UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIAS DE COMPUTAÇÃO

Projeto 2 – Programação Dinâmica

Disciplina: SCC0218 – Algoritmos Avançados e Aplicações – 2º Semestre de 2015

Prof. Dr.: Gustavo Enrique de Almeida Prado Alves Batista

Data de entrega: 23 de Novembro de 2015

Aluno:	**NºUSP:**
Fábio A. M. Pereira	7987435
Mateus Abrahão Cardoso	8658332
Matheus Cabral Manoel	9066470

Sumário

1 – Introdução

Nesse relatório explicamos brevemente a forma como implementamos o algoritmo de classificação de séries temporais: Dynamic Time Warping (DTW), além de um breve estudo sobre a efetividade de seu uso para a classificação de movimentos capturados com um acelerômetro.

2 – Implementação

Utilizamos a linguagem Java para implementar os algoritmos do projeto. Tal escolha foi feita uma vez que o Java abstrai o gerenciamento de memória, permitindo que nós focássemos nos detalhes algorítmicos do problema. Além disso os membros do grupo possuem maior familiaridade com a linguagem Java do que com outras linguagens, facilitando o trabalho em equipe.

2.1 – Classe Movimento

Representamos cada linha dos arquivos de teste e treino por meio de uma classe java chamada Movimento. Nesta classe armazenamos em um **int** o tipo de movimento que foi realizado, que é um número de 1 a 12, e os valores da série temporal em uma **ArrayList** de valores numéricos do tipo **Double**.

Segue abaixo a representação da classe:

```java
import java.util.ArrayList;

public class Movimento {

    private int classe;                 // Tipo de movimento realizado, de 1 a 12: direita, esquerda, etc.
    private ArrayList<Double> serie;    // Lista de valores que representam a série temporal

    // Construtor da Classe Movimento
    public Movimento(int classe) {...
    }
    // Insere um valor numérico da série temporal
    public void addNumber(double num) {
        this.serie.add(num);
    }
    // Retorna o "i"ésimo valor numérico da série temporal
    public double get(int i) {
        return serie.get(i);
    }
    // Retorna a quantidade de valores numéricos que a série possui
    public int getSerieSize() {
        return this.serie.size();
    }
    // Retorna o tipo de movimento que a série representa: Um valor de 1 a 12
    public int getClasse() {
        return this.classe;
    }
    // Retorna uma string contendo o tipo de movimento + os valores numéricos da série
    public String toString() {...
    }
}
```

2.2 – Algoritmo DTW

Apresentamos abaixo a nossa implementação do algoritmo DTW seguindo os moldes da relação de recorrência passada na especificação do projeto:

```java
public double dtwDistance(Movimento a, Movimento b) {

    double dtw[][] = new double[a.getSerieSize()+1][b.getSerieSize()+1];

    // Inicializamos a primeira linha e coluna das séries "a" e "b" como Double.MAX_VALUE (equivalente ao INFINITY)
    for(int i=0; i<=a.getSerieSize(); i++)
        dtw[i][0] = Double.MAX_VALUE;
    for(int i=0; i<=b.getSerieSize(); i++)
        dtw[0][i] = Double.MAX_VALUE;
    dtw[0][0] = 0;

    // Aqui calculamos a distância DTW entre as séries "a" e "b" conforme a relação
    // de recorrência passada na especificação do projeto
    for(int i=1; i<=a.getSerieSize(); i++) {
        for(int j=1; j<=b.getSerieSize(); j++) {
            dtw[i][j] = d(a.get(i-1), b.get(j-1)) + Math.min(Math.min(dtw[i-1][j], dtw[i][j-1]), dtw[i-1][j-1]);
        }
    }

    return dtw[a.getSerieSize()][b.getSerieSize()];
}
```

2.3 – Classificação das Séries

Utilizando o classificador do 1-vizinho mais próximo tomamos cada série temporal do do arquivo “teste.txt” e procuramos pela série temporal do arquivo “treino.txt.”mais similar a ela utilizando a distância DTW. Se a classe do arquivo de teste for igual à classe do arquivo de treino, então contabilizamos um acerto.

Segue abaixo a implementação dessa classificação:

```java
public double calculaTaxaAcerto() {
    ArrayList<Double> dtws;
    int menorIndex = 0;
    double menor = 0;
    int hit = 0;
    double taxaAcerto;

    //pegando movimento por movimento do arquivo de testes para comparar com os do arquivo treino
    for(int i=0; i<teste.size(); i++) {
        dtws = new ArrayList<Double>();

        //pegando os movimentos de treino
        for(int j=0; j<treino.size(); j++) {

            //adicionando em um vetor o resultado do dtw realizado entre cada movimento
            dtws.add(dtwDistance(teste.get(i), treino.get(j)));

            //guardando sempre a menor distancia para que possamos comparar sua classe futuramente
            if(j==0) {
                menorIndex = 0;
                menor = dtws.get(0);
                continue;
            }

            if(dtws.get(j) <= menor) {
                menor = dtws.get(j);
                menorIndex = j;
            }
        }

        //verificando se a classe do movimento de treino que mais se assemelha ao movimento testando é a mesma deste
        //contabilizando um acerto caso seja
        if(teste.get(i).getClasse() == treino.get(menorIndex).getClasse())
            hit++;
    }

    taxaAcerto = (double) hit/teste.size();

    return taxaAcerto;
}
```

3 – Análise

A medida final do desempenho é a taxa de acertos, ou seja, compara-se a classe da série A do arquivo de teste com a classe da série B (do arquivo de treino), que é a mais similar a A de acordo com a distância DTW, e contabiliza-se 1 a mais no número de acertos. A taxa de acerto é o número de acertos dividido pelo total de séries temporais testadas.

Nosso algoritmo implementado teve uma taxa de acerto de 84.79%, ou seja, 84,79% dos movimentos foram reconhecidos corretamente, conforme mostrado abaixo:

```
                                    \Projeto2\Código>java DTW
Taxa de Acerto: 0.8479166666666667
Tempo de Execucao: 12928ms
```

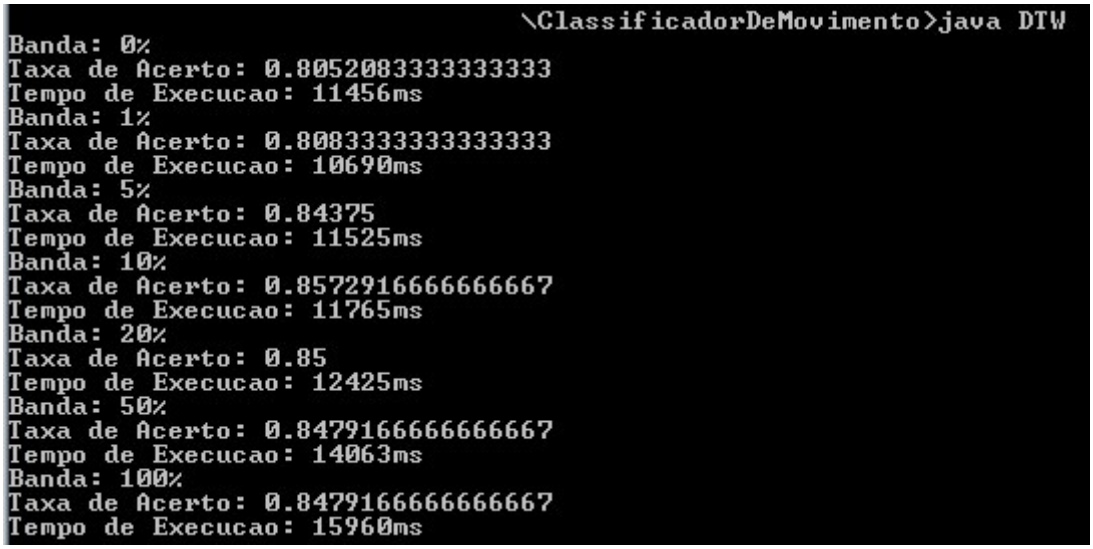
Como não temos conhecimento de qual a taxa de acerto mínimo é necessária para se jogar no console Nintendo Wii de forma satisfatória, não podemos afirmar com precisão se o algoritmo aplicado é bom o suficiente, ou não, para reconhecer os movimentos do controle. Assim, nos limitamos apenas a afirmar que foi obtida uma taxa de acerto de 84,79%.

4 – Extensão 1

Nesta extensão implementamos o algoritmo de distância DTW com adição da banda de restrição Sakoe-Chiba. Para adicionar a banda tivemos de modificar o método "dtwDistance" para ficar da seguinte forma:

```
62  public double dtwDistanceSakoeChiba(Movimento a, Movimento b, double window) {
63      double dtw[][] = new double[a.getSerieSize()+1][b.getSerieSize()+1];
64
65      // Inicialmente, escolhemos o maior entre o tamanho da janela "window" e a diferença entre o tamanho das séries "a" e "b"
66      // Fazemos isso, pois existe uma propriedade que afirma que: a adição da banda só faz sentido se | a - b | <= window
67      window = Math.max(Math.abs(a.getSerieSize()-b.getSerieSize()),window);
68
69      // Diferente do DTW normal, inicializamos todos elementos como Double.MAX_VALUE (equivalente ao INFINITY)
70      for(int i=0; i<=a.getSerieSize(); i++)
71          for(int j=0; j<=b.getSerieSize(); j++)
72              dtw[i][j] = Double.MAX_VALUE;
73
74      dtw[0][0] = 0;
75
76      // Aqui a única modificação é que restringimos os valores da série "b" (percorridos com "j") que serão casados
77      // com os da série "a" (percorridos com "i").
78      // Só pegaremos valores da série "b" que estão dentro da "window" especificada
79      for(int i=1; i<=a.getSerieSize(); i++) {
80          for(int j=(int)Math.max(1.0,i-window); j<=Math.min(b.getSerieSize(),i+window); j++) {
81              dtw[i][j] = d(a.get(i-1), b.get(j-1)) + Math.min(Math.min(dtw[i-1][j], dtw[i][j-1]), dtw[i-1][j-1]);
82          }
83      }
84
85      return dtw[a.getSerieSize()][b.getSerieSize()];
86  }
```

Testando o algoritmo para bandas de 0% (que equivale a distância euclidiana),1%,5%, 10%, 20%, 50% e 100%, conforme pedido no documento de especificação do projeto, obtivemos os seguintes resultados:



4 – Análise da Extensão 1

Dos resultados que obtivemos com os testes acima pudemos concluir o seguinte:

Em relação ao tempo de execução do algoritmo, tivemos que as bandas de 0%, 1%, 5%, 10% e 20% mantiveram seu tempo de execução sempre menor ou bem próximo do tempo de execução do algoritmo sem utilizar a banda. Isso demonstra que o uso da banda pode diminuir levemente o tempo de execução do algoritmo.

Já em relação à taxa de acerto das bandas testadas, percebemos que seu uso apresenta um pequeno ganho de desempenho para as bandas de 10% e 20%. Para todas as outras bandas testadas, a taxa de acerto ou foi igual à do algoritmo sem uso de banda ou inferior a ele.

5 – Extensão 2

Nesta extensão, implementamos o algoritmo de distância DTW, só que desta vez levando em consideração não apenas uma dimensão, mas três: Dos eixos X, Y e Z. No artigo citado na especificação do projeto (“On the Non-Trivial Generalization of Dynamic TimeWarpingtotheMulti-DimensionalCase”) é dito que existem duas fomas básicas de se estender o algoritmo DTW para casos multidimensionais. Para o caso em que cada dimensão

é considerada independente uma da outra, a extensão recebe o nome de DTW_I, enquanto que no caso em que as dimensões não são consideradas independentes a extensão recebe o nome de DTW_D.

Para o caso do trabalho, tivemos fortes indícios durante a leitura do artigo que a extensão a ser utilizada deveria ser a DTW_I, uma vez que é apresentado um problema similar de classificação, no caso classificar o tipo de movimento realizado por um jogador de tênis, em que o autor sugere o uso do DTW_I. Transcrevemos abaixo o trecho que motivou o nosso raciocínio

> "Suppose we have accelerometers on both wrists of a tennis player, and our classification task is to label data into the following shot types {serve|forehand|lob|other}. For many exemplars we might expect DTWI to work best, since the hands are generally loosely coupled in tennis. However, for some classes, such as the backhand, most players use a two-handed grip, temporarily coupling the two accelerometers. This would give us a class-by-class level difference in the suitability of the warping technique."

O autor revela também que no caso de um saque utilizando as duas mãos, talvez o DTW_I não seria o mais adequado. No entanto, como temos apenas um acelerômetro de três eixos e não dois, esse caso não se aplica ao nosso problema. Assim o algoritmo estendido ficou da seguinte forma:

```
69      public double dtwDistance(Movimento3D a, Movimento3D b) {
70
71          double dtwX[][] = new double[a.getSerieXSize()+1][b.getSerieXSize()+1];
72          double dtwY[][] = new double[a.getSerieYSize()+1][b.getSerieYSize()+1];
73          double dtwZ[][] = new double[a.getSerieZSize()+1][b.getSerieZSize()+1];
74          double dtw3DResult;
75
76          // Inicializamos a primeira linha e coluna dos eixos X,Y e Z das séries "a" e "b" como Double.MAX_VALUE (equivalente ao INFINITY)
77          for(int i=0; i<=a.getSerieXSize(); i++){
78              dtwX[i][0] = Double.MAX_VALUE;
79          }
80          for(int i=0; i<=a.getSerieYSize(); i++){
81              dtwY[i][0] = Double.MAX_VALUE;
82          }
83          for(int i=0; i<=a.getSerieZSize(); i++){
84              dtwZ[i][0] = Double.MAX_VALUE;
85          }
86
87          for(int i=0; i<=b.getSerieXSize(); i++){
88              dtwX[0][i] = Double.MAX_VALUE;
89          }
90          for(int i=0; i<=b.getSerieYSize(); i++){
91              dtwY[0][i] = Double.MAX_VALUE;
92          }
93          for(int i=0; i<=b.getSerieZSize(); i++){
94              dtwZ[0][i] = Double.MAX_VALUE;
95          }
96
97          dtwX[0][0] = 0;
98          dtwY[0][0] = 0;
99          dtwZ[0][0] = 0;
100
101         // Aqui calculamos a distância DTW entre as séries "a" e "b" conforme a relação
102         // de recorrência passada na especificação do projeto
103         for(int i=1; i<=a.getSerieXSize(); i++) {
104             for(int j=1; j<=b.getSerieXSize(); j++) {
105                 dtwX[i][j] = d(a.getX(i-1), b.getX(j-1)) + Math.min(Math.min(dtwX[i-1][j], dtwX[i][j-1]), dtwX[i-1][j-1]);
106             }
107         }
108         for(int i=1; i<=a.getSerieYSize(); i++) {
109             for(int j=1; j<=b.getSerieYSize(); j++) {
110                 dtwY[i][j] = d(a.getY(i-1), b.getY(j-1)) + Math.min(Math.min(dtwY[i-1][j], dtwY[i][j-1]), dtwY[i-1][j-1]);
111             }
112         }
113         for(int i=1; i<=a.getSerieZSize(); i++) {
114             for(int j=1; j<=b.getSerieZSize(); j++) {
115                 dtwZ[i][j] = d(a.getZ(i-1), b.getZ(j-1)) + Math.min(Math.min(dtwZ[i-1][j], dtwZ[i][j-1]), dtwZ[i-1][j-1]);
116             }
117         }
118
119         dtw3DResult = dtwX[a.getSerieXSize()][b.getSerieXSize()] + dtwY[a.getSerieYSize()][b.getSerieYSize()] + dtwZ[a.getSerieZSize()][b.getSerieZSize()];
120
121         return dtw3DResult;
122     }
```

5.1 – Análise da Extensão 2

Ao testar o algoritmo, obtivemos o seguinte resultado:

Como podemos observar, levando em conta as três dimensões invés de uma apenas, tivemos uma perda na taxa de acerto na faixa dos 7%. Além disso, observamos também que o tempo de execução do algoritmo praticamente quintuplicou. Para o caso unidimensional tínhamos tempos de execução próximos de 12-15 segundos, enquanto que agora o tempo de execução subiu para próximo de 1 minuto.

Em relação à queda na taxa de acertos, não tivemos uma conclusão clara de o por quê ela ocorreu. No entanto, podemos afirmar que o aumento do tempo de execução se deu uma vez que no DTW_I, agora temos de inicializar os valores de três eixos: X, Y e Z invés de apenas um eixo, e além disso temos que calcular o DTW para cada eixo separadamente, ou seja, calcular o DTW 3 vezes invés de uma, e só depois é que somamos o resultado.