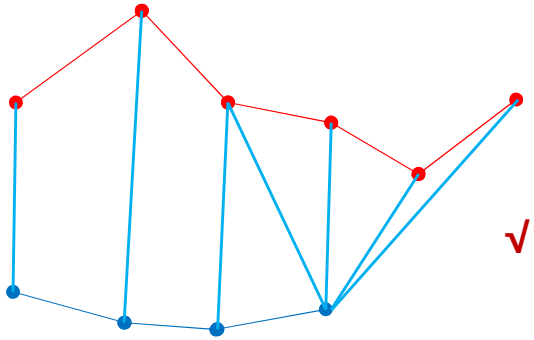
动态时间规整（DTW）

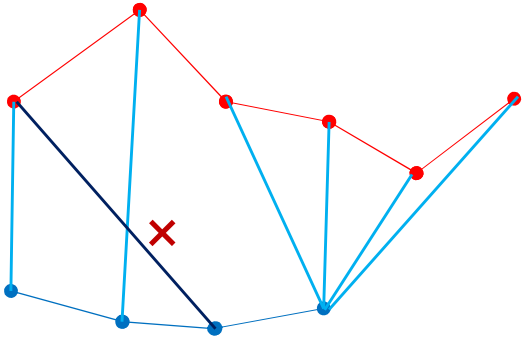
于泓
鲁东大学
信息与电气工程学院
2019.11.13

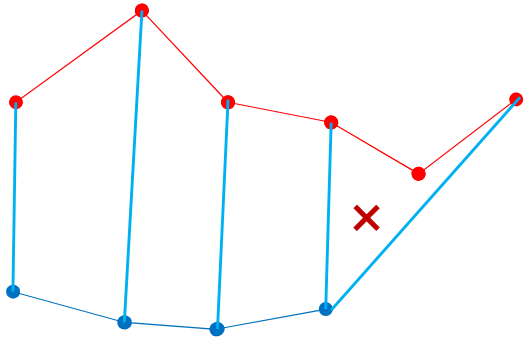
DTW（Dynamic Time Warping）

- 按照距离最近的原则，构建两个序列元素之间的对应的关系，评估两个序列的相似性。

- 要求（1）单向对应，不能回头
  （2）一、一对应 不能有空
  （3）对应之后，距离最近

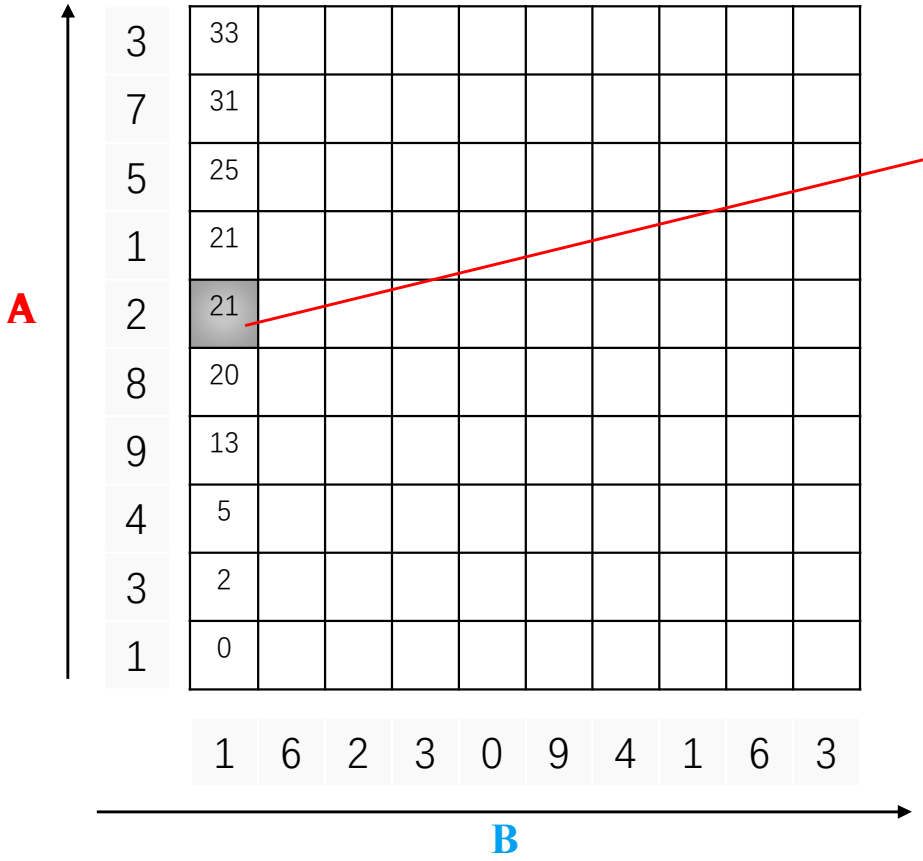






算法实现

累计距离矩阵(D)



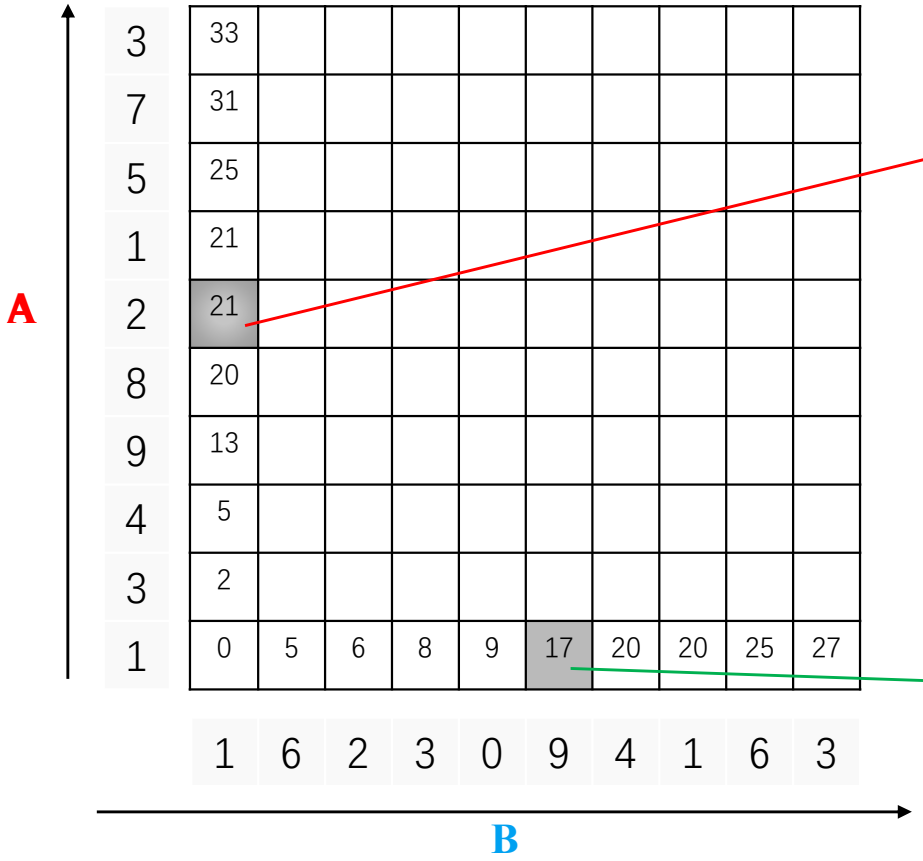
（1） 填充累计距离矩阵　　dis(x,y)=|x-y|

对于最左边一列：$D[i,0]=dis(A_i,B_0)+D[i-1,0]$

D[5,0]=dis(2,1)+D[4,0]=|2-1|+20 = 21

算法实现

累计距离矩阵(D)



（1） 填充距离矩阵 dis(x,y)=|x-y|

对于最左边一列：$D[i,0]=dis(A_i,B_0)+D[i-1,0]$

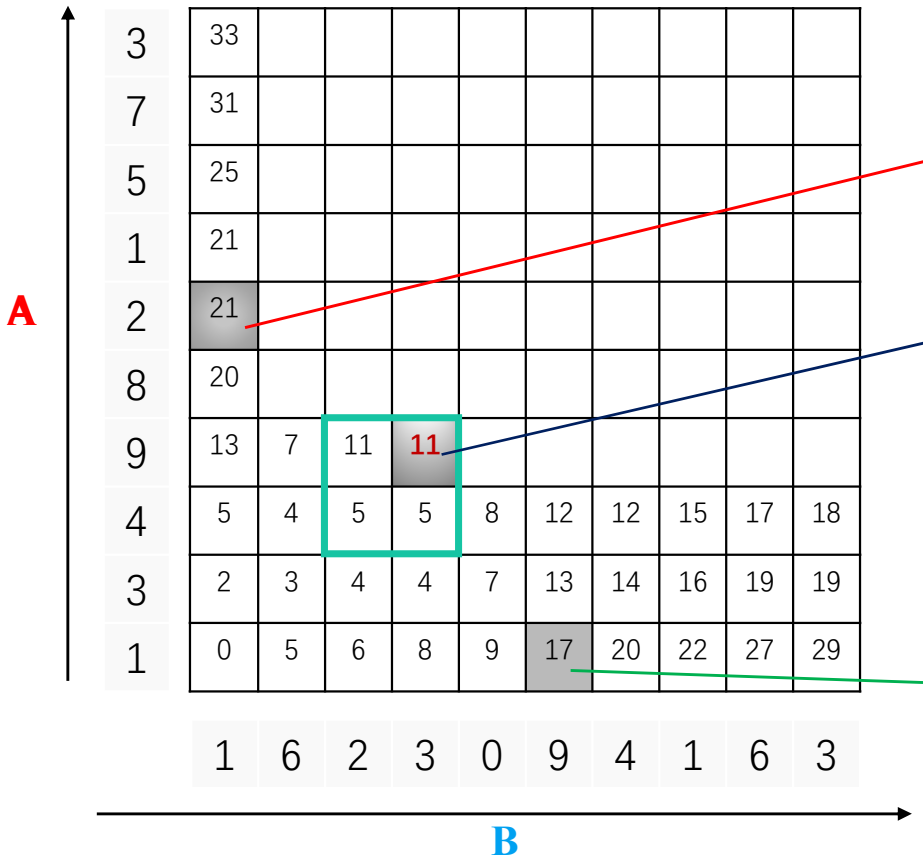
$D[5,0]=dis(2,1)+D[4,0]=|2-1|+20=21$

对于最下边一行：$D[0,j]=dis(A_0,B_j)+D[0,j-1]$

$D[0,5]=dis(1,17)+D[0,4]=|1-9|+9=17$

算法实现

累计距离矩阵(D)



（1） 填充距离矩阵 dis(x,y)=|x-y|

对于最左边一列：$D[i,0]=dis(A_i,B_0)+D[i-1,0]$

D[5,0]=dis(2,1)+D[4,0]=|2-1|+20 = 21

左下3元素

对于其他：$D[i,j]=dis(A_i,B_j)+min(D[i-1,j],D[i,j-1],D[i-1,j-1])$

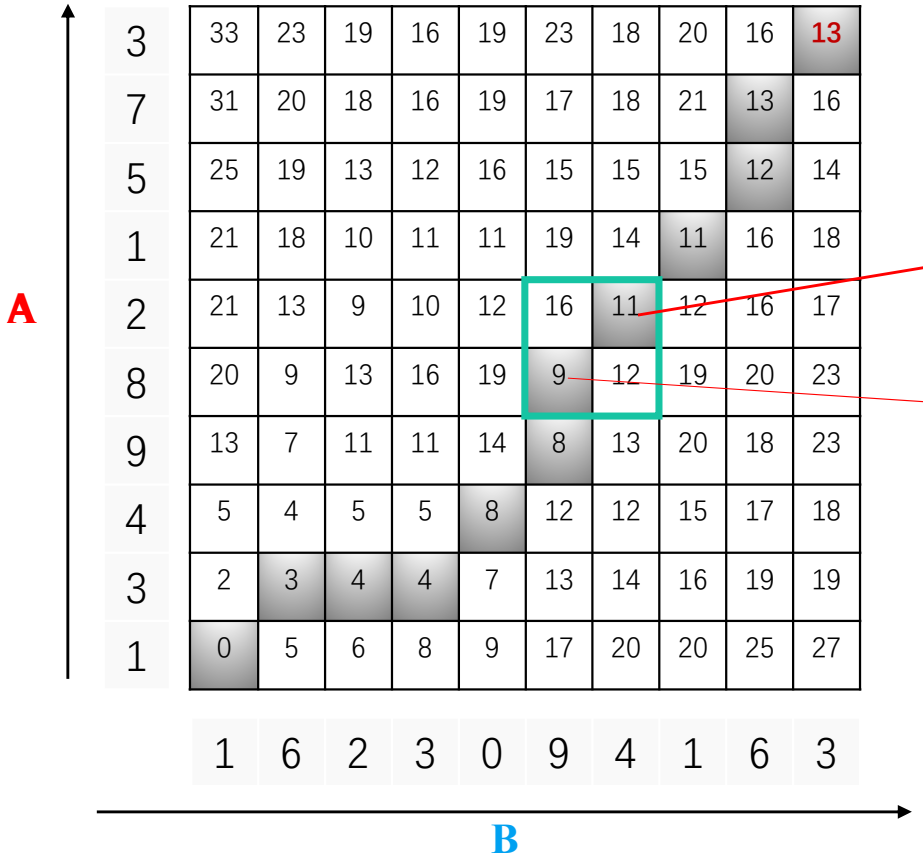
D[3,3]=dis(A_3,B_3)+min(D[3,2],D[2,3],D[2,2])
=|9-3|+min(11,5,5) = 6+5 = 11

对于最下边一行：$D[0,j]=dis(A_0,B_j)+D[0,j-1]$

D[0,5]=dis(1,17)+D[0,4]=|1-9|+9 = 17

算法实现

累计距离矩阵(D)



（2） 从右上角开始，向左下找寻配准路径
找到左下三点中，较小的那个作为下一节点。

关键：距离函数 dis的定义

当前点

下一点

（3） 输出配准结果

结果[(A_0,B_0),(A_1,B_1),(A_1,B_2),(A_1,B_3),(A_2,B_4),….(A_9,B_9)]

代码实现

```
def dis_abs(x, y):
    return abs(x-y)[0]

def estimate_twf(A,B,dis_func=dis_abs):

    N_A = len(A)
    N_B = len(B)

    D = np.zeros([N_A,N_B])
    D[0,0] = dis_func(A[0],B[0])

    # 左边一列
    for i in range(1,N_A):
        D[i,0] = D[i-1,0]+dis_func(A[i],B[0])
    # 下边一行
    for j in range(1,N_B):
        D[0,j] = D[0,j-1]+dis_func(A[0],B[j])
    # 中间部分
    for i in range(1,N_A):
        for j in range(1,N_B):
            D[i,j] = dis_func(A[i],B[j])+min(D[i-1,j],D[i,j-1],D[i-1,j-1])
```

```
# 路径回溯
i = N_A-1
j = N_B-1
count =0
d = np.zeros(max(N_A,N_B)*3)
path = []
while True:
    if i>0 and j>0:
        path.append((i,j))
        m = min(D[i-1, j],D[i, j-1],D[i-1,j-1])
        if m == D[i-1,j-1]:
            d[count] = D[i,j] - D[i-1,j-1]
            i = i-1
            j = j-1
            count = count+1

        elif m == D[i,j-1]:
            d[count] = D[i,j] - D[i,j-1]
            j = j-1
            count = count+1

        elif m == D[i-1, j]:
            d[count] = D[i,j] - D[i-1,j]
            i = i-1
            count = count+1

    elif i == 0 and j == 0:
        path.append((i,j))
        d[count] = D[i,j]
        count = count+1
        break
    elif i == 0:
        path.append((i,j))
        d[count] = D[i,j] - D[i,j-1]
        j = j-1
        count = count+1

    elif j == 0:
        path.append((i,j))
        d[count] = D[i,j] - D[i-1,j]
        i = i-1
        count = count+1

mean = np.sum(d) / count
return mean, path[::-1],D
```

```
if __name__=="__main__":
    a = np.array([1,3,4,9,8,2,1,5,7,3])
    b = np.array([1,6,2,3,0,9,4,1,6,3])
    a = a[:,np.newaxis]
    b = b[:,np.newaxis]
    dis,path,D = estimate_twf(a,b,dis_func=dis_abs)
    print(dis)
    print(path)
    print(D)
```

```
yuhong@admin2:/home/sdh$ python test_dtw.py
1.0833333333333333
[(0, 0), (1, 1), (1, 2), (1, 3), (2, 4), (3, 5), (4, 5), (5, 6), (6, 7), (7, 8), (8, 8), (9, 9)]
[[ 0.  5.  6.  8.  9. 17. 20. 20. 25. 27.]
 [ 2.  3.  4.  4.  7. 13. 14. 16. 19. 19.]
 [ 5.  4.  5.  5.  8. 12. 12. 15. 17. 18.]
 [13.  7. 11. 11. 14.  8. 13. 20. 18. 23.]
 [20.  9. 13. 16. 19.  9. 12. 19. 20. 23.]
 [21. 13.  9. 10. 12. 16. 11. 12. 16. 17.]
 [21. 18. 10. 11. 11. 19. 14. 11. 16. 18.]
 [25. 19. 13. 12. 16. 15. 15. 15. 12. 14.]
 [31. 20. 18. 16. 19. 17. 18. 21. 13. 16.]
 [33. 23. 19. 16. 19. 23. 18. 20. 16. 13.]]
```

DTW 的Python 包

- pip inatall dtw
- pip install dtw_c
- pip install fastdtw

DTW的应用

- 计算两个序列之间的相似性 （取dis）

  动作识别：

  A ： 录制动作时传感器数据　　B：测试动作时传感器数据

- 获取匹配特征对 （取path）

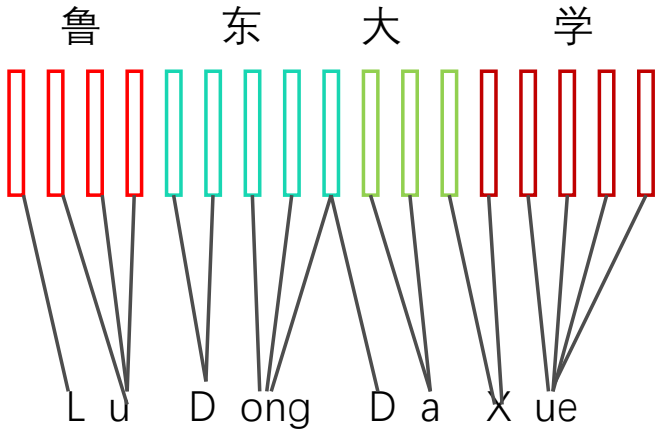
  Voice conversion

  A ： 说话人A的特征　　B：说话人B的特征

- 选择不同的dis

  语音识别/英文发音质量检测

  A ： 语音特征　　B：文本序列标签　　dis = 概率分布

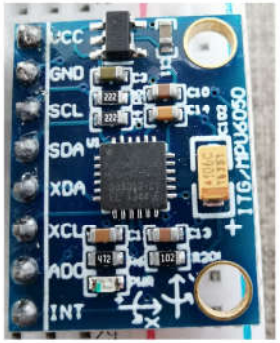


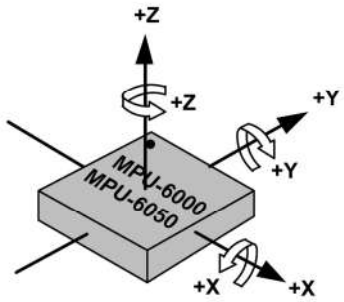
DTW的应用

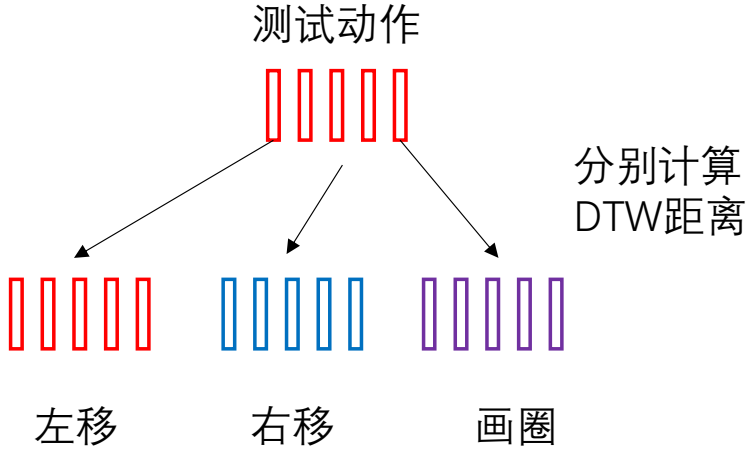
- 计算两个序列之间的相似性 （取dis）

  动作识别：

  A ： 测试动作时传感器数据　　B：录制动作时传感器数据





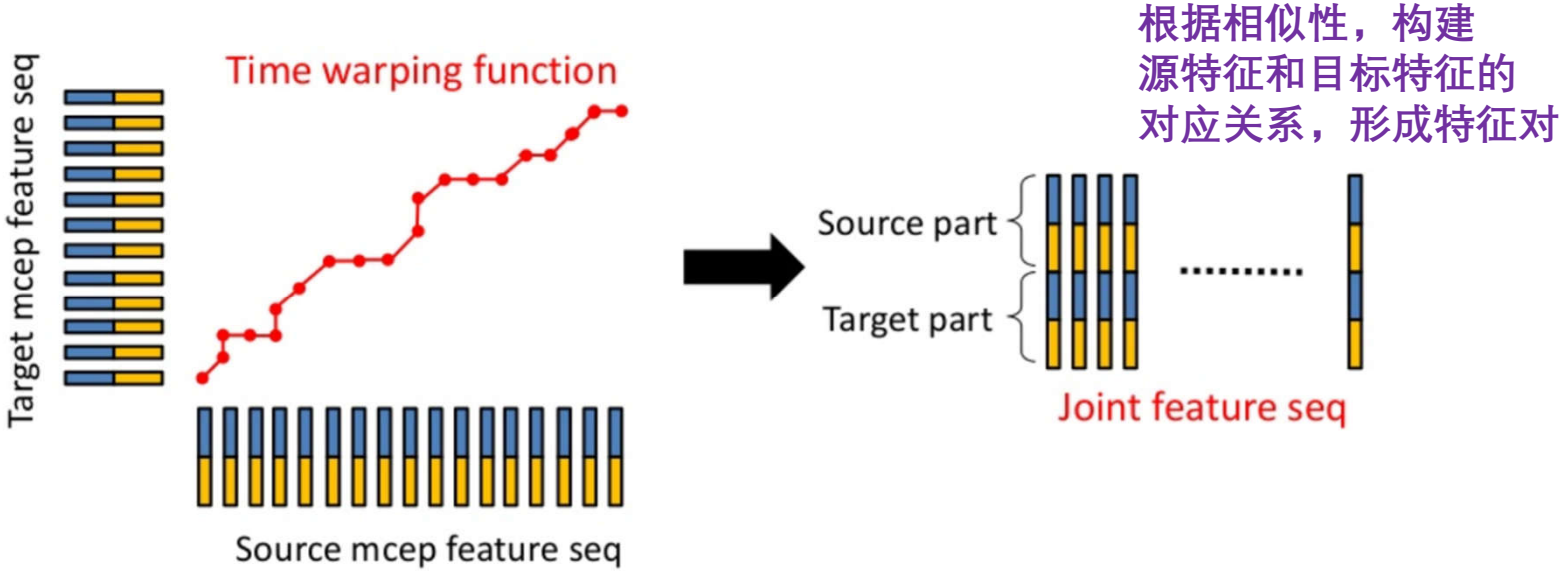


DTW的应用

- 获取匹配特征对 （取path）

  Voice conversion

  A ：目标说话人的特征　　B：源说话人的特征



DTW的应用

- 选择不同的dis，A、B序列属性不同

  语音识别/英文发音质量检测

  A：语音特征　　B：文本序列标签　　dis = 概率分布