Problema do Máximo Subarray
Uma Análise de sua Complexidade

Matheus Garay Trindade[1], **Guilherme de Freitas Gaiardo**[1]

[1]Departamento de Eletrônica e Computação – Universidade Federal de Santa Maria (UFSM)
97.105-900 – Santa Maria – RS – Brazil

{mtrindade,ggaiardo}@inf.ufsm.br

Abstract. *This paper analises the classic computing problem of the maximum subarray. Three solutions with different complexities will be presented. This complexities will then be proved. For better understanding of what these complexities actually mean, various simulations were conducted with inputs of different sizes. With so, it is possible to perceive how the algorithm complexity impacts on its performance.*

Resumo. *Este artigo faz uma análise do clássico problema do subarray máximo. Serão apresentadas três soluções com diferentes complexidades. Essas complexidades serão então demonstradas. Para melhor entendimento do que essas complexidades significam, diversas simulações foram feitas com entradas de diferentes tamanhos. Com isso, é possível perceber o quanto a forma de crescimento do algoritmo em função da entrada têm um impacto direto na performance.*

1. Definição matemática

As deduções mostradas a seguir são baseadas em [Winston 2010]. Para esta técnica, devemos escolher o hiperplano que melhor separa duas classes de dados em um espaço n-dimensional. Para ilustrar o ponto de partida, temos a Figura , onde os pontos positivos estão acima do hiperplano e os pontos negativos abaixo.

Para encontrar o hiperplano de fato, é mais conveniente começar por definir o classificador. Para isso, define-se uma função que, dependendo do seu resultado, mostrará se o dado é de uma classe ou de outra, e.g., positivo ou negativo. Assumindo que se conhece hiperplano, pega-se um vetor $\vec{w}$ perpendicular à ele. Temos que a projeção P de um dado $\vec{d}$ sobre esse vetor tem valor dado pela seguinte função:

$$P(\vec{d}) = \vec{d} \cdot \vec{w} = \|\vec{d}\|.\|\vec{w}\|.\cos(\theta) = C.\|\vec{d}\|.\cos(\theta) \tag{1}$$

Onde θ é o ângulo entre $\vec{d}$ e $\vec{w}$. Como a Equação 1 mostra, temos em $\|\vec{d}\|.\cos(\theta)$ o comprimeto projeção de um dado sobre o vetor $\vec{w}$. Caso essa projeção seja maior que uma constante c, fica evidente que o dado está acima do hiperplano, sendo positivo, se for menor, está abaixo, sendo negativo, e se for igual a essa constante, é ponto do hiperplano. Formalmente, temos:

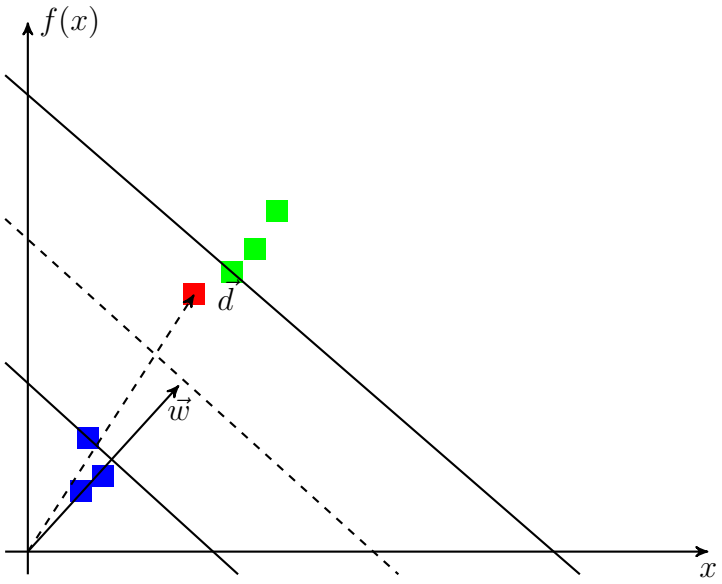

Figura 1. Dados de Exemplo

$$f(n) = \begin{cases} \vec{w}.\vec{d} > c, & \text{se } \vec{d} \text{ é positivo} \\ \vec{w}.\vec{d} < c & \text{se } \vec{d} \text{ é negativo} \\ \vec{w}.\vec{d} = c & \text{se } \vec{d} \text{ está no hiperplano} \end{cases} \tag{2}$$

Ou ainda:

$$f(n) = \begin{cases} \vec{w}.\vec{d} + b > 0, & \text{se } \vec{d} \text{ é positivo} \\ \vec{w}.\vec{d} + b < 0 & \text{se } \vec{d} \text{ é negativo} \\ \vec{w}.\vec{d} + b = 0 & \text{se } \vec{d} \text{ está no hiperplano} \end{cases} \tag{3}$$

A seguir, deve-se definir algumas condições. O classificador terá resultado 1 para vetores de suporte positivo e -1 para vetores de suporte negativo:

$$f(n) = \begin{cases} \vec{w}.\vec{d} + b = 1, & \text{se } \vec{d} \text{ é positivo e é vetore de suporte} \\ \vec{w}.\vec{d} + b = -1 & \text{se } \vec{d} \text{ é negativo e é vetore de suporte} \end{cases} \tag{4}$$

Assim, podemos então maximizar essa função de forma a encontrar o hiperplano que maximiza a distância entre as classes. Para isso, como se tem um função com restrições, para maximizá-la, usa-se multiplicadores de Lagrange. Assim, encontra-se o classificador ótimo. Vale que ressaltar que usa-se uma solução númerica para encontrá-lo, mas como o espaço do problema é convexo, ela sempre converge para o classificador ótimo.

2. SVM Multiclasses

SVM é intrinsicamente uma técnica para classificação binária, entretanto pode facilmente ser convertida para classificar um dado entre mais de duas classes. A seguir, serão apresentadas técnicas usadas para obter essa funcionalidade [Hsu and Lin 2002]. Para as sessões seguintes, os dados a serem usados são mostrados na Figura 2.

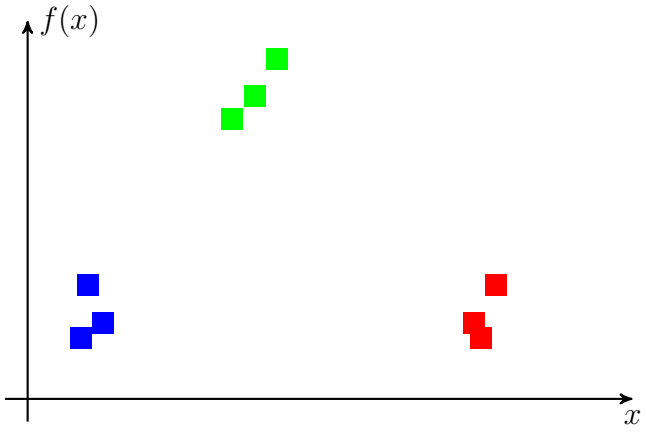

Figura 2. Dados de Exemplo

2.1. One-vs-All

Essa técnica consiste em criar uma SVM para cada classe separando-a das outras. Ou seja, se o problema for constituído de 3 classes, 3 SVMs serão construídas: uma que separa Azul de Vermelho e Verde, uma Vermelho separado de Azul e Verde e outra que separa Verde de Azul e Vermellho. Após isso, entramos com um dado nessas 3 SVMs e avaliamos o resultado em cada.

Idealmente, apenas uma SVM classificará o dado como sendo de apenas uma classe, e essa será adotada como classificação final. Por exemplo, apenas a SVM que separa Verde de Azul e Vermelho classificou como sendo de uma classe, nesse caso Verde, sendo essa a classe final. No caso de confusão, i.e., mais de uma classificando como sendo de uma classe só, pode-se usar como critério de desempate a distância para o hiperplano em cada SVM empatada. Um exemplo de Verde contra Azul e Vermelho é mostrada na Figura 3

2.2. One-vs-One

Para essa técnica, as classes serão combinadas duas as duas. Ou seja, serão geradas a seguinte quantidade de combinações:

$$C * \frac{(C-1)}{2} \tag{5}$$

Onde C é o número de classes do problema. Assim, no caso exemplo, serão geradas 6 partições de dados. Para cada uma dessas partições será gerada uma SVM. Um exemplo é mostrado na Figura 5.

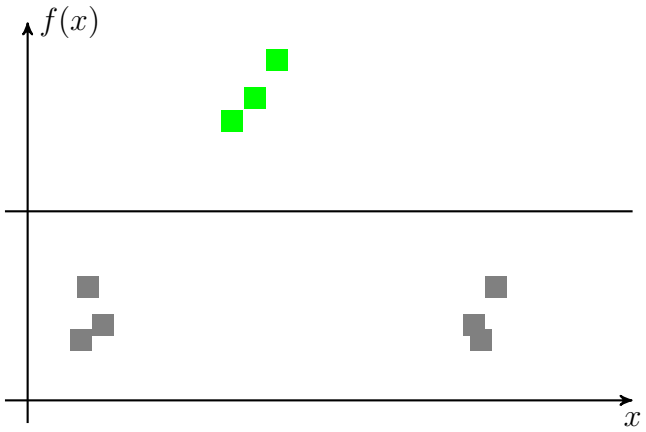

Figura 3. Uma das SVM da técnica One-vs-All

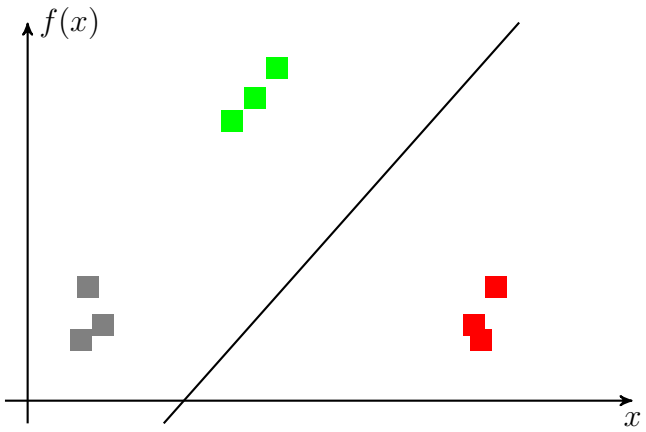

Figura 4. Uma das SVM da técnica One-vs-One (Verde-vs-Vermelho, Azul é descartado)

Após isso, avaliamos o resultado de cada SVM e faz-se uma votação. Suponha uma SVM que separe a classe i da classe j. Se essa SVM classificar como i, soma-se um voto para i, caso contrário soma-se um voto para j. Ao final, avalia-se qual classe recebeu mais votos. A que tiver vencido a votação será o resultado.

Vale ressaltar um ponto, por mais que essa técnica gere mais SVMs do que a técnica One-vs-All, ela ainda pode ser mais rápida de treinar. Isso deve-se ao fato de que cada SVM possui um conjunto de dados de treinamento menor na técnica One-vs-One do que na One-vs-All, o que pode compensar o fato de possuir mais SVMs a serem treinadas em geral

2.3. Error Correcting Output Code

Error Correcting Output Code (ECOC) tenta reduzir o problema de um dos vários classificadres não treinar muito bem e introduzir erro no resultado. Para isso, cada classe recebe um código binário conforme mostrado na Tabela 1.

Classe	f_1	f_2	f_3	f_4	f_5
Vermelho	1	0	1	0	1
Azul	0	0	1	1	1
Verde	1	1	0	0	0

Tabela 1. Códigos para a técnica ECOC

A ideia por trás dessa técnica é, dado um dado de entrada, encontrar o seu código e classificá-lo como sendo a que possui o código mais próximo. Para isso, uma SVM binária será criada para cada coluna. Todos os elementos cuja coluna está marcada como 1 definirão uma classe e todos marcados com 0 serão a outra classe. Classifica-se um dado novo sobre essa SVM e descobre-se se essa coluna é 0 ou 1 para ele. Fazendo isso sobre todas as colunas encontra-se o valor do código do dado, bastando então classificando-o como sendo da classe do código mais próximo.

Note que cada classe possui no minimo dois bits de diferença para todas as outras . Isso garante um certo grau de tolerância a possíveis erros em classificadors de uma coluna. Essa tolerância pode ser melhorada aumentando o número de colunas (até certo ponto) a custo de processamento devido ao número de SVMs extras para calcular essas colunas.

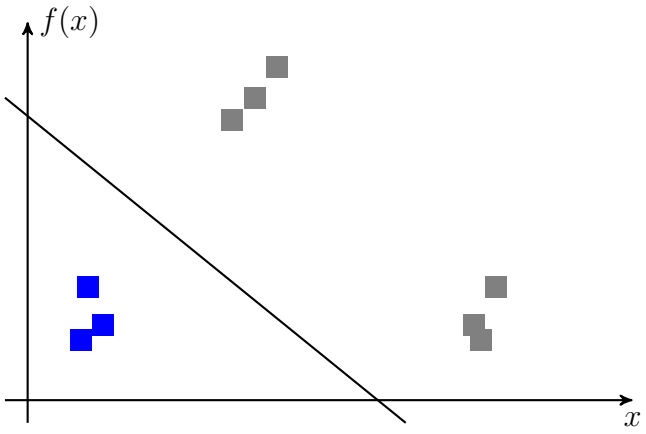

Figura 5. SVM da primeira coluna da Tabela 1

Referências

Hsu, C.-W. and Lin, C.-J. (2002). A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425.

Winston, P. (2010). 6.034 artificial intelligence. (Massachusetts Institute of Technology: MIT OpenCourseWare), http://ocw.mit.edu (Accessado 28 Jun, 2016). License: Creative Commons BY-NC-SA.