Universidade de São Paulo – USP São Carlos
Instituto de Ciências Matemáticas e de Computação
SCC5900 – Projetos de Algoritmo – Prof. Dr. Marcelo Manzato
Damares Oliveira de Resende – #11022990

Relatório de Atividades – Projeto III

Algoritmos de análise de séries temporais são fundamentais em diversas áreas como a robótica, biometria, medicina, reconhecimento de gestos, sismologia, processamento de voz e etc. Neste contexto, a maioria das aplicações requerem uma análise em tempo real dos dados, logo necessitam um processamento robusto e rápido.

Um dos algoritmos mais consolidados para a classificação de séries temporais é o DTW, do inglês *Dynamic Time Warping*. Ele consiste em analisar duas séries temporais ponto a ponto de forma a calcular a correspondência entre eles. Como resultado, o DTW gera a série temporal de similaridade, e calcula distância entre as séries analisadas.

Neste trabalho foram implementadas três versões deste algoritmo para a análise de séries: 1) DTW (algoritmo comum); 2) DTW Constrained (algoritmo otimizado no tempo; 3) DTW Constrained Plus (algoritmo otimizado no tempo e espaço). A base de dados utilizada foi extraída de acelerômetros e consiste de 240 séries de treino, 960 de teste e podem ser classificadas em 12 diferentes grupos. Vale salientar que as séries analisadas não possuem o mesmo número de amostras necessariamente. Detalhes da implementação e resultados obtidos são descritos a seguir.

O Algoritmo DTW

A implementação básica deste algoritmo faz uso de programação dinâmica que constrói uma matriz de memoização que armazena os valores referentes ao custo entre de cada ponto da série A com um ponto da série B. O cálculo deste custo segue a seguinte relação de recorrência:

$$DTW(A_i, B_j) = (A_i - B_j)^2 + \min(DTW(A_{i-1}, B_{j-1}), DTW(A_i, B_{j-1}), DTW(A_{i-1}, B_j))$$

Equação 1: Equação de recorrência do DTW

A matriz de memoização gerada é preenchida de cima para baixo e da esquerda para a direita. A vizinhança de sua diagonal contém a série de similaridade onde o último valor (M[m][n]) corresponde à distância entre as séries A (representada nas linhas) e B (representada nas colunas). Esta matriz é ilustrada na Figura 1a.

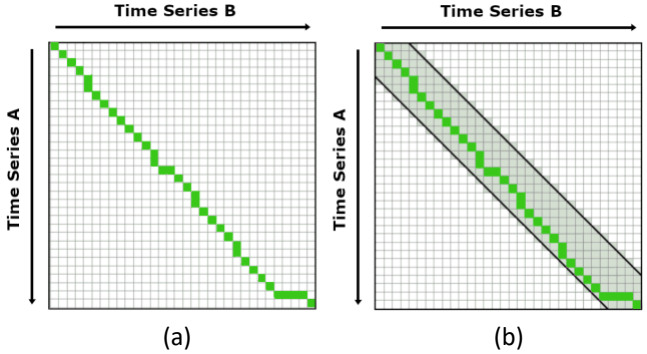

Figura 1: Matriz de memoização gerada pelo algoritmo DTW. (a) DTW Comum. (b) DTW Otimizado.

Nesta versão do algoritmo toda a matriz é preenchida. Sua complexidade de tempo e de espaço é igual a O(m*n), onde m é o tamanho da série temporal A e n o tamanho da série temporal B.

O Algoritmo DTW Constrained

A Figura 1b ilustra a matriz de memoização do algoritmo DTW otimizado. Essa versão do código tem por objetivo minimizar o espaço de busca da matriz de memoização. Como a série de similaridade se dá na diagonal da matriz, uma boa aproximação do problema consiste em estabelecer uma janela de busca de tamanho w ao longo da diagonal de forma que apenas os elementos que se encontram na parte destacada em cinza são computados.

Essa versão do algoritmo possui complexidade de tempo igual a O(m*w), porém ainda tem complexidade de espaço equivalente a O(m*n). Neste trabalho a janela w é definida por *max(5, |m – n|)*.

O Algoritmo DTW Constrained Plus

A implementação desse algoritmo segue a mesma lógica da implementação do algoritmo DTW Constrained, porém busca reduzir a complexidade de espaço. Nesta versão, ao invés de armazenar os valores computados em cada iteração em uma matriz m x n, eles são guardados em um array 2 x m. A primeira linha desta estrutura e os primeiros m-j elementos da segunda linha, armazenam os valores das iterações passadas. A cada nova iteração em i há uma troca (swap) entre as duas linhas.

Desta forma a complexidade de espaço é reduzida para O(m), porém a complexidade de tempo continua sendo O(m *w). A Figura 2 ilustra esta estrutura de memoização.

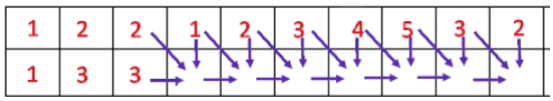

Figura 2: Matriz de memoização do algoritmo DTW Consrained Plus

O Algoritmo 1-NN

Este algoritmo é utilizado para classificar as instâncias de teste. Ele compara cada exemplo de entrada com cada exemplo da base de dados de treino, e determina o rótulo da série a partir do exemplo que possui a menor distância em relação a ela. Essa distância é computada pelo DTW. Para melhorar a performance a consulta à base de dados de treino foi paralelizada.

A complexidade de tempo deste algoritmo quando não paralelizado é O(k), onde k é o número de exemplos de treino. Considerando a complexidade do DTW interno a ele, sua performance final é definida por O(k*m*w). Por outro lado, a complexidade de espaço deste algoritmo é O(k*x), onde x é o número de amostras da maior série temporal na base de dados de treino.

Implementação

O código foi implementado em Python 3 e está dividido em dois módulos: 1) *dataparser.py* é responsável por ler os dados; 2) *tsanalysis.py* contém os algoritmos DTW e de classificação. Ele usa como padrão o algoritmo DTW Constrained. Esse algoritmo foi escolhido por ter a melhor performance. Para rodar o código com o algoritmo padrão basta executar a seguinte linha de comando dentro da pasta **src**: **python tsanalysis.py**. Para escolher outros algoritmos parâmetros adicionais podem ser inseridos. Os parâmetros aceitos são: **-dtw**, **-dtwcp** e **-dtwc**.

Resultados

A Tabela 1 sumariza os resultados de performance dos algoritmos implementados assim como a complexidade de cada um deles. Pode-se observar que o tempo de execução do algoritmo DTW Constrained é menor que o do DTW Comum, mas sua performance em termos de precisão se mantém. Por outro lado, em relação a acurácia o algoritmo DTW Constrained Plus perde bastante. Isso se dá pelo fato de seu espaço de busca ser extremamente limitado. Esse algoritmo é preciso apenas quando as séries temporais possuem aproximadamente a mesma quantidade de amostras.

Os testes foram realizados em um MacOS Pro Mid 2014, cujo processador é um Intel Core i5 da quarta geração (4278U) de 2.6 GHz e memória DDR3L SDRAM de 8 GB.

	Comum	**Const.**	**Const. Plus**
Comp. Tempo	O(m*n)	O(m*w)	O(m*w)
Comp. Espaço	O(m*n)	O(m*n)	O(2*m)
Tempo Total	7 min 25 s	5 min 42 s	4 min 18 s
Acurácia	84.79%	84.47%	57.60%

Tabela 1: Comparação de performance entre os algoritmos implementados

Ademais, é curioso o fato da performance no tempo do DTW Constrained Plus ser melhor que a do DTW Constrained. O primeiro possui apenas otimização de memória em relação ao segundo, então acredita-se que isso é causado pela execução da linha de alocação da memória. A alocação de uma matriz m*n (`memo = [[float('inf') for _ in range(n)] for _ in range(m)]`) é feita 240 * 960 vezes no DTW Constrained, enquanto que o DTW Constrained Plus aloca 240 * 960 vezes uma matriz 2 * m (`memo = [[float('inf') for _ in range(n)] for _ in range(2)]`).

Conclusão

Embora suficiente, a performance dos algoritmos implementados ainda não é a ideal. Como dito anteriormente, o DTW é bastante robusto e é utilizado em inúmeras aplicações, por esse motivo vários estudos de otimização foram realizados para deixa-lo mais rápido. Uma otimização interessante dele é paralelizar a busca preenchendo a matriz de memoização diagonalmente, começando de cima para baixo e da esquerda para a direita. Dessa forma, a complexidade de tempo do algoritmo é definida em O(n). Apesar de mais performática, esta versão do algoritmo não foi implementada pois foi descoberta no mesmo dia da entrega final do trabalho.

Referências

[1] Salvador, S. and Chan, P., 2007. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, *11*(5), pp.561-580.