Lista de Exercícios 3

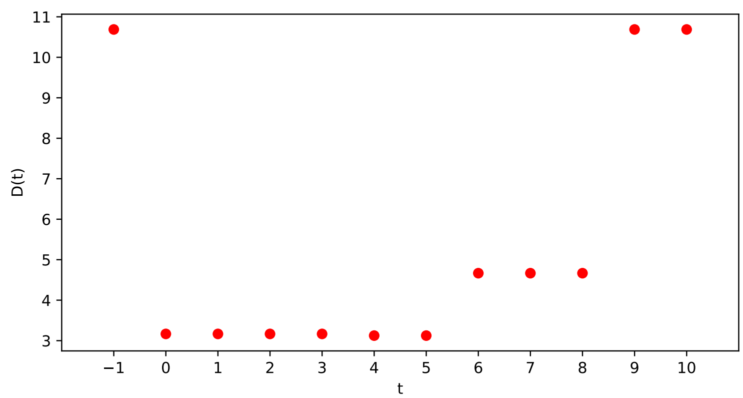
Otimização Natural

Patrícia Kovaleski

5 de abril de 2018

Questão 1) *D.A. para Quantização Escalar com 1 Bit*

(a) Gráfico de $D(t)$ para $t \in [-1, 10]$.



Sendo:

t	$D(t)$
-1	10.688
0	3.167
1	3.167
2	3.167
3	3.167
4	3.125
5	3.125
6	4.667
7	4.667
8	4.667
9	10.688
10	10.688

(b) Matriz não-normalizada p', calculada utilizando $\epsilon^{-d_{xy}/T}$:

$$p' = \begin{bmatrix} 1.2341 \times 10^{-4} & 3.6788 \times 10^{-1} & 1.2341 \times 10^{-4} & 2.3195 \times 10^{-16} \\ 9.5402 \times 10^{-6} & 6.9768 \times 10^{-1} & 1.1592 \times 10^{-3} & 2.4017 \times 10^{-14} \end{bmatrix}$$

Vetor μ, calculado pela soma das colunas da matriz não-normalizada:

$$\mu = \begin{bmatrix} 1.3295 \times 10^{-4} & 1.0655 & 1.2826 \times 10^{-3} & 2.4249 \times 10^{-14} \end{bmatrix}$$

Matriz $p(y|x)$, calculada pela divisão da matriz p' pelo vetor μ :

$$p(y|x) = \begin{bmatrix} 0.9282 & 0.3452 & 0.0962 & 0.0096 \\ 0.0717 & 0.6547 & 0.9038 & 0.9904 \end{bmatrix}$$

(c)

$$D = \sum_{\mathbf{X}} p(x) \sum_{\mathbf{Y}} p(y|x) d(x, y)$$

$$D = 0.25 \times (d_{x_1 y_1} \times p(y_1|x_1) + d_{x_1 y_2} \times p(y_2|x_1) + d_{x_2 y_1} \times p(y_1|x_2) + d_{x_2 y_2} \times p(y_2|x_2) + \\ d_{x_3 y_1} \times p(y_1|x_3) + d_{x_3 y_2} \times p(y_2|x_3) + d_{x_4 y_1} \times p(y_1|x_4) + d_{x_4 y_2} \times p(y_2|x_4))$$

$$D = 0.25 \times (9.0 \times 0.9282 + 11.56 \times 0.0717 + 1.0 \times 0.3452 + 0.36 \times 0.6547 + \\ 9.0 \times 0.0962 + 6.76 \times 0.9038 + 36.0 \times 0.0096 + 31.36 \times 0.9904)$$

$$D = 12.036$$

(d) Os novos centróides são calculados de acordo com:

$$y_i = \frac{\sum_x p(y_i|x) x}{\sum_x p(y_i|x)}$$

Logo,

$$y_1 = 1.4822$$
$$y_2 = 6.4698$$

(e) Repetindo as letras (b), (c) e (d) para T = 0.1

$$p(y|x) = \begin{bmatrix} 1.000 & 1.659 \times 10^{-3} & 1.870 \times 10^{-10} & 7.059 \times 10^{-21} \\ 7.622 \times 10^{-12} & 0.998 & 1.000 & 1.000 \end{bmatrix}$$

$$D = 11.8703$$
$$y_1 = 0.0066$$
$$y_2 = 6.3346$$

(f) Repetindo as letras (b), (c) e (d) para T = 50.0

$$p(y|x) = \begin{bmatrix} 0.5128 & 0.4968 & 0.4888 & 0.4768 \\ 0.4872 & 0.5032 & 0.5112 & 0.5232 \end{bmatrix}$$

$$D = 13.0881$$
$$y_1 = 4.6635$$
$$y_2 = 4.8344$$

(g) Comparando os items (d), (e) e (f), vemos que quanto menor a temperatura, menor o erro encontrado em D. Porém, utilizar de primeira um valor muito baixo de temperatura não costuma nos levar ao mínimo global da função, uma vez que não permite aumentar o valor de D, apenas diminuí-lo. Para T = 0.1, foi encontrado o menor valor para D, com probabilidades condicionais próximas a 1 ou 0 para cada cluster. Isto evidencia um *Hard Clustering* e muito provavelmente o resultado encontrado trata-se de um mínimo local.

Já utilizando um valor muito alto de temperatura, como T = 50, a distribuição de probabilidade tornou-se uniforme entre os clusteres. Com isso, há máxima incerteza sobre a distribuição dos clusteres. Este pode ser visto como uma das etapas do *Soft Clustering*, que, após maximizar a incerteza e encontrar o mínimo nessas condições, diminui o valor da temperatura e recalcula as partições e centróides para a nova condição. Nessa abordagem não necessariamente será encontrado o mínimo global da função, mas permitirá escapar de alguns mínimos locais.

Questão 2) A função escolhida a ser minimizada é a função de Rosenbrock, definida por:

$$f(x) = \sum_{i=0}^{n-1} [100 \times (x_{i+1} - x_i^2)^2 + (1 - x_i)^2],$$

Com ponto mínimo $f(x) = 0$ e $x_i = 1 \forall i \in n$.

O código utilizado para esta questão encontra-se em anexo com o nome *lista3-q2.py*.

A princípio, o vetor **x** inicial foi gerado com números uniformemente distribuídos entre -10 e 10. Porém, o algoritmo estava com muita dificuldade de chegar próximo ao mínimo para os valores: $T0 = 1.0$ e $\epsilon = 0.001$. Avaliando os J_{min} gerados, foi visto que tal configuração não permitia ao algoritmo agir da melhor forma, ficando preso em valores extremamente altos para a função custo. Em seguida, após testar diferentes valores para a temperatura a configuração $T0 = 50.0$ e $\epsilon = 0.05$ levou aos seguintes resultados:

$$\begin{aligned} x_{min} = [&0.484, 0.046, 0.265, -0.122, -0.069, -0.103, 0.034, 0.195, 0.121, -0.027, \\ &-0.063, -0.405, 0.362, 0.288, -0.135, 0.011, 0.11, 0.052, -0.14, 0.471] \end{aligned}$$

$$J_{min} : 90.7560$$

Apesar haver uma minimização considerável para a função custo (antes os resultados eram na ordem de 10^6) ainda é muito distante do valor mínimo desejado: 0.

Após novos testes foi visto que os valores sorteados para o vetor inicial $\mathbf{x}$ influenciam consideravelmente os resultados. Assim, utilizando agora um gerador uniforme entre 0 e 1 para o vetor $\mathbf{x}$ inicial, $T0 = 0.5$, $\epsilon = 0.005$, os seguintes resultados foram encontrados:

$$\begin{aligned} x_{min} = [&0.9861, 1.0130, 1.0034, 0.9983, 0.9997, 0.9989, 0.9817, 0.9859, 0.9618, 0.9603, \\ &0.9730, 0.9292, 0.8212, 0.6657, 0.4336, 0.2406, 0.0809, 0.0003, -0.0021, 0.0330] \end{aligned}$$

$$J_{min} : 5.2614$$

Aumentando o número de iterações de 10^5 para 10^6 e mantendo as configurações anteriores o algoritmo chegou ainda mais perto do mínimo global. Ou seja, o número elevado de variáveis a se minimizar requer um grande número de iterações para se alcançar o mínimo esperado.

$$\begin{aligned} x_{min} = [&0.9915, 0.9975, 0.9878, 0.9851, 0.9983, 1.0113, 1.0086, 1.0073, 0.9998, 0.9899, \\ &0.988, 0.9936, 0.9685, 0.9759, 0.9789, 0.9606, 0.9472, 0.9357, 0.8622, 0.755] \end{aligned}$$

$$J_{min} : 0.7501$$