Algoritmos de Programación Dinámica

Algorítmica

Grupo D2

Pablo Mariano Moreno Mancebo

Descripción del trabajo

El objetivo de la práctica consiste en que sea capaz de analizar un problema y plantear una solución al mismo mediante una técnica de diseño de algoritmos de programación dinámica.

Para ello siguiendo el ejercicio guiado se resuelve el ejercicio asignado al DNI : 76067676 - Ejercicio nº 3 : Comparación de señales con Dynamic Time Warping

### 

Índice

[**Enunciado del problema - Ejercicio 3**](#_tkmiyq8oz7m6) **3**

[**Análisis y notación a utilizar**](#_hefv3vajzcxt) **3**

[**Diseño de componentes**](#_s0r1sgpjpbkz) **4**

[-Algoritmo diseñado](#_xpeffw98ycd) 4

[**Eficiencia**](#_b1vo0gqfr5y9) **6**

[**Detalles de implementación**](#_uld2ioxumtg5) **6**

[**Pruebas**](#_pxijvobdfhd2) **7**

# Enunciado del problema - Ejercicio 3

**Comparación de señales con Dynamic Time Warping**

En el análisis de señales con aprendizaje automático, es común comparar dos secuencias de datos (números enteros o reales) X=(x1, x2, ..., xm) e Y=(y1, y2, ..., yn), para conocer su similitud. Un ejemplo de este caso es el análisis de la marcha y la contabilización de pasos con los datos obtenidos por acelerómetros de teléfonos móviles o pulseras de actividad. El algoritmo Dynamic Time Warping (DTW) es una solución eficiente que nos permite conocer cómo de similares son dos señales de datos X e Y proporcionadas como entrada, buscando una medida de emparejamiento de una señal en otra. Un ejemplo de este emparejamiento se muestra en la siguiente figura, para dos señales Q (rojo) y C (azul).

Para comparar dos señales X=(x1, x2, ..., xm) e Y=(y1, y2, ..., yn), el valor DTW[i,j] devuelve la distancia mínima existente entre la señal X=(x1, x2, ..., xi) y la señal Y=(y1, y2, ..., yj). Se calcula como:

• DTW[i,j]= |X[i]-Y[j]| + DTW[i-1,j], si la distancia mínima de emparejamiento hasta X[i] e Y[j] (sin contar las posiciones i y j) es DTW[i-1,j],

• DTW[i,j]= |X[i]-Y[j]| + DTW[i,j-1], si la distancia mínima de emparejamiento hasta X[i] e Y[j] (sin contar las posiciones i y j) es DTW[i,j-1],

• DTW[i,j]= |X[i]-Y[j]| + DTW[i-1,j-1], si la distancia mínima de emparejamiento hasta X[i] e Y[j] (sin contar las posiciones i y j) es DTW[i-1,j-1].

Se pide: Implementar un algoritmo de programación dinámica que calcule la distancia DTW entre dos secuencias de números reales.

# Análisis y notación a utilizar

* Se dispone de una secuencia X para analizar. Para evitar problemas de notación, lo renombramos y llamaremos Sec1
* Se dispone de una secuencia Y para analizar. Para evitar problemas de notación, lo renombramos y llamaremos Sec2
* Se debe de tener un total de 2 secuencias.
* Se nos dice que devolvamos la distancia mínima entre dos secuencias

# Diseño de componentes

En primer lugar, asumimos que las secuencias están ordenados de menor a mayor

sin pérdida de generalidad. Esto será requisito para que el algoritmo a diseñar funcione en todos los casos correctamente.

Veamos si el problema se puede resolver con Programación Dinámica:

* **El problema a resolver es de minimización**. En concreto se debe encontrar la suma mínima de la resta de ambas secuencias.
* **El problema se puede resolver por etapas:** Se puede resolver recorriendo una matriz dada y vamos acumalon siempre la minima en M[n-1][m-1]
* **El problema debe poder modelarse mediante una ecuación recurrente**, viene dada en el enunciado.
* **Casos base:**
* Matriz vacía

minimo = 0 → M[0][0] = 0

* Columnas == 0

minimo = DTW[i-1][j] → Sec1[i] - Sec2[j] + minimo

* Para la filas == 0

minimo = DTW[i][j-1] → Sec1[i] - Sec2[j] + minimo

* **Se debe cumplir el Principio de Optimalidad de Bellman** en el cual se resume en que si una secuencia es óptima, entonces necesariamente cualquier subsecuencia que haya llevado a ésta también debe serlo.

Para comprobarlo en mi caso se cumple en todos los casos dado que al eliminar cualquier la suma siempre al restar ambas secuencias va a ser minima como en los casos de fibonacci.

## -Algoritmo diseñado

* Los datos los iremos guardando en una tabla T[i][j], donde las filas se corresponderán con sec1 y las columnas con sec2. Asumimos que pueden tomar valores tanto enteros como reales.
* La forma de rellenar la tabla será por filas y, en cada fila, de izquierda a derecha para facilitar la forma de asociar los datos DTW[i-1][j] , DTW[i-1][j-1] y DTW[i][j-1]
* Asumiremos no válidos los valores T[i][j] cuando i<0 ó j<0.En este caso, por simplicidad,no se considerarán en la ecuación en recurrencias aquellos casos no válidos.
* Para conocer , dado el valor T[n][M], cuál es el valor de DTW para entonces pero no de la suma total, para la suma total voy sumando la ‘diagonal’ del final a principio pero no justo el valor de la diagonal si no el valor que rodean a la diagonal (DTW[i-1][j] , DTW[i-1][j-1] y DTW[i][j-1])

Así, el algoritmo para el cálculo de los valores DTW [i][j] parará cuando se conozca el valor DTW [n][M] que es el que se desea saber, y tendrá como entrada:

* suma, el valor desde los casos hasta la susodicha ‘diagonal mínima’ nombrada anteriormente
* Sec1[1..n], un array con la secuencia 1
* Sec2[1..n], un array con la secuencia 2
* DTW, el valor que se obtiene.

**Según el diseño planteado anteriormente, asumimos que las secuencias están ordenadas de menor a mayor**

| **Algoritmo DTW (Sec1[1..n],Sec2[1..n])**  //Para rellenar los datos  Para i = 0 .. n hacer:  Para j = 0 .. m hacer:  resta = |sec1[i]-sec2[j]|  Si i==0 && j == 0 { Casos base }  minimo=0  En otro caso Si i==0 { Casos base }  minimo= DTW[i][j-1];  En otro caso Si j==0 { Casos base }  minimo= DTW[i-1][j];  Si no { Casos generales }  minimo=BuscarMinimo(DTW[i-1][j],DTW[i][j-1],DTW[i-1][j-1])  DTW[i][j]=resta + minimo  Fin para  Fin para  //Para dar la salida  Mientras i = n != 0 && j = m != 0  Si ( j== 0 )  i--;  suma += DTW[i][j];  En otro caso Si( i == 0)  j--;  suma += DTW[i][j];  Si no  minimo=BuscarMinimo(DTW[i-1][j],DTW[i][j-1],DTW[i-1][j-1])  Si (minimo == DTW[i-1][j-1])  i--;j--  En otro caso Si (minimo == DTW[i][j-1])  j--  En otro caso Si(minimo == DTW[i-1][j])  i--  suma += minimo |
| --- |

# Eficiencia

Al tener dos for anidados de 0 hasta n y m respectivamente y con operaciones elementales de O(1) sería **O(n²)**

la parte del while -- Mientras también tiene O(n²) pero al hacer el máximo sale el mismo **O(n²)**

Con respecto a la recurrencia tenemos : O(2^n ) por lo que hemos mejorado

# Detalles de implementación

* El lenguaje de programación utilizado para resolver la práctica ha sido C++. Se han tomado las siguientes decisiones de implementación:
* El índice para las secuencias irá de 0 a n-1, en lugar de 1 a n como se ha hecho en el diseño, por simplicidad para el manejo del lenguaje C++. Esto se extiende al cálculo de la memoria y de la ecuación en recurrencias.
* La memoria DTW se implementará como un vector de vectores
* Los arrays de secuencias se usa las bibliotecas de la stl :: vector por simplicidad
* Tenemos la precondición de que los elementos estén ordenados
* Toda la práctica se ha implementado bajo un único fichero main.cpp, que puede compilarse con la orden:

g++ -O2 -o eje main.cpp

**Algoritmo main:**

Leer valores de Sec1 por línea de comandos

Leer valores de Sec2 por línea de comandos

Aplicar algoritmo

Muestra el resultado del algoritmo -- la suma

# Pruebas

