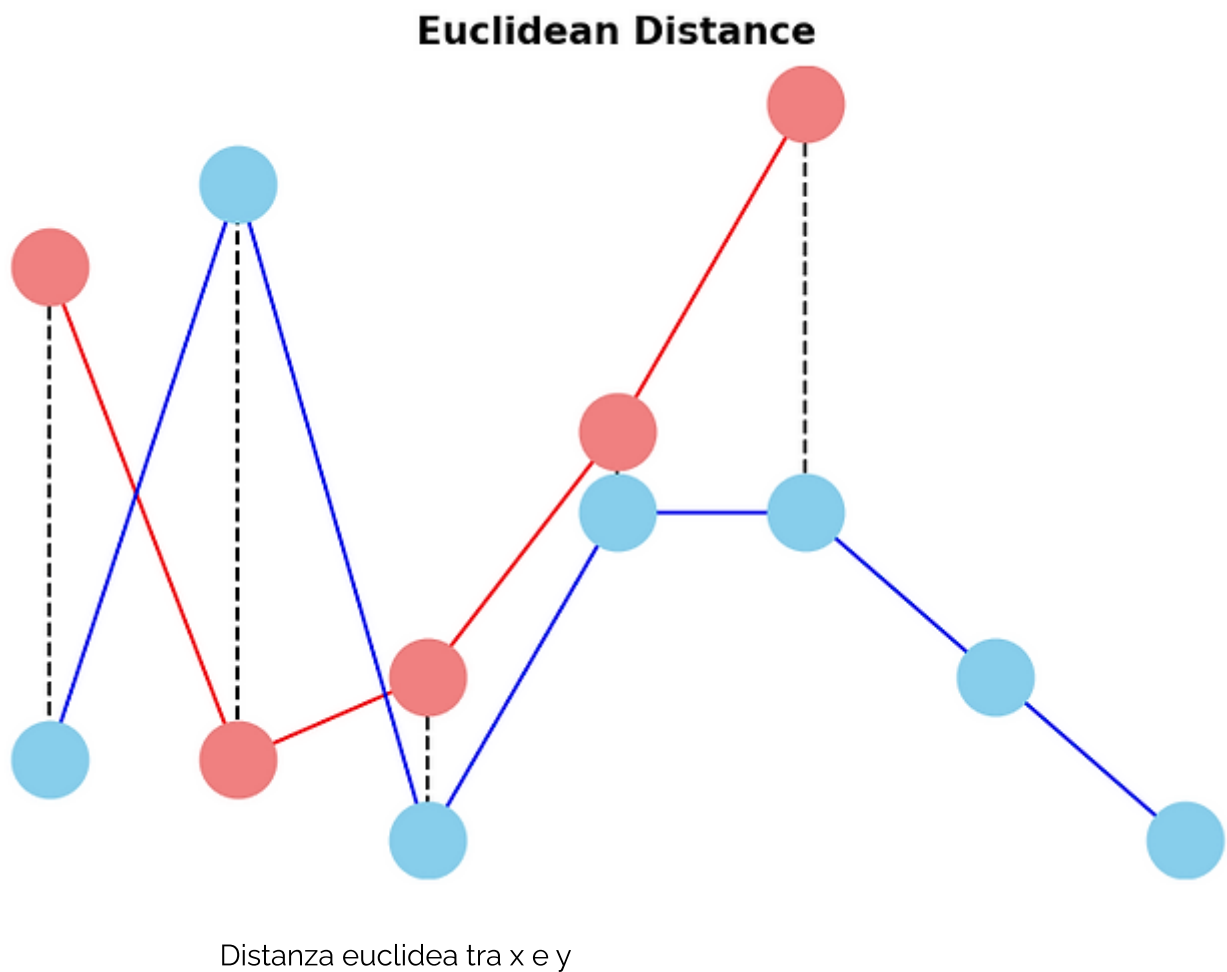
xx = [ ( io , x [ io ] ) per io in np . arange ( 0 , len ( x )
) ]
yy = [ ( j , y [ j ] ) per j in np . arange ( 0 , len ( y ) )
]

per i , j in zip ( xx , yy [ : - 2 ] ) :
    ax . plot ( [ i [ 0 ] , j [ 0 ] ] , [ i [ 1 ] , j [ 1 ] ] ,
'--k' , linewidth = 1 )

ascia . plot ( x , '-ro' , label = 'x' , linewidth = 1 ,
markersize = 20 , markerfacecolor = 'lightcoral' ,
markeredgcolor = 'lightcoral' )
ax . plot ( y , '-bo' , label = 'y' , linewidth = 1 , markersize
= 20 , markerfacecolor = 'skyblue' , markedgcolor = 'skyblue' )

```

```
ax . set_title ( "Distanza euclidea" , fontsize = 10 , fontweight
= "bold" )
```



Calcola la distanza DTW e il percorso di curvatura

Molti pacchetti Python calcolano il DTW semplicemente fornendo le sequenze e il tipo di distanza (solitamente euclidea per impostazione predefinita). Qui, utilizziamo una popolare implementazione Python di DTW che è FastDTW che è un algoritmo DTW approssimativo con complessità di tempo e memoria inferiori.

```
dtw_distance , warp_path = fastdtw ( x , y , dist = euclidean )
```

Si noti che stiamo usando la funzione di distanza Euclidean di SciPy che abbiamo importato in precedenza. Per una migliore comprensione del percorso di curvatura, calcoliamo prima la matrice dei costi accumulati e quindi visualizziamo il percorso su una griglia. Il codice seguente tratterà una heatmap della matrice dei costi accumulati.

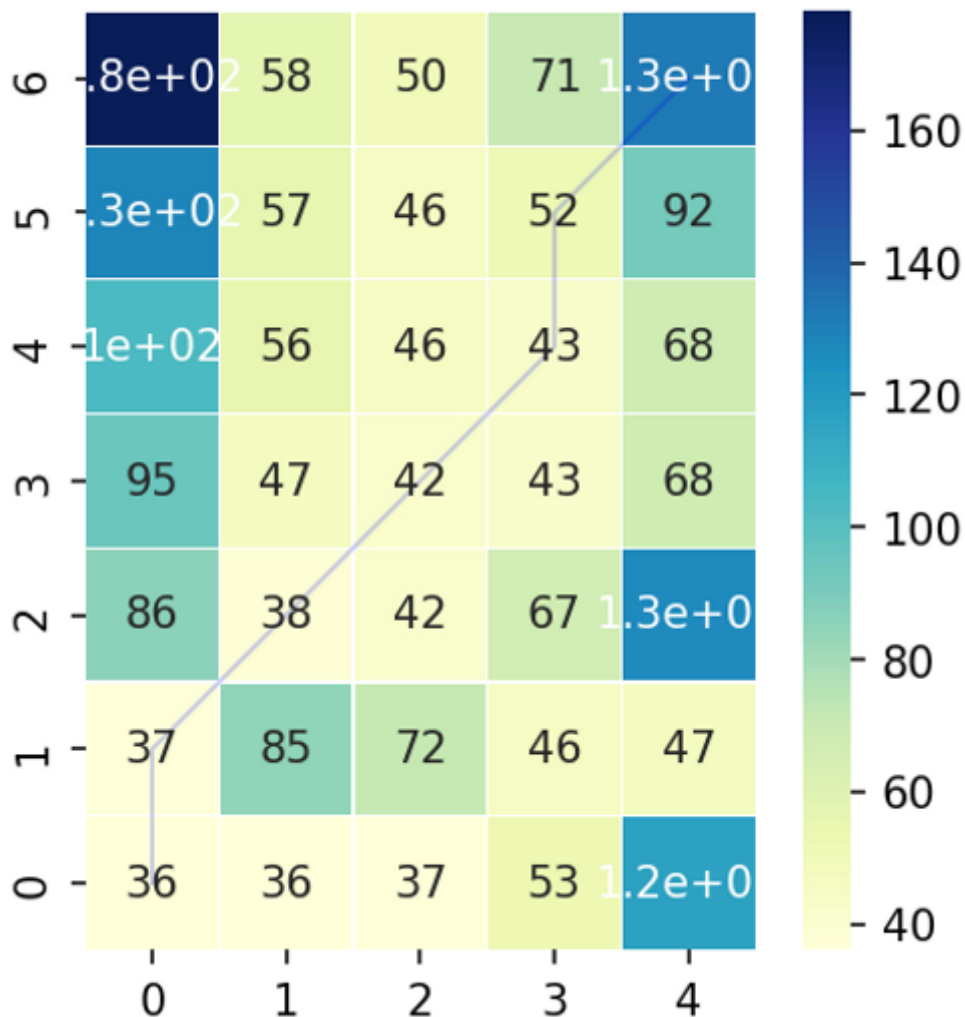
```
matrice_costo = calcola_matrice_costo_accumulato ( x , y )

fig , ax = plt . sottotrame ( figsize = ( 6 , 4 ) )
ax = sbn . heatmap ( cost_matrix , annot = True , square = True ,
linewidths = 0.1 , cmap = "YlGnBu" , ax = ax )
ax . invertire_asse_y ( )

# Ottieni il percorso di curvatura nelle direzioni x e y
path_x = [ p [ 0 ] for p in warp_path ]
path_y = [ p [ 1 ] for p in warp_path ]

# Allinea il percorso dal centro di ogni cella
percorso_xx = [ x + 0,5 per x in percorso_x ]
percorso_yy = [ y + 0,5 per y in percorso_y ]

ascia . plot ( path_xx , path_yy , color = 'blue' , linewidth = 1
, alpha = 0.2 )
```



La barra dei colori mostra il costo di ogni punto della griglia. Come si può vedere, il percorso di curvatura (linea blu) sta attraversando il costo più basso sulla griglia. Vediamo la distanza DTW e il percorso di curvatura stampando queste due variabili.

```
print ( "DTW distance: " , dtw_distance )  
print ( "Warp path: " , warp_path )
```

Distanza DTW: 23.0

Percorso di curvatura: [(0, 0), (0, 1), (1, 2), (2, 3), (3, 4), (3, 5), (4, 6)]

Il percorso di deformazione inizia al punto (0, 0) e termina a (4, 6) per 6 mosse. Calcoliamo anche il costo accumulato maggiormente utilizzando le funzioni che abbiamo definito in precedenza e confrontiamo i valori con la heatmap.

```
cost_matrix = compute_accumulated_cost_matrix ( x , y )  
print ( np . flipud ( cost_matrix ) )
```

```
[[178.  58.  50.  71. 133.]  
 [129.  57.  46.  52.  92.]  
 [104.  56.  46.  43.  68.]  
 [ 95.  47.  42.  43.  68.]  
 [ 86.  38.  42.  67. 127.]  
 [ 37.  85.  72.  46.  47.]  
 [ 36.  36.  37.  53. 117.]]
```

La matrice dei costi stampata sopra ha valori simili alla heatmap.

Ora tracciamo le due sequenze e colleghiamo i punti di mappatura. Di seguito è riportato il codice per tracciare la distanza DTW tra x e y.

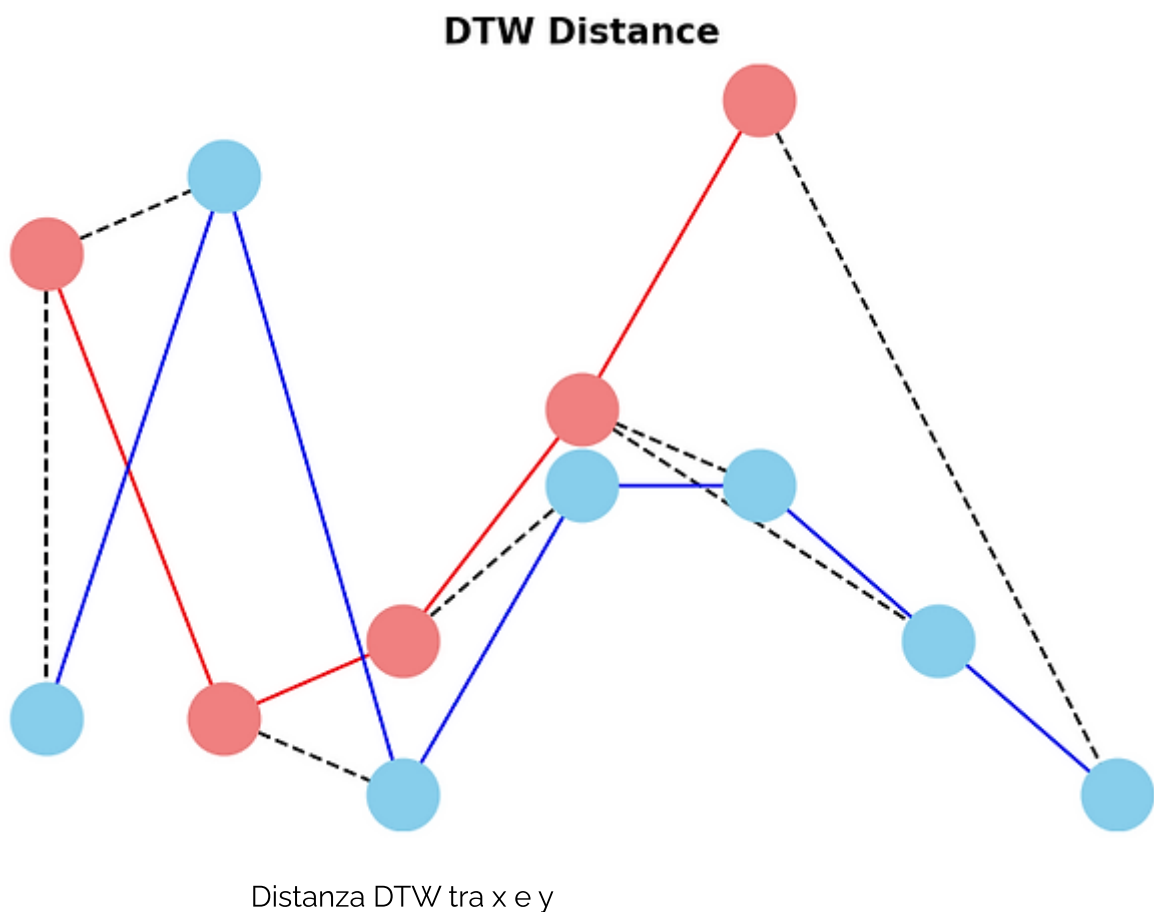
```
fig , ax = plt . sottotrame ( figsize = ( 6 , 4 ) )  
  
# Rimuovi il bordo e le zecche degli assi  
fig . topa . set_visible ( falso )  
ascia . asse ( 'off' )  
  
for [ map_x , map_y ] in warp_path :  
    ax . plot ( [ map_x , map_y ] , [ x [ map_x ] , y [ map_y ]  
] , '--k' , linewidth = 1 )
```

```

ascia . plot ( x , '-ro' , label = 'x' , linewidth = 1 ,
markersize = 20 , markerfacecolor = 'lightcoral' ,
markeredgcolor = 'lightcoral' )
ax . plot ( y , '-bo' , label = 'y' , linewidth = 1 , markersize
= 20 , markerfacecolor ='azzurro cielo' , markeredgcolor =
'azzurro cielo' )

ascia . set_title ( "Distanza DTW" , fontsize = 10 , fontweight =
"bold" )

```



Puoi scaricare il codice da [Github](#) .

Applicazioni di DTW

1. Per rilevare somiglianze nel camminare. Se una persona camminava più velocemente dell'altra, o se ci sono state accelerazioni e decelerazioni durante un'osservazione.

2. Applicazioni di riconoscimento vocale. Sono usati per abbinare un comando vocale campione con i comandi di altri anche se la persona parla più velocemente o più lentamente rispetto alla voce campione preregistrata.

3. Analisi del potere di correlazione

Originariamente pubblicato su [KDnuggets](#) .

Riferimenti

<https://medium.com/walmartglobaltech/time-series-similarity-using-dynamic-time-warping-explained-9d09119e48ec>

https://en.wikipedia.org/wiki/Dynamic_time_warping

<https://databricks.com/blog/2019/04/30/understanding-dynamic-time-warping.html>

http://www.mathcs.emory.edu/~lxiong/cs730_s13/share/slides/searching_sigkdd2012_DTW.pdf

<https://ealizadeh.com/blog/introduction-to-dynamic-time-warping>