references.bib)

LINFO2275 : project 2

Hand gesture recognition

Rousseau Mathieu, 67001800

UCLouvain
Belgium
21/05/2023

1 Introduction

We want to implement a hand gesture recognition system for a smart user interface. A 3D tracking of the position vector $\tilde{\mathbf{r}}(t) = (x(t), y(t), z(t))^T$ of the hand is recorded as a sequence of the 3 coordinates in function of the time t [1]. The user is executing a sketch that is recorded and recognized by the system.

The system will be trained in order to recognize two different kind of sketches.

A first dataset contains the records of the numbers $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ in three dimensions. We asked 10 users to draw 10 times each of these numbers. We have therefore a total of 1000 sequences.

A second dataset contains the records of three-dimensional figures. As for the precedent dataset, 10 users have drawn each figure 10 times.

2 Preprocessing

We first preprocessed the data. We embedded the datas in a list of 1000 entries. Each entry consist in a matrix of dimension ($n\_timepoints$, $vec\_dimensions$) where $n\_timepoints$ is the length of the time series and $vec\_dimensions$ is the dimension of the position vector which is 3 in our case. Notice that each time series has different length, that's important for what is to follow. After that, we resampled each time series into $N = 64$ points as proposed by.

3 Dynamic time warping

3.1 Introduction

Given a time series of three-dimensional unknown vector data points, we would like to predict the sketch represented by these points. In order to do this, we used a K-Nearest Neighbors approach where we find the k most similar time series in the dataset and use the most represented target value among the k time series and use it as the prediction output.

To find the k most similar time series, we use a distance function as the similarity measure. However, we cannot use the often used euclidean distance as it produces pessimistic results when dealing with time series of different lengths.

Dynamic time warping - a often used technique in speech recognition problem - is a solution to this problem, it finds the optimum alignment between two time series by "warping" non-linearly a time series stretching or shrinking its time axis. Once we found the optimal alignment, we can determine the similarity between these twos.

Let $s = (s_1, \ldots, s_n)$ and $t = (t_1, \ldots, t_m)$ be two time series of length n and m respectively where s_i, t_j are the position vector $\in \mathbb{R}^3$ for $i = 1, \ldots, n$ and $j = 1, \ldots, m$. We define a warping of order $n \times m$ as a sequence of k points $x = (x_1, \ldots, x_k)$ where $x_l = (i_l, j_l) \in [1, n] \times [1, m]$. Each points x_l of the path aligns the vector s_{i_l} with the vector t_{i_l}.

The following constraints apply,

- **boundary conditions** : $x_1 = (1, 1)$ and $x_k = (n, m)$. This ensures that every index of the time series are used in the warp path.
- **locality** : $x_{l+1} - x_l \in \{(1, 0), (1, 1), (0, 1)\}, \quad \forall l \in [1, k-1]$.

The locality condition implies a monotonicity condition : $s_1 \leq \cdots \leq s_n$ and $t_1 \leq \ldots s_m$. This monotonicity implies that the lines representing the warp path do not overlap.

---

1. Note that we will not consider the precise recorded time of the observations.

Among all the warping path we can construct, want to find the optimal warping path x^* that consists in the warping path that minimizes the square root of the following cost given for any warping path x,

$$C_x(s,t) = \sum_{l=0}^{k} ||s_{i_l} - t_{j_l}||_2^2 \tag{1}$$

We can then compute the DTW distance by computing the total cost of the warping path x^*.

In practice, to find the optimal path, we use a dynamic programming approach which consists in the following algorithm,

1. We define a distance matrix $D \in \mathbb{R}^{n \times m}$ where $D_{i,j}$ is the minimum-distance for the warping path between $s' = (s_1, \ldots, s_i)$ and $t' = (t_1, \ldots, t_j)$.
2. Each entry is computed by using the following recursion formula,

$$D_{i,j} = \sqrt{||s_i - r_j||} - \min(D_{i-1,j}, D_{i-1,j-1}, D_{i,j-1}) \tag{2}$$

   with the initial condition $D_{1,1} = 0$
3. The entry $D(n,m)$ contains the minimum-distance for the warping path between s and t.

This algorithm has $\mathcal{O}(nm)$ time complexity. We can however speed up the algorithm by applying **contraints** that consists in limiting the number of cells that are evaluated in the distance matrix (e.g. using a Sakoe-Chiba Band window [2]) and **data abstraction** that consists in performing the algorithm on a reduced representation of the data. Instead of running the algorithm on the full distance matrix, we reduce the size of the time series which implies that the number of cells in the distance matrix are reduced. Once the warping path is found on this low-resolution matrix, we map it back to a higher-resolution matrix. This projected path is used as an initial guess for a new warping path on this new distance matrix. In other words, the projected path is refined on the hiher-resolution matrix by making **local adjustments** [3] which implies that this matrix is only filled in the neighborhood projected path. This results in a algorithm with linear time complexity since the warping path grows linearly with the size of the time series.

Researches **keylist** have shown that using a single neighbor in the KNN algorithm gives the best results. Moreover, trying different combination of number of neighbors and radius hyperparameters, we notices that a good tradeoff between efficiency and rapidity was not only obtained with a single neighbor but also with a radius of 1.

3.2 Results

3.2.1 User-independent cross-validation

3.2.2 User-dependent cross-validation

4 $P-recognizer

In this algorithm, we consider each gesture as a set of points : $\{ pi = (x_i, y_i) : i = 1, ..., N \}$. In our case, because of the resampling made in the preprocessing step, each gesture is represented by a set of $N = 64$ points. It is worth noticing that gesture are seen as a cloud of points so that there does not need to have order in the set (i.e. p_1 is not the starting point nor p_i follow p_{i-1}) even though it's the case for us.

---

2. In a nutshell, this window assumes that the best path will not stray too far away from the main diagonal of the distance matrix.

3. These local adjustments can be controlled through a radius parameter that controls the number of cells that will be evaluated around the path.

Let C and T be clouds of points. The first one correspond to a gesture candidate whereas the second one is a template in the training set. The aim of this algorithm is to match C to T by the use of a nearest neighbor classifier (KNN) in the same idea as in the precedent section. Here we obviously do not use a DTW algorithm to compute the distance but the sum of euclidean distances for all pairs of points (C_i, T_j) where $C_i \in C$ is matched to $T_j \in T$ through some function $f : \mathbb{R}^2 \to \mathbb{R}^2, \quad C_i \to f(T_j)$,

$$d(C, T) = \sum_{i=1}^{N} ||C_i - T_j||_2 \tag{3}$$

However this algorithm require to search for the minimum matching distance between C and T from $n!$ possible alignments. The library we used to run this algorithm use a special heuristic called Greedy-5**keylist**. The idea is that for each point $C_i \in C$, we find the closest point in T that has not been matched yet. Once it is matched with C_i, we continue with C_{i+1} and so on until all the points of C are matched with a point of T. The closeness is computed through a weighted euclidean distance,

$$d(C, T) = \sum_{i=1}^{N} w_i ||C_i - T_j||_2 \tag{4}$$

where w_i indicate the confidence in each pair (C_i, T_j).

Basically, the first match has a weight of 1 since C_i has all the points of T to choose for the closest match. As long as the possibility of matching reduce, the confidence also. The formula for the weights translate a linearly decreasing weighting,

$$w_i = 1 - \frac{i-1}{N} \tag{5}$$

This heuristic allows us to have a time complexity of $\mathcal{O}(n^{2+\epsilon})$ which is comparable to the DTW algorithm without any optimization.

4.1 Results

4.1.1 User-independent cross-validation

4.1.2 User-dependent cross-validation

Appendix