AI学习笔记--sklearn--DTW算法

在孤立词语音识别中，最为简单有效的方法是采用DTW（Dynamic Time Warping，动态时间归整）算法，该算法基于动态规划（DP）的思想，解决了发音长短不一的模板匹配问题，是语音识别中出现较早、较为经典的一种算法，用于孤立词识别。HMM算法在训练阶段需要提供大量的语音数据，通过反复计算才能得到模型参数，而DTW算法的训练中几乎不需要额外的计算。所以在孤立词语音识别中，DTW算法仍然得到广泛的应用。

- 算法原理

无论在训练和建立模板阶段还是在识别阶段，都先采用端点算法确定语音的起点和终点。以存入模板库的各个词条称为参考模板，一个参考模板可表示为R={R（1），R（2），……，R（m），……，R（M）}，m为训练语音帧的时序标号，m=1为起点语音帧，m=M为终点语音帧，因此M为该模板所包含的语音帧总数，R（m）为第m帧的语音特征矢量。所要识别的一个输入词条语音称为测试模板，可表示为T={T（1），T（2），……，T（n），……，T（N）}，n为测试语音帧的时序标号，n=1为起点语音帧，n=N为终点语音帧，因此N为该模板所包含的语音帧总数，T（n）为第n帧的语音特征矢量。参考模板与测试模板一般采用相同类型的特征矢量（如MFCC，LPC系数）、相同的帧长、相同的窗函数和相同的帧移。

假设测试和参考模板分别用T和R表示，为了比较它们之间的相似度，可以计算它们之间的距离 D[T，R]，距离越小则相似度越高。为了计算这一失真距离，应从T和R中各个对应帧之间的距离算起。设n和m分别是T和R中任意选择的帧号，d[T（n），R（m）]表示这两帧特征矢量之间的距离。距离函数取决于实际采用的距离度量，在DTW算法中通常采用欧氏距离。

若N=M则可以直接计算，否则要考虑将T（n）和R（m）对齐。对齐可以采用线性扩张的方法，如果N<M可以将T线性映射为一个M帧的序列，再计算它与{R（1），R（2），……，R（M）}之间的距离。但是这样的计算没有考虑到语音中各个段在不同情况下的持续时间会产生或长或短的变化，因此识别效果不可能最佳。因此更多的是采用动态规划（DP）的方法。

若把测试模板的各个帧号n=1~N在一个二维直角坐标系中的横轴上标出，把参考模板的各帧号m=1~M在纵轴上标出，通过这些表示帧号的整数坐标画出一些纵横线即可形成一个网络，网络中的每一个交叉点（n，m）表示测试模式中某一帧的交汇点。DP算法可以归结为寻找一条通过此网络中若干格点的路径，路径通过的格点即为测试和参考模板中进行计算的帧号。路径不是随意选择的，首先任何一种语音的发音快慢都有可能变化，但是其各部分的先后次序不可能改变，因此所选的路径必定是从左下角出发，在右上角结束

为了描述这条路径，假设路径通过的所有格点依次为（n_1，m_1），……，（n_i，m_j），……，（n_N，m_M），其中（n_1，m_1）=（1，1），（n_N，m_M）=（N，M）。路径可以用函数m = Oslash;（n ）描述，其中n =i，i=1，2，……，N，Ø（1）=1，Ø（N）=M。为了使路径不至于过倾斜，可以约束斜率在0.5~2的范围内，如果路径已经通过了格点（n ，m ），那么下一个通过的格点（n ，m ）只可能是下列三种情况之一：

（n ，m ）=（n +1，m）

（n ，m ）=（n +1，m +1）

（n ，m ）=（n ，m+1 ）

用r表示上述三个约束条件。求最佳路径的问题可以归结为满足约束条件r时，求最佳路径函数m =Ø（n ），使得沿路径的积累距离达到最小值，即：

搜索该路径的方法如下：搜索从（n, m）点出发，可以展开若干条满足η的路径，假设可计算每条路径达到（n, m）点时的总的积累距离，具有最小累积距离者即为最佳路径。易于证明，限定范围的任一格点（n, m）只可能有一条搜索路径通过。对于（n, m），其可达到该格点的前一个格点只可能是（n-1, m）、（n-1, m -1）和（n, m-1），那么（n, m）一定选择这3个距离之路径延伸而通过（n, m），这时此路径的积累距离为：

D[(n, m)]=d[T(n),R(m)]+min{D(n-1,m),D(n-1,m-1),D(n,m-1)}

这样可以从（n ，m ）=（1，1）出发搜索（n ，m ），对每一个（n ，m ）都存储相应的距离，这个距离是当前格点的匹配距离与前一个累计距离最小的格点（按照设定的斜率在三个格点中进行比较）。搜索到（n ，m ）时，只保留一条最佳路径。如果有必要的话，通过逐点向前寻找就可以求得整条路径。这套DP算法便是DTW算法。

DTW算法可以直接按上面描述来实现，即分配两个N×M的矩阵，分别为积累距离矩阵D和帧匹配距离矩阵d，其中帧匹配距离矩阵d（i，j）的值为测试模板的第i帧与参考模板的第j帧间的距离。D（N，M）即为最佳匹配路径所对应的匹配距离

- 算法比较

DTW算法由于没有一个有效地用统计方法进行训练的框架，也不容易将低层和顶层的各种知识用到语音识别算法中，因此在解决大词汇量、连续语音、非特定人语音识别问题时较之HMM算法相形见绌。HMM是一种用参数表示的,用于描述随机过程统计特性的概率模型。而对于孤立词识别,HMM算法和DTW算法在相同条件下,识别效果相差不大, 又由于DTW算法本身既简单又有效，但HMM算法要复杂得多。它需要在训练阶段提供大量的语音数据,通过反复计算才能得到参数模型,而DTW算法的训练中几乎不需要额外的计算。

- 算法解析

动态时间规整DTW是一个典型的优化问题，它用满足一定条件的的时间规整函数W(n)描述测试模板和参考模板的时间对应关系，求解两模板匹配时累计距离最小所对应的规整函数。

假设我们有两个时间序列Q和C，他们的长度分别是n和m：（实际语音匹配运用中，一个序列为参考模板，一个序列为测试模板，序列中的每个点的值为语音序列中每一帧的特征值。例如语音序列Q共有n帧，第i帧的特征值（一个数或者一个向量）是qi。至于取什么特征，在这里不影响DTW的讨论。我们需要的是匹配这两个语音序列的相似性，以达到识别我们的测试语音是哪个词）

Q = q1, q2,…,qi,…, qn ;

C = c1, c2,…, cj,…, cm ;

如果n=m，那么就用不着折腾了，直接计算两个序列的距离就好了。但如果n不等于m我们就需要对齐。最简单的对齐方式就是线性缩放了。把短的序列线性放大到和长序列一样的长度再比较，或者把长的线性缩短到和短序列一样的长度再比较。但是这样的计算没有考虑到语音中各个段在不同情况下的持续时间会产生或长或短的变化，因此识别效果不可能最佳。因此更多的是采用动态规划（dynamic programming）的方法。

为了对齐这两个序列，我们需要构造一个n x m的矩阵网格，矩阵元素(i, j)表示qi和cj两个点的距离d(qi, cj)（也就是序列Q的每一个点和C的每一个点之间的相似度，距离越小则相似度越高。这里先不管顺序），一般采用欧式距离，d(qi, cj)= (qi-cj)2（也可以理解为失真度）。每一个矩阵元素(i, j)表示点qi和cj的对齐。DP算法可以归结为寻找一条通过此网格中若干格点的路径，路径通过的格点即为两个序列进行计算的对齐的点。

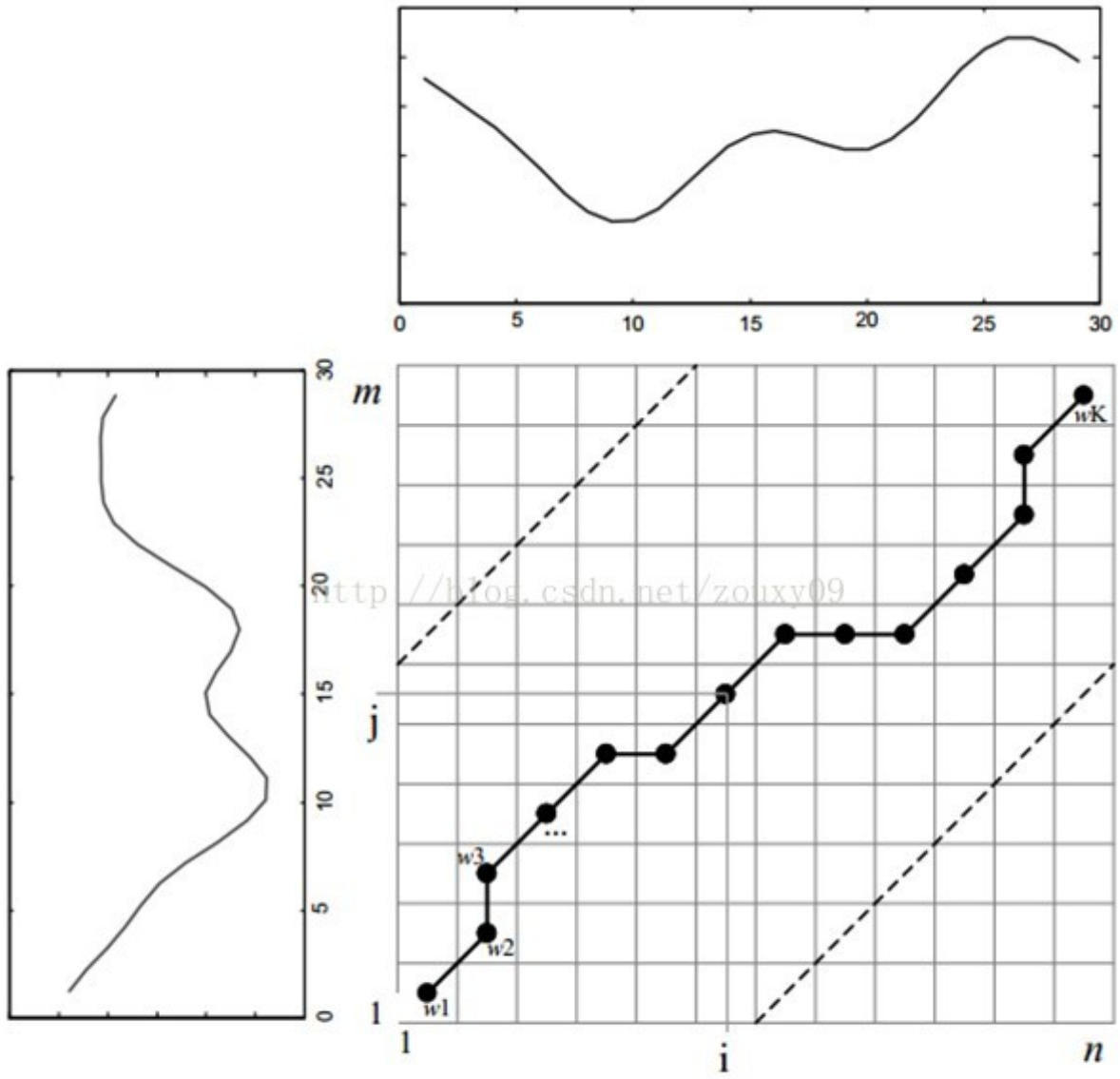


Figure 3: An example warping path.

那么这条路径我们怎么找到呢？那条路径才是最好的呢？也就是刚才那个问题，怎么样的warping才是最好的。

把这条路径定义为warping path规整路径，并用W来表示， W的第k个元素定义为wk=(i,j)k，定义了序列Q和C的映射。这样我们有：

$$W = w_1, w_2, \ldots, w_k, \ldots, w_K \qquad \max(m,n) \le K < m+n-1$$

首先，这条路径不是随意选择的，需要满足以下几个约束：

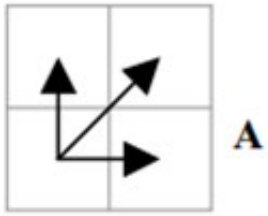
1）边界条件：w1=(1, 1)和wK=(m, n)。任何一种语音的发音快慢都有可能变化，但是其各部分的先后次序不可能改变，因此所选的路径必定是从左下角出发，在右上角结束。

2）连续性：如果wk-1= (a’, b’)，那么对于路径的下一个点wk=(a, b)需要满足 (a-a’) <=1和 (b-b’) <=1。也就是不可能跨过某个点去匹配，只能和自己相邻的点对齐。这样可以

保证Q和C中的每个坐标都在W中出现。

3）单调性：如果wk-1= (a', b')，那么对于路径的下一个点wk=(a, b)需要满足0<=(a-a')和0<= (b-b')。这限制W上面的点必须是随着时间单调进行的。以保证图B中的虚线不会相交。

结合连续性和单调性约束，每一个格点的路径就只有三个方向了。例如如果路径已经通过了格点(i, j)，那么下一个通过的格点只可能是下列三种情况之一：(i+1, j)，(i, j+1)或者(i+1, j+1)。



满足上面这些约束条件的路径可以有指数个，然后我们感兴趣的是使得下面的规整代价最小的路径：

分母中的K主要是用来对不同的长度的规整路径做补偿。我们的目的是什么？或者说DTW的思想是什么？是把两个时间序列进行延伸和缩短，来得到两个时间序列性距离最短也就是最相似的那一个warping，这个最短的距离也就是这两个时间序列的最后的距离度量。在这里，我们要做的就是选择一个路径，使得最后得到的总的距离最小。

这里我们定义一个累加距离cumulative distances。从(0, 0)点开始匹配这两个序列Q和C，每到一个点，之前所有的点计算的距离都会累加。到达终点(n, m)后，这个累积距离就是我们上面说的最后的总的距离，也就是序列Q和C的相似度。

累积距离γ(i,j)可以按下面的方式表示，累积距离γ(i,j)为当前格点距离d(i,j)，也就是点qi和cj的欧式距离（相似性）与可以到达该点的最小的邻近元素的累积距离之和：

$$\gamma(i,j) = d(q_i,c_j) + \min\{\gamma(i-1,j-1), \gamma(i-1,j), \gamma(i,j-1)\}$$

最佳路径是使得沿路径的积累距离达到最小值这条路径。这条路径可以通过动态规划（dynamic programming）算法得到。

下面是一个例子来理解上述原理：

DTW为(Dynamic Time Warping,动态时间归准)的简称。应用很广，主要是在模板匹配中，比如说用在孤立词语音识别，计算机视觉中的行为识别，信息检索等中。可能大家学过这些类似的课程都看到过这个算法，公式也有几个，但是很抽象，当时看懂了但不久就会忘记，因为没有具体的实例来加深印象。

这次主要是用语音识别课程老师上课的一个题目来理解DTW算法。

首先还是介绍下DTW的思想：假设现在有一个标准的参考模板R，是一个M维的向量，即R={R（1），R（2），……，R（m），……，R（M）}，每个分量可以是一个数或者是一个更小的向量。现在有一个才测试的模板T，是一个N维向量，即T={T（1），T（2），……，T（n），……，T（N）}同样每个分量可以是一个数或者是一个更小的向量，注意M不一定等于N，但是每个分量的维数应该相同。

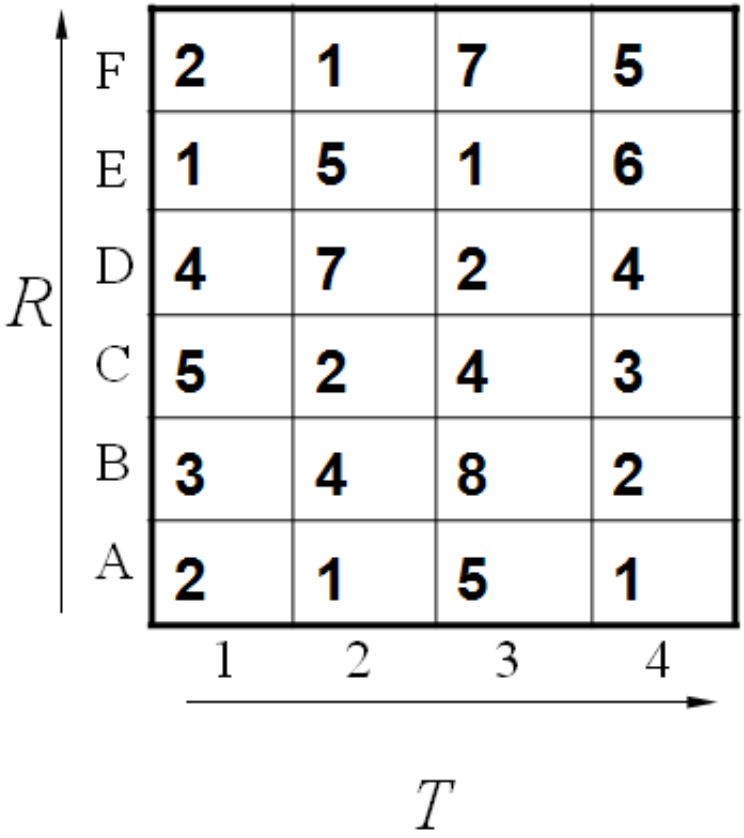
由于M不一定等于N，现在要计算R和T的相似度，就不能用以前的欧式距离等类似的度

量方法了。那用什么方法呢？DTW就是为了解决这个问题而产生的。

首先我们应该知道R中的一个分量R（m）和T中的一个分量T（n）的维数是相同的，它们之间可以计算相似度（即距离）。在运用DTW前，我们要首先计算R的每一个分量和T中的每一个分量之间的距离，形成一个M*N的矩阵。（为了方便，行数用将标准模板的维数M，列数为待测模板的维数N）。

然后下面的步骤该怎么计算呢？用个例子来看看。

这个例子中假设标准模板R为字母ABCDEF(6个)，测试模板T为1234(4个)。R和T中各元素之间的距离已经给出。如下：

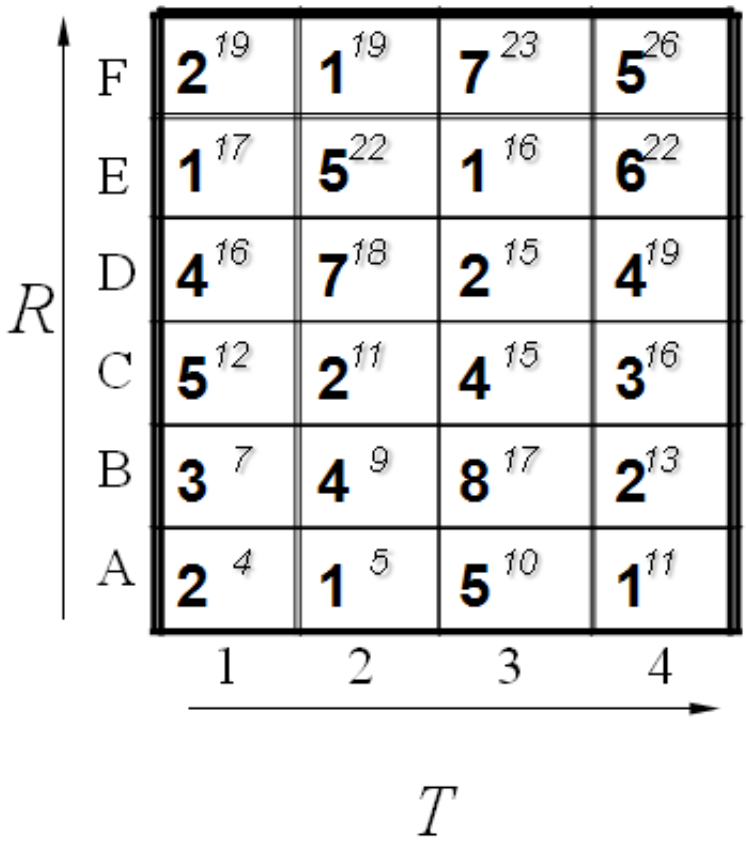


既然是模板匹配，所以各分量的先后匹配顺序已经确定了，虽然不是一一对应的。现在题目的目的是要计算出测试模板T和标准模板R之间的距离。因为2个模板的长度不同，所以其对应匹配的关系有很多种，我们需要找出其中距离最短的那条匹配路径。现假设题目满足如下的约束：当从一个方格((i-1,j-1)或者(i-1,j)或者(i,j-1))中到下一个方格(i,j)，如果是横着或者竖着的话其距离为d(i,j)，如果是斜着对角线过来的则是2d(i,j).其约束条件如下图像所示：

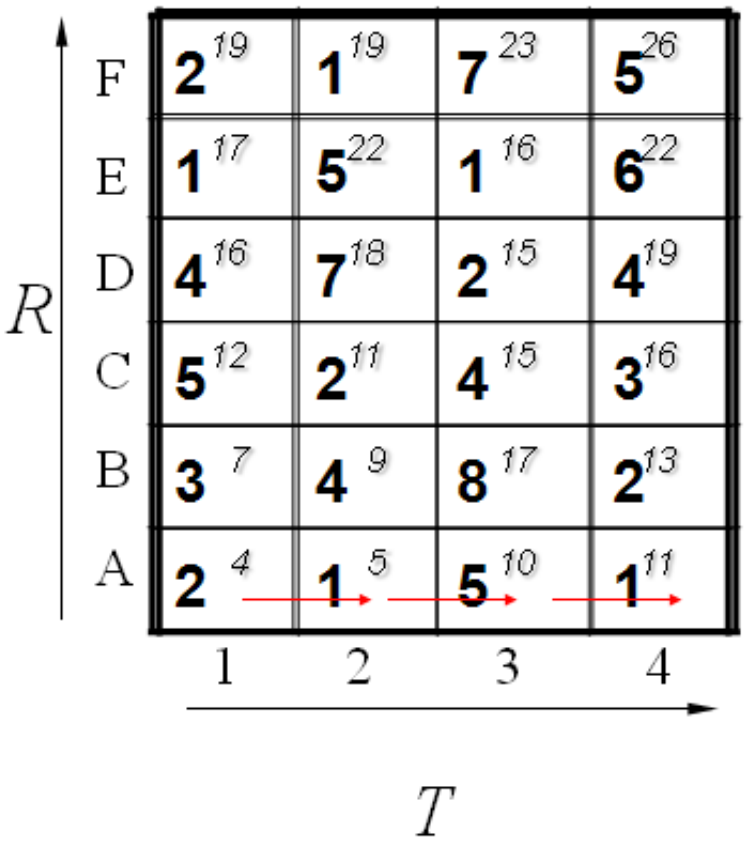
$$g(ck)=g(i_k,\ j_k)=g(i,\ j)=\min\begin{cases} g(i\text{-}1,\ \ j)+\ d(i,\ j) \\ g(i\text{-}1,\ j\text{-}1)+2d(i,\ j) \\ g(i,\ \ j\ \text{-}1)+\ d(i,\ j) \end{cases}$$

其中g(i,j)表示2个模板都从起始分量逐次匹配，已经到了M中的i分量和T中的j分量，并且匹配到此步是2个模板之间的距离。并且都是在前一次匹配的结果上加d(i,j)或者2d(i,j),然后取最小值。

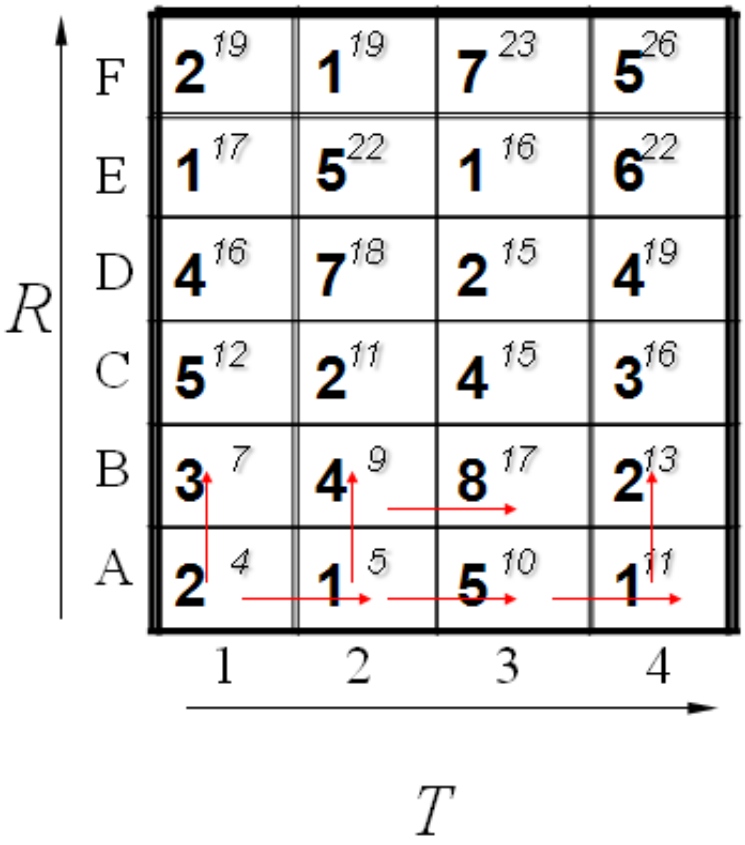
所以我们将所有的匹配步骤标注后如下：



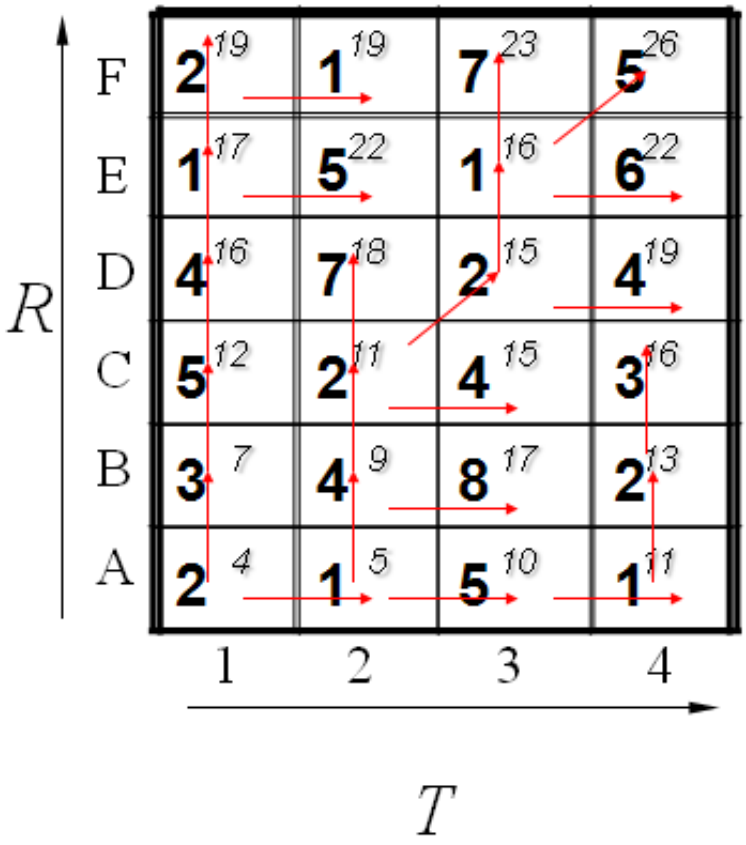
怎么得来的呢？比如说g(1,1)=4, 当然前提都假设是g(0,0)=0,就是说g(1,1)=g(0,0)+2d(1,1)=0+2*2=4.　g(2,2)=9是一样的道理。首先如果从g(1,2)来算的话是g(2,2)=g(1,2)+d(2,2)=5+4=9,因为是竖着上去的。如果从g(2,1)来算的话是g(2,2)=g(2,1)+d(2,2)=7+4=11,因为是横着往右走的。如果从g(1,1)来算的话，g(2,2)=g(1,1)+2*d(2,2)=4+2*4=12.因为是斜着过去的。综上所述，取最小值为9. 所有g(2,2)=9.当然在这之前要计算出g(1,1),g(2,1),g(1,2).因此计算g(I,j)也是有一定顺序的。其基本顺序可以体现在如下：



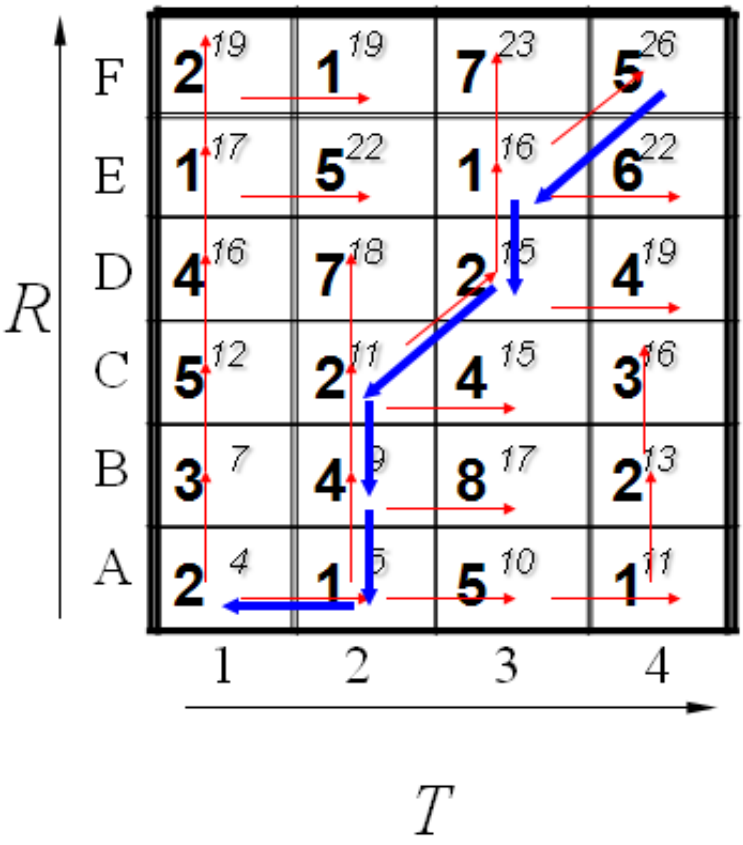
计算了第一排，其中每一个红色的箭头表示最小值来源的那个方向。当计算了第二排后的结果如下：



最后都算完了的结果如下：



到此为止，我们已经得到了答案，即2个模板直接的距离为26. 我们还可以通过回溯找到最短距离的路径，通过箭头方向反推回去。如下所示：



到这里，估计大家动手算一下就会明白了。其实很简单，通过例子的学习后再回去看那些枯燥的理论公式就发现很容易了

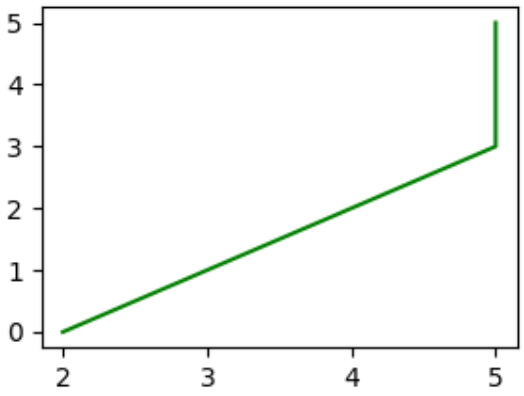
在实际应用中，比如说语音识别中的孤立词识别，我们首先训练好常见字的读音，提取特征后作为一个模板。当需要识别一个新来的词的时候，也同样提取特征，然后和训练数据库中的每一个模板进行匹配，计算距离。求出最短距离的那个就是识别出来的字了。

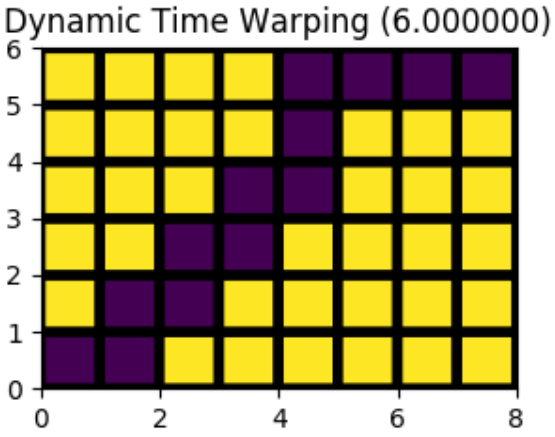
- 算法Demo

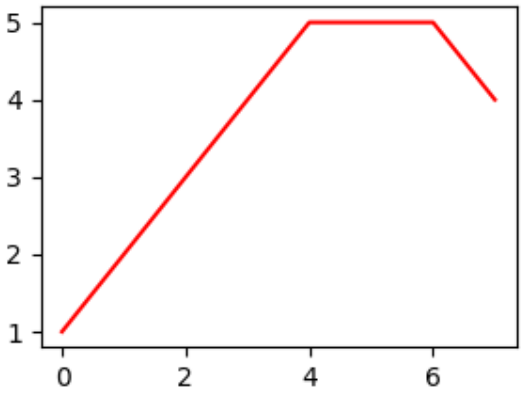
```
from math import *
import matplotlib.pyplot as plt
import numpy
def print_matrix(mat):
    print '[matrix] width : %d height : %d' % (len(mat[0]), len(mat))
    print '-----------------------------------'
    for i in range(len(mat)):
        print mat[i]  # [v[:2] for v in mat[i]]

def dist_for_float(p1, p2):
    dist = 0.0
```

```
    elem_type = type(p1)
    if elem_type == float or elem_type == int:
        dist = float(abs(p1 - p2))
    else:
        sumval = 0.0
        for i in range(len(p1)):
            sumval += pow(p1[i] - p2[i], 2)
        dist = pow(sumval, 0.5)
    return dist

def dtw(s1, s2, dist_func):
    w = len(s1)
    h = len(s2)

    mat = [([[0, 0, 0, 0] for j in range(w)]) for i in range(h)]

    # print_matrix(mat)

    for x in range(w):
        for y in range(h):
            dist = dist_func(s1[x], s2[y])
            mat[y][x] = [dist, 0, 0, 0]

    # print_matrix(mat)

    elem_0_0 = mat[0][0]
    elem_0_0[1] = elem_0_0[0] * 2

    for x in range(1, w):
        mat[0][x][1] = mat[0][x][0] + mat[0][x - 1][1]
        mat[0][x][2] = x - 1
        mat[0][x][3] = 0

    for y in range(1, h):
        mat[y][0][1] = mat[y][0][0] + mat[y - 1][0][1]
        mat[y][0][2] = 0
        mat[y][0][3] = y - 1

    for y in range(1, h):
        for x in range(1, w):
            distlist = [mat[y][x - 1][1], mat[y - 1][x][1], 2 * mat[y - 1][x - 1][1]]
            mindist = min(distlist)
            idx = distlist.index(mindist)
            mat[y][x][1] = mat[y][x][0] + mindist
            if idx == 0:
                mat[y][x][2] = x - 1
                mat[y][x][3] = y
            elif idx == 1:
                mat[y][x][2] = x
                mat[y][x][3] = y - 1
            else:
```

```
                mat[y][x][2] = x - 1
                mat[y][x][3] = y - 1

    result = mat[h - 1][w - 1]
    retval = result[1]
    path = [(w - 1, h - 1)]
    while True:
        x = result[2]
        y = result[3]
        path.append((x, y))

        result = mat[y][x]
        if x == 0 and y == 0:
            break

    # print_matrix(mat)

    return retval, sorted(path)

def display(s1, s2):
    val, path = dtw(s1, s2, dist_for_float)
    w = len(s1)
    h = len(s2)
    mat = [[1] * w for i in range(h)]
    for node in path:
        x, y = node
        mat[y][x] = 0
    mat = numpy.array(mat)
    plt.subplot(2, 2, 2)
    c = plt.pcolor(mat, edgecolors='k', linewidths=4)
    plt.title('Dynamic Time Warping (%f)' % val)
    plt.subplot(2, 2, 1)
    plt.plot(s2, range(len(s2)), 'g')
    plt.subplot(2, 2, 4)
    plt.plot(range(len(s1)), s1, 'r')
    plt.show()

s1 = [1, 2, 3, 4, 5, 5, 5, 4]
s2 = [3, 4, 5, 5, 5, 4]
s2 = [1, 2, 3, 4, 5, 5]
s2 = [2, 3, 4, 5, 5, 5]
# val, path = dtw(s1, s2, dist_for_float)
display(s1, s2)
```


5
4
3
2
1
0
2
3
4
5


Dynamic Time Warping (6.000000)
6
5
4
3
2
1
0
0
2
4
6
8


5
4
3
2
1
0
2
4
6