**Projeto - programação dinâmica**

Pedro Henrique Rampim Natali - 10310655

**1.0) Introdução**

Foi pedido para que fosse implementado primeiramente um DTW(dynamic time warping) e depois com a ajuda dessa função, um trabalho seria feito utilizando além desses conceitos, K vizinhos mais próximos e outros.

**2.0)O código**

O trabalho foi feito em python pelo fato da linguagem ser mais intuitiva e mais “fácil” de se trabalhar. Facilitando muito quando em relação a C, por exemplo, ainda que seu desempenho em termos de velocidade seja menor.

Como o código é demasiadamente grande para um pequeno relatório, acredito que é interessante deixar a explicação do mesmo na implementação, por isso, o código está comentado, e pode ser analisado por meio do arquivo trabalho.py

**2.1) Complicações do Código**

Houveram algumas complicações no código, a primeira delas é de que não consegui resolver o problema cuja resolução poderia bonificar a nota ( de um valor ser igualmente identificado como dois, ou seja, um valor de tipo 1 ser identificado com mesmo número como tipo 1 e tipo 2 )

A segunda complicação foi o fato de que meu trabalho inteiro foi compilado utilizando uma máquina virtual, pelo fato de não ter conseguido interpretar o python no windows. Devido a isso, o tempo de execução apresentou-se como muito alto(rodando todas as posições passaram 30 minutos rodando e não consegui obter resultado, por causa disso resolvi aumentar o passo para que a função executasse mais rapidamente e resultados proporcionais fossem obtidos). Devido a isso, rodei o range das posições de 0 a 960 com passo 10. Assim foi possível encontrar um valor de tempo de execução proporcional ao real.

**2.2) Medição de tempo**

O tempo foi medido usando a função time.time() da biblioteca time. Dessa forma, é possível exemplificar como foi medido um dos tempos na prática, e os outros foram feitos da mesma maneira. Logo, será utilizado aqui o exemplo de como o tempo do DTW foi encontrado :

(função antes)

ini = time.time()

valores\_pos\_dtw = matriz\_de\_distancias(serie11,serie22)

fim = time.time()

a = a + (fim-ini)

(função depois)

(Onde a foi definido como 0 antes do loop)

Esse mesmo a foi printado várias vezes e colocando esses valores no excel, a função

SUM foi usada para encontrar o tempo total. Assim tivemos os valores para o DTW e o mesmo foi feito para o QuickSort e para o conjunto de execução dos K-vizinhos.

E assim, foi possível encontrar a tabela abaixo :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Valor dado ao K | | | | |
| Operação Analisada | 1 | 3 | 5 | 10 | 20 |
| DTW | 185,87(s) | 188,88(s) | 190,45(s) | 194,81(s) | 197,94(s) |
| QuickSort | 0,187(s) | 0,124(s) | 0,144(s) | 0,115(s) | 0,220(s) |
| Todo conjunto de execução do K-vizinhos | 187,894(s) | 189,717(s) | 193,778(s) | 196,546(s) | 199,88(s) |

**2.3)Taxa de acertos**

A taxa foi definida calculando a soma de acertos e erros, e fazendo a divisão de acertos pelo total. Assim, para cada iteração do k (1,3,5,10,20) foi calculada a taxa, resultando na relação abaixo :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Taxa de Acertos | 87% | 82% | 80% | 71% | 65% |

**2.4) Algoritmo de Ordenação**

O algoritmo de ordenação utilizado foi o QuickSort, pelo fato de ser um algoritmo que me senti mais seguro de trabalhar, principalmente pelo fato de seu tempo de execução ser muito pequeno.

Note que o QuickSort de fato foi uma ótima escolha para nossa implementação, visto que o tempo de execução dele nao se compara ao do código.

**2.5) Complexidade**

A partir da análise da complexidade do código, nota-se que a complexidade resultante é de O(), um código relativamente lento em questão de velocidade.

**3.0)Conclusão:**

O Algoritmo não se apresentou como extremamente eficiente, visto que para valores de K grandes, a partir de 20, o algoritmo teve somente 65% de eficiência. Porém tratando-se de um algoritmo relativamente simples se levado em conta Machine Learning ( ainda que seja avançado para um aluno de Algoritmos 1 ) ele teve um desempenho bom. Principalmente para valores de K menores, fato que ficou claro na execução de K = 1, resultando em 87% de eficiência.

Outro porém deve ser feito que o valor real de tempo de execução é bem maior, e provavelmente a eficiência real seja por volta de 85% (1ºcaso). Pois rodei com passo 10 como já explicado anteriormente.

Ainda assim, acredito que o algoritmo de ordenação utilizado foi o correto, a complexidade de O() foi relativamente esperada, visto que várias iterações foram utilizadas no código.