SMNE UFPR
Simpósio de Métodos Numéricos em Engenharia
25 a 27 de outubro, 2017

Métodos de Busca Direta para Seleção de Parâmetros em Máquinas de Vetores Suporte

Natalha Cristina da Cruz Machado Benatti
Lucas Garcia Pedroso
Departamento de Matemática
Universidade Federal do Paraná
Curitiba, Brasil
natalha.benatti@ufpr.br, lucaspedroso@ufpr.br

***Resumo*—Neste trabalho apresentamos uma revisão de conceitos acerca das máquinas de vetores suporte, utilizadas para problemas de classificação e regressão no contexto de aprendizagem de máquina. Tendo em vista que escolher adequadamente os parâmetros utilizados nos problemas de Otimização envolvidos pode melhorar o desempenho do classificador, estudamos dois métodos de Otimização sem derivadas, a busca padrão e a busca direta com malha adaptável, e discutimos como estes podem auxiliar nessa tarefa. Para avaliar a eficiência de tais métodos nesse contexto, nós os comparamos com a clássica busca por *grid* na aplicação a trinta e quatro problemas de classificação binária da biblioteca LIBSVM. Constatamos que a busca padrão utilizada e a busca direta com malha adaptável foram capazes de encontrar parâmetros tão bons quanto a busca por *grid*, com baixa taxa de vetores suporte e alta taxa de acerto no conjunto de teste, resultando em horas ou até dias de economia de processamento.**

***Palavras-chave*—Máquinas de vetores suporte, Otimização sem derivadas, parâmetros algorítmicos, busca padrão, método de busca direta com malha adaptável.**

I. Introdução

Otimização é um campo da Matemática de grande interesse por seu potencial de aplicação a problemas práticos. Neste trabalho voltamos nossa atenção aos problemas de Otimização que definem as máquinas de vetores suporte. Estas máquinas são algoritmos no âmbito de reconhecimento de padrões e classificação de dados. Sua eficácia depende da função *kernel* e da seleção dos parâmetros da máquina, sendo eles a constante que regulariza os termos da função objetivo do problema e os parâmetros da função *kernel* utilizada. Estes interferem diretamente nos resultados, sua escolha pode provocar situações indesejadas como o *overfitting* e o *underfitting*, resultando em modelos que não generalizam bem os dados. Com o objetivo de encontrar um método para seleção de modelos competente como a busca por *grid*, que é a mais utilizada na prática mas que demanda de alto custo computacional, estudamos métodos de Otimização sem derivadas.

Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES.

II. Máquinas de Vetores Suporte

As máquinas de vetores suporte para classificação têm o intuito de classificar objetos n-dimensionais $x = (x_1, x_2, \ldots, x_n)^T$ utilizando, para isso, um hiperplano. No caso mais simples, consideramos duas classes, às quais chamamos classe positiva X^+ e negativa X^-, denominado classificação binária. O caso de classificação em mais de duas classes é chamado multiclasses, e este não será abordado neste trabalho. Para encontrar o padrão dos elementos analisados, são utilizadas amostras onde já se conhece a classe a que cada elemento do conjunto de dados pertence, criando um modelo para futuras avaliações.

Consideremos m pontos $(x^{(1)}, y_1), \ldots, (x^{(m)}, y_m) \in X \times \{-1, 1\}$, com $X \subset \mathbb{R}^n$ constituindo o conjunto dos dados sendo a união disjunta dos conjuntos X^+ e X^-, onde $y_i = 1$ quando $x^{(i)} \in X^+$ e $y_i = -1$ quando $x^{(i)} \in X^-$. O objetivo é encontrar um hiperplano ${w^*}^T x + b^* = 0$ que separe os pontos de classes

distintas, definindo o modelo $m(x) = sinal({w^*}^T x + b^*)$. De tal modo, um novo dado x poderá ser classificado como pertencente à classe positiva X^+ se ${w^*}^T x + b^* > 0$, isto é, $m(x) = 1$, ou à classe negativa X^- se ${w^*}^T x + b^* < 0$, isto é, $m(x) = -1$.

A. *Classificação utilizando margens rígidas*

Consideremos um caso particular dos problemas de classificação, onde o conjunto de dados tem a característica de ser linearmente separável, o que acontece quando é possível separar corretamente os pontos de classes diferentes por pelo menos um hiperplano.

O desempenho dos classificadores está associado a uma distância, a qual chamamos de margem, que é a distância do classificador ao ponto $x \in X$ mais próximo.

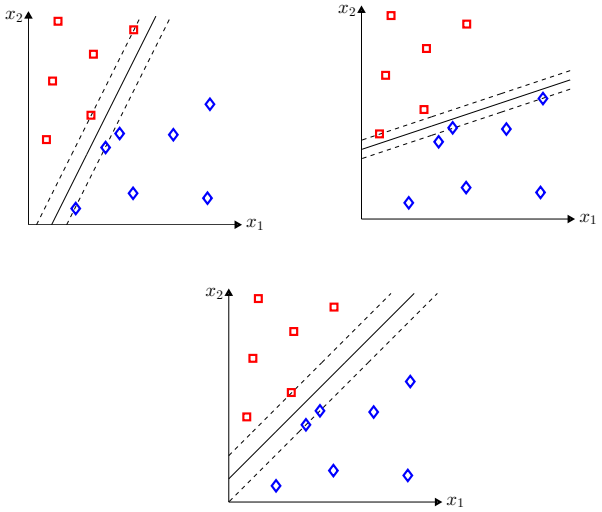

Figura 1: Alguns classificadores com diferentes valores de margens.

Resultados teóricos descritos por Vapnik [7] mostraram que o hiperplano definido com a maior margem implica em uma maior generalização do algoritmo de aprendizagem, isto é, o hiperplano classifica melhor elementos que não estavam previamente no conjunto de dados. Tal hiperplano de margem máxima é chamado hiperplano ótimo.

Impondo que um hiperplano classificador cumpra

$$\begin{cases} w^T x^{(i)} + b \geq 1, & \text{para todo } i \text{ tal que } \ y_i = 1; \\ w^T x^{(i)} + b = 1, & \text{para pelo menos um } x^{(i)} \in X; \\ w^T x^{(i)} + b \leq -1, & \text{para todo } i \text{ tal que } \ y_i = -1; \\ w^T x^{(i)} + b = -1, & \text{para pelo menos um } \ x^{(i)} \in X. \end{cases} \tag{1}$$

e por resultados básicos de Geometria Analítica, a margem é dada por $\rho = \dfrac{1}{\|w\|}$.

Dentre todos os hiperplanos que separam as classes, nosso interesse está voltado ao que proporciona maior margem, ou seja,

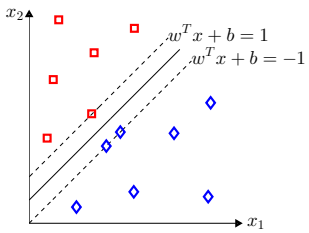

Figura 2: Ilustração do hiperplano ótimo que separa o conjunto linearmente separável.

desejamos o valor máximo de ρ, o que é equivalente a encontrar $\|w\|$ mínimo. Portanto devemos resolver o problema de Otimização quadrático e convexo, ao qual denominamos problema primal[1]

$$\begin{array}{ll} \min\limits_{w \in \mathbb{R}^n,\, b \in \mathbb{R}} & \frac{1}{2}\|w\|^2 \\ \text{s. a} & y_i(w^T x^{(i)} + b) \geq 1 \qquad i = 1, \ldots, m. \end{array} \tag{2}$$

Consideremos a formulação do dual de Wolfe, uma vez que o problema considerado é convexo e continuamente diferenciável. A fim de construirmos o problema dual, consideremos a função Lagrangiana do problema (2), com $\alpha \in \mathbb{R}^m$, com $\alpha_i \geq 0$ para todo $i = 1, \ldots, m$, sendo o multiplicador de Lagrange. Então após algumas manipulações algébricas temos que o problema dual assume a forma

$$\begin{array}{ll} \max\limits_{\alpha \in \mathbb{R}^m} & \sum\limits_{i=1}^{m} \alpha_i - \frac{1}{2} \sum\limits_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j {x^{(i)}}^T x^{(j)} \\ \text{s. a} & \sum\limits_{i=1}^{m} \alpha_i y_i = 0 \qquad\qquad i = 1, \ldots, m \\ & \alpha \geq 0. \end{array} \tag{3}$$

É possível provar que estes problemas têm solução, o que pode ser encontrado em [3]. Consideremos os vetores suporte, que são os dados $x^{(i)}$ correspondentes aos α_i não nulos. Estes desempenham um importante papel por definirem o classificador e sua margem, pois encontramos o valor de w através do cálculo do gradiente da Lagrangiana, obtendo $w = \sum\limits_{i=1}^{m} \alpha_i y_i x^{(i)}$. Assim, após resolvermos o problema dual temos o valor ótimo α^* e então podemos explicitar w^* ótimo. Na expressão para o vetor w^* os α_i^* nulos não têm peso, apenas os pontos $x^{(i)}$ tais que $\alpha_i^* \neq 0$ estão envolvidos, é por essa razão que estes pontos são chamados de vetores suporte.

Em [3] é demonstrado que a solução do problema dual (3) cumpre as condições de KKT associadas ao problema (2). Portanto pela condição de complementaridade de Karush-Kunh-Tucker (KKT),

$$\alpha_i^* \left(y_i \left({w^*}^T x^{(i)} + b^* \right) - 1 \right) = 0, \qquad i = 1, \ldots, m, \tag{4}$$

de onde podemos explicitar b^*. O par (w^*, b^*) encontrado define o

[1] O termo $\frac{1}{2}$ é introduzido para simplificar as expressões do gradiente e da hessiana da função objetivo.

hiperplano ótimo, a partir do qual temos o modelo a seguir, onde I=$\{i : \alpha_i^* > 0\}$ e $|I|$ sua cardinalidade, $m(x)$ é dado por

$$sinal\left(\sum_{i\in I}\alpha_i^* y_i {x^{(i)}}^T x + \frac{1}{|I|}\sum_{i\in I}\left(y_i - \sum_{i\in I}\alpha_i^* y_i {x^{(i)}}^T x^{(i)}\right)\right). \tag{5}$$

Como vemos, o classificador que define este modelo depende apenas de um subconjunto dos dados, os vetores suporte, reduzindo assim o custo computacional de sua avaliação. É possível mostrar que os pontos $x^{(i)} \in X$ que não são vetores suporte poderiam ser retirados da modelagem e ainda assim seria encontrado o mesmo hiperplano ótimo que separa as classes.

B. Classificação no caso não linear

Nosso objetivo agora é encontrar um classificador mesmo que nosso conjunto de dados não seja linearmente separável, e isso será possível se permitirmos que alguns dados do conjunto sejam classificados incorretamente. Para lidar com este caso são introduzidas as variáveis de folga, $\xi_i \geq 0$, $i = 1, \ldots, m$, suavizando as restrições impostas na determinação do hiperplano ótimo. As variáveis de folga são acrescentadas em cada uma das restrições do problema original e penalizadas na função objetivo. Isto é, dada a constante de regularização $C > 0$, o problema de Otimização que determina o hiperplano ótimo de margens flexíveis torna-se

$$\begin{aligned} \min_{w\in\mathbb{R}^n,\, b\in\mathbb{R},\, \xi\in\mathbb{R}^m} \quad & \tfrac{1}{2}\|w\|^2 + C\sum_{i=1}^m \xi_i \\ \text{s. a} \quad & y_i(w^T x^{(i)} + b) \geq 1 - \xi_i \qquad i = 1, \ldots, m \\ & \xi \geq 0, \end{aligned} \tag{6}$$

onde a constante de regularização $C > 0$ pondera a importância que se dá à maximização da margem ou à minimização dos erros. De maneira similar ao algoritmo de margens rígidas, o problema primal também é transformado em seu correspondente dual, dado por

$$\begin{aligned} \max_{\alpha\in\mathbb{R}^m} \quad & -\tfrac{1}{2}\sum_{i,j=1}^m \alpha_i\alpha_j y_i y_j {x^{(i)}}^T x^{(j)} + \sum_{i=1}^m \alpha_i \\ \text{s. a} \quad & \sum_{i=1}^m \alpha_i y_i = 0 \qquad i = 1, \ldots, m \\ & 0 \leq \alpha \leq C. \end{aligned} \tag{7}$$

Mesmo permitindo alguns erros no processo de obtenção do classificador não conseguimos encontrar um hiperplano que classifique bem o exemplo dado na Fig. 3. Notemos que uma quadrática pode separar os dados deste exemplo. Podemos generalizar as máquinas de vetores suporte para o caso não linear utilizando o chamado "truque do *kernel*". Com tal objetivo são introduzidas funções que mapeiam os dados do espaço de origem em um espaço onde o conjunto de amostras seja linearmente separável. A ideia é levar os dados $x^{(i)} \in X \subseteq \mathbb{R}^n$ a $\mathcal{S}$, onde $\dim(\mathcal{S}) > n$ através de uma função $\varphi : \mathbb{R}^n \to \mathcal{S}$, sendo o espaço $\mathcal{S}$ chamado de espaço de

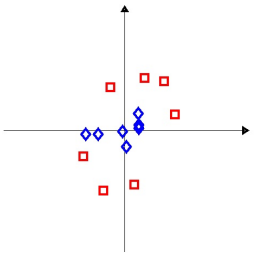

Figura 3: Exemplo de conjunto que não é linearmente separável.

características. O "truque do kernel" é exatamente o processo de mapear os dados nesse espaço de maior dimensão por intermédio da função de mapeamento φ, e construir um hiperplano classificador com margem máxima nesse espaço de maior dimensão, o que gera um modelo não linear no espaço de origem.

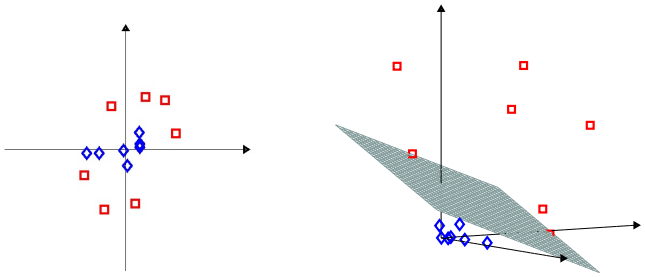

Figura 4: Os dados em $\mathbb{R}^2$ são mapeados para o espaço de características $\mathbb{R}^3$ via função de mapeamento $\varphi(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$.

Definição: Dada uma função $\varphi : X \to \mathcal{S}$, sendo $\mathcal{S}$ um espaço dotado de produto interno, chamamos de *kernel* a aplicação $k(x^{(i)}, x^{(j)}) = \langle\varphi(x^{(i)}), \varphi(x^{(j)})\rangle$.

As Condições de Mercer nos dizem propriedades matemáticas para verificarmos se uma aplicação $k(x^{(i)}, x^{(j)})$ é um produto interno em algum espaço de características $\mathcal{S}$, isto é, estabelece condições para que uma aplicação $k(x^{(i)}, x^{(j)})$ seja uma função *kernel*. Utilizando tal truque nas máquinas de vetores suporte, temos que a formulação dual passa a ser:

$$\begin{aligned} \max_{\alpha\in\mathbb{R}^m} \quad & \sum_{i=1}^m \alpha_i - \tfrac{1}{2}\sum_{i,j=1}^m \alpha_i\alpha_j y_i y_j k(x^{(i)}, x^{(j)}) \\ \text{s. a} \quad & \sum_{i=1}^m \alpha_i y_i = 0 \qquad i = 1, 2, \ldots, m \\ & 0 \leq \alpha \leq C, \end{aligned} \tag{8}$$

A construção do modelo ocorre de forma equivalente ao caso com margens rígidas, entretanto, o vetor normal w^*, antes determinado já não pode mais ser descrito explicitamente, pois nem sempre conhecemos a função φ e o espaço onde esta está levando os dados, basta-nos conhecer a função *kernel* definida por $\varphi(x^{(i)})^T\varphi(x^{(j)})$.

Os dados que tem $\alpha_i = 0$ relacionado não têm relevância, e então,

de forma análoga aos casos anteriores, somente os vetores suporte se fazem importante. Dado um novo dado x modelo neste caso é

$$sinal\left(\sum_{i\in I}\alpha_i y_i k(x^{(i)},x)+\frac{1}{|I|}\sum_{i\in I}\left(y_i-\sum_{j\in I}\alpha_j y_j k(x^{(j)},x^{(i)})\right)\right) \quad (9)$$

sem a dependência da função de mapeamento $\varphi(x)$, sendo necessário somente o conhecimento da função *kernel* $k(x,\overline{x})$.

As funções *kernel* mais utilizadas para a resolução de problemas de máquinas de vetores suporte são destacadas na Tabela I.

Tabela I: Funções *kernel* mais utilizadas no contexto de máquinas de vetores suporte.

Kernel	$\boldsymbol{k(x,\overline{x})}$	**Parâmetros**
Linear	$x^T\overline{x}$	Não há
Polinomial	$((x^T\overline{x})+\eta)^p$	p: grau do polinômio η: constante
Sigmoidal	$\tanh(\gamma(x^T\overline{x})+\eta)$	γ: constante positiva η: constante
Gaussiano	$\exp\left(-\gamma\|x-\overline{x}\|^2\right)$	γ: constante positiva

III. Métodos de Busca Direta

Dentro da vasta área de métodos de Otimização sem derivadas, uma importante classe é a de métodos de busca direta [6], pois nesta classe além de não calcular a derivada ou aproximá-la, também não se faz uso do valor explícito da função nas operações do seu algoritmo, bastando saber para quaisquer dois pontos qual tem o menor valor de função objetivo.

A. Busca Padrão

Nesta classe de métodos, para cada iteração é realizada a avaliação da função objetivo em um conjunto de pontos, definidos por conjuntos de direções com características apropriadas sendo o fato do conjunto de direções gerar positivamente o espaço de busca, o que permite escolher a nova aproximação para o minimizador.

Definição: Dizemos que um conjunto $D=\{d_1,d_2,\ldots,d_k\}\subseteq\mathbb{R}^n$ gera positivamente $\mathbb{R}^n$ se dado qualquer vetor $x\in\mathbb{R}^n$ existem escalares $t_1,t_2,\ldots,t_k\in\mathbb{R}$ não negativos tal que

$$x=\sum_{i=1}^{k}t_k d_k.$$

Para sintetizar a ideia da busca vamos considerar um iterando atual $x_k\in\mathbb{R}^n$ e um valor $t_k\in\mathbb{R}_+$ denominado tamanho de passo ou de malha. O objetivo da iteração k é determinar um novo ponto $x_{k+1}=x_k+t_k d$ tal que $f(x_{k+1})<f(x_k)$.

1) Busca Coordenada: A busca coordenada é a busca padrão mais intuitiva, pois usa a base positiva maximal

$$D_\oplus=\{e_1,\ e_2,\ \ldots,\ e_n,\ -e_1,\ -e_2,\ \ldots\ ,-e_n\},$$

onde e_i é o i-ésimo vetor da base canônica de $\mathbb{R}^n$. Então dado $x_k\in\mathbb{R}^n$ iterando corrente e $t_k>0$ tamanho do passo corrente, a busca coordenada avalia a função nos pontos do conjunto

$$P_k=\{x_k+t_k d : d\in D_\oplus\} \quad (10)$$

seguindo uma ordem antes determinada. Os conjuntos $D_\oplus$ e P_k são conhecidos como conjunto de direções e conjunto dos pontos de votação (*poll points*), respectivamente.

Vemos na Fig. 5 uma sequência de tamanhos de passo cumprindo $t_0=t_1=t_2$, posteriormente uma atualização do tamanho de passo na iteração $k=3$ com $\beta=\frac{1}{2}$. Temos $t_3=\frac{t_0}{2}=t_4=t_5$.

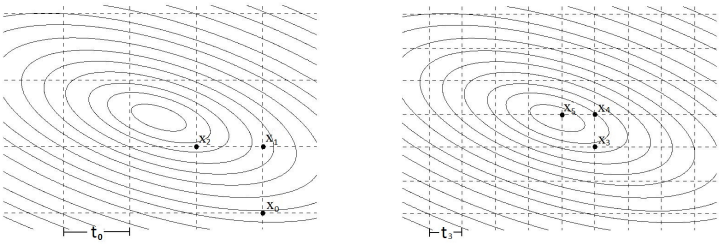

Figura 5: Ilustração de 5 primeiros passos da busca coordenada. Fonte: Diniz-Ehrhardt, Lopes e Pedroso, 2012, p.40.

Quando se encontra um ponto em P_k que decresça o valor da função, a iteração k é declarada como bem-sucedida, e quando isso acontece se atualiza o novo iterando $x_{k+1}=x_k+t_k d$ e se mantém ou se aumenta o tamanho do passo t_k, geralmente fazendo-se $t_{k+1}=t_k$ ou $t_{k+1}=2t_k$. Já quando não existe nenhum ponto em P_k que decresça o valor da função objetivo, a iteração k é considerada mal-sucedida, e se faz $x_{k+1}=x_k$ e se reduz o valor de t_k, no geral é feito $t_{k+1}=\frac{1}{2}t_k$. A abordagem de busca padrão, descrita a seguir, generaliza a busca coordenada, permitindo o uso de um conjunto mais amplo de direções de busca em cada iteração.

2) Busca Padrão Generalizada: Os algoritmos de busca padrão generalizada (GPS, do inglês *Generalized Pattern Search*) têm por objetivo resolver o problema de minimização irrestrito

$$\min_{x\in\mathbb{R}^n}\ f(x),$$

onde $f:\mathbb{R}^n\to\mathbb{R}$. Para tal, consideremos então, como na subseção anterior, um iterando corrente $x_k\in\mathbb{R}^n$, um tamanho de passo corrente, ou tamanho de malha, $t_k>0$. Consideramos ainda um conjunto $\mathcal{D}$ de conjuntos D_k tais que $D_k=\{d_1,d_2,\ldots,d_{r(k)}\}$[2] é um conjunto de $r(k)$ direções, com $r(k)\geq n+1$, que geram positivamente o $\mathbb{R}^n$.

O objetivo da busca na iteração k é encontrar um novo ponto

[2]Por abuso de linguagem vamos considerar, quando conveniente, o conjunto D_k como uma matriz $n\times|D_k|$ cujas colunas são d_i, para cada $i=1,2,\ldots,r$.

x_{k+1} tal que $f(x_{k+1}) < f(x_k)$. O processo consiste em duas etapas, o passo de busca (*search step*) e o passo de votação (*poll step*).

O passo de busca é opcional e fundamenta-se em avaliar a função objetivo em um número finito de pontos escolhidos arbitrariamente, e se move para algum deles somente se houver decréscimo da função objetivo nesse ponto. Pode ser através de um algoritmo heurístico, por exemplo, com o objetivo de melhorar a eficácia do método. O passo de votação é executado quando o passo de busca tiver sido mal-sucedido ou não tiver sido computado, e se baseia numa procura local em torno do iterando corrente, explorando os pontos do conjunto de votação P_k definido como

$$P_k = \{x_k + t_k d : d \in D_k\}, \quad \text{onde} \quad D_k \in \mathcal{D}. \tag{11}$$

Consideremos ainda na iteração k a malha M_k, sendo esta centrada em torno do iterando corrente x_k, e sua espessura definida através do tamanho de malha t_k, dada da seguinte forma

$$M_k = \left\{ x_k + t_k D_k u : u \in \mathbb{Z}_+^{|D_k|} \right\}, \tag{12}$$

onde $\mathbb{Z}_+^{|D_k|}$ é o conjunto dos vetores com componentes inteiras não negativas cuja dimensão é a cardinalidade de D_k. O ingrediente chave na definição da malha são as combinações lineares não negativas dos elementos de um conjunto D_k que gera positivamente $\mathbb{R}^n$.

Algoritmo 1 - Busca Padrão Generalizada
Passo 0 *Inicialização*:
Sejam $k = 0$, $x_0 \in \mathbb{R}^n$, $t_0 > 0$, $0 < \beta_1 \leq \beta_2 < 1$, $\lambda \geq 1$ e $\mathcal{D}$.
Passo 1 *Passo de Busca*:
Tente encontrar $x \in \mathbb{R}^n$ tal que $f(x) < f(x_k)$. Se for bem sucedido faça $x_{k+1} = x$, declare a iteração bem-sucedida e vá para Passo 3.
Passo 2 *Passo de Votação*:
Tome $D_k \in \mathcal{D}$. Se encontrar $\overline{x} \in P_k$ tal que $f(\overline{x}) < f(x_k)$, então faça $x_{k+1} = \overline{x}$ e declare a iteração k como bem-sucedida. Caso contrário faça $x_{k+1} = x_k$ e declare a iteração mal-sucedida.
Passo 3 *Atualização do Parâmetro da Malha*:
Se a iteração foi bem-sucedida faça $t_{k+1} \in [t_k, \lambda t_k]$. Caso contrário faça $t_{k+1} \in [\beta_1 t_k, \beta_2 t_k]$.
Passo 4 *Nova Iteração*:
Faça $k \leftarrow k + 1$ e volte ao Passo 1.

A busca coordenada é um caso particular do GPS, sendo qualquer algoritmo de busca padrão que use apenas as direções e_i e $-e_i$ para $i = 1, \ldots, n$ como conjunto de direções de busca.

3) Convergência Global da Busca Padrão Generalizada: O estudo de convergência a pontos estacionários de primeira ordem dos métodos de busca padrão generalizada é abordado em [**?**], [6], [10]. Tal estudo é realizado sob algumas hipóteses, que podem ser simplificadas como se segue.

Hipótese 1: O conjunto de nível $L(x_0) = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$ é compacto.

Hipótese 2: O conjunto $\mathcal{D}$ de bases positivas em $\mathbb{R}^n$ é finito.

Hipótese 3: Decréscimo simples com treliças inteiras.

Teorema [6, Teorema 7.4]:
Considere válidas as Hipóteses 1, 2 e 3. Então a sequência de iterações (x_k) tem um ponto limite x^*, para o qual $\nabla f(x^*) = 0$.

A prova deste teorema pode ser generalizada para funções não suaves, e também para o caso em que o conjunto de bases positivas $\mathcal{D}$ é qualquer, mas para o último caso deve-se considerar hipóteses mais fortes que o decréscimo simples.

B. Método de Busca Direta com Malha Adaptável

Apesar da grande aplicabilidade dos algoritmos de busca padrão, a utilização de um número finito de direções de busca implica que seu desempenho pode não ser satisfatório no caso em que a função objetivo do problema é não suave. Isto ocorre pois se a derivada da função é descontínua em algum ponto, não há garantias de que existe ao menos uma direção de descida mesmo que o conjunto de direções de busca gerem positivamente o $\mathbb{R}^n$. Audet e Dennis introduziram em [2] o método chamado de busca direta com malha adaptável (MADS, do inglês, *mesh adaptive direct search method*).

O algoritmo MADS para Otimização não linear é uma extensão para a classe de algoritmos de busca padrão generalizada. Tal método utiliza do decréscimo simples com treliças inteiras para garantir sua convergência global, mesmo no caso não suave. Além disso, o MADS é aplicável tanto a problemas restritos quanto irrestritos, utilizando a técnica de barreira extrema, isto é, se o conjunto viável é dado por $\Omega \subset \mathbb{R}^n$, então definimos $f(x) = +\infty$ caso $x \notin \Omega$.

O método é iterativo e também dividido em duas fases principais, sendo elas o passo de busca e o passo de votação. O passo de busca é exatamente como utilizado no GPS, mas no passo de votação o MADS introduz um parâmetro de votação (*poll size parameter*) $\Delta_k \in \mathbb{R}_+$ para a iteração k. Este parâmetro define a magnitude da distância entre os pontos gerados pelo passo de votação e o iterando atual x_k. O GPS é um caso particular do MADS, no sentido que se tomarmos $t_k = \Delta_k$ recuperamos o algoritmo GPS.

Consideremos a malha M_k dada em (12) onde cada base positiva $D_k \in \mathcal{D}$ cumpre a hipótese de estar em treliças inteiras, abordado em [6], [10]. Além disso, o passo de busca, que é opcional, é feito sobre a malha M_k. Caso o passo de busca falhe ou não seja realizado, o passo de votação é realizado sobre o conjunto P_k, conhecido por estrutura (*frame*) definida da mesma maneira dada em (11), onde cada $d \in D_k$ cumpre três condições. A primeira é que d é uma combinação inteira não negativa das colunas de D. Além disso, a distância entre x_k e o ponto $x_k + t_k d_k$ tende para zero se, e somente se, t_k satisfaz

$$\lim_{k \in K} \ t_k \|d_k\| = 0 \ \text{ equivale a } \lim_{k \in K} \ t_k = 0$$

para qualquer subsequência infinita K. Finalmente, a terceira condição é que o limite de todas subsequências convergentes de

$$\overline{D}_k = \left\{ \frac{d_k}{\|d_k\|} : d_k \in D_k \right\} \tag{13}$$

seja uma base positiva.

Analogamente ao que ocorre na busca padrão, o tamanho de malha pode aumentar se um ponto melhor for encontrado, e deve diminuir caso contrário.

Algoritmo 2 - MADS
Passo 0 *Inicialização*:
Sejam $k = 0$, $x_0 \in \mathbb{R}^n$, $0 < t_0 \leq \Delta_0$ onde $t_0, \Delta_0 \in \mathbb{R}$ e $\mathcal{D}$.
Passo 1 *Passo de Busca*:
Tente encontrar $x \in \mathbb{R}^n$ tal que $f(x) < f(x_k)$. Se for bem sucedido faça $x_{k+1} = x$, declare a iteração bem-sucedida e vá para Passo 3.
Passo 2 *Passo de Votação*:
Tome $D_k \in \mathcal{D}$. Se encontrar $\overline{x} \in P_k$ onde $t_k \leq \Delta_k$ tal que $f(\overline{x}) < f(x_k)$, então faça $x_{k+1} = \overline{x}$ e declare a iteração k bem-sucedida. Caso contrário faça $x_{k+1} = x_k$ e declare a iteração k mal-sucedida.
Passo 3 *Atualização dos Parâmetros*:
Defina o parâmetro t_k para se situar em treliças inteiras.
Atualize $\Delta_{k+1} \geq t_{k+1}$.
Passo 4 *Nova Iteração*:
Faça $k \leftarrow k + 1$ e volte ao Passo 1.

A Fig. 6 mostra exemplos de estruturas em $\mathbb{R}^2$. Os pontos p^1, p^2 e p^3 são possíveis escolhas para formarem a estrutura, através das direções $p^i - x_k$ para $i = 1, 2, 3$, na figura a malha M_k é representada pela interseção de todas as linhas.

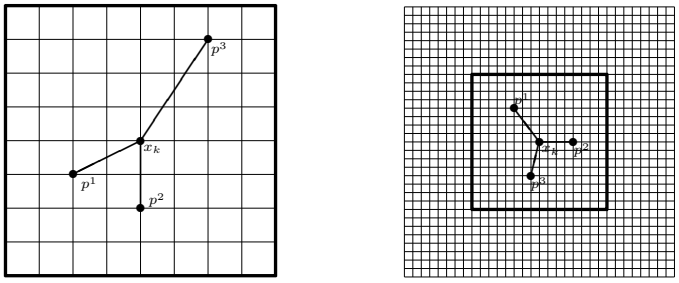

Figura 6: Exemplo de estruturas $P_k = \{x_k + t_k d : d \in D_k\} = \{p^1, p^2, p^3\}$ para diferentes valores de t_k e Δ_k.
Fonte: Audet e Dennis, 2006, pp. 195.

A análise de convergência do MADS pode ser encontrada em [2], e podemos resumi-la a dois pontos principais. O primeiro deles é que existem subsequências para as quais a malha fica cada vez mais refinada, isto é $t_k \to 0$ quando $k \to \infty$, e o segundo é que os pontos de acumulação da sequência $\{x_k\}$ são estacionários, com respeito a Clarke, obtendo $f^o(x^*; v) \geq 0$.

IV. Aplicações

Apresentamos resultados numéricos realizados para seleção de parâmetros para as máquinas de vetores suporte para classificação. Utilizamos os métodos de busca padrão e MADS e os comparamos com a busca por *grid*. Discutimos os resultados das comparações entre os métodos estudados na resolução de problemas da *Library for Support Vector Machines* (LIBSVM) [5] mediante gráficos de perfil de desempenho.

A. Influência da Seleção de Parâmetros

Na prática, se usássemos todo o conjunto de dados X para construir o modelo não haveria possibilidade de testá-lo, isto é, investigar a taxa de acerto deste modelo quando apresentados novos dados. Por isso, é comum particionarmos o conjunto X em dois subconjuntos disjuntos cuja união seja X, denominados conjunto de treinamento e conjunto de teste. O conjunto de treinamento é utilizado para obtenção do classificador, enquanto que o conjunto de teste é utilizado para conhecer sua taxa de acerto.

Na construção do classificador, é interessante evitar duas situações, a saber, *underfitting* e *overffiting*. A primeira delas acontece quando se determina uma função muito simples, cuja flexibilidade é insuficiente para capturar informações importantes dos padrões de entrada, desta forma, o modelo se ajusta muito pouco aos dados de treinamento, não os representando adequadamente, e como consequência não consegue generalizar para novos dados. A segunda situação, o *overffiting*, acontece quando o classificador se ajusta excessivamente aos dados de treinamento, obtendo baixo desempenho quando aplicados dados diferentes, onde considera-se que o algoritmo se adaptou inclusive aos ruídos dos dados de treinamento. Essas duas situações são as maiores causas para o mau desempenho dos algoritmos de aprendizagem de máquina, pois em ambos os casos o classificador não consegue prever corretamente as saídas dos dados de teste, perdendo assim sua capacidade de generalização.

Desta forma, a construção de um classificador e a sua capacidade de generalização dependem significativamente da escolha do parâmetro C, e dos parâmetros da função *kernel* utilizada. A seleção inadequada desses parâmetros pode levar à ocorrência de *overfitting*, onde a taxa de acertos na fase de treinamento é alta e na etapa de teste é baixa, ou de *underfitting*, onde ambas taxas de acerto do treinamento e teste são baixas. Vale salientar que estes parâmetros influenciam de maneira diferente para cada problema considerado, isto é, não há possibilidade de apresentar um conjunto de parâmetros ótimo que se adapte a qualquer problema.

Segundo [4], o *underfitting* muitas vezes não é discutido, pois é fácil de detectá-lo, e na tentativa de remediá-lo a ideia é considerar funções *kernel* diferentes ou algoritmos alternativos de aprendizagem de máquinas. Já a ocorrência de *overfitting* é perceptível por impactar diretamente na quantidade de vetores suporte do problema. Desta forma, o número de vetores suporte fornece informações importantes quanto à qualidade do modelo obtido, pois altas taxas de vetores suporte indicam *overfitting*, enquanto que uma baixa taxa reflete numa melhor generalização dos dados.

Concluímos então que devemos nos preocupar em tomar o parâmetro C e os parâmetros da função *kernel* utilizada de forma a obter uma taxa de acerto alta no conjunto de teste e uma quantidade de vetores suporte baixa. Com tal objetivo procuramos funções que

dependem da taxa de acerto e da taxa de vetores suporte que pudessem nos dar um com equilíbrio entre essas duas medidas de desempenho.

B. Banco de Dados, Softwares e Calibração de Parâmetros

Todos os bancos de dados dos problemas que testamos foram retirados da biblioteca LIBSVM, sendo esta uma biblioteca para máquinas de vetores suporte que está em desenvolvimento desde 2000, por Chang e Lin [5]. Além disso todos os cálculos relativos à classificação, no processo de treinamento e teste, foram executados por meio de pacotes computacionais disponibilizados pela biblioteca. Para alguns bancos de dados a biblioteca disponibiliza separadamente os conjuntos de treinamento e teste, e podemos observar que a proporção de dados no conjunto de treinamento varia de 5% a 96% do conjunto total de dados, dependendo do problema. Para os bancos de dados com conjuntos não disponibilizados separadamente realizamos uma escolha aleatória de cerca de 80% dos dados para o conjunto de treinamento e 20% para o conjunto de teste.

Dentre as diversas técnicas existentes para seleção de parâmetros, a busca por *grid* é a mais utilizada por ser fácil de usar e compreender, e ainda pelo fato desta fornecer bons resultados. Contudo, essa técnica é muito cara computacionalmente, pois ela avalia todos os pontos de uma malha, isto é, todas as combinações de parâmetros. Em virtude da busca por *grid* ser uma busca exaustiva, o tamanho do seu espaço de busca impacta diretamente no tempo computacional necessário para encontrar bons parâmetros, e esta é a principal motivação para trabalharmos com outros modelos para seleção de parâmetros para máquinas de vetores suporte. Observemos ainda que as dimensões dos conjuntos de treinamento e teste, determinadas pelo número de dados de amostra e pelo número de características de cada amostra também influenciam no custo computacional.

Utilizamos 34 problemas de classificação binária da biblioteca LIBSVM, empregamos os mesmos conjuntos de treinamento e teste para cada metodologia de busca. Vale ressaltar que na grande maioria dos casos estes são problemas do mundo real, isto é, dados de bancos reais. Por exemplo, o conjunto de dados referentes ao problema *breast-cancer* tem informações acerca do câncer de mama e foi obtido através do Instituto de Oncologia da *University Medical Center*, em Jugoslávia.

Os algoritmos foram implementados no software Matlab versão 8.3 (R2014a). Para a implementação do MADS utilizamos uma versão para o Matlab do *software* livre NOMAD [1].

Utilizamos o *kernel* Gaussiano em todos os problemas tomados, pois este é o mais utilizado na prática. Logo nosso objetivo é encontrar o melhor par de parâmetros (C, γ) para cada problema, tendo como objetivo maximizar a taxa de acerto e minimizar a taxa de vetores suporte.

Inicialmente rodamos os 34 problemas com os parâmetros $(C, \gamma) = (1, 1)$ para analisarmos o tempo médio desprendido para rodar cada problema, então dividimos os problemas em três subconjuntos, o primeiro deles é o subconjunto dos problemas mais rápidos, que utilizavam menos de 1 segundo para resolver o problema de Otimização, representando 14 problemas, o segundo deles o subconjunto de problemas que demandavam entre 1 e 30 segundos para ser resolvido, compondo 10 problemas, e o terceiro é dos problemas que demandavam mais de 30 segundos, sendo eles na maioria demorando cerca de 8 minutos para resolver o problema com (C, γ) definidos, constituindo os 10 problemas restantes. Isso foi feito para que pudéssemos realizar os primeiros testes para calibração de parâmetros entre a acurácia e a taxa de vetores suporte.

Vamos denotar a taxa de acerto, ou acurácia, por ac e a taxa de vetores suporte por vs. Sabemos que $0 \leq ac, vs \leq 100$. Utilizando o primeiro subconjunto de problemas, os mais rápidos, propusemos e testamos 4 diferentes funções de mérito, tais funções a serem minimizadas dependem de ac e vs, apresentadas a seguir.

$$\Psi_1(ac, vs) = -ac + \vartheta_1 \max\{0, vs - \omega_1\}; \tag{14}$$

$$\Psi_2(ac, vs) = -ac + \vartheta_1 (\max\{0, vs - \omega_1\})^2; \tag{15}$$

$$\Psi_3(ac, vs) = -ac - \vartheta_1 \frac{1}{1 + vs}; \tag{16}$$

$$\Psi_4(ac, vs) = -ac + \vartheta_1 \max\{0, vs - \omega_1\} + \vartheta_2 \max\{0, vs - \omega_2\}. \tag{17}$$

Onde $\vartheta_i \in \mathbb{R}_+$ e $\omega_i \in \mathbb{Z} \cap [0, 100]$ para $i = 1, 2$. Observemos que o termo ϑ_i faz uma penalização na taxa de vetores suporte quando esta taxa é maior que ω_i. A intenção aqui foi encontrar uma função que dependa da acurácia mas que não deixe de levar em consideração a taxa de vetores suporte. Notemos que se estivéssemos apenas maximizando a acurácia, poderíamos usar $\Psi_0 = -ac$. Porém, estaríamos sujeitos a encontrar classificadores com *overfitting*, conforme já explicado. Tivemos uma etapa de calibragem dos parâmetros das funções e testes sobre qual seria a melhor função para a realização dos testes práticos.

A função Ψ_1 com $\vartheta_1 = 10$ e $\omega_1 = 0$ apresentou as menores taxas de vetores suporte e ainda acurácias parecidas com as obtidas minimizando a função Ψ_0 que considera somente a acurácia, então concluímos que a função Ψ_1 nos dava o maior balanceamento dentre elas, escolhida assim para realizar os testes na íntegra.

C. Resultados Numéricos

A busca dos parâmetros (C, γ) para as máquinas de vetores suporte no método de busca por *grid* foi realizada numa malha de tamanho 41×41, com as coordenadas de $C, \gamma \in \{2^{-10}, 2^{-9.5}, 2^{-9}, 2^{-8.5}, \ldots, 2^{9.5}, 2^{10}\}$. Definimos esta malha baseado no que foi sugerido no trabalho de Hsu, Chang e Lin [9]. Além disso, para a busca coordenada e MADS, utilizamos como ponto inicial $(C, \gamma) = (2^0, 2^0)$, tamanho de passo inicial $t_0 = 2^{10}$ e critério de parada $t_k < t_{tol} = 2^{0.5}$, proporcionando uma busca sobre a malha utilizada no *grid* mesmo que estas possam eventualmente sair do espaço delimitado pela malha.

Realizamos a comparação entre os métodos mediante gráfico de perfil de desempenho proposto por Dolan e Moré [8]. Para tal, utilizamos o número de avaliações de função como medida de desempenho estando altamente relacionada com o tempo de execução. Por uma avaliação de função entendemos que o método resolveu um

problema de Otimização de máquinas de vetores suporte conforme variação dos valores de C e γ e o testou para obter a taxa de acerto da máquina.

Fixando um problema, ao utilizarmos os três métodos para encontrar o melhor par de parâmetros e obtermos as taxas de acerto e de vetores suporte relativos a cada método, tomamos a maior taxa de acerto e menor taxa de vetores suporte e denotamos ac^* e vs^*, respectivamente, então declaramos o fracasso de um método quando sua taxa de acerto é menor que $0.95ac^*$ ou taxa de vetores suporte maior que $1.20vs^*$.

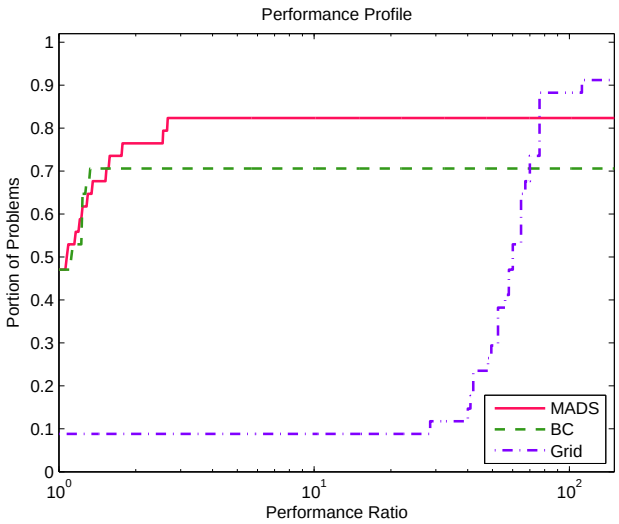

Figura 7: Perfil de desempenho.

Em 47.1% dos problemas os métodos de seleção de modelos MADS e Busca Coordenada foram os mais eficientes, enquanto a busca por *grid* foi mais eficiente em 8.8% dos problemas. Já em termos de robustez, a busca por *grid* levou 112.1 vezes o melhor trabalho para atingir 91.2% de robustez, seguido pelo MADS que levou 2.7 vezes o melhor trabalho para atingir 82.4% de robustez. O menos robusto foi o método de busca coordenada, que levou 1.3 vezes o melhor trabalho para resolver 70.6% dos problemas.

A busca por *grid* faz uma busca em toda a malha, o que torna a busca muito competitiva quando falamos de robustez, porém obtivemos 82.4% de robustez com o MADS, isto é, dos 34 problemas apenas 6 obtiveram uma acurácia com diferença maior que 5% da acurácia obtida com a busca por *grid* ou taxa de vetores suporte 20% menor que a encontrada por alguma das outras duas buscas. Como exemplo de fracasso do MADS e da busca coordenada citamos o problema *madelon*, porém para tal problema facilmente vemos que o *kernel* Gaussiano não é o mais recomendado, pois todas as buscas encontraram taxa de 100% de vetores suporte, o que nos indicam o fenômeno *overfitting*.

Considerando toda análise feita durante esta seção, concluímos que o método MADS obteve bons resultados em um contexto geral, sendo portanto indicado na busca de parâmetros ótimos para máquinas de vetores suporte para classificação. Esta avaliação não desconsidera a hipótese da recomendação da busca coordenada como método de seleção de parâmetros, pois além de ser uma busca de fácil implementação esta também obteve resultados satisfatórios. Além do fato que o método MADS apresentou em média 97.9% de redução de avaliações de função em relação ao *grid*, e a busca coordenada reduziu 98.1% o número de avaliações de função em relação ao *grid*.

V. Conclusão

As máquinas de vetores suporte têm grande aplicabilidade em casos reais, sendo utilizadas em áreas como na Medicina, Informática, Economia entre inúmeras outras. Levando em consideração os aspectos apresentados no texto, sabemos que para resolvermos o problema de Otimização definido pelas máquinas de vetores suporte precisamos definir previamente a constante C, que tem o papel de fazer uma regularização nos dados, a função *kernel* a ser utilizada e seus respectivos parâmetros. Ao obter uma solução α^* podemos construir o hiperplano ótimo, obtendo assim um modelo para classificação de novos dados. Porém a boa performance desse modelo depende de uma adequada seleção dos parâmetros mencionados.

Quando comparamos os resultados obtidos com o MADS e a busca coordenada concluímos que o MADS mantém o baixo custo computacional e a baixa taxa de vetores suporte da busca coordenada, porém o supera relativamente a taxas de acerto nos conjuntos de teste, já em relação a maior simplicidade do algoritmo e facilidade de implementação a busca coordenada se sobressai.

As metodologias que propusemos podem ser bastante úteis em situações onde o pesquisador tem um conjunto de dados de alta dimensão, o que é normal em bases de dados reais, sendo obrigado a empregar um método de tentativa e erro ou a exaustiva busca por *grid* para conseguir configurar os parâmetros do seu modelo.

Referências

[1] M. A. Abramson, C. Audet, G. Couture, J. E. Dennis Jr., S. Le Digabel e C. Tribes. *The NOMAD project.* Software disponível em https://www.gerad.ca/nomad. Consultado em 18/04/2017.

[2] C. Audet e J. E. Dennis Jr. *Mesh Adaptive Direct Search Algorithms for Constrained Optimization*. SIAM Journal on Optimization. V. 17, 2006. pp. 188–217.

[3] N. C. C. M. Benatti. *Métodos de Busca Direta para Seleção de Parâmetros em Máquinas de Vetores Suporte.* Dissertação de Mestrado, Universidade Federal do Paraná, Curitiba, 2017.

[4] B. E. Boser, I. M. Guyon e V. N. Vapnik. *A Training Algorithm for Optimal Margin Classifiers*. COLT'92: Proceedings of the Fifth Annual Workshop on Computational Learning Theory. ACM Press, New York, 1992 pp. 144–152.

[5] C. C. Chang e C. J. Lin. *LIBSVM: A Library for Support Vector Machines.* ACM Transactions on Intelligent Systems and Technology, V. 2, N. 3, artigo 27, New York, 2011. pp. 1-27.

[6] A. R. Conn, K. Scheinberg e L. N. Vicente. *Introduction to Derivative-Free Optimization.* Society for Industrial and Applied Mathematics, Philadelphia, 2009.

[7] C. Cortes e V. N. Vapnik. *Support Vector Networks*. Machine Learning, V 20, n 3. Kluwer Academic Publishers, Boston, 1995. pp. 273–297.

[8] E. D. Dolan, J. J. Moré. *Benchmarking Optimization Software with Performance Profiles.* Mathematical Programming, V. 91, 2002. pp. 201-213.

[9] C. W. Hsu, C. C. Chang e C. J. Lin. *A Practical Guide to Support Vector Classification.* Technical report, Department of Computer Science, National Taiwan University, 2016.

[10] V. Torczon. *On the Convergence of Pattern Search Algorithms*. SIAM Journal on Optimization. V. 7, 1997. pp. 1-25.