CHAPTER

5 Template Matching

5.1 INTRODUCTION

In this chapter, we assume that each class is represented by a single pattern. A set of such *reference* patterns (or *prototypes*) is available and stored in a database. Given an unknown *test* pattern, template matching consists of searching the database for the reference pattern most "similar" to the given test pattern. This is equivalent to defining a matching cost that quantifies similarity between the test pattern and the reference patterns.

Template-matching techniques are very common in string matching, speech recognition, alignment of molecular sequences, image retrieval, and so forth. They often come with different names depending on the application. For example, in speech recognition the term *dynamic time warping* is used, whereas in string matching *Edit* (or *Levenstein*) *distance* is quite common.

This chapter is devoted to a series of examples of increasing complexity, culminating in an example from speech recognition.

5.2 THE EDIT DISTANCE

A *string* pattern is defined as an *ordered sequence of symbols* taken from a discrete and finite set. For example, if the finite set consists of the letters of the alphabet, the strings are words. The *Edit distance* between two string patterns A and B, denoted $D(A,B)$, is defined as the minimum total number of changes (C), insertions (I), and deletions (R) required to change pattern A into pattern B,

$$D(A,B) = \underbrace{\min}_{j}[C(j) + I(j) + R(j)] \tag{5.1}$$

where j runs over all possible combinations of symbol variations in order to obtain B from A. If the two strings are exactly the same, then $D(A,B) = 0$. For every symbol "change," "insertion," or "deletion," the cost increases by one.

The required minimum is computed by means of the dynamic programming methodology [Theo 09, Section 8.2.1]. That is, an optimal path is constructed in the 2-dimensional grid formed by the two sequences in the 2-dimensional space, by locating one sequence across the horizontal axis and the other across the vertical axis. The Edit distance is commonly used in spell-checking systems where the prototypes stem from the vocabulary of words.

DOI: 10.1016/B978-0-12-374486-9.00005-1

Example 5.2.1. Compute the Edit distance between the words "book" and "bokks," taking the former as the reference string. Plot the optimal matching path and comment on the sequence of operations needed to change "bokks" to "book." Repeat for the words "template" (reference) and "teplatte."

Solution. Use function *editDistance* by typing

```
[editCost,Pred]=editDistance('book','bokks');
```

The Edit distance equals 2 and is stored in the variable *editCost*. The value of 2 is the result of a symbol change (*k* to *o*) and a deletion (*s* at the end of the word). To extract the matching path, give matrix *Pred* as input to function *BackTracking* as follows:

```
L_test=length('bokks'); % number of rows of the grid
L_ref=length('book'); % number of columns of the grid
[BestPath]=BackTracking(Pred,L_test,L_ref,1,'book','bokks');
% The fourth input argument indicates that a plot of the best
% path will be generated. The last two arguments serve as labels for the
resulting axes.
```

The resulting best path is stored in the vector variable *BestPath* and is presented in Figure 5.1(a), where the reference pattern has been placed on the horizontal axis. Each element of vector *BestPath* is a complex number and stands for a node in the path. The real part of the element is the node's row index and the imaginary part is its column index. Inspection of the path in Figure 5.1(a) reveals that the first

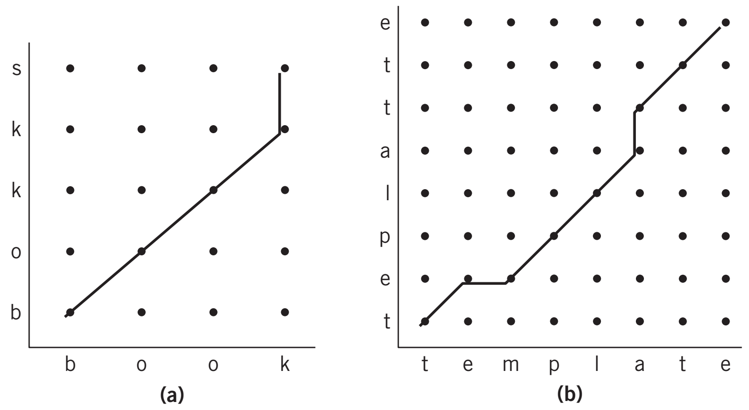

FIGURE 5.1

(a) Optimal matching path between "book" and "bokks." (b) Optimal matching path between "template" and "teplatte." A diagonal transition between two successive nodes amounts either to zero cost (if the corresponding symbols are the same) or to one (if the corresponding symbols are different). Horizontal transitions (symbol insertions) and vertical transitions (deletions) contribute one to the total cost.

occurrence of k in “bokks” has been changed to o. In addition, the vertical segment of the best path is interpreted as the deletion of s in “bokks.” A total of one symbol change and one symbol deletion is needed to convert “bokks” to “book” in an optimal way.

Similarly, the Edit cost for matching “teplatte” against “template” equals 2, and the resulting best path can be seen in Figure 5.1(b). In this case, an insertion (horizontal segment) and a deletion (vertical segment) are required to convert “teplatte” to the reference pattern in an optimal way. ■

Exercise 5.2.1

Given the words “impose,” “ignore,” and “restore” as prototypes, determine which one stands for the most likely correction of the mistyped word “igposre” in terms of the edit distance. Note that, as is the case with most spell checkers, ties result in multiple correction possibilities.

5.3 MATCHING SEQUENCES OF REAL NUMBERS

In this section, we focus on a slightly different task, that of matching sequences of real numbers. In contrast to Section 5.2, where the goal was to change one string pattern to another, the aim here is to measure how similar/dissimilar are two given *ordered* sequences of numbers. For example, if we are given two real numbers, x, y, their similarity can be quantified by the absolute value of their difference. If we are given two vectors (i.e., two strings of real numbers of *equal* length), we can use the respective Euclidean distance. A more interesting case is when two sequences of numbers are of different length. One approach to this problem is to allow local “stretching”/“compressing,” known as *warping*, achieved by constructing the optimal (low-cost) path through nodes of the respective grid. The grid is formed by the two sequences in the 2-dimensional space by locating one sequence along the horizontal axis and the other along the vertical axis.

Assuming that the reference sequence is placed on the horizontal axis, the dimensions of the grid are $J \times I$, where I and J are the lengths of the reference and test sequences, respectively. In the simplest case, the cost assigned to each node of the grid is equal to the absolute value of the difference between the respective numbers associated with a specific node. The type and the allowable degree of expansion/compression are determined by the so-called local constraints. Popular choices include the Sakoe-Chiba and Itakura constraints [Theo 09, Section 8.2]. Basically, these are constraints imposed on the allowable jumps among nodes in the grid.

The purpose of matching sequences of numbers is pedagogical and is used as an intermediate step to help the reader acquire a better understanding of the concepts underlying dynamic time warping for speech recognition, which is treated in the next section (see [Theo 09, Section 8.2.3]).

Example 5.3.1. Let $P = \{-1, -2, 0, 2\}$ be the prototype and $T = \{-1, -2, -2, 0, 2\}$ be the unknown pattern.

1. Compute the matching cost and the resulting best path by adopting the Sakoe-Chiba local constraints. Comment on the shape of the resulting best path.
2. Repeat with $T = \{-1, -2, -2, -2, -2, 0, 2\}$.

Solution

Step 1. For $T = \{-1,-2,-2,-2,-2,0,2\}$, use function *DTW Sakoe* and type

```
P=[-1,-2,0,2];
T=[-1,-2,-2,0,2];
[MatchingCost,BestPath,D,Pred]=DTWSakoe(P,T,1);
```

where D is the array having as elements the costs associated with optimally reaching each node of the grid. The value 1 is used if a plot is required; if not, 0 is used. Although the two sequences differ by one symbol, the matching cost equals 0. This is due to the fact that a) the absolute difference was employed as the (node) cost and b) the only difference between the two sequences is symbol repetition.

To further interpret the result, observe the respective best path in Figure 5.2(a). It can be seen that the vertical segment of the path corresponds to a local stretching operation; that is, the symbol -2 of the prototype (horizontal axis) is matched against two consecutive occurrences of -2 in sequence T.

Step 2. To repeat the experiment with $T = \{-1,-2,-2,-2,-2,0,2\}$ type

```
P=[-1,-2,0,2];
T=[-1,-2,-2,-2,-2,0,2];
[MatchingCost,BestPath,D,Pred]=DTWSakoe(P,T,1);
```

The matching cost remains 0 and the resulting best path is presented in Figure 5.2(b). It can be seen that the vertical segment of the path is now four nodes long. This should not come as a surprise

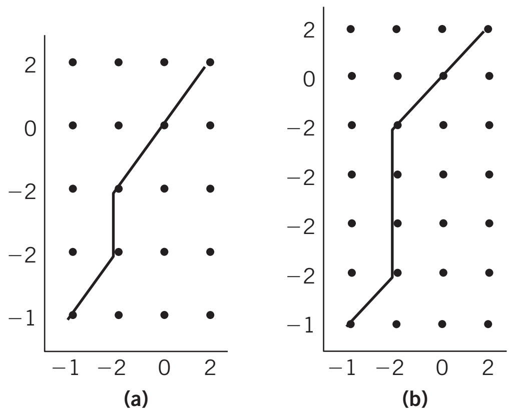

FIGURE 5.2

(a) Best path for $P = \{-1,-2,0,2\}$ and $T = \{-1,-2,-2,0,2\}$. (b) Best path for $P = \{-1,-2,0,2\}$ and $T = \{-1,-2,-2,-2,-2,0,2\}$.

because, as before, T differs from P in terms of *symbol repetition*. *The length of repetition does not affect the cost*; it only causes a more intense time-warping effect. To draw an analogy with speech, we may have the same phoneme but one time it is said fast and another time slowly. Long horizontal or vertical segments are very common when the Sakoe-Chiba local constraints are employed. If no global constraints are specified, the horizontal (vertical) segments can become arbitrarily long. This behavior is often undesirable with real-world signals, such as in speech.

Example 5.3.2. Let $P = \{1,0,1\}$ be the prototype, and let $T_1 = \{1,1,0,0,0,1,1,1\}$, $T_2 = \{1,1,0,0,1\}$ be two unknown patterns. Compute the matching cost for the standard Itakura local constraints between P and T_1 and between P and T_2.

Solution. To compute the matching cost for the two unknown patterns using the standard Itakura local constraints, use function *DTWItakura* and type

```
P=[1,0,1];
T1=[1,1,0,0,0,1,1,1];
T2=[1,1,0,0,1];
[MatchCost1,BestPath1,D1,Pred1]=DTWItakura(P,T1,1);
[MatchCost2,BestPath2,D2,Pred2]=DTWItakura(P,T2,1);
```

The returned value of *MatchCost*1 is ∞, whereas the value of *MatchCost*2 is 0. This is because one property of the standard Itakura constraints is that the maximum allowed stretching factor for the prototype is 2. In other words, the length of the unknown pattern has to be, in the worst case, twice the length of the prototype. If this rule is violated, the *DTWItakura* function returns ∞. In the case of P and T_2, this rule is not violated and the best path can be seen in Figure 5.3.

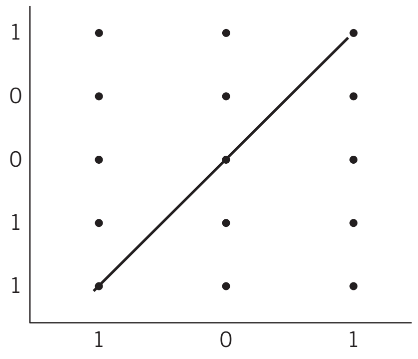

FIGURE 5.3

Best path for $P = \{1,0,1\}$ and $T = \{1,1,0,0,1\}$ using the standard Itakura local constraints.

Example 5.3.3. This example demonstrates the importance of the endpoint constraints [Theo 09, Section 8.2.3]. Let the sequence $P = \{-8, -4, 0, 4, 0, -4\}$ be a prototype. Also let the sequence $T = \{0, -8, -4, 0, 4, 0, -4, 0, 0\}$ be the unknown pattern.

1. Compute the matching cost by adopting the Sakoe-Chiba local constraints and comment on the result.
2. Repeat, allowing for endpoint constraints. Specifically, omit at most two symbols from each endpoint of T.

Solution

Step 1. For the first case, type

```
P=[-8,-4,0,4,0,-4];
T=[0,-8,-4,0,4,0,-4,0,0];
[MatchingCost,BestPath,D,Pred]=DTWSakoe(P,T,1);
```

The matching cost turns out to be 16. In practice, the cost is normalized by dividing it by the length of the best path. In this example, the best path is 9 nodes long, as can be seen in Figure 5.4(a), and so the normalized cost is 1.778. Hence, although P and T can be considered practically the same (they only differ in the trailing zeros), the matching cost is nonzero.

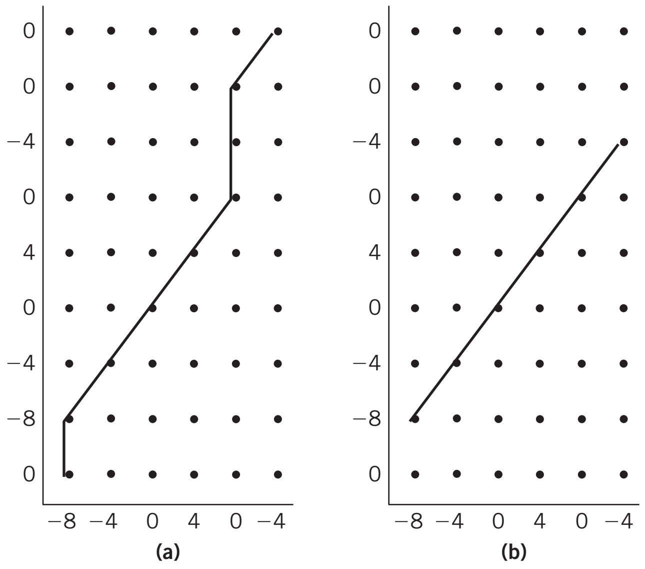

FIGURE 5.4

(a) Best path for $P = \{-8, -4, 0, 4, 0, -4\}$ and $T = \{0, -8, -4, 0, 4, 0, -4, 0, 0\}$, no endpoint constraints.
(b) Previous sequences matched while allowing endpoint constraints.

Inspection of Figure 5.4(a) reveals that time stretching occurs at the beginning and end of the path and is due to the existence of zeros at the endpoints. This is a common situation; that is, the unknown pattern contains "garbage" near the endpoints. As a remedy, we can resort to a variation of the standard matching scheme, where it is possible to omit a number of symbols at the endpoints of the unknown pattern; it suffices to specify the maximum number of symbols to omit. This type of enhancement to the standard matching mechanism can be easily embedded in the dynamic programming methodology.

Step 2. To employ the endpoint constraints in the current example, use function *DTWSakoeEndp* and type

```
P=[-8,-4,0,4,0,-4];
T=[0,-8,-4,0,4,0,-4,0,0];
[MatchingCost,BestPath,D,Pred]=DTWSakoeEndp(P,T,2,2,1);
```

In this function call, the third and fourth arguments stand for the number of symbols that can be omitted at each endpoint (2 in this example). The fifthindicates that a plot of the best path will be generated. The resulting matching cost is zero, as can be verified from Figure 5.4(b), where the first row and the last two rows of the grid have been skipped by the algorithm.

Endpoint constraints are very useful in speech recognition because the unknown pattern usually has silence periods around the endpoints, whereas the prototypes are free of such phenomena. ■

5.4 DYNAMIC TIME WARPING IN SPEECH RECOGNITION

Here we focus on a simple task in speech recognition known as *isolated word recognition (IWR)*. We assume that the spoken utterance consists of discrete words; that is, there exist sufficient periods of silence between successive words (hence the term "isolated"). This is a convenient assumption, since it allows for employing segmentation algorithms capable of locating the boundaries (endpoints) of the spoken words with satisfactory precision. Note that in practice a certain amount of silence/noise is likely to exist close to the endpoints of the detected word after the segmentation stage.

At the heart of any IWR system is an architecture consisting of a set of reference patterns (prototypes) and a distance measure. Recognition of a test (unknown) pattern is achieved by searching for the best match between the test and each one of the reference patterns, on the basis of the adopted measure.

As a first stage, a feature extraction algorithm converts each signal into a sequence of feature vectors—instead of matching sequences of real numbers, the task is computing the matching cost between two sequences of vectors. However, the rationale is the same, and the only difference lies in replacing the absolute value with the Euclidean distance between vectors. In speech recognition, this type of matching is known as *dynamic time warping* (DTW).

We now develop a simple, speaker-dependent IWR system for a 10-word vocabulary consisting of "zero," "one," . . . , "nine" and uttered by a single male speaker. We are actually building an "isolated digit recognition" system. We need a total of 10 utterances to use as prototypes and a number of utterances for testing.

At this point, you may record your own audio files or you may use the files available via this book's website. If you record your own files, name the prototypes as follows: *zero.wav*, *one.wav*, and so on, for the sake of compatibility, and place all 10 in a single folder (this is the naming convention we have adopted for the audio samples on the website). Note that "clean" prototypes are desirable. That is, make sure silence/noise has been removed, at least to the extent possible, from the endpoints of the prototypes before storing. Such care need not be taken with the samples that will be used for testing. In addition, the file names for the test patterns need not follow any naming convention and it suffices to store the unknown patterns in the folder where the prototypes are held.

To build the system, we use short-term energy and short-term zero-crossing rate as features [Theo 09, Section 7.5.4], so that each signal is represented by a sequence of 2-dimensional feature vectors. Note that this is not an optimal feature set in any sense and has only been adopted for simplicity.

The feature extraction stage is accomplished by typing the following code:

```
protoNames={'zero','one','two','three','four','five',...
   'six','seven','eight','nine'};
for i=1:length(protoNames)
   [x,Fs,bits]=wavread(protoNames{i}]);
   winlength = round(0.02*Fs); % 20 ms moving window length
   winstep = winlength; % moving window step. No overlap
   [E,T]=stEnergy(x,Fs,winlength,winstep);
   [Zcr,T]=stZeroCrossingRate(x,Fs,winlength,winstep);
   protoFSeq{i}=[E;Zcr];
end
```

which performs feature extraction per prototype and uses a single cell array (*protoFSeq*) to hold the feature sequences from all prototypes. To extract the features, a moving windows technique is employed [Theo 09, Section 7.5.1]. The length of the moving windows is equal to 20 milliseconds and there is no overlap between successive windows.

To find the best match for an unknown pattern—for example, a pattern stored in file *upattern1.wav*—type the following code:

```
[test,Fs,bits]=wavread('upattern1');
winlength = round(0.02*Fs); % use the same values as before
winstep = winlength;
[E,T]=stEnergy(test,Fs,winlength,winstep);
[Zcr,T]=stZeroCrossingRate(test,Fs,winlength,winstep);
Ftest=[E;Zcr];

tolerance=0.1;
LeftEndConstr=round(tolerance/winstep); % left endpoint constraint
RightEndConstr = LeftEndConstr;
for i=1:length(protoNames)
   [MatchingCost(i),BestPath{i},D{i},Pred{i}]=DTWSakoeEndp(...
         protoFSeq{i},Ftest,LeftEndConstr,RightEndConstr,0);
end
```

```
[minCost,indexofBest]=min(MatchingCost);
fprintf('The unknown pattern has been identified as %s \n',...
       protoNames{indexofBest});
```

This code uses the standard Sakoe local constraints and allows for endpoint constraints. Specifically, it is assumed that the length of silence/noise at each endpoint may not exceed 0.1 seconds (i.e., at most 5 frames can be skipped from each endpoint of the test utterance). Note that, instead of using the function *DTWSakoeEndp* to compute the matching cost, we could have used *DTWItakuraEndp*. To avoid repeating the code for each unknown pattern, the whole system is available on the website as a single m-file under the name *IsoDigitRec.m*.

Remarks

- Errors may occur in the experiments. This is also expected in practice, and is even more true here since only two features have been used for pedagogic simplicity.
- Moreover, this a speaker-dependent speech recognition example. Hence, if you record your own voice and test the system using the provided prototypes, the performance may not to be a good one, especially if the accent of the speaker is very different from the accent we used to record the prototypes. You can reconstruct the whole example by using your own voice for both the test data and the prototypes.