1

Rodrigo de Castro Cosme

# Máquinas de vetores suporte usando o algoritmo evolução diferencial com busca local para classificação de dados

**Vitória**
**2011**

Rodrigo de Castro Cosme

# Máquinas de vetores suporte usando o algoritmo evolução diferencial com busca local para classificação de dados

Dissertação apresentada à Universidade Federal do Espírito Santo, para a obtenção do Título de Mestre em Informática, na Área de Inteligência Computacional.

Orientador: Renato Antonio Krohling

**Vitória**
**2011**

Dissertação de Mestrado sob o título "Máquinas de vetores suporte usando o algoritmo evolução diferencial com busca local para classificação de dados", defendida por Rodrigo de Castro Cosme e aprovada em 30 de janeiro de 2012, em Vitória, Estado do Espírito Santo, pela banca examinadora constituída pelos professores:

---

Prof. Dr. Renato Antonio Krohling
Universidade Federal do Espírito Santo
Orientador

---

Profª. Drª. Maria Claudia Silva Boeres
Universidade Federal do Espírito Santo
Membro Interno

---

Prof. Dr. João Marques Salomão
Instituto Federal do Espírito Santo
Membro Externo

*Dedico este trabalho à minha família, pelo apoio incondicional.*

# Declaração de Autoria

Eu declaro que essa dissertação foi escrita por mim. Qualquer ajuda recebida durante o trabalho de pesquisa e na escrita tem sido reconhecida. Além disso, eu certifico que todas as fontes de informação e literatura usadas estão indicadas na dissertação.

---

Rodrigo de Castro Cosme

# Epígrafe

O único lugar onde o sucesso vem antes do trabalho é no dicionário.

Albert Einstein

# Agradecimentos

Aos professores, que compartilharam seus conhecimentos e me ensinaram o que eu precisava aprender.

Ao professor Renato Krohling, por sua orientação neste projeto e na vida.

Aos membros da banca Prof.ª Maria Claudia Silva Boeres e Prof. João Marques Salomão, por comporem a banca de avaliação da dissertação.

À Fundação de Apoio à Ciência e Tecnologia do Espírito Santo (FAPES) pelo fomento à pesquisa no Espírito Santo, em especial, pela bolsa de estudo a mim concedida.

Aos colegas de Mestrado, em especial Gilberto Segundo, que dividiram experiências e ajudaram a desenvolver conhecimento para tornar possível este trabalho.

Aos amigos, pelo apoio.

À família, pela dedicação e pela confiança em mim depositada.

*Resumo*

Mineração de dados é uma área chave para diversos campos da ciência e engenharia. Neste contexto, um método de aprendizado estatístico, conhecido como máquinas de vetores suporte tem se apresentado como um método promissor para solucionar classificação de dados. Geralmente, o problema de máquinas de vetores suporte (inglês: *Support Vector Machines* - SVM) é formulado como um problema de otimização não-linear sujeito a restrições. Técnicas de otimização convencionais que utilizam a abordagem Lagrangiana são usadas para solucionar este tipo de problema. No caso de classificação de dados ruidosos as técnicas convencionais apresentam deterioração de desempenho, já que o problema de otimização resultante é multidimensional e pode apresentar muitos mínimos locais. Neste trabalho, é proposto o algoritmo Evolução Diferencial combinado com uma técnica de busca local, uma hibridização de busca tabu com o método Nelder-Mead, para encontrar os parâmetros ótimos dos classificadores SVM aplicados a dados ruidosos.

**Palavras-chave:** classificação, máquinas de vetores suporte, Evolução Diferencial, Busca Tabu, Nelder-Mead

*Abstract*

Data mining is a key area for many fields of science and engineering. In this context, a statistical learning method, known as Support Vector Machines has presented itself as a promising method to solve data classification. Usually, the SVM problem is formulated as a nonlinear optimization problem subject to constraints. Conventional optimization techniques using the Lagrangian approach are used to solve this kind of problem. In the case of classification of noisy data the conventional techniques show performance deterioration, since the resulting optimization problem is multidimensional and may present many local minima. In this work, it is proposed the Differential Evolution algorithm (DE) combined with a local search technique, a hybridization of tabu search with Nelder-Mead method, to find the optimal parameters of SVM classifiers applied to noisy data.

**Keywords:** classification, Support Vector Machines, Differential Evolution, Tabu Search, Nelder-Mead

# Lista de Figuras

# Lista de Tabelas

# Lista de Símbolos

| | | |
|---|---|---|
| $SVM$ | Support Vector Machine | vii |
| $PSO$ | Particle Swarm Optimization | 4 |
| $GA$ | Genetic Algorithm | 4 |
| $DE$ | Differential Evolution | 4 |
| $RN$ | Rede Neural | 4 |
| $LS$ | Local Search | 13 |
| $b$ | Constante de deslocamento do hiperplano separador | 7 |
| $w$ | Vetor normal ao hiperplano separador | 7 |
| $C$ | Limite superior de todos os multiplicadores de Lagrange | 9 |
| $\xi$ | Variáveis slack | 9 |
| $l$ | Número de multiplicadores de Lagrange | 9 |
| $N_{car}$ | Número de características | 7 |
| $\alpha$ | Multiplicadores de Lagrange | 9 |
| $\beta$ | Vetor do núcleo Gaussiano modificado | 12 |
| $N_p$ | Tamanho da população | 14 |
| $F_m$ | Fator de mutação | 15 |
| $rand$ | Função que retorna número aleatório entre 0 e 1 | 16 |
| $C_r$ | Taxa de recombinação | 16 |
| $\alpha_l$ | Fator de escala do componente local do núcleo Gaussiano modificado | 19 |

# Sumário

# Capítulo 1

# Introdução

Para realizar a tarefa de classificação existem várias técnicas amplamente utilizadas, dentre elas: abordagem bayesiana (LUTS et al., 2010; LUUKKA; LAMPINEN, 2011), máquinas de vetores suporte (SOUZA, 2011; LUUKKA; LAMPINEN, 2011), k vizinhos mais próximos (LUUKKA; LAMPINEN, 2011), análise discriminativa (LUUKKA; LAMPINEN, 2011), Redes Neurais artificiais (LUUKKA; LAMPINEN, 2011).

Em 1995, Vapnik (1995) introduziu a fundamentação para máquinas de vetores suporte, um método estatístico de aprendizado de máquina. Em sua forma básica, o objetivo da SVM é definir funções de decisão através de hiperplanos separadores. A distância dos elementos pertencentes às duas classes distintas deve ser máxima em relação ao hiperplano separador. Quando as classes a que pertencem os elementos são conhecidas a priori, o problema é chamado de classificação. Os conjuntos de dados usados para calcular o limite de decisão entre as classes é chamado conjunto de treinamento, enquanto o conjunto usado para testar a eficácia do método é chamado conjunto de validação.

## 1.1 O Problema de classificação

Inicialmente, o problema de classificação consiste em separar duas classes, por isso chamado de problema de classes binário. A classificação de mais classes introduziu um novo nível de dificuldade, pois SVM foi matematicamente proposto para julgar apenas duas classes. Para classificar problemas de múltiplas classes (KIM et al., 2003) alguns métodos foram propostos. O conjunto de SVMs (inglês: *SVM ensemble*) (LLERENA, 2011) é um desses métodos. Ele consiste em aplicar SVM às $m$ classes do problema em grupos de dois, então cada saída desses SVMs binários é agregado para resolver o problema de múltiplas classes original. Alguns métodos para agregar SVMs binários para resolver problemas de múltiplas classes são dados por voto da maioria e escolha ponderada (KIM et al., 2003).

Adicionalmente, há casos em que as classes não são linearmente separáveis. Em tais casos, uma transformação não-linear é aplicada aos dados para atingir um plano onde os dados podem ser separados. Esta transformação é chamada função núcleo. Várias funções núcleo foram estudadas para melhorar o desempenho e para incorporar outras propriedades como a seleção de características (DU; PENG; TERLAKY, 2009).

Ao utilizar SVMs no contexto de classificação, uma série de parâmetros devem ser configurados de forma a minimizar o erro de classificação das instâncias de validação. Sendo esse um problema de otimização multidimensional, há que se evitar ficar preso em mínimos locais. Entre as alternativas para solucionar o problema de otimização, encontram-se os métodos determinísticos de otimização quadrática como gradientes conjugados, Quasi-Newton e pontos interiores Primal-Dual (PAQUET; ENGELBRECHT, 2003; SEMOLINI, 2002). Esses métodos normalmente são eficientes computacionalmente, mas podem levar a mínimos locais resultando em estagnação. Por outro lado, métodos baseados em computação evolutiva (EBERHART; SHI, 2007)

(algoritmos genéticos, evolução diferencial, otimização via enxame de partículas) têm se mostrado eficazes para solucionar estes tipos de problemas.

## 1.2 Objetivo

Geralmente, os métodos de busca ou otimização que trabalham com população são mais custosos computacionalmente que métodos baseados em gradiente mas podem prover soluções mais robustas. Em alguns casos, não há sequer uma solução definida ou um algoritmo exato aplicável ao problema. Pois não se trata de uma otimização de uma função matemática diferenciável, que seja possível calcular os parâmetros (coordenadas) que fazem com que a função cresça ou diminua, mas de uma otimização de um sistema com várias variáveis que regulam seu funcionamento. Nesse sentido, o processo de treinamento da SVM utilizando um algoritmo evolutivo inspirado biologicamente é desejável. Há várias formas de abordar o problema, uma delas é usar um algoritmo evolutivo para encontrar os parâmetros da SVM (LIN et al., 2008).

Ao aplicar qualquer metodologia para resolver problemas de otimização, a resolução do problema em si acontece depois da coleta de dados que caracterizam o problema. É durante a coleta e manipulação dos dados que podem ocorrer falhas (mecânicas, humanas, ambientais, no meio de transmissão dos dados, etc) que podem ser muito difíceis ou custosas de eliminar, de tal modo que o desenvolvimento de técnicas para manipular o ruído em meio aos dados é mais viável. O objetivo deste trabalho é encontrar os parâmetros da SVM usando algoritmo evolutivo para classificar dados que podem estar corrompidos por ruído e analisar a robustez da solução para problemas bem estabelecidos (FRANK; ASUNCION, 2010). Para isso, será investigado o impacto de diferentes tipos de ruído em diferentes partes dos dados e como isso afeta a precisão da classificação.

## 1.3 Metodologia

As SVMs são utilizadas em várias áreas da engenharia, incluindo aplicações para classificação de imagens, classificação de padrões, mineração de dados e predição. Muitos desses problemas estão sujeitos à interferência e ruído inerentes ao ambiente da aplicação, portanto qualquer método ou combinação de métodos aplicada deve ser capaz de lidar com tais ruídos e ainda prover soluções robustas. Alguns métodos que lidam com ruído incluem Rede Neural(RN) e SVMs.

Em (LIN et al., 2008) é mostrada uma abordagem para determinação de parâmetros e seleção de características do SVM utilizando o algoritmo de otimização via enxame de partículas (inglês: *Particle Swarm Optimization* - PSO). Outros algoritmos de otimização foram explorados no intuito de realizar a mesma tarefa, como Algoritmo Genético (HUANG; WANG, 2006) (inglês: *Genetic Algorithm* - GA) e o algoritmo Evolução Diferencial (inglês: *Differential Evolution* - DE) para otimização do modelo de SVM (LI, 2010; SUN et al., 2010). Em (DU; PENG; TERLAKY, 2009) é apresentada uma extensão da função núcleo gaussiano, comumente usada na teoria de aprendizado de máquina que pode realizar extração de características. Em (NETTLETON; ORRIOLS-PUIG; FORNELLS, 2010) é realizada uma análise do impacto na taxa de erro de classificação ao introduzir-se ruído em diferentes partes do conjunto de dados. Em (LUUKKA; LAMPINEN, 2011) são introduzidas e analisadas variáveis ruidosas, uma forma de introdução de ruído definida pelo autor.

Outros trabalhos focam em agrupamento de dados para produzir resultados significativos em classificação de dados ruidosos (BANERJEE, 2009), com a desvantagem de que o número de agrupamentos é conhecido a priori.

De modo a ter uma boa classificação e reduzir parâmetros necessários que o projetista deve configurar, nesta dissertação uma abordagem SVM+DE é proposta. Esta

abordagem já foi utilizada anteriormente para outras aplicações, como predição de tempestades (JUN; JIAN, 2009) e predição de carga de gás (SUN et al., 2010). Neste trabalho será proposto o uso de DE com busca local aplicado ao problema de classificação de dados ruidosos. Esta técnica híbrida tem o benefício do poder estatístico do SVM enquanto o DE é usado para encontrar os valores dos parâmetros do SVM em conjunto com um algoritmo de busca local híbrido que consiste de Busca Tabu (MASHINCHI; ORGUN; PEDRYCZ, 2011) e do método Nelder-Mead (NELDER; MEAD, 1965), para evitar cair em mínimos locais.

## 1.4 Estrutura

No Capítulo 2 é descrito o método SVM. O Capítulo 3 apresenta o algoritmo DE. O Capítulo 4 descreve os algoritmos Busca Tabu e Nelder-Mead usados para busca local. O Capítulo 5 ilustra o método híbrido. O Capítulo 6 mostra os resultados e discussões. No Capítulo 7 as conclusões são apresentadas.

# Capítulo 2

# Máquina de Vetores Suporte

A modelagem de dados desempenha um papel fundamental em muitos processos de apoio à tomada de decisão. Com base em dados, um modelo do sistema é construído com o propósito de dar respostas sobre observações que ainda serão feitas. Abordagens tradicionais como Redes Neurais têm sofrido dificuldades de generalização, podendo gerar modelos que se superadequam aos dados como consequência dos algoritmos usados para selecionar os parâmetros e os métodos usados para selecionar o melhor modelo. Resultados promissores têm sido obtidos com SVM (KIM et al., 2003). Sua formulação engloba o princípio da Minimização do Risco Estrutural (inglês: *Structural Risk Minimization*), que se mostrou superior ao princípio tradicional de Minimização do Risco Empírico (inglês: *Empirical Risk Minimization*), usado nas redes neurais convencionais (GUNN, 1998). Originalmente SVM foi desenvolvida para classificação, mas posteriormente foi extendida para regressão (VAPNIK; GOLOWICH; SMOLA, 1996) e predição (SAPANKEVYCH; SANKAR, 2009). Uma discussão mais detalhada sobre SVM para classificação é descrita a seguir.

## 2.1 Fundamentação teórica

Dado um conjunto de treinamento $Z = \{(x^1,y^1),(x^2,y^2),\ldots,(x^l,y^l)\}$ de $l$ instâncias onde cada instância é composta de $N_{car}$ características ($x^i = (x^i_1,\ldots,x^i_{N_{car}})^T \in R^{N_{car}}$) e um rótulo de classificação $y^i \in \{1,-1\}$ pertencente a uma de duas classes. A tarefa de classificação consiste em separar duas classes com um hiperplano.

$$f(\mathbf{w},b) = sign(\mathbf{w}\cdot\mathbf{x}+b) \tag{2.1}$$

O vetor $w$ é um vetor de valor desconhecido da normal do hiperplano $wx - b = 0$. As retas $wx - b = -1$ e $wx - b = 1$ são hiperplanos auxiliares (ou margens) separadas por uma distância de $\frac{2}{||w||}$, o parâmetro $b$ (chamado bias) denota uma constante de deslocamento do hiperplano e é calculado usando dois vetores suporte (pontos que estão em cima dos hiperplanos auxiliares), mas pode ser calculado usando todos os vetores suporte na margem para dar estabilidade e evitar que o cálculo fique dependente de um subconjunto dos dados (GUNN, 1998; VAPNIK; GOLOWICH; SMOLA, 1996), obtendo um maior poder de generalização. A Equação 2.1 retorna como resultado se um vetor $x$ está à direita ou à esquerda do plano, pertencendo à classe $+1$ ou $-1$ respectivamente.

A Figura 2.1 ilustra o hiperplano separador e os vetores suporte para um exemplo sintético em que os dados são criados artificialmente de acordo com a distribuição normal (YAZDI; EFATI; SABERI, 2007). Para este exemplo duas classes de dados são geradas a partir de duas funções de densidade de probabilidade pré-estabelecidas. A classe 1, em vermelho, e a classe 2, em azul, foram geradas por distribuições normais com $\mu_1 = 1$, $\sigma_1 = 0{,}1$ e $\mu_2 = -1$, $\sigma_2 = 0{,}4$ respectivamente.

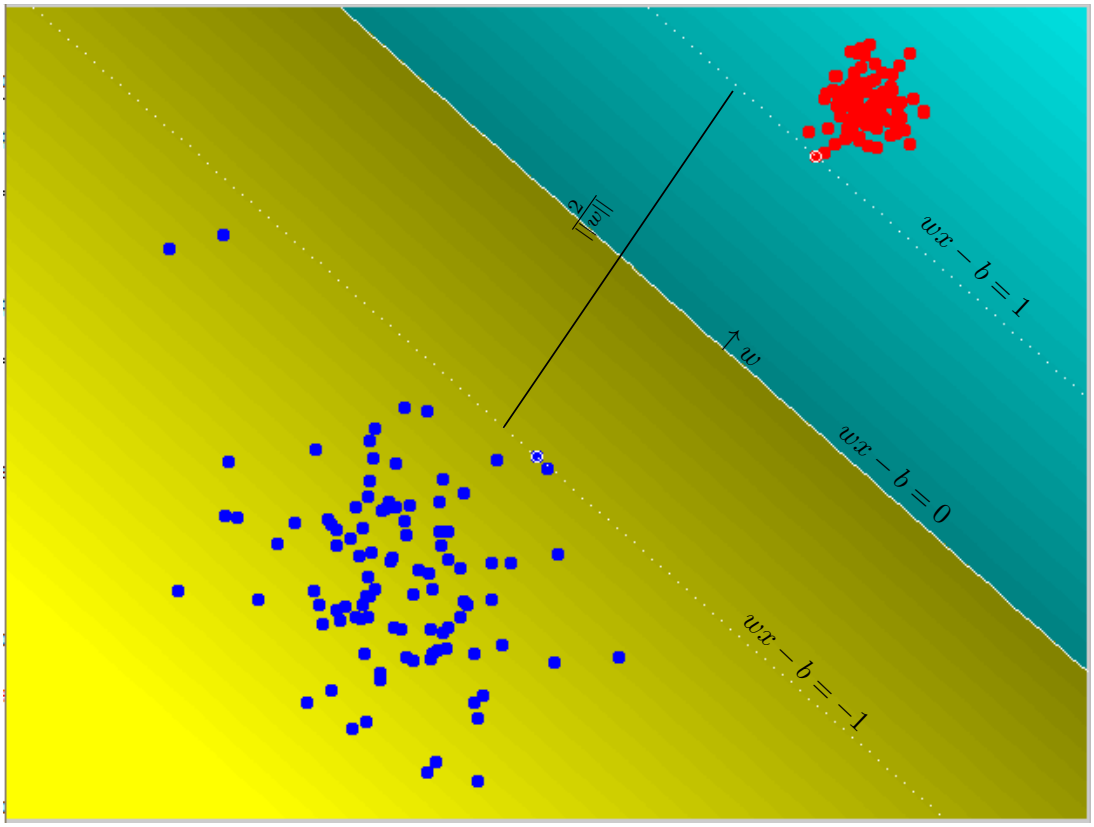



Figura 2.1: Hiperplano separador ótimo.

## 2.2 Formulação como problema de otimização

Embora a Equação 2.1 possa separar qualquer parte dos espaços de características, é necessário estabelecer um hiperplano separador ótimo (HSO) (KAWANO et al., 2009) (**w***,**b***). Para separar otimamente o conjunto de vetores, eles devem ser separados de modo que minimize o erro de classificação de uma nova instância e a distância entre os vetores mais próximos ao hiperplano seja máxima. O hiperplano pode ser determinado pela solução do seguinte problema de minimização sujeito a (abreviação: s. a.) restrições.

$$\begin{aligned} &\min \tfrac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{l}\xi_i \\ &\text{s.a. } y^i[\mathbf{w}\cdot\mathbf{x}^i + b] \geq 1 - \xi_i \end{aligned} \tag{2.2}$$

em que $C$ é o limite superior de todos os multiplicadores de Lagrange e as variáveis $\xi_i \geq 0$ são introduzidas na construção do hiperplano separador ótimo, quando os dados são linearmente não-separáveis.

A solução do problema de otimização descrito pela Equação 2.2 é dada pelo ponto de cela do funcional Lagrangiano.

$$L(w,b,\alpha) = \frac{1}{2}||w|| - \sum_{i=1}^{l}\alpha_i\{[(x_i w) - b]y_i - 1\} \tag{2.3}$$

onde $l$ é o número de multiplicadores de Lagrange $L(w,b,\alpha)$ que deve ser minimizada em relação a $w$,$b$ e maximizada em relação a $\alpha_i \geq 0$. Dadas as seguintes propriedades do hiperplano:

1. Os coeficientes $\alpha_i$ do hiperplano ótimo devem satisfazer as condições

$$\sum_{i=1}^{l}\alpha_i y_i = 0, a_i \geq 0, i = 1,\ldots,l \tag{2.4}$$

2. Os parâmetros do hiperplano ótimo são combinações lineares dos vetores do conjunto de treinamento

$$w = \sum_{i=1}^{l} y_i\alpha_i x_i, a_i \geq 0, i = 1,\ldots,l \tag{2.5}$$

3. A solução deve satisfazer a seguinte condição de Karush-Kuhn-Tucker

$$\alpha_i[(x_i w) - b]y_i - 1 \tag{2.6}$$

Substituindo-se a Equação 2.5 no Lagrangiano e levando-se em consideração a condição de Karush-Kuhn-Tucker, obtém-se a forma Dual do problema de otimização, dependente somente de $\alpha$

$$\begin{aligned} &\max \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \\ &\text{s.a.} \quad 0 \leq \alpha_i \leq C, 0 \leq i \leq l \\ &\qquad \sum_{j=1}^{l} \alpha_j y_j = 0 \end{aligned} \tag{2.7}$$

Em 1992, foi observado (VAPNIK, 1995) que para descrever o hiperplano separador ótimo no espaço de características e estimar os coefiecientes de expansão correspondentes do hiperplano separador pode-se usar o produto interno de dois vetores $\phi(x_1)$ e $\phi(x_2)$, que são imagens no espaço de características dos vetores de entrada $x_1$ e $x_2$. Portanto, se pudermos estimar o produto interno de dois vetores no espaço de características $\phi(x_1)$ e $\phi(x_2)$ como uma função de duas variáveis no espaço de entrada

$$(\Phi(x_i)\Phi(x_j)) = K(x_i,x_j) \tag{2.8}$$

é possível construir soluções que são equivalentes ao hiperplano ótimo no espaço de características. Para obter essa solução deve-se substituir o produto interno $(x_i,x_j)$ pela função $K(x_i,x_j)$, chamada função núcleo, descrita pela Equação 2.8. Esta equação, dada pelo Teorema de Mercer (WANG et al., 2009), é utilizada para resolver o caso de superfícies de decisão não-linear (quando as classes não são linearmente separáveis).

Várias funções núcleo foram propostas e estudadas ao longo dos anos, dentre elas as funções núcleo polinomial, sigmoidal, Gaussiano, função de base radial (inglês: *Radial Basis Function* - RBF). A Figura 2.2 ilustra uma função núcleo, que é um produto interno de vetores, como pode ser visto na Equação 2.9. O núcleo Gaussiano, dado pela Equação 2.9, e a função de decisão do SVM binário descrita pela Equação 2.1, podem

ser combinados e reescritos de acordo com a Equação 2.10.

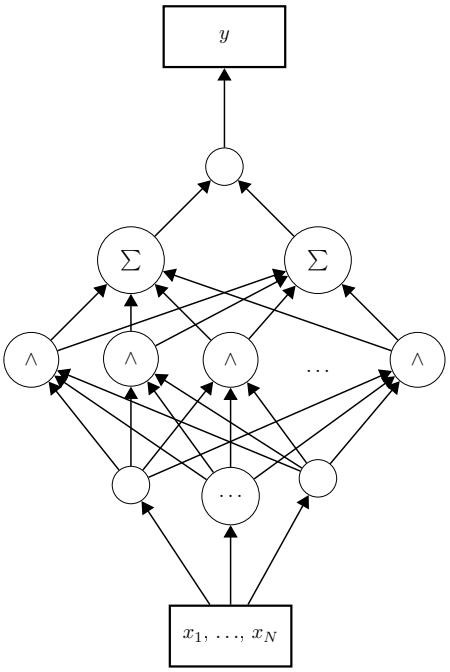



Figura 2.2: Ilustração de uma função núcleo.

$$K(x_i,x_j) = exp(\frac{-||x_i - x_j||^2}{2\sigma^2}) \tag{2.9}$$

$$g(x) = sign(\sum_{i=1}^{l} \alpha_i y_i K(x_i,x) + b) \tag{2.10}$$

Reescrevendo a Equação 2.10 e utilizando a Equação 2.5, temos a equação que define o hiperplano separador $w\Phi(x) + b = 0$. Um ponto $x$ pertence à primeira classe se $g(x) = sign(w\Phi(x) + b)$ for igual a $+1$ ou à segunda classe se $g(x)$ for igual à $-1$.

O núcleo Gaussiano mostra-se com boa capacidade de generalização (WU et al., 2007), por isso uma extensão deste núcleo, proposta em (DU; PENG; TERLAKY, 2009), que adiciona as propriedades de seleção e de mapeamento de características, é utilizada neste trabalho. A seleção de características é feita pelo vetor de parâmetros

$\beta$, descrito na Equação 2.11. Quanto menor o valor de $\beta_k$ na Equação 2.11, menos relevante a característica $k$ é para a classificação. Portanto, remover esta característica em questão não afetaria o classificador.

$$K(x_i,x_j) = exp(\frac{-\sum_{k=1}^{N_{car}} \beta_k(x_{ik} - x_{jk})^2}{2\sigma_k^2}) \tag{2.11}$$

Além da escolha da função núcleo, a seleção de amostras para treinamento e validação afeta muito o desempenho do classificador. Está bem estabelecido na literatura a proporção de 70% do conjunto de dados usados para treinamento e 30% para validação (KIM et al., 2003; BHATTACHARYYA, 2004), entretanto, quais amostras reservar para cada tarefa ainda é uma questão em aberto. Alguns algoritmos foram propostos para resolver esse problema. Em (KHOSHGOFTAAR; HULSE; NAPOLITANO, 2011) são comparadas técnicas de *Boosting* e *Bagging* (RÜBESAM, 2004). O método boosting surgiu como reposta ao seguinte problema teórico: "Suponha que existe um método de classificação que é ligeiramente melhor do que uma escolha aleatória, para qualquer distribuição em $X$. Esse método é chamado de *weak learner*, ou classificador fraco. A existência de um classificador fraco implica na existência de um classificador forte (*strong learner*), com erro pequeno em todo o espaço $X$. Em 1990, Schapire (1990) mostrou que era possível obter um classificador forte a partir de um fraco. Já o método bagging baseia-se em criar preditores em amostras bootstrap dos dados, e depois agregá-los, ou combiná-los, para formar um preditor (que espera-se) seja melhor.

Com o crescimento do conjunto de dados a tarefa de classificação torna-se consideravelmente mais complexa à medida que cresce o número de características e de vetores de características, devido ao número de produtos internos necessários ao mapeamento do espaço de características no espaço multidimensional. Em virtude do crescimento do número de vetores de características, como visto na Equação 2.3, cresce o número de parâmetros desconhecidos que devem ser otimizados, como $\alpha_i$, com $i = 1,\ldots,l$. A

seguir, serão descritos os algoritmos para otimizar os parâmetros do SVM.

Muitas pesquisas foram feitas no campo de SVM com ruído para definir métodos de remoção dos mesmos ou para acomodar o modelo de aprendizagem aos dados perturbados. Tópicos de pesquisa incluem o agrupamento de dados para remover o ruído (ZHANG; RAMAKRISHNAN; LIVNY, 1997; BANERJEE, 2009), classificadores com evolução diferencial (LUUKKA; LAMPINEN, 2011) e seleção de características relevantes (BYEON; RASHEED, 2008). Nesse trabalho, as abordagens SVM+DE e o equivalente com busca local (inglês: Local Search - LS), SVM+DE+LS, são propostas para classificação de dados (KROHLING, 2011). Portanto, esta é uma primeira abordagem usando SVM+DE+LS para gerar o modelo de classificação para dados ruidosos.

# Capítulo 3

# Evolução Diferencial

O algoritmo de otimização Evolução Diferencial (DE) foi introduzido por Storn (1995). Como um algoritmo evolutivo baseado em população, cada membro da população é uma solução candidata, uma possível solução do problema. Cada membro da população é sujeito a operações básicas de mutação, recombinação e seleção. As principais características desse algoritmo são as de fácil paralelização e boa convergência. Soluções candidatas são representadas por vetores de parâmetros. Os componentes dos vetores são variáveis do problema e são otimizados de acordo com uma função de avaliação (função objetivo). A população consiste de um conjunto de vetores e seus membros são representados por $x_i = [x_{i1}, x_{i2}, \ldots, x_{iD}]^T$ com componentes referenciados como $x_{ij}$, onde os índices $i$ representam cada indivíduo na população de tamanho $N_p$ e $j$ são componentes dos indivíduos no espaço D-dimensional. Os limites inferior e superior de uma solução candidata qualquer são $\underline{x_j}$ e $\overline{x_j}$, respectivamente, com $\underline{x_j} \leq x_{ij} \leq \overline{x_j}$, $i = 1, \ldots, N_p$, $j = 1, \ldots, D$. Os $N_p$ indivíduos da população são inicializados aleatoriamente no espaço de busca D-dimensional e após a inicialização o valor da função objetivo de cada vetor é calculado. Os três operadores: mutação, recombinação e seleção são mostrados na Figura 3.1 e descritos a seguir (STORN; PRICE, 1997).

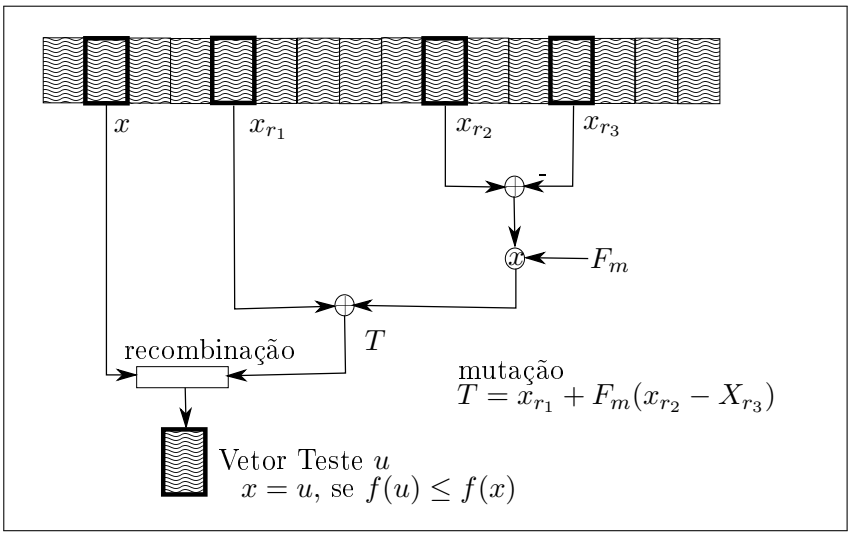



Figura 3.1: Ilustração dos operadores usados no algoritmo DE.

## 3.1 Operadores evolutivos

Durante o processo evolutivo, são realizadas as operações de mutação, recombinação e seleção para cada membro da população, até que um critério de parada seja satisfeito, seja ele número máximo de iterações ou número de iterações sem melhora do menor valor objetivo (no caso de minimização). Os operadores evolutivos são descritos a seguir.

***Mutação***

O operador de mutação é responsável pelo que é chamado de exploração, que significa introduzir informação nova no processo de busca implicando na procura em novas áreas do espaço de busca. Cada vetor solução $x_i$ na geração $t$ ($x_i^t$) origina um vetor mutante $u_i^t$ definido por

$$u_i^t = x_{r1}^t + F_m(x_{r2}^t - x_{r3}^t) \tag{3.1}$$

onde $r_1$, $r_2$ e $r_3$ são inteiros aleatórios e diferentes mutualmente ($i \neq r_1 \neq r_2 \neq r_3$) no intervalo [1,2,...,$N_p$], portanto é necessário que $N_p \geq 4$. O fator de escala $F_m$

(chamado fator de mutação) introduz diversidade entre dois indivíduos para evitar estagnação na busca. O fator de mutação é usualmente extraído do intervalo [0,5; 1] como indicado em (STORN; PRICE, 1997).

### *Recombinação*

Após a operação de mutação, a recombinação realiza a intensificação ou refinamento da busca (SILVA, 2009) (inglês: *exploitation*), que significa procurar nas áreas perto das soluções atuais. Cada vetor alvo é comparado com o seu equivalente mutante $v_i^t$, gerando um vetor teste $u_i$ de acordo com

$$u_{i,j}^t = \begin{cases} v_{i,j}^t, & \text{se } rand_j \leq C_r \text{ ou } j = k \\ u_{i,j}^t, & \text{caso contrário} \end{cases} \tag{3.2}$$

onde $rand_j(0,1)$ é um número aleatório amostrado da distribuição uniforme para cada variável $j$. O índice $k = 1, \ldots, D$ é um índice das variáveis selecionadas aleatoriamente para cada $i$ para assegurar que, pelo menos uma variável, é sempre selecionada do vetor mutante $v_i^t$. A taxa de recombinação, $C_r$ , controla a fração de valores mutantes usados, o seu valor normalmente usado está no intervalo [0,8; 1] como indicado em (STORN; PRICE, 1997).

### *Seleção*

Durante a operação de seleção, o melhor valor entre o vetor pai $x_i^t$ e o vetor teste $u_i^t$ é selecionado, de acordo com seu valor de custo calculado pela função objetivo $f()$.

Para problemas de minimização, o vetor selecionado é dado por

$$x_i^{t+1} = \begin{cases} u_i^t, & \text{se } f(u_i^t) \leq f(x_i^t) \\ x_i^t, & \text{caso contrário} \end{cases} \tag{3.3}$$

## 3.2 O algoritmo DE

O desempenho do algoritmo DE pode ser afetado pelo tamanho da população $N_p$, pelo fator de mutação $F_m$ e pela taxa de recombinação $C_r$. Diferentes variantes do DE foram desenvolvidas e classificadas nos anos recentes usando a seguinte notação: $DE/\alpha_D/\beta_D/\gamma_D$, onde $\alpha_D$ indica o método para selecionar o indivíduo pai para formar a base do vetor mutante; $\beta_D$ indica o número de vetores diferença usados para perturbar a base do indivíduo, e $\gamma_D$ indica o método de recombinação usado para criar a próxima população.

Em cada geração $t$ do algoritmo, DE percorre cada um dos indivíduos $x_i^t$ criando o vetor teste $u_i^t$, como mostra o Algoritmo 3.1 apresentado a seguir. Esta é a versão mais comum de DE: DE/rand/1/bin (LUUKKA; LAMPINEN, 2011). Nesta versão do algoritmo: o valor $\alpha_D = rand$ significa que o vetor a ser perturbado é um vetor aleatório e não o elemento de melhor solução da população (DE/best/1/bin); o valor $\beta_D = 1$ significa que só uma diferença ponderada por um fator de mutação é adicionada e não duas como $v_i^t = x_{r1}^t + F_{m_1}(x_{r2}^t - x_{r3}^t) + F_{m_2}(x_{r4}^t - x_{r5}^t)$ (DE/rand/2/bin); e o valor $\gamma_D = bin$ significa que o tipo de perturbação utilizada é extraída da distribuição binomial e não a exponencial (DE/rand/1/exp).

```
1  input
2    N_p: tamanho da população
3    D: dimensão dos indivíduos
4    X^0: população inicial, composta por x_1^0,...,x_{N_p}^0
5    F_m: fator de mutação
6    C_r: taxa de recombinação
7  output
8    X^t: população otimizada após t iterações
9  begin
10   Inicializa população
11   do
12     for i=1:N_p
13       r1,r2,r3 ∈ {1,...,N},r1 ≠ r2 ≠ r3
14       j_rand = floor(rand_i[0,1] * D) + 1
15       for j=1:D
16         if rand[0,1] ≤ C_r %recombinação de acordo com a Equação 3.2 para o método DE/rand/1/bin
17           u_{i,j}^t = x_{r1,r1}^t + F_m(x_{r2,j}^t − x_{r3,j}^t) %mutação de acordo com a Equação 3.1
18         else
19           u_{i,j}^t = x_{i,j}^t
20         endif
21       endfor
22       if f(u_i^t) ≤ f(x_i^t) %seleção de acordo com a Equação 3.3
23         x_i^t = u_i^t
24       endif
25     endfor
26   until o número de iterações ou critério de parada seja satisfeito
27 end
```

Algoritmo 3.1: Evolução Diferencial.

Para melhor equilibrar exploração e intensificação, foi proposto em Das et al. (2009) o uso de mutação baseada em vizinhança (DAS; SUGANTHAN, 2011). Neste método a vizinhança local, onde o melhor indivíduo está em uma pequena vizinhança, e a vizinhança global, onde o melhor indivíduo está na população inteira da iteração atual, são usadas em conjunto. A combinação do modelo de vizinhança local e global é feita por um novo parâmetro $w_m$, o fator peso, resultando em um vetor de mutação baseado

em vizinhança.

Para criar o vetor local, o melhor vetor na vizinhança e dois outros vetores são escolhidos, como descrito pela Equação 3.4:

$$L_i = X_i + \alpha_l(X_{best_i} - X_i) + \beta_v(X_p - X_q) \tag{3.4}$$

onde $X_{best_i}$ é o melhor vetor na vizinhança de $X_i$, tal que os índices dos vizinhos de $X_i$, $p$ e $q$, estão no intervalo $[i - k_r,i + k_r]$, onde $p \neq q \neq i$ e $k_r$ é o raio da vizinhança.

Analogamente, o vetor global é calculado de acordo com:

$$G_i = X_i + \alpha_l(X_{best} - X_i) + \beta_v(X_{r1} - X_{r2}) \tag{3.5}$$

onde $X_{best}$ é o melhor vetor da população com $r_1,r_2 \in [1,N]$ onde $r_1 \neq r_2 \neq i$. Os parâmetros $\alpha_l$ e $\beta_v$ são fatores de escala.

O vetor resultante da mutação é formado como uma combinação de $L_i$ e $G_i$ dado por:

$$V_i = w_m G_i + (1 - w_m)L_i \tag{3.6}$$

Com respeito a vizinhança, a topologia é usada para construir o conceito de proximidade. Algumas topologias foram propostas para o algoritmo otimização via enxame de partículas (inglês: *Particle Swarm Optimization* - PSO) (BASTOS-FILHO et al., 2008), tais como estrela, anel e circular, mas experimentos mostraram que a topologia em anel, ilustrada na Figura 3.2, é melhor para o caso do DE (DAS et al., 2009). Neste caso, em uma população de $N_p$ indivíduos, os vizinhos do indivíduo $X_i$ são $X_{i-k_r} \ldots X_i \ldots X_{i+k_r}$, onde $k_r \in [1,N_p]$, conforme mostrado na Figura 3.2, em que os vizinhos de $x_i$ em um

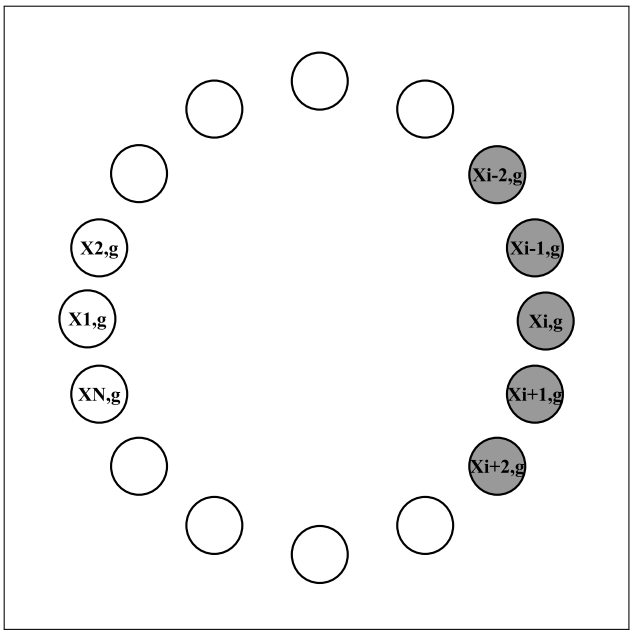



Figura 3.2: Topologia anel para indivíduos na geração $g$.

raio $k_r = 2$ são mostrados em negrito na iteração $g$ do algoritmo.

Motivado por (KROHLING; COELHO, 2006), onde os coeficientes do PSO foram gerados por uma distribuição de probabilidade exponencial com bons resultados, a mesma distribuição é usada neste trabalho para gerar o fator de escala no DE. A distribuição de probabilidade exponencial com função de densidade $f(r)$ é descrita pela Equação 3.7 a seguir:

$$f(r) = \frac{1}{2b_e} exp(-|r - a_e|/b_e), \ -\infty \leq r \leq \infty, a_e, b_e \geq 0 \tag{3.7}$$

Para controlar a variância pode-se variar os parâmetros $a_e$ e $b_e$. Neste trabalho foram configurados $a_e = 0$ e o fator de escala $b_e = 1$. Resultados experimentais indicam que o uso de um fator de escala extraído da distribuição de probabilidade exponencial tem contribuído de maneira positiva para o desempenho do DE.

Durante o processo de otimização, alguns indivíduos podem convergir para um

mínimo local, atraindo a população para o mesmo ponto. Em tais casos, o algoritmo pode não conseguir encontrar soluções melhores. Com esse propósito, um algoritmo de busca local será usado para que, quando o processo evolutivo ficar preso a uma solução, a busca local possa encontrar um ponto vizinho melhor, caso ele exista.

# Capítulo 4

# Métodos de Busca Tabu e Nelder-Mead

Com o objetivo de evitar mínimos locais durante processo de otimização, foi adotado o uso de Busca Tabu (inglês: *Tabu Search* - TS) (HERTZ; TAILLARD; WERRA, 1995) em conjunto com o método Nelder-Mead para intensificar a busca na vizinhança local numa tentativa de escapar da estagnação durante o processo evolutivo.

## 4.1 Método de Busca Tabu

Busca Tabu é uma metaheurística, método que visa aproximar a solução ideal do problema de otimização (normalmente otimização combinatorial) de forma genérica e cujo resultado não é necessariamente o melhor. No seu funcionamento, o método Busca Tabu pode ser aplicado diretamente a declarações verbais ou simbólicas (GLOVER; LAGUNA, 2002), não há necessidade de transformar o problema de decisão em formulação matemática. A otimização do método depende apenas da avaliação da função a ser otimizada para diferentes pontos. A notação matemática é introduzida para descrever uma grande classe de problemas e descrever algumas características de TS. Estes

problemas são caracterizados por otimização (minimização ou maximização) de uma função $f(x)$ sujeita a $x \in X$, onde $f(x)$ pode ser linear ou não linear e o conjunto $X$ representa as restrições no vetor de variáveis de decisão. As restrições podem incluir desigualdades lineares e não lineares e podem obrigar todos ou alguns componentes de $x$ a receberem valores discretos. Embora esta representação seja útil para discutir algumas considerações sobre a resolução de problemas, em muitas aplicações de otimização o problema pode não ser facilmente formulado como uma função objetivo sujeita a restrições.

TS começa como qualquer busca local ou busca em vizinhança, procedendo iterativamente de um ponto (solução) para outro até que um critério de parada seja satisfeito. Cada $x \in X$ tem uma vizinhança associada $N(x) \subset X$ e cada solução $x' \in N(x)$ é alcançada a partir de $x$ através de uma operação chamada movimento.

Como ponto de partida, TS pode ser comparada ao método de descida simples para minimização de $f(x)$ (ou de subida simples para maximização de $f(x)$). Tais métodos permitem mover-se para soluções que melhorem o valor da função objetivo atual terminando quando nenhuma melhora é alcançada. O método de descida genérico é mostrado no Algoritmo 4.1. O $x$ final obtido por um método de descida é conhecido por mínimo local, já que ele é melhor ou no mínimo igual a todas as outras soluções na vizinhança. A desvantagem do método de descida é que um ótimo local não é garantidamente um ótimo global.

```
1  input
2    X: espaço de busca
3  output
4    x: ponto ótimo no espaço de busca
5  begin
6    Escolha x ∈ X para começar o processo
7    Encontre x' na vizinhança de x tal que f(x') < f(x)
8    Se nenhum x' puder ser encontrado, x será o ótimo local e o método termina
9    Senão, atribua x = x' e volte para a linha 7
10 end
```

Algoritmo 4.1: Método de descida genérico.

Em Busca Tabu, um número $R$ de vizinhos aleatórios é gerado e o melhor ponto (uma solução no espaço de busca) avaliado é escolhido como ponto inicial, solução atual ($C_S$) e melhor solução ($S$). A lista usada para armazenar pontos recentes, lista tabu de curta duração (inglês: *Short-Term Tabu List* - STTL), é atualizada com $C_S$. Então $R$ vizinhos são gerados aleatoriamente em torno de $C_S$ baseados numa estratégia de busca e classificados de acordo com seus desempenhos. O melhor ponto vizinho é copiado para $C_S$ se já não for membro da STTL. Um novo vizinho é escolhido como próximo movimento (será a solução atual na próxima iteração) se sua avaliação na função objetivo for menor que a avaliação de $S$, no caso de minimização. Para intensificar a busca, $S$ é adicionado à lista tabu de longa duração (inglês: *Long-Term Tabu List* - LTTL). Portanto, se $C_S$ é melhor que $S$, então $S$ é substituído por $C_S$. O processo de gerar vizinhos e selecionar o melhor termina apenas quando o número máximo de iterações é atingido ou quando não há melhoras após algumas iterações. O método de Busca Tabu descrito anteriormente é mostrado no Algoritmo 4.2 em forma de pseudo código.

```
1  input
2    C (resultado da otimização via DE)
3  output
4    S (resultado da otimização local)
5  begin
6    for k=1:iterationsMax
7      Gerar R vizinhos em torno de C
8      Calcular vizinho com melhor função objetivo
9      if vizinho não está na lista STTL
10       C = vizinho
11       calcular valor objetivo de C // f(C)
12       incluir C em STTL
13       //critério de aspiração
14       if f(C) > f(S) // solução atual melhor que melhor
                solução
15         S = C
16         incluir S em LTTL
17       end
18     else
19       C = último elemento de LTTL
20     end
21   end
22 end
```

Algoritmo 4.2: Algoritmo Busca Tabu.

## 4.2 Método Nelder-Mead

De modo a encontrar o mínimo de uma função, a estratégia do método de busca Nelder-Mead (NM) gera $M$ pontos em um espaço de busca M-dimensional, de $P_1, \ldots, P_M$, em torno de um ponto de partida $P_0$. O ponto de partida pode ser escolhido aleatoriamente ou ser o resultado de um algoritmo executado previamente. Cada ponto $P_i$ avaliado tem seu valor de função objetivo denominado $Y_i$, onde $Y_i = f(P_i)$ é o resultado da avaliação do ponto $P_i$ por uma função $f$ que é objeto de minimização. O maior e menor dentre os valores objetivos dos $M$ pontos avaliados são $Y_h = \underset{i}{max}(Y_i)$ e

$Y_l = \underset{i}{min}(Y_i)$, respectivamente. Em cada etapa, $P_h$ é substituído por um novo ponto.

As operações usadas para a busca NM são reflexão, expansão e contração, gerando os pontos $P_r$ , $P_e$ e $P_c$ , respectivamente. Outro ponto usado nestas transformações é o centróide, denotado por $\overline{P}$. Estas operações são definidas a seguir:

$$\overline{P} = \frac{1}{n}\sum_{i=1}^{n} P_i \tag{4.1}$$

$$P_r = (1 + \alpha_{NM})\overline{P} - \alpha_{NM} P_h \tag{4.2}$$

$$P_e = \gamma_{NM} P_r + (1 - \gamma_{NM})\overline{P} \tag{4.3}$$

$$P_c = \beta_{NM} P_h + (1 - \beta_{NM})\overline{P} \tag{4.4}$$

onde $\alpha_{NM}$ , $\beta_{NM}$ e $\gamma_{NM}$ são valores no intervalo [0,1].

Dado que $P^*$ recebe a reflexão do ponto $P_h$ , tem-se que $P^*$ está na linha que une $P_h$ e $\overline{P}$. Se $Y^*$ está entre $Y_h$ e $Y_l$, então $P_h$ é substituído por $P^*$ e o processo recomeça do princípio. Se $Y^* < Y_l$, a reflexão produziu um novo mínimo, então $P^*$ é expandido para $P^{**}$. Se $Y^{**} < Y_l$, $P_h$ é substituído por $P^{**}$ e o processo reinicia. Mas se $Y^{**} > Y_l$ então tem-se uma expansão falha e $P_h$ é substituído por $P^*$ antes de reiniciar. Se ao refletir $P$ para $P^*$ for obtido $Y^* > Y_l$ para todo $i \neq h$, de modo que a substituição de $P$ por $P^*$ torna $Y^*$ máximo, então um novo valor é atribuído a $P_h$ seja ele o antigo valor de $P_h$ ou $P^*$, o que possuir o menor valor de $Y$.

Então $P^{**}$ é atualizado com a contração de $P_h$ e o método reinicia, a não ser que $Y^{**} > min(Y_h, Y^*)$, de modo que o ponto contraído é pior que o melhor entre $P_h$ e $P^*$. Para tal falha na contração, todos os $P_i$'s são substituídos por $(P_i + P_l)/2$ e o método reinicia.

O método de busca NM está listado conforme o Algoritmo 4.3. A aplicação do

método no trabalho é descrita no Capítulo 5.

1 **input**
2 $P_0$: (ponto de partida)
3 $M$: número de pontos gerados na vizinhança de $P_0$
4 **output**
5 $P_l$: melhor ponto encontrado na vizinhança de $P_0$
6 **begin**
7 Realizar reflexão para o ponto $P_h$
8 Copiar $P_r$ para $P_{temp}$
9 **if** $f(P_{temp}) \leq Y_l$ substituir $P_l$ por $P_{temp}$ e $Y_l$ por $f(P_{temp})$ **else** vá para passo 12
10 Realizar expansão em ponto $Y_l$
11 Copiar $P_e$ para $P_{temp}$ vá para passo linha 9
12 **if** ($f(P_{temp}) < Y_h$ e $f(P_{temp}) > Y_i, i \neq h$) substituir $P_h$ por $P_{temp}$ e $f(P_h)$ por $f(P_{temp})$
13 Realizar contração de $P_h$
14 Copiar $P_c$ para $P_{temp}$
15 **if** $f(P_{temp}) > Y_h$ substituir todos $P_i$s por $((P_i + P_l))/2$
16 **if** condição de parada não foi alcançada vá para linha 7
17 Retorne $P_l$ e $Y_l$
18 **end**

Algoritmo 4.3: Algoritmo Nelder-Mead.

onde o critério de parada no passo 16 é dado por:

$$\frac{1}{D}\sum_{i=1}^{n} \parallel P_i^k - P_i^{k+1} \parallel^2 < \epsilon \tag{4.5}$$

onde $P_i^k$ e $P_i^{k+1}$ são os pontos nas iterações $k$ e $k+1$, respectivamente, e $\epsilon$ é um número real pequeno.

# Capítulo 5

# Otimização dos parâmetros das máquinas de vetores suporte

Para realizar a otimização dos parâmetros de vetores suporte, os algoritmos e métodos descritos nos capítulos anteriores são utilizados em conjunto. Os parâmetros do SVM, descritos no Capítulo 2, são otimizados através de algoritmo evolutivo. Dentre esses parâmetros, aqueles utilizados para otimização do classificador são explicitados neste capítulo.

Devido à natureza multidimensional do indivíduo (membro da população do algoritmo evolutivo), o processo evolutivo pode ficar estagnado a uma solução que não necessariamente é a ótima, um mínimo local. Para evitar a estagnação da solução nos mínimos locais, um método de busca local é utilizado para guiar a otimização e escapar de mínimos locais.

Na configuração de um algoritmo de otimização a função objetivo, que dita o quão boa é uma solução, desempenha um papel fundamental no processo evolutivo. A função objetivo utilizada bem como a representação dos parâmetros e a busca local são detalhados a seguir.

## 5.1 Função objetivo

O método consiste em encontrar os parâmetros da função núcleo e selecionar o número mínimo de características enquanto maximizamos a taxa de acerto de classificação do SVM, isto é, minimizamos os erros de classificação feitos pelo SVM. Dessa forma, definimos a função objetivo como o inverso da acurácia da classificação, isto é, o número de classificações feitas corretamente no total de instâncias que foram classificadas.

$$fobj_i = erro_i = \frac{1}{acc_i}, \forall i = 1, \ldots, N \tag{5.1}$$

## 5.2 Representação dos parâmetros

Como visto nas Equações 2.2, 2.3 e 2.11, os vetores $\alpha$ (multiplicadores de Lagrange), $\beta$, $\gamma$ (relacionados à função núcleo Gaussiana modificada (DU; PENG; TERLAKY, 2009) como apresentado na Equação 2.9) e a constante $C$ são parâmetros que não dependem dos dados ($x$,$y$) e cujos valores deseja-se descobrir para a tarefa de classificação de dados. Por isso a representação dos parâmetros do SVM nos indivíduos é feita pelos parâmetros $0 \leq C \leq 1$ (das Equações 2.2 e 2.7), $0 \leq \sigma \leq 1$, $0 \leq \beta \leq 1$ e pelos multiplicadores de Lagrange $0 \leq \alpha \leq 1$. Os componentes $\sigma$ e $\beta$ do indíviduo são variáveis multidimensionais relacionadas ao núcleo gaussiano modificado definido pela Equação 2.11, como mostrado na Figura 5.1. A conexão dos dois algoritmos é feita como mostra a Figura 5.2, cada membro da população do DE é um vetor de parâmetros como mostrado na Figura 5.1. Durante a execução do algoritmo evolutivo, o resultado da avaliação da função objetivo (linha 20 do Algoritmo 3.1) é obtido após realizar a classificação dos dados utilizando como parâmetros do SVM os valores dos indivíduos do DE, como mostrado no Algoritmo 5.1.

| C | σ1 | σ2 | ... | σN | α1 | α2 | ... | αl | β1 | β2 | ... | βN |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

Figura 5.1: Representação dos parâmetros da SVM nos indivíduos do DE.

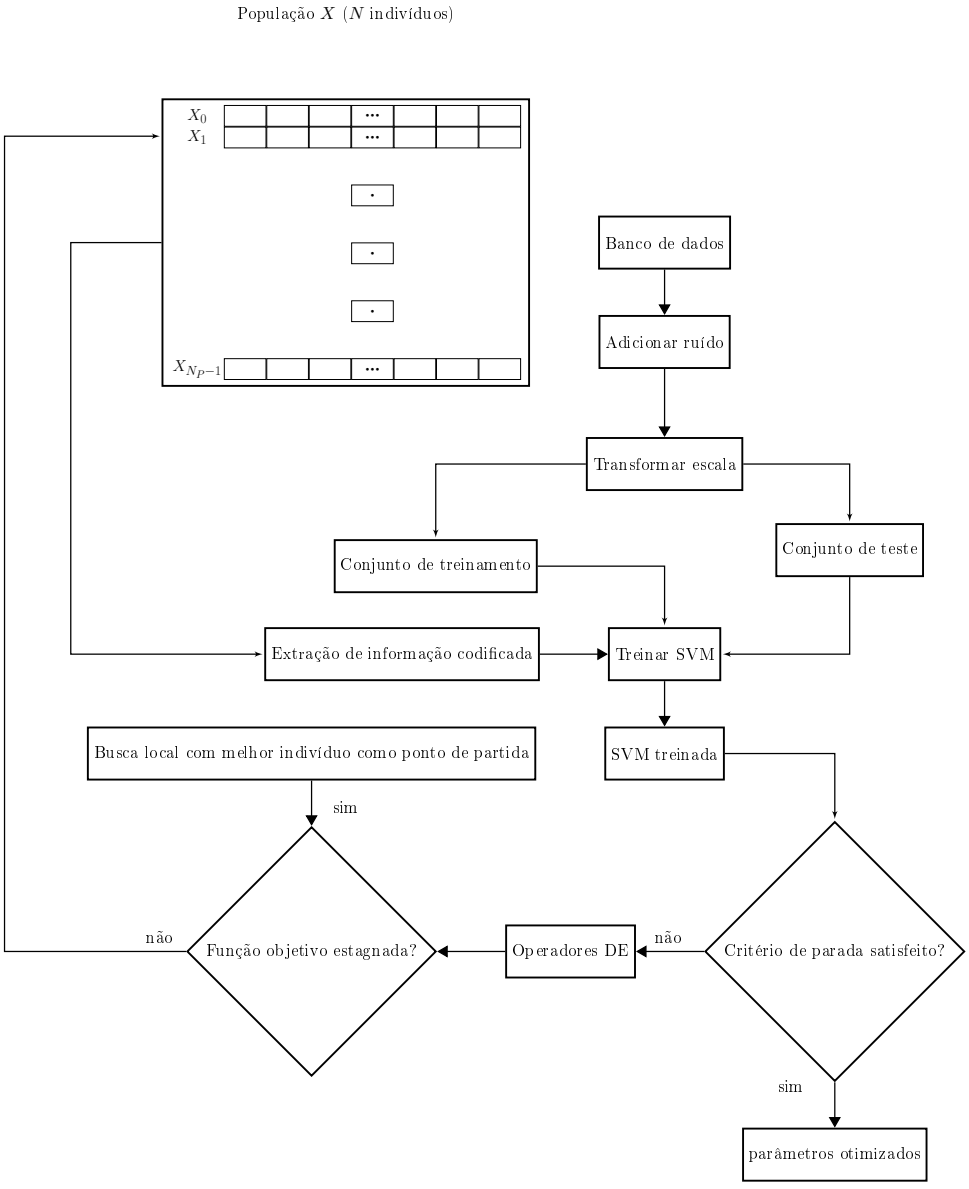



Figura 5.2: Conexão entre DE+SVM+TS.

## 5.3 O Algoritmo híbrido

O Algoritmo 5.2 descreve o processo de treinamento e classificação da SVM (GUNN, 1998) descrito no Capítulo 2, utilizado na avaliação da função objetivo do método híbrido (Algoritmo 5.1). Neste algoritmo, o problema de otimização descrito pela Equação 2.7 pode ser expresso em notação matricial $min\frac{1}{2}\alpha^T H\alpha + C^T\alpha$, onde os elementos $ij$ da matriz são formados por $y_i y_j K(x_i x_j)$. Durante a fase de treinamento, a constante $b$ é calculada a partir da média dos vetores suporte na margem. Utilizando o princípio que vetores suporte na margem têm $0 < \alpha < C$.

O resultado da classificação é obtido na fase de validação do algoritmo, em que cada predição $g_i$, rótulo de classificação do vetor $i$, é obtido pela multiplicação da linha $i$ da matriz $H$ pelo vetor $\alpha$: $g_i = H_i\alpha + b$.

De acordo com (DU; PENG; TERLAKY, 2009), após o treinamento da SVM com $N$ características, pode-se validar a SVM com menos características e obter resultados similares, porque um valor pequeno em um elemento de $\beta$ significa que a característica correspondente não contribui muito para a classificação, consequentemente ela pode ser removida do conjunto de dados sem afetar a acurácia da classificação. Este processo de eliminar características sem perder acurácia na classificação, chamado extração de características, é responsável por aumentar o desempenho do classificador (WU et al., 2010), pois a dimensão dos vetores de características diminui, influenciando diretamente nas operações do SVM sobre os vetores de características nas Equações 2.7 e 2.11. Os valores de $\beta$ foram configurados de tal forma que os índices que satisfazem $\beta_i \leq 0{,}5$, $i = 1,\ldots,N$, determinam que as características de mesmo índice podem ser removidas do conjunto de características como feito em (DU; PENG; TERLAKY, 2009).

```
1  input
2    N_p: tamanho da população
3    D: dimensão dos indivíduos
4    X^0: população inicial, composta por x_1^0,...,x_{N_p}^0
5    F_m: fator de mutação
6    C_r: taxa de recombinação
7  output
8    X^t: população otimizada após t iterações
9  begin
10   Inicializa população
11   do
12     for i=1:N_p
13       mutação
14       recombinação
15       //Modificação em relação ao Algoritmo 3.1
16       Extrair os parâmetros α_i,β_i,γ_i do elemento x_i e contruir um SVM_i com esses parâmetros
17       Atribuir a f_i a acurácia do classificador SVM_i obtida // Algoritmo 5.2
18       Extrair os parâmetros α_u,β_u,γ_u do elemento u_i e contruir um SVM_u com esses parâmetros
19       Atribuir a f_u a acurácia do classificador SVM_u // Algoritmo 5.2
20       if f_u ≤ f_i %seleção de acordo com a Equação 3.3
21         x_i^t = u_i^t
22       endif
23     endfor
24   until número de iterações ou critério de parada satisfeito
25 end
26 %
```

Algoritmo 5.1: Método híbrido.

```
1  input
2    X: vetores de características
3    Y: rótulos de classificação
4    α: multiplicadores de Lagrange
5    C: limite superior (caso não separável)
6  output
7    predictedY: vetor resultado da classificação
8  begin
9      //treinamento da SVM
10     //Construindo a matriz kernel com os vetores de treinamento
11     H = zeros(n,n);
12     for i=1:n
13       for j =1: n
14         H(i,j) = Y(i)*Y(j)*svkernel (ker,X(i,:),X(j,:));
15       end
16     end
17     obter vetor α de um algoritmo otimizador
18     //Calcular o número de vetores suporte
19     epsilon = 10e-6*(alpha);
20     svi = find(alpha>epsilon);
21     nsv = length(svi);
22     svii = find(alpha>epsilon & alpha<(C-epsilon));
23     if length(svii)>0
24       b0 = (1/length(svii))*sum(Y(svii)-H(svii,svi)*alpha(svi).*Y(svii));
25     else
26       Não há vetores suporte na margem - sem possibilidade de calcular bias
27     end
28     //validação
29     for i=1:m
30       for j=1:n
31         H(i,j) = trnY(j)*svkernel(ker,tstX(i,:),trnX(j,:));
32       end
33     end
34     predictedY = sign(H*alpha+bias);
35   end
36 end
```

Algoritmo 5.2: Treinamento e classificação do SVM.

## 5.4 Combinação dos algoritmos Busca Tabu e Nelder-Mead

Durante o processo evolutivo, o objetivo é a maximização da função objetivo. Quando o melhor valor da função objetivo fica estagnado por um número de iterações (configurado neste trabalho para $MAX_{\mathrm{e}} = 10$), um algoritmo de busca local é aplicado para intensificar a busca nas regiões próximas ao ponto de melhor solução encontrado até então, tendo como ponto inicial o melhor indivíduo ou o vetor local, com probabilidade de 50%. Este algoritmo é a combinação de Busca Tabu com o método Nelder-Mead (MASHINCHI; ORGUN; PEDRYCZ, 2011), em que NM é usado para melhorar o valor da função objetivo da solução atual ($C_S$) de TS, como mostrado na Figura 5.3. Para isso, nas linhas 4 e 5 do Algoritmo 4.2, o método Nelder-Mead gera os vizinhos do ponto atual ($C$), os avalia e retorna o melhor dentre eles.

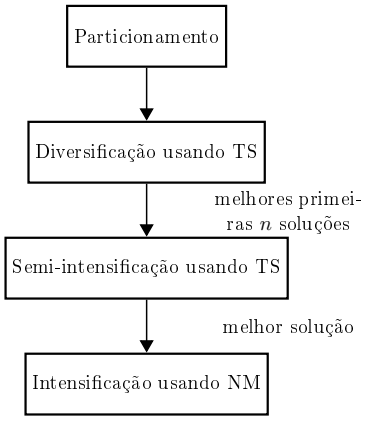



Figura 5.3: Ilustração da combinação dos algoritmos de busca local TS e NM.

# Capítulo 6

# Resultados de simulação

## 6.1 Estudo de caso 1

### 6.1.1 Problemas de testes convencionais em classificação

Para criar o conjunto de testes, três conjuntos de dados foram selecionados do repositório da UCI (FRANK; ASUNCION, 2010) e então foi adicionado ruído a eles. Conjuntos de dados que continham $m$ classes (onde $m > 2$) foram transformados em $m$ conjuntos de dados de 2 classes e seus resultados são mostrados para cada classe separadamente. Esta transformação consiste em atribuir um rótulo diferente às classes que não a atual. Este procedimento é realizado para cada uma das $m$ classes. Adicionalmente, os conjuntos de dados foram perturbados com diferentes níveis de ruído.

A geração de ruído pode ser classificada de modos diferentes (NETTLETON; ORRIOLS-PUIG; FORNELLS, 2010). Neste estudo duas categorias foram levadas em consideração, que são distribuição e localização. A distribuição escolhida para adicionar ruído aos dados foi a Gaussiana; e as localizações escolhidas para introduzir o ruído foram rótulo de saída, dados de treinamento, dados de validação, ou uma combinação dos dois. Outra possibilidade é introduzir variáveis ruidosas nos dados de treinamento

como feito em (LUUKKA; LAMPINEN, 2011). Aqui também foi introduzido nos dados de validação e em ambos.

A porcentagem de dados a serem perturbados foi configurada para 10% e 50%. Uma vez que a porcentagem $\eta$ de ruído foi determinada, $\eta L_s$ vetores de características são selecionados aleatoriamente de acordo com a distribuição uniforme para serem perturbados, onde $L_s$ é o tamanho do conjunto de dados para ser perturbado que pode ser o conjunto de treinamento, de validação ou ambos.

O método de seleção de vetores de características que serão perturbados, definida em (NETTLETON; ORRIOLS-PUIG; FORNELLS, 2010), é descrita a seguir. Para cada característica, $N_{\text{ruído}}$ números aleatórios são gerados pela distribuição normal, onde $N_{\text{ruído}}$ é o número de vetores do conjunto de dados multiplicado pela porcentagem de ruído a ser introduzido. Assim que os vetores a serem alterados são selecionados, a perturbação é introduzida da seguinte maneira. As variáveis perturbadas são substituídas por números aleatórios amostrados de uma distribuição normal com o valor original como média e um desvio padrão de valor unitário. Um exemplo do método de seleção e geração de ruído descrito é mostrado na Tabela 6.1.

Tabela 6.1: Primeiros 5 valores gerados para variáveis da iris. Legenda: VO (valor original), VA (valor aleatório), IM (índice do exemplo modificado).

| Valor | Var 1 | | Var 2 | | Var 3 | | Var 4 | |
|---|---|---|---|---|---|---|---|---|
| | VO, VA | IM | VO, VA | IM | VO, VA | IM | VO, VA | IM |
| 1 | 7,7, 6,1 | 15 | 3,8, 2,9 | 12 | 1,6, 1,5 | 20 | 0,2, 2,5 | 15 |
| 2 | 6,8, 5,0 | 19 | 3,0, 3,3 | 7 | 4,4, 2,3 | 14 | 1,8, 0,3 | 1 |
| 3 | 5,1, 5,4 | 6 | 2,3, 3,1 | 11 | 4,4, 6,0 | 6 | 1,2, 1,8 | 20 |
| 4 | 4,4, 5,6 | 4 | 2,6, 2,9 | 18 | 5,7, 5,6 | 13 | 2,0, 0,8 | 7 |
| 5 | 4,6, 5,2 | 17 | 3,2, 4,0 | 6 | 5,6, 1,6 | 4 | 1,0, 1,2 | 16 |

O número de linhas da Tabela 6.1 significam o número de vetores de características que serão perturbados com ruído. Este número é referente à porcentagem de ruído multiplicada pelo número de vetores de características da base de dados. Definido o número de vetores que serão perturbados, os índices deste vetores são gerados aleatori-

amente. Para este exemplo, uma base de dados fictícia com 4 características é utilizada e os vetores de índice de 1 a 5 serão perturbados. Para cada característica, os vetores escolhidos são perturbados. No exemplo, para a primeira característica: o vetor 1 com valor original 7,7 é perturbado pela distribuição normal com média 7,7 e variância 1 ($N(7{,}7;\ 1)$), resultando em 6,6. Os valores dos vetore 2, 3, 4 e 5 são trocados, respectivamente, por $N(6{,}8;\ 1) = 5{,}0$, $N(5{,}1;\ 1) = 5{,}4$, $N(4{,}4;\ 1) = 5{,}6$ e $N(4{,}6;\ 1) = 5{,}2$. O mesmo processo é repetido para cada característica.

Os tipos de ruído são separados em Gaussiano e variáveis ruidosas. A adição de ruído Gaussiano funciona como descrito anteriormente. Para a adição de variáveis ruidosas, depois da adição de novas variáveis aleatórias uniformes, o mesmo princípio do ruído Gaussiano é aplicado. Qualquer um dos dois pode ser adicionado aos dados de treinamento, validação ou ambos (treinamento/validação). Para facilidade de referência nós nomeamos os ruídos no treinamento como ruído A, na validação como ruído B, no treinamento/validação como ruído C e nos rótulos dos dados de treinamento como ruído D. Ruídos A, B, C e D foram estudados em (NETTLETON; ORRIOLS-PUIG; FORNELLS, 2010). Todos os tipos de ruídos são adicionados aos dados de entrada, exceto o ruído D, que atribui aos dados de saída (o vetor rótulo) um valor aleatório uniforme escolhido entre dois possíveis valores $\{-1{,}1\}$. Em (LUUKKA; LAMPINEN, 2011) o tipo de ruído chamado variáveis ruidosas adiciona variáveis aos dados de treinamento, que é chamado aqui de ruído E. O princípio deste ruído é adicionar características, colunas, nulas à base de dados. Em seguida, a seleção de vetores e a introdução de ruído seguem os mesmos passos definidos pelo ruído Gaussiano, mostrado no exemplo anterior. Os tipos de ruídos restantes seguem o mesmo princípio, mas adicionam ruído em diferentes partes: ruído F nos dados de validação e ruído G nos dados de treinamento/validação. O número de variáveis ruidosas adicionadas neste trabalho foi $2N_{\text{car}}$, ou seja, duas vezes o número de características do conjunto de dados a ser classificado (LUUKKA; LAMPINEN, 2011).

Para medir a eficácia de classificação, uma característica que deve ser respeitada para estimar a taxa de erro verdadeira diz respeito a distribuição aleatória dos exemplos entre os conjuntos utilizados para treinar e validar os classificadores (SOUZA, 2011). A validação cruzada representa uma classe entre os métodos de reamostragem. Neste método a amostra é dividida em $N$ partições (inglês: folds) mutualmente exclusivas. A cada experimento uma partição é utilizada para validar o classificador enquanto as outras são utilizadas para treinamento. A taxa de erro é a média das taxas de erros calculadas para o total de $N$ experimentos realizados, como ilustrado na Figura 6.1. Este tipo de validação cruzada, entre outros existentes, é chamado de validação cruzada com $N$-partições.

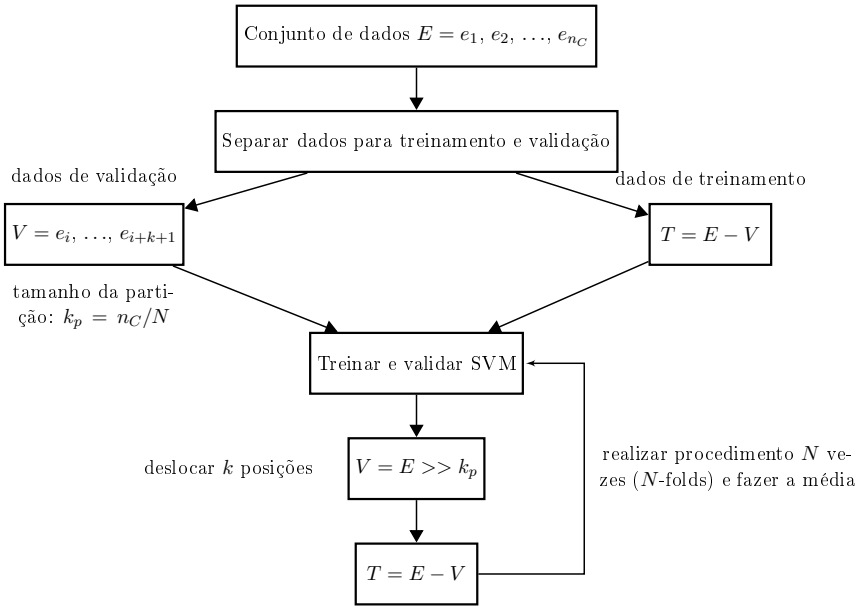



Figura 6.1: Validação cruzada.

O bootstrap foi introduzido como um método para calcular intervalos de confiança para parâmetros em circunstâncias em que outros métodos não podem ser aplicados (ZOUBIR; BOASHASH, 1998). Por exemplo, quando há poucos dados disponíveis, de modo que métodos de aproximação que utilizam grandes quantidades de amostras é inaplicável. Para calcular um estimador baseado em amostras dos dados, o experi-

mento deve ser repetido por um número suficiente de vezes para aproximar o parâmetro procurado pela distribuição dos dados. A escolha da distribuição utilizada para aproximar a distribuição geradora dos dados analisados pode incluir informações parciais da distribuição original dos dados, mas essa escolha não é única. Para aproximar uma distribuição normal de média $\mu$ e variância $\sigma^2$ desconhecidas, extrai-se amostras de tamanho $n_b$ com $\widehat{\mu}$ e $\widehat{\sigma}^2$ estimados do conjunto de dados $X$. O intervalo de confiança para $\mu$ pode ser calculado determinando-se a distribuição de $\widehat{\mu}$ (sobre repetidas amostragens de tamanho $n_b$ da distribuição desconhecida). O método bootstrap é descrito no Algoritmo 6.1 a seguir.

```
1  input
2    X: conjunto de dados
3    N_r: número de reamostragens
4  output
5    v̂: estimador da distribuição de X
6  begin
7    Conduzir experimento e obter amostras aleatórias X = X_1,...,X_{N_r} e calcular o estimador v̂ da
         amostra X.
8    Construir uma distribuição F̂ que dê peso 1/N_b a cada observação
9    Após a seleção de F̂, extrair x* = X*_1,...,X*_{N_r} chamado reamostragem bootstrap
10   Aproximar a distribuição de v̂ pela distribuição de v̂* derivada de X*
11 end
```

Algoritmo 6.1: Algoritmo bootstrap.

Ao conduzir o experimento, suponha que a amostra seja $X = \{-2{,}41,\ 4{,}86,\ 6{,}06,\ 9{,}11,\ 10{,}20,\ 12{,}81,\ 13{,}17,\ 14{,}10,\ 15{,}77,\ 15{,}79\}$ de tamanho $n_b = 10$, com $\mu = 9{,}946$ de todos os valores em $X$.

No passo 2, reamostragem, usando um gerador de números aleatórios, extrair uma amostra de mesmo tamanho de $X$, 10 valores, com repetição. Por exemplo $X_1^* = \{9{,}11,\ 9{,}11,\ 6{,}06,\ 13{,}17,\ 10{,}20,\ -2{,}41,\ 4{,}86,\ 12{,}81,\ -2{,}41,\ 4{,}86\}$, com média dos 10 valores

$\mu_1^* = 6{,}54$. Os passos 1 e 2 são repetidos uma grande quantidade de vezes para obter um total de $N_r$ estimativas bootstrap $\mu_1^*$, $\mu_2^*$, ..., $\mu_{N_b}^*$. Por exemplo, $N_r = 1000$.

Passo 4, aproximação da distribuição de $\widehat{\mu}$. Ordenar as estimativas de bootstrap para obter $\widehat{\mu}_1^* \leq \widehat{\mu}_2^* \leq \ldots \leq \widehat{\mu}_{1000}^*$. Suponha que os valores sejam 3,48, 3,39, 4,46,..., 8,86, 8,88, 8,89, ..., 10,07, 10,08,..., 14,46, 14,53, 14,66. Com média $\mu = 10$ e variância $\sigma^2/N_r = 25$.

Passo 5, intervalo de confiança. O intervalo de confiança bootstrap $(1-\alpha)100\%$ desejado é $(\widehat{\mu}_{q1}^*, \widehat{\mu}_{q2}^*)$, onde $q_1 = \lfloor N_r\alpha/2 \rfloor$ é a parte inteira de $q_1 = N_r\alpha/2$ e $q_2 = N_r - q_1 + 1$. Os índices $q_1$ e $q_2$ são mostrados na Figura 6.2. Para $\alpha = 0{,}05$ e $N_r = 1000$, temos $q_1 = 25$ e $q_2 = 976$ e o intervalo de confiança desejado é (6,27; 13,19) onde $q_1 = N_r\alpha/2$ é a parte inteira de $q_1 = N_r\alpha/2$ e $q_2 = N_r - q_1 + 1$. Para $\alpha = 0{,}05$ e $N_r = 1000$, temos $q_1 = 25$ e $q_2 = 976$ e o intervalo de confiança de 95% é (6,27; 13,19).

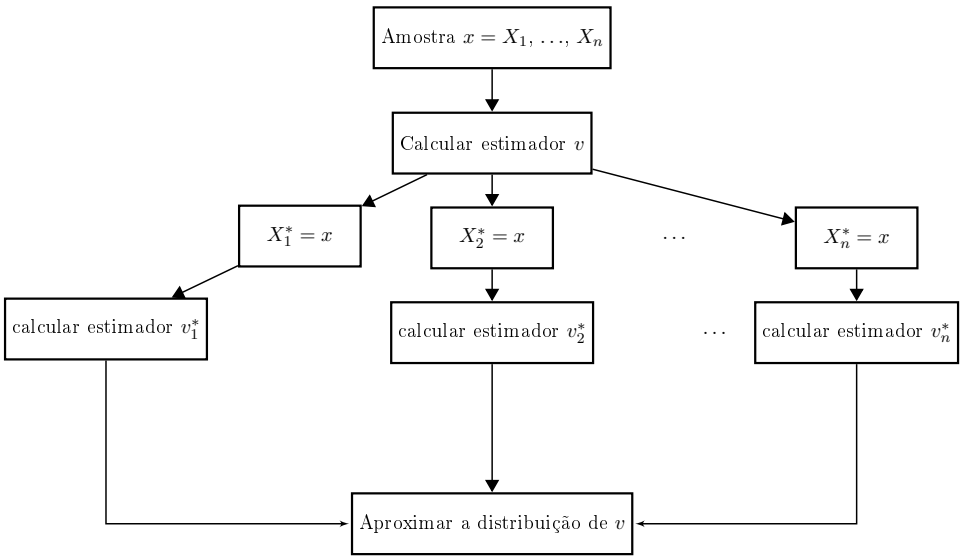



Figura 6.2: Método bootstrap.

Durante o processo evolutivo os indivíduos foram classificados baseados em valores de validação cruzada com $N$ partições (LUTS et al., 2010; SOUZA, 2011), com o número de partições configurado para três. Para validar os classificadores otimizados

uma nova validação foi feita com os conjuntos de dados selecionados aleatoriamente e dividido em 70% para treinamento e 30% para validação. Os dados pertencentes à cada classe foram divididos nas mesmas proporções, 70% de cada classe para treinamento e 30% para validação. Finalmente, para validar estatisticamente o método foi realizado um bootstrap (ZOUBIR; BOASHASH, 1998) com número de reamostragens $N_r = 100$ amostras para os classificadores treinados. Bootstrap é utilizado para verificar a acurácia do estimador do parâmetro. O método consiste em obter um estimador dentro de um intervalo de confiança para um parâmetro, por exemplo média ($\mu$) e desvio padrão ($\sigma$), de um conjunto de variáveis independentes $x = \{X_1, \ldots, X_{N_r}\}$ amostradas de uma distribuição desconhecida. A distribuição $\mu$ depende da distribuição dos $X_i$s, que é desconhecida. No caso em que $M$ é grande a distribuição de $\mu$ pode ser aproximada pela distribuição normal. O método bootstrap assume que $x = \{X_1, \ldots, X_{N_r}\}$ constitui a própria distribuição; de modo que reamostrando $x$ várias vezes e calculando $\mu$ para cada uma dessas reamostragens é obtida uma distribuição bootstrap de $\mu$ da qual um intervalo de confiança é derivado. Tal fato é garantido pelo teorema central do limite (ZOUBIR; BOASHASH, 1998), que diz que quando $N_r$ é grande a distribuição de $\widehat{\mu}$ pode ser aproximada pela distribuição normal.

### 6.1.2 Resultados e discussões

A efetividade do método híbrido proposto, SVM+DE+LS, foi avaliada realizando experimentos para três problemas de classificação do repositório da UCI (FRANK; ASUNCION, 2010), dados do coração (heart), dados do câncer de mama (breast-cancer) e os dados da iris. Esses três problemas são mostrados na Tabela 6.2, que contém o número de vetores de características e a quantidade de características para os três problemas de classificação. Os resultados estão resumidos na Tabela 6.3 para os dados do coração, na Tabela 6.4 para os dados do câncer de mama e na Tabela 6.5 para os dados da iris. Os valores são médias de resultados de validação cruzada obtidos

em 10 execuções do processo evolutivo, enquanto o valor objetivo usado para medir o desempenho dos classificadores no algoritmo DE foi o inverso da acurácia, como descrito no Capítulo 5.

As bibliotecas de funções de evolução diferencial, desenvolvidas em (SEGUNDO; KROHLING; COSME, 2011), e SVM, desenvolvidaas neste trabalho, utilizadas para executar os experimentos foram desenvolvidas em C++. Os experimentos foram executados em um ambiente multitarefas não dedicado, com processador Intel(R) Core(TM) i3 CPU 540 @ 3.07GHz, 4 MB de cache, 4 GB de RAM.

As informações contidas nas tabelas 6.3 a 6.5 representam: na primeira linha, a porcentagem de ruído introduzido, na segunda linha, os resultados para o algoritmo SVM+DE sem busca local. Na terceira linha os resultados para o algoritmo proposto com busca local, SVM+DE+LS. As colunas de 2 a 9, representam os conjuntos de experimentos separados por tipo de ruído introduzido aos dados. Os resultados estão dispostos iniciando pelos dados originais, sem ruído, seguido por ruídos A, B, C, D, E, F, G: respectivamente, ruído Gaussiano introduzido nos dados de treinamento, nos dados de validação, nos dados de treinamento e validação, nos rótulos de treinamento; variáveis ruidosas introduzidas nos dados de treinamento, nos dados de validação, nos dados de treinamento e validação.

Tabela 6.2: Bancos de dados utilizados.

| Nome do banco | nº de instâncias | nº de características | nº de classes |
|---|---|---|---|
| heart_scale (coração) | 270 | 13 | 2 |
| breast_cancer (câncer de mama) | 683 | 10 | 2 |
| iris_scale (iris) | 150 | 4 | 3 |

Na Tabela 6.3, na comparação dos resultados para SVM+DE e SVM+DE+LS, não foi percebida uma diferença significativa, não ultrapassando 1%. O resultado obtido de

Tabela 6.3: Valores de validação cruzada obtidos com SVM+DE e SVM+DE+LS para o problema do coração.

| algoritmo | sem ruído | ruído A | | ruído B | | ruído C | | ruído D | | ruído E | | ruído F | | ruído G | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 10% | 50% | 10% | 50% | 10% | 50% | 10% | 50% | 10% | 50% | 10% | 50% | 10% | 50% |
| SVM+DE | **83.8** | 82.0 | **79.7** | 81.6 | 81.2 | 81.9 | 76.4 | 82.9 | 83.0 | **83.9** | 81.0 | **84.0** | **82.5** | **84.2** | 82.6 |
| SVM+DE+LS | 83.2 | **83.7** | 78.9 | **82.9** | **81.9** | **82.6** | **76.7** | **86.1** | **87.3** | 83.0 | **84.6** | 82.7 | 80.5 | 83.2 | **84.6** |

Tabela 6.4: Valores de validação cruzada obtidos com SVM+DE e SVM+DE+LS para o problema do câncer de mama.

| algoritmo | sem ruído | ruído A | | ruído B | | ruído C | | ruído D | | ruído E | | ruído F | | ruído G | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 10% | 50% | 10% | 50% | 10% | 50% | 10% | 50% | 10% | 50% | 10% | 50% | 10% | 50% |
| SVM+DE | 96.7 | 94.6 | 93.2 | 95.4 | 93.0 | 88.3 | 90.1 | **96.8** | **96.3** | 96.7 | 90.0 | **97.1** | 96.2 | 94.8 | 96.2 |
| SVM+DE+LS | 96.7 | **95.8** | **93.6** | **95.7** | **93.3** | **94.8** | **91.1** | 95.7 | 95.4 | **96.8** | **95.7** | 95.9 | **96.3** | **96.7** | **96.6** |

Tabela 6.5: Valores de validação cruzada obtidos com SVM+DE e SVM+DE+LS para o problema da iris.

| algoritmo | sem ruído | ruído A | | ruído B | | ruído C | | ruído D | | ruído E | | ruído F | | ruído G | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 10% | 50% | 10% | 50% | 10% | 50% | 10% | 50% | 10% | 50% | 10% | 50% | 10% | 50% |
| SVM+DE | | | | | | | | | | | | | | | |
| iris0 | 99.7 | 99.7 | **94.0** | 98.3 | 96.2 | 96.8 | **92.4** | 99.7 | 99.7 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | **100.0** |
| iris1 | 91.6 | 84.9 | 76.2 | 83.2 | 82.0 | 74.3 | 67.2 | 83.6 | 85.5 | 82.8 | 84.3 | 81.9 | 91.3 | 84.7 | 89.6 |
| iris2 | **94.8** | 93.4 | 81.9 | 91.1 | 88.7 | 91.8 | 83.3 | 94.8 | 95.4 | 91.2 | **96.0** | **95.6** | 94.4 | 93.4 | 95.5 |
| SVM+DE+LS | | | | | | | | | | | | | | | |
| iris0 | **100.0** | 99.7 | 93.0 | **99.7** | **98.3** | **99.4** | 88.1 | 99.7 | **100.0** | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 99.7 |
| iris1 | **94.4** | **93.3** | **79.6** | **94.3** | **89.7** | **93.9** | **74.0** | **94.9** | **94.8** | **94.8** | **90.0** | **95.2** | **93.3** | **94.6** | **93.4** |
| iris2 | 93.5 | **94.0** | **85.7** | **94.6** | **90.7** | **94.2** | **85.3** | **95.6** | 95.4 | **96.5** | 95.3 | 95.0 | **95.4** | **95.0** | **96.8** |

acordo com (LUUKKA; LAMPINEN, 2011) para o problema de classificação da base de dados do coração (problema do coração, para simplificar) sem ruído a acurácia de classificação foi de 83,21, enquanto neste trabalho foi obtido 83,8. A função objetivo utilizada pelo autor citado, para minimização, foi o número de classificações incorretas no conjunto de dados de treino. Enquanto neste trabalho a função a ser minimizada é o valor de validação cruzada com 3 partições. Na Tabela 6.3 é feita uma comparação entre os algoritmos SVM+DE e SVM+DE+LS, com os melhores resultados em negrito. Porém o único caso em que há diferença evidente é para o ruído D, enquanto para os outros ruídos a diferença é mínima e ora o SVM+DE é melhor, ora SVM+DE+LS é melhor. A Tabela 6.3 mostra que, apesar da introdução do ruído, a acurácia do

classificador aumentou em alguns casos. Uma possível razão pode ser devido ao ruído contribuir para aumentar a distância entre as amostras e as margens do hiperplano separador, tornando a tarefa de classificação mais precisa. A introdução de 10% de ruído aumentou ou manteve a acurácia para todos os tipos de ruído. A introdução de 50% de ruído diminuiu o desempenho para todos os tipos de ruído exceto para os ruídos D e G. Em (LUUKKA; LAMPINEN, 2011), quando 20 variáveis ruidosas foram adicionadas com 100% de probabilidade, a acurácia média foi 80,99, enquanto nós obtivemos 86.

A comparação dos algoritmos sem e com busca local na Tabela 6.4 mostra que, com exceção do ruído D, a busca local introduziu uma pequena melhora, embora não significativa. A Tabela 6.4 mostra os resultados para o problema de classificação da base de dados do câncer de mama (problema do câncer de mama). Todos os tipos de ruído diminuiram ou mantiveram os níveis de acurácia, os tipos de ruído que tiveram a maior perda de acurácia com 10% e 50% foram ruídos A e C, ambos afetando os dados de treinamento. De acordo com (NETTLETON; ORRIOLS-PUIG; FORNELLS, 2010), ruído no treinamento tem grande impacto na acurácia dos classificadores.

Para o problema de classificação da base de dados da iris (problema da iris), como discutido na Seção 6.1, o problema de decisão original de 3 classes foi transformado em 3 problemas de decisão: iris classe 0 (irisc0), iris classe 1 (irisc1) e iris classe 2 (irisc2). Neste problema a comparação favorece o algoritmo SVM+DE+LS em relação ao SVM+DE para todos os tipos de ruídos, perdendo apenas na comparação de uma das 3 classes em cada um dos tipos de ruído, sendo que a diferença foi maior nos ruídos B e principalmente C, apresentando uma diferença de 20% para o ruído C com 10% de ruído no subproblema iris1. Como mostrado na Tabela 6.5, a introdução de 10% de ruído não afetou a acurácia em nenhum dos tipos de ruídos. A introdução de 50% de ruído teve um grande impacto nos ruídos A, B e C, deteriorando o desempenho,

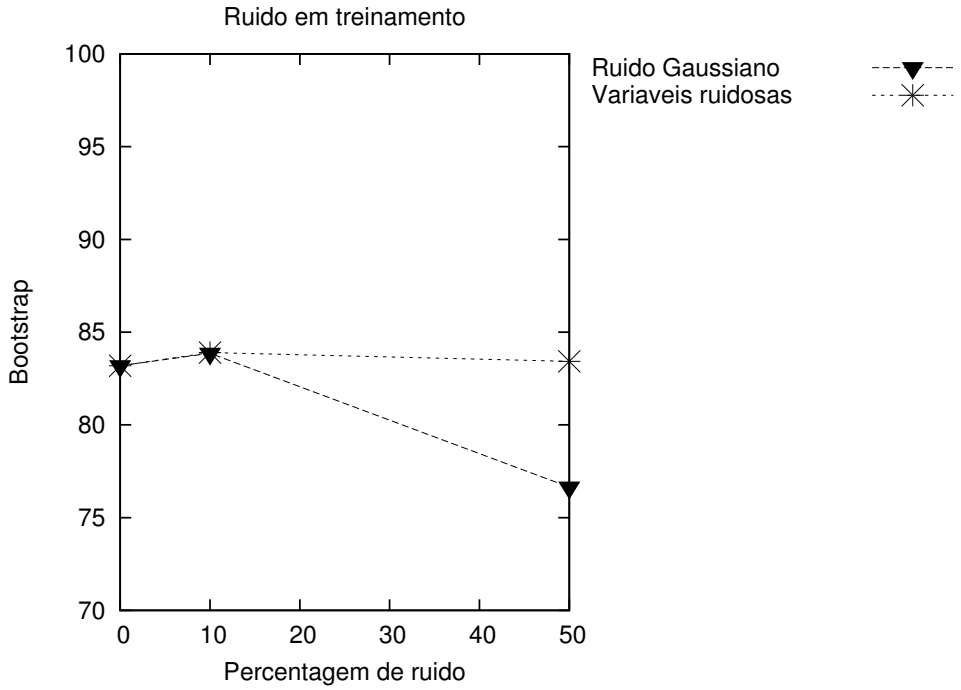
enquanto os outros mantiveram o desempenho. A maior deterioração de desempenho ocorreu com o ruído C com 50% de nível de ruído.

Além disso, os classificadores treinados foram submetidos a um método bootstrap para mostrar estatisticamente que o método alcança bons resultados apesar da aleatoriedade. Os pontos plotados são valores resultantes obtidos fazendo a média dos valores de bootstrap de 10 execuções para estimar a acurácia sem nenhuma influência que possa ser introduzida pela aleatoriedade da seleção de dados de ambos treinamento e validação. Nas Figuras 6.3 a 6.7 os valores de bootstrap obtidos para valores crescentes de ruído são mostrados para cada tipo de ruído introduzido, para os problemas do coração, câncer de mama e irisc0, irisc1, irisc2, respectivamente. Como descrito na Seção 6.1.1, valores altos de bootstrap significam que os resultados são robustos, apesar do fator aleatório envolvendo a escolha dos conjuntos de dados de treinamento e validação.

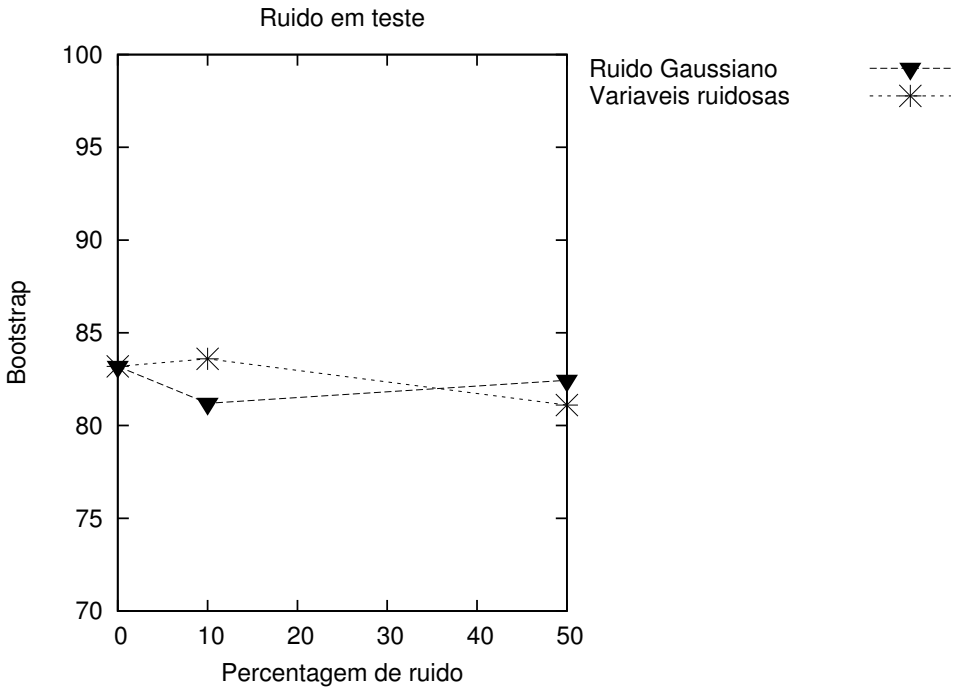
A análise do gráfico resultante mostra que, para o problema do coração, ruídos A e C causaram uma maior deterioração do desempenho quando o ruído aumentou, com acurácias atingindo valores abaixo de 80%. A análise do gráfico para o problema do câncer de mama mostra que o ruído C fez com que os classificadores apresentassem uma maior queda de desempenho quando a porcentagem desse tipo de ruído introduzida nos dados aumentou. Para o problema da íris, ruídos A e C tiveram um maior impacto negativo na classificação. Como mencionado em (NETTLETON; ORRIOLS-PUIG; FORNELLS, 2010), ruído nos dados de treinamento causa maior impacto na classificação, o que explica a maior influência causada por esses dois tipos de ruído.

Para mostrar o tempo computacional para cada banco de dados, o tempo gasto para executar os experimentos para todos os tipos de ruídos descritos anteriormente é mostrado na Tabela 6.6. Por envolver evolução populacional, cuja função de avaliação é realizada por validação cruzada, avaliação bootstrap e validação final, o tempo computacional é custoso. Portanto, as durações descritas são em minutos (m) para o tempo

médio de 10 execuções.

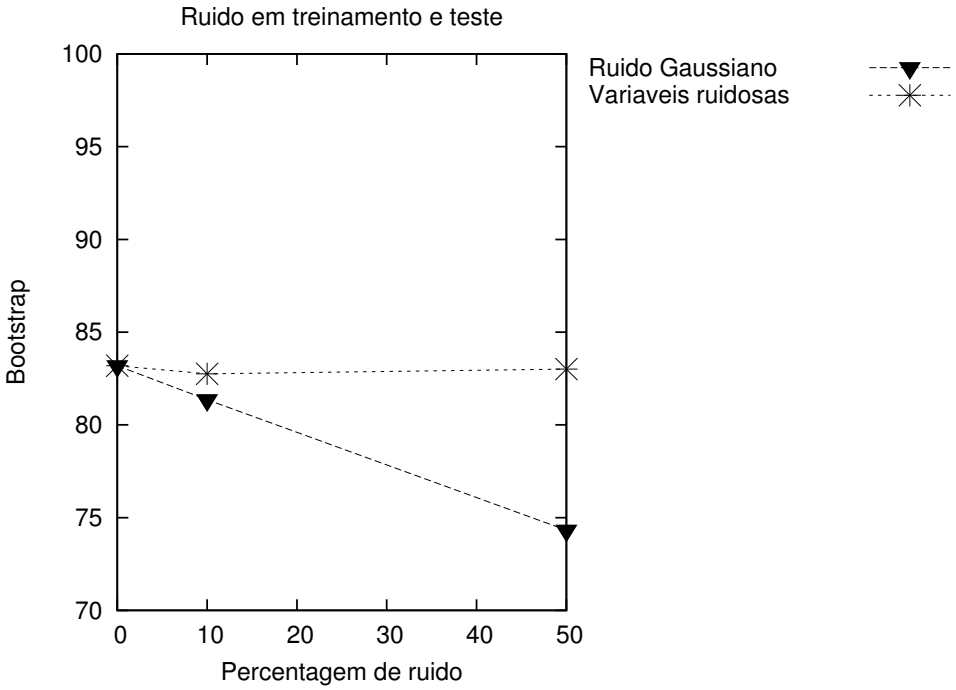



(a) Ruídos A (ruído no conjunto de treinamento) e E (variável ruidosa no conjunto de treinamento).

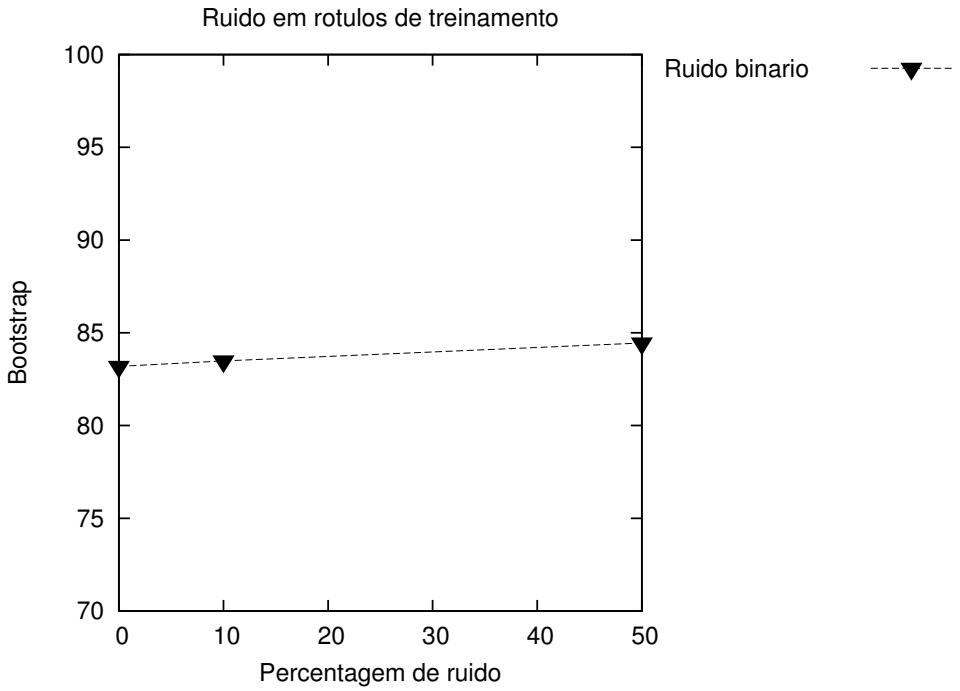



(b) Ruídos B (ruído no conjunto de validação) e F (variável ruidosa no conjunto de validação).

Figura 6.3: Valores de bootstrap da função objetivo com ruído crescente para o problema do coração.

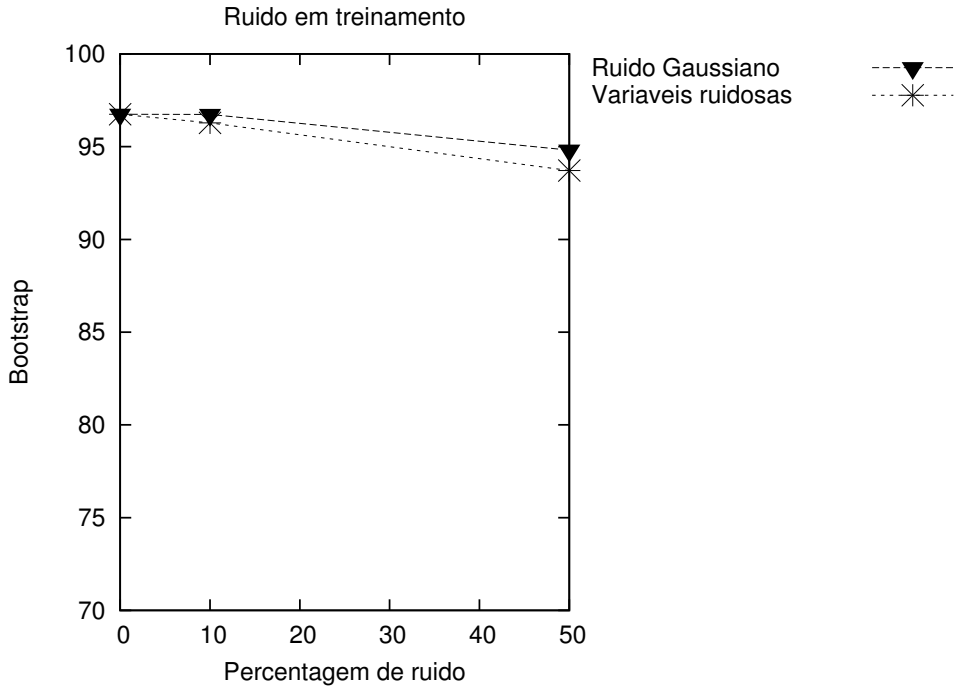



(c) Ruídos C (ruído nos conjuntos de treinamento/validação) e G (variáveis ruidosas nos conjuntos de treinamento/validação).

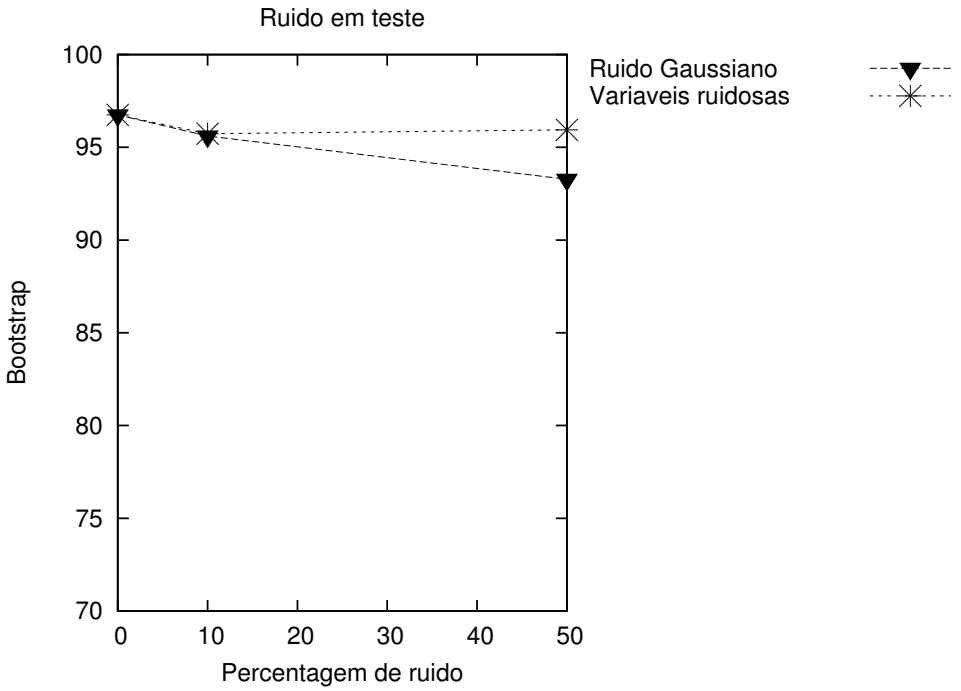



(d) Ruído D (ruído binário nos rótulos de treinamento).

Figura 6.3: Valores de bootstrap da função objetivo com ruído crescente para o problema do coração.

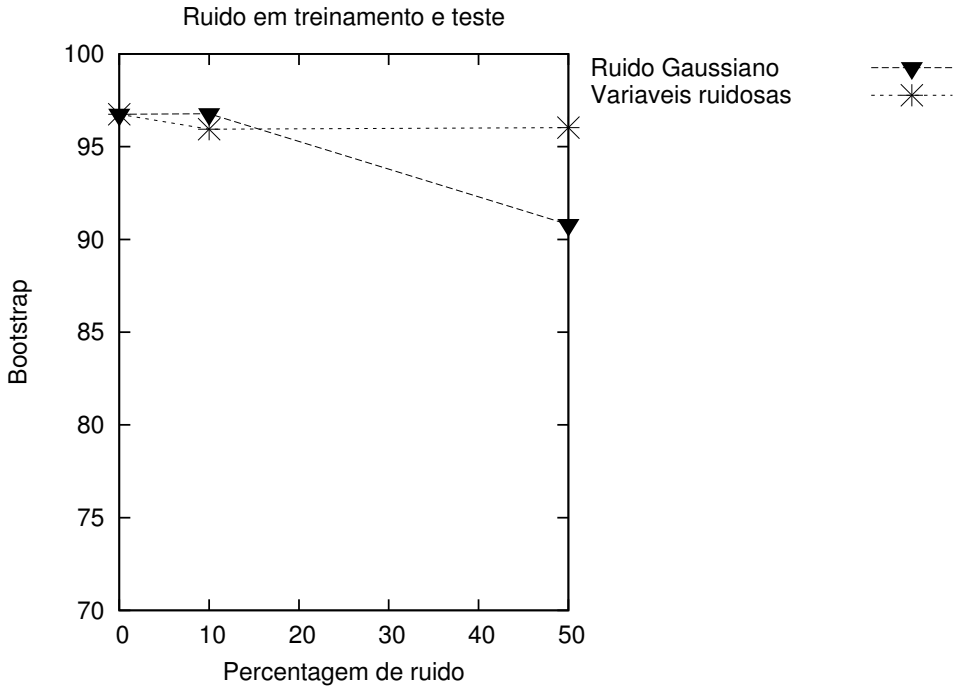



(a) Ruídos A (ruído no conjunto de treinamento) e E (variável ruidosa no conjunto de treinamento).

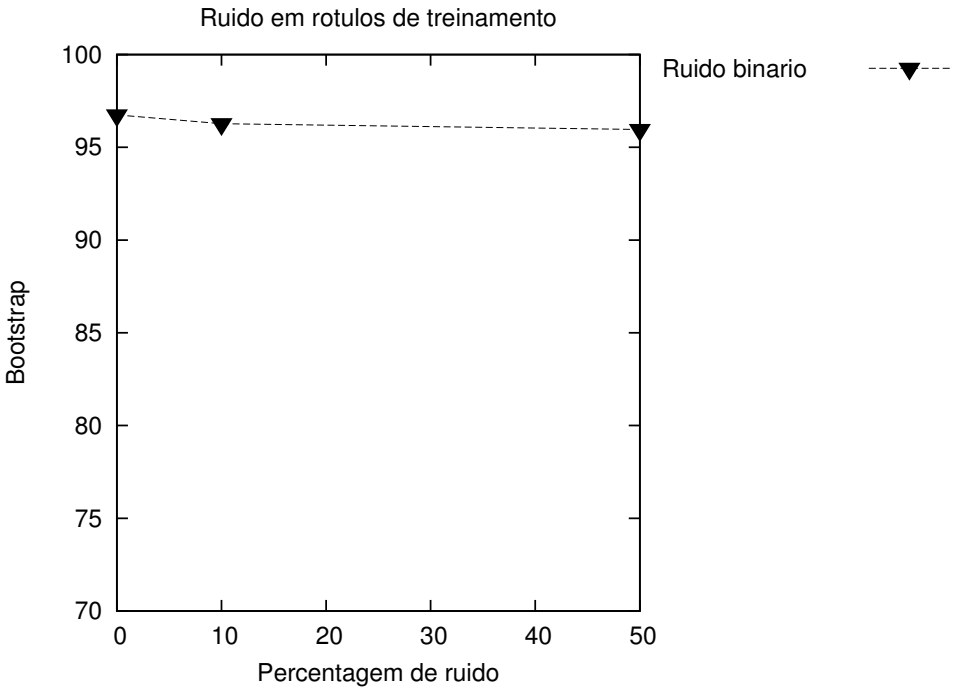



(b) Ruídos B (ruído no conjunto de validação) e F (variável ruidosa no conjunto de validação).

Figura 6.4: Valores de bootstrap da função objetivo com ruído crescente para o problema do câncer de mama.

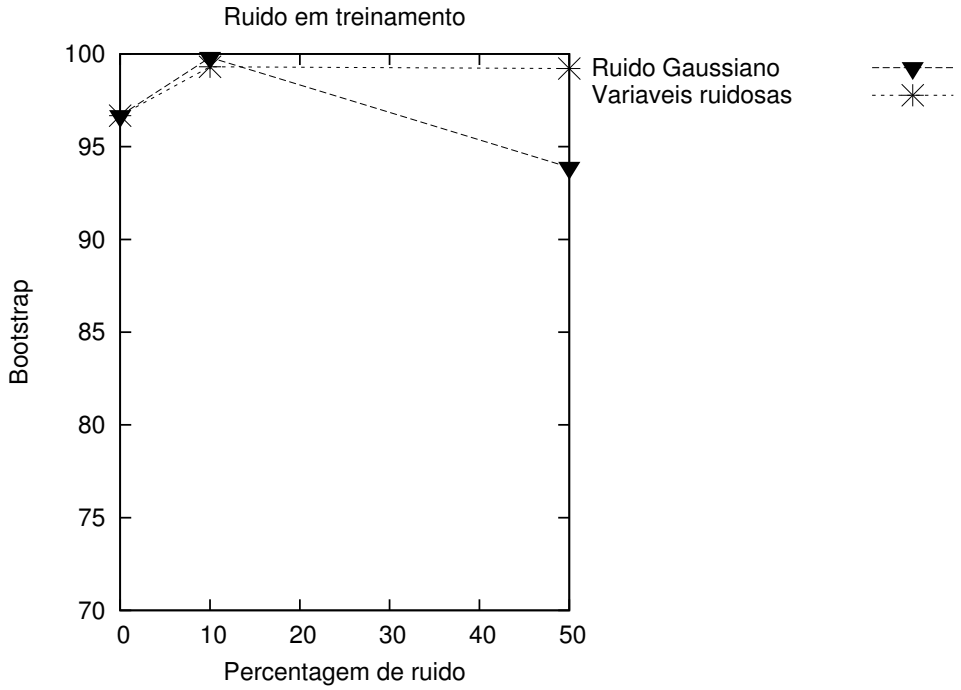



(c) Ruídos C (ruído nos conjuntos de treinamento/validação) e G (variáveis ruidosas nos conjuntos de treinamento/validação).

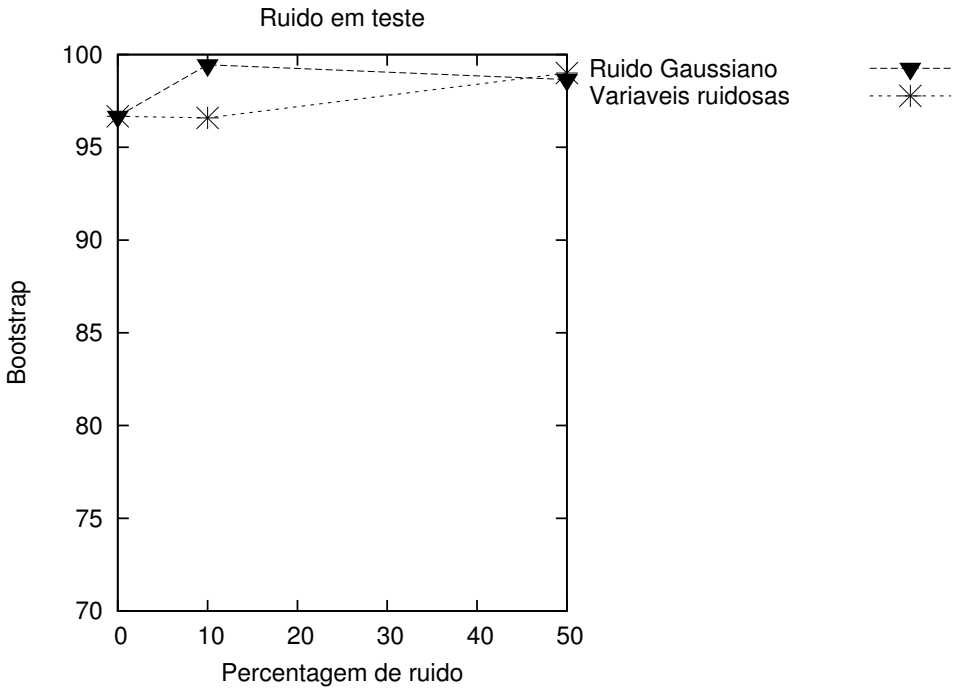



(d) Ruído D (ruído binário nos rótulos de treinamento).

Figura 6.4: Valores de bootstrap da função objetivo com ruído crescente para o problema do câncer de mama.

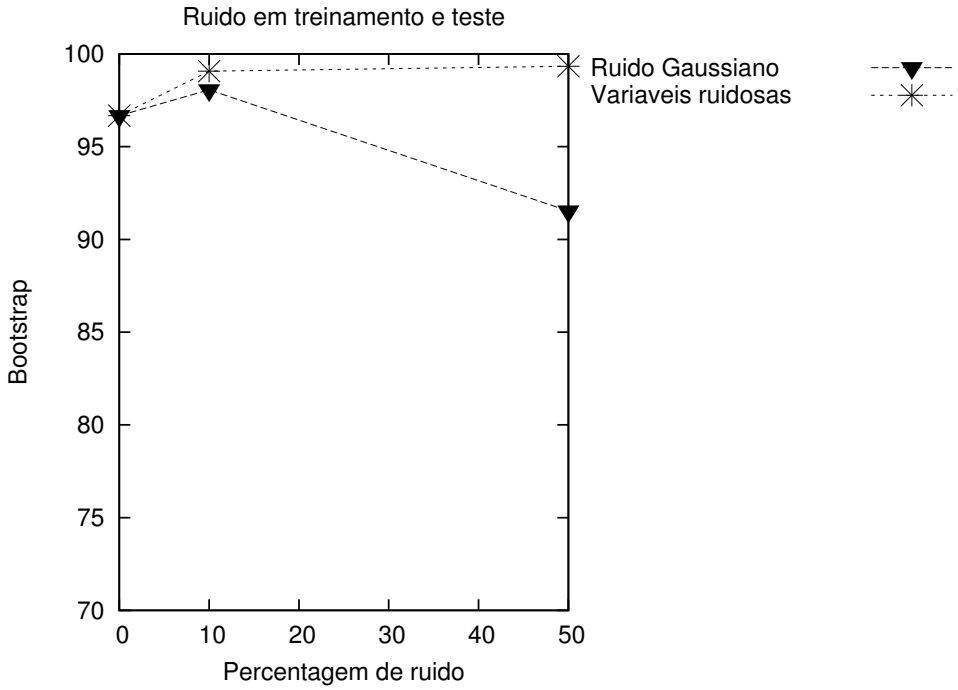



(a) Ruídos A (ruído no conjunto de treinamento) e E (variável ruidosa no conjunto de treinamento).

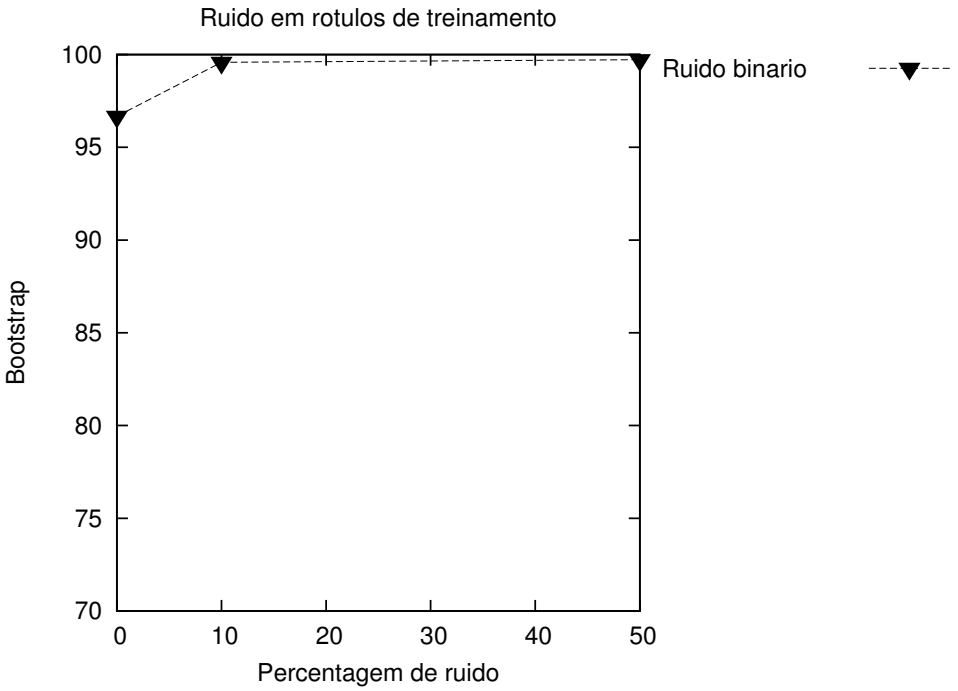



(b) Ruídos B (ruído no conjunto de validação) e F (variável ruidosa no conjunto de validação).

Figura 6.5: Valores de bootstrap da função objetivo com ruído crescente para o problema do iris classe0.

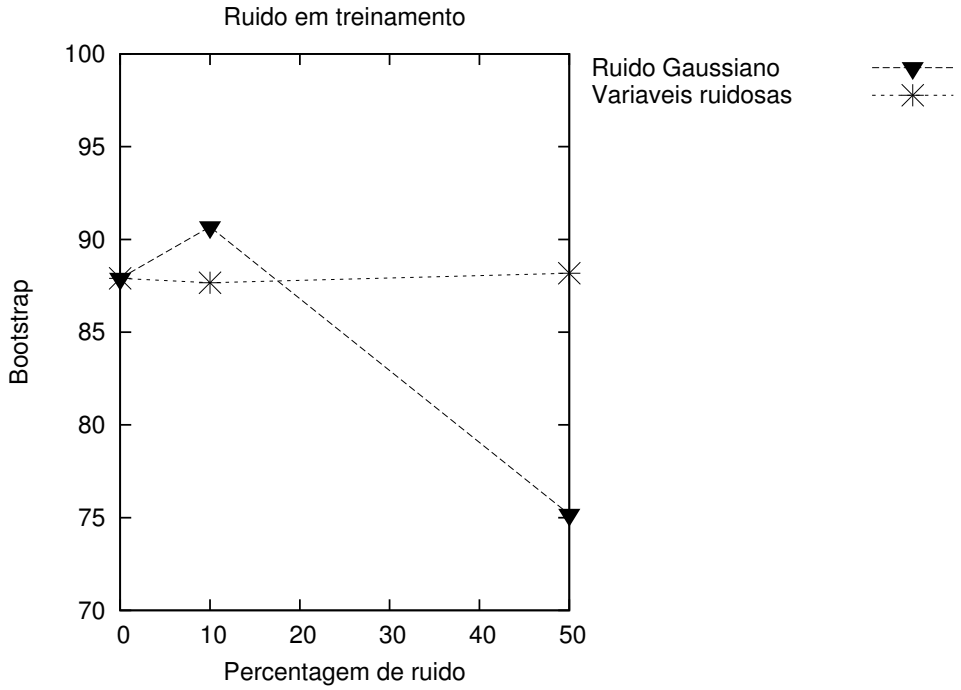



(c) Ruídos C (ruído nos conjuntos de treinamento/validação) e G (variáveis ruidosas nos conjuntos de treinamento/validação).

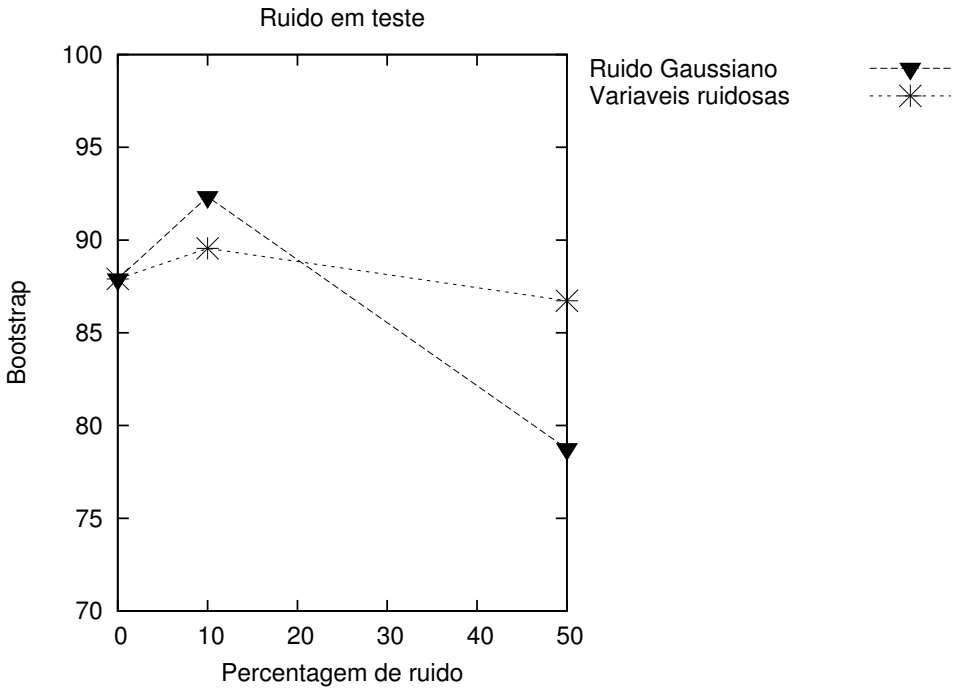



(d) Ruído D (ruído binário nos rótulos de treinamento).

Figura 6.5: Valores de bootstrap da função objetivo com ruído crescente para o problema do iris classe0.

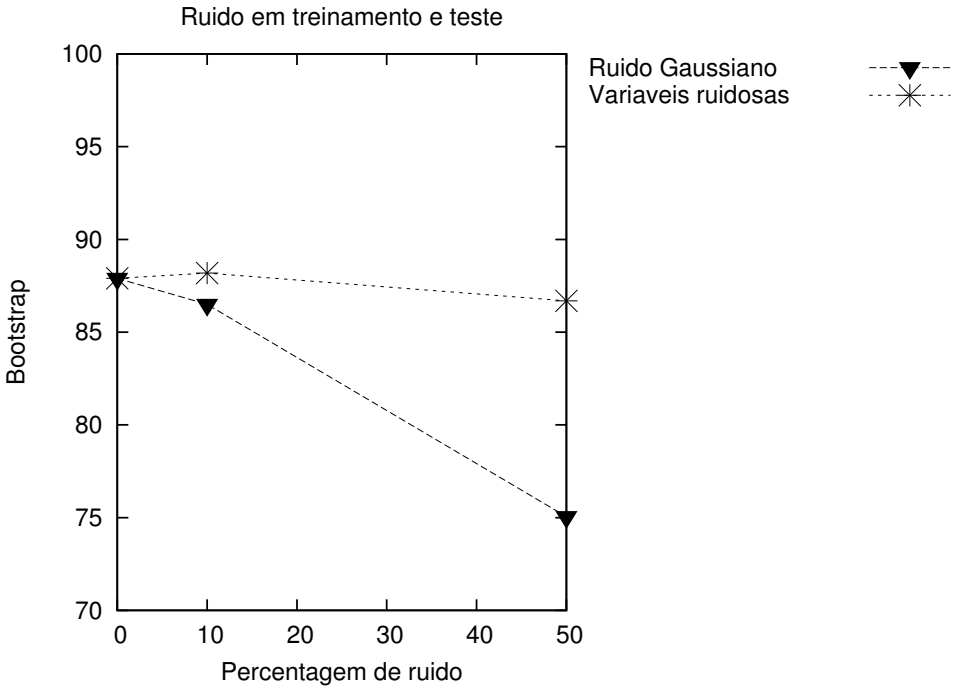



(a) Ruídos A (ruído no conjunto de treinamento) e E (variável ruidosa no conjunto de treinamento).

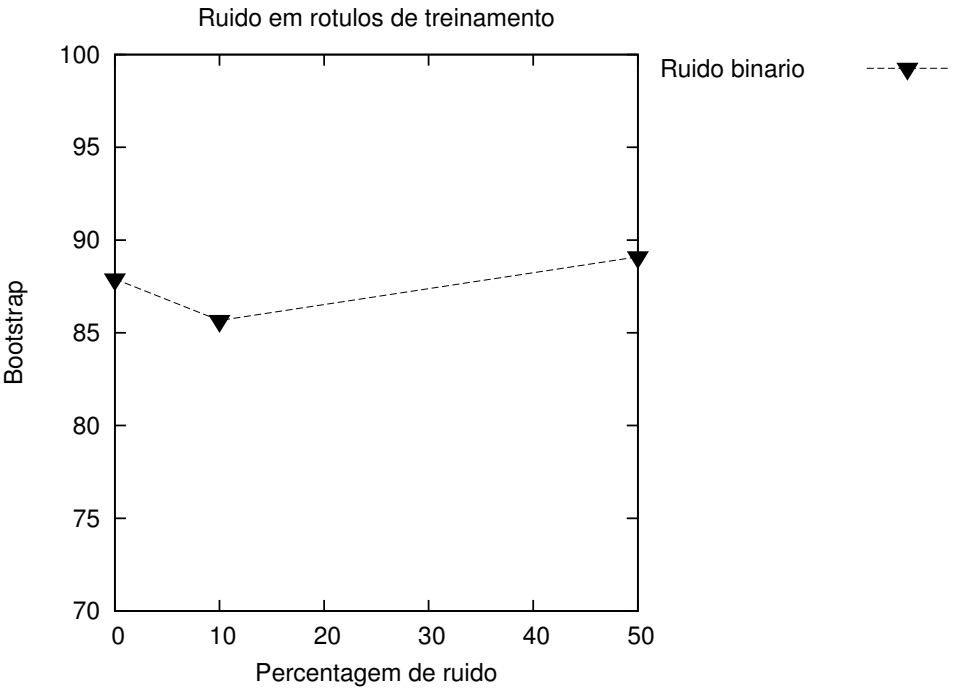



(b) Ruídos B (ruído no conjunto de validação) e F (variável ruidosa no conjunto de validação).

Figura 6.6: Valores de bootstrap da função objetivo com ruído crescente para o problema do iris classe1.

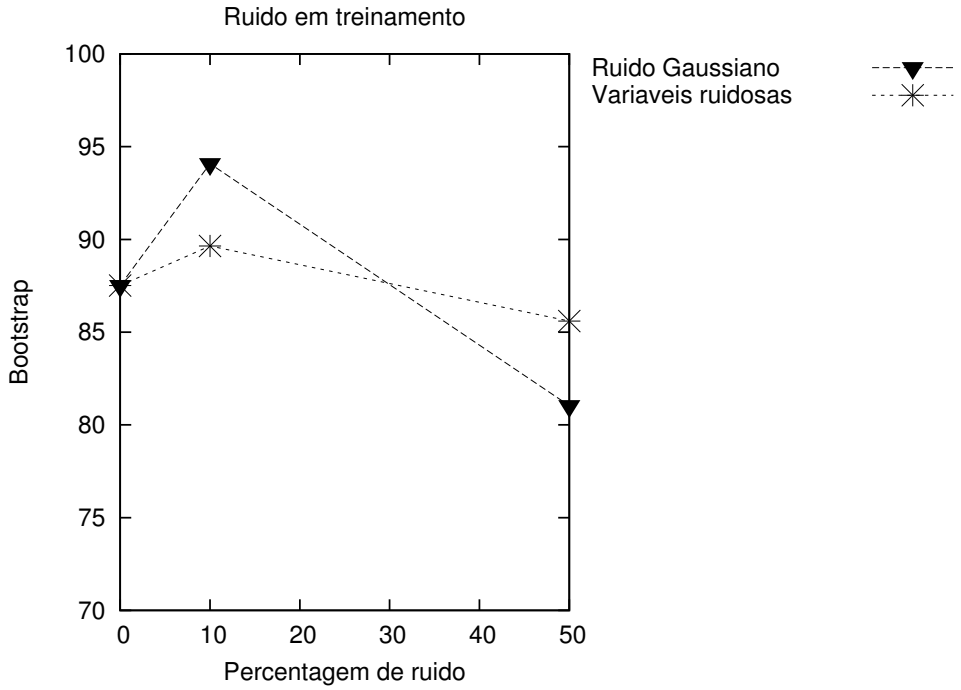



(c) Ruídos C (ruído nos conjuntos de treinamento/validação) e G (variáveis ruidosas nos conjuntos de treinamento/validação).

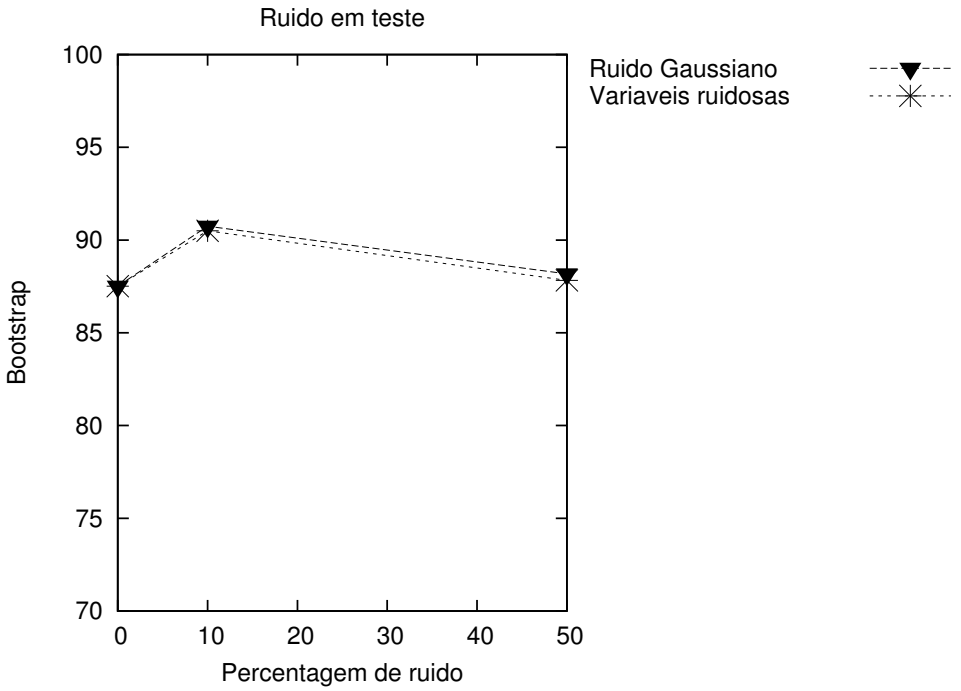



(d) Ruído D (ruído binário nos rótulos de treinamento).

Figura 6.6: Valores de bootstrap da função objetivo com ruído crescente para o problema do iris classe1.

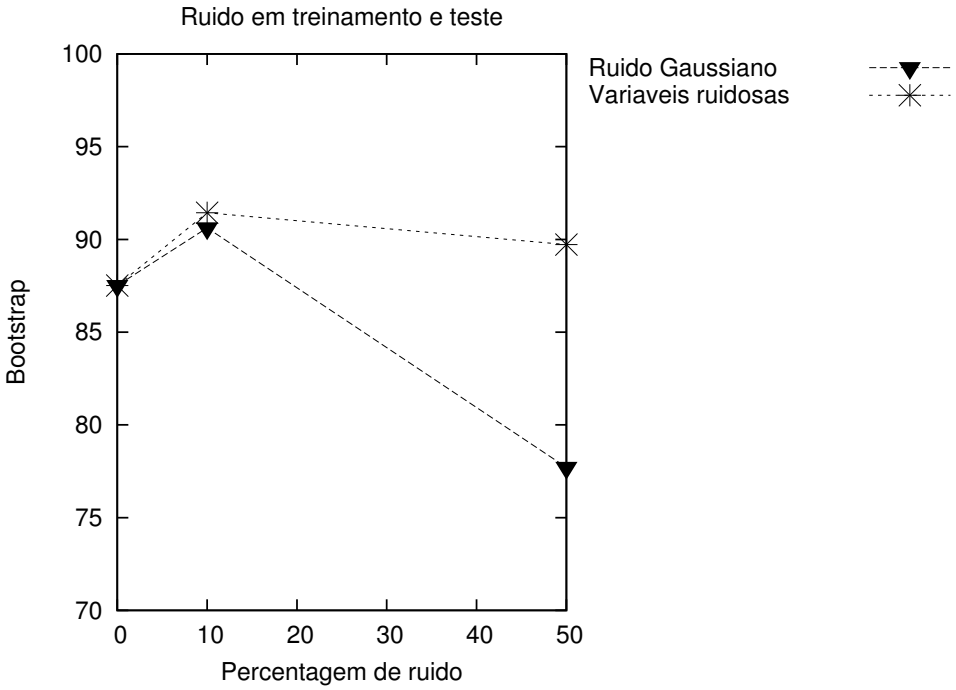



(a) Ruídos A (ruído no conjunto de treinamento) e E (variável ruidosa no conjunto de treinamento).

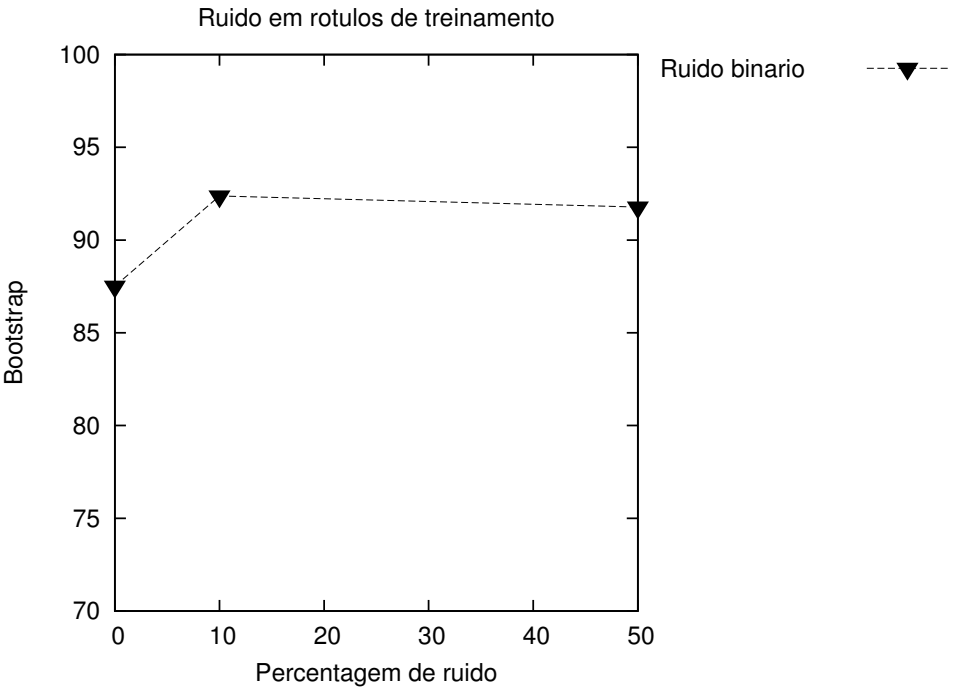



(b) Ruídos B (ruído no conjunto de validação) e F (variável ruidosa no conjunto de validação).

Figura 6.7: Valores de bootstrap da função objetivo com ruído crescente para o problema do iris classe2.




(c) Ruídos C (ruído nos conjuntos de treinamento/validação) e G (variáveis ruidosas nos conjuntos de treinamento/validação).




(d) Ruído D (ruído binário nos rótulos de treinamento).

Figura 6.7: Valores de bootstrap da função objetivo com ruído crescente para o problema do iris classe2.

Tabela 6.6: Duração dos experimentos.

| Nome do banco | duração(m) |
|---|---|
| Algoritmo SVM+DE | |
| heart_scale | 4597 |
| breast_cancer_scale | 19680 |
| iris_scale | 1764 |
| Algoritmo SVM+DE+LS | |
| heart_scale | 1699735,1 |
| breast_cancer_scale | [1]1865916 |
| iris_scale | 13232 |

## 6.2 Estudo de caso 2

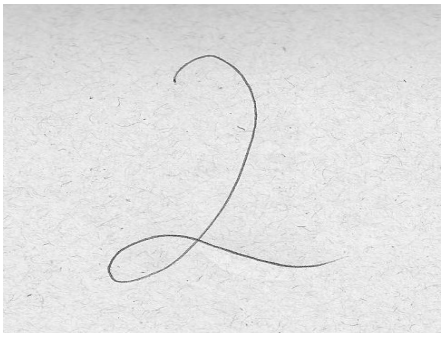
### 6.2.1 Problema de reconhecimento de dígitos escritos à mão

Um quarto problema de classificação foi analisado neste trabalho, a classificação de números escritos à mão. O banco de dados para esse problema foi gerado por coleta de dados. Para isso, foi solicitado que 3 pessoas escrevessem 50 números cada em papéis separados, 25 números "2" e 25 números "5". As imagens foram digitalizadas e preparadas - as bordas foram retiradas - para a extração de características. Em seguida as imagens foram transformadas para escala de cinza, eliminando cor e saturação, mas mantendo o valor da luminância no modelo Cor, Saturação e Luminosidade - ou Intensidade - (inglês: Hue Saturation and Intensity, HSI), para então ser normalizada para a dimensão 64x64 pixels. A imagem em escala de cinza por sua vez tem na sua representação matricial números de luminosidade. Então, utilizando uma constante limite de luminosidade $L_{lum}$ essa matriz de cinza ($M_c$) foi transformada em matriz binária ($M_b$), tal que os elementos de $M_c$ são formados conforme a Equação 6.1.

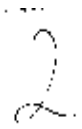
[1]Média de somente 5 experimentos realizados em um total de 10, devido ao alto custo computacional deste banco de dados (ver Tabela 6.2)

$$M_{b_{ij}} = \begin{cases} 1, & \text{se } M_{c_{ij}} \geq L_{lum} \\ 0, & \text{caso contrário} \end{cases} \tag{6.1}$$

A imagen mostrada na Figura 6.8 exemplifica esses passos até a geração da imagem binária, cuja forma matricial é utilizada para o processo de extração de características. A extração ocorre da seguinte forma: a matriz binária é dividida em zonas $Z_i$, com $i = 1, \ldots, n_Z$, $n_Z = 4$ , inferior esquerda, inferior direita, superior esquerda e superior direita. A extração de características é feita usando o método centróide da imagem e centróide da zona (ICZ-ZCZ, inglês: Image Centroid and Zone Centroid) baseado em métricas de distância (RAJASHEKARARADHYA, 2008), de modo que o número de características geradas é igual a $2n_Z$, onde $n_Z$ é o número de zonas. A descrição do ICZ-ZCZ é apresentada no Algoritmo 6.2. Foi feita a opção por gerar o banco de dados para esse problema através de coleta de dados e por não utilizar os bancos de dados disponíveis (FRANK; ASUNCION, 2010), para que utilizando o algoritmo ICZ-ZCZ para extração de características da imagem, com valores diferentes de $n_Z$, fosse possível analisar o impacto da introdução de ruído no banco de dados com diferentes números de características.



(a)



(b)

Figura 6.8: Transformação de imagem em matriz binária.

```
1  input
2    M_b: matriz binária da imagem
3    n_Z: número de zonas para se dividir a imagem
4  output
5    características extraídas: 2n_Z características
6  begin
7    Calcule o centróide da imagem
8    Divida a imagem em n_Z zonas iguais
9    Calcule a distância entre o centróide da imagem e cada pixel da zona
10   Repita a linha 9 para cada pixel na zona
11   Calcule a distância média entre estes pontos
12   Calcule o centróide da zona
13   Calcule a distância entre o centróide da zona e cada pixel da zona
14   Repita a linha 13 para cada pixel da zona
15   Calcule a distância média entre estes pontos
16   Repita as linhas 9−15 sequencialmente para toda a zona
17   Ao final 2n características serão obtidas para classificação
18 end
```

Algoritmo 6.2: Algoritmo ICZ-ZCZ.

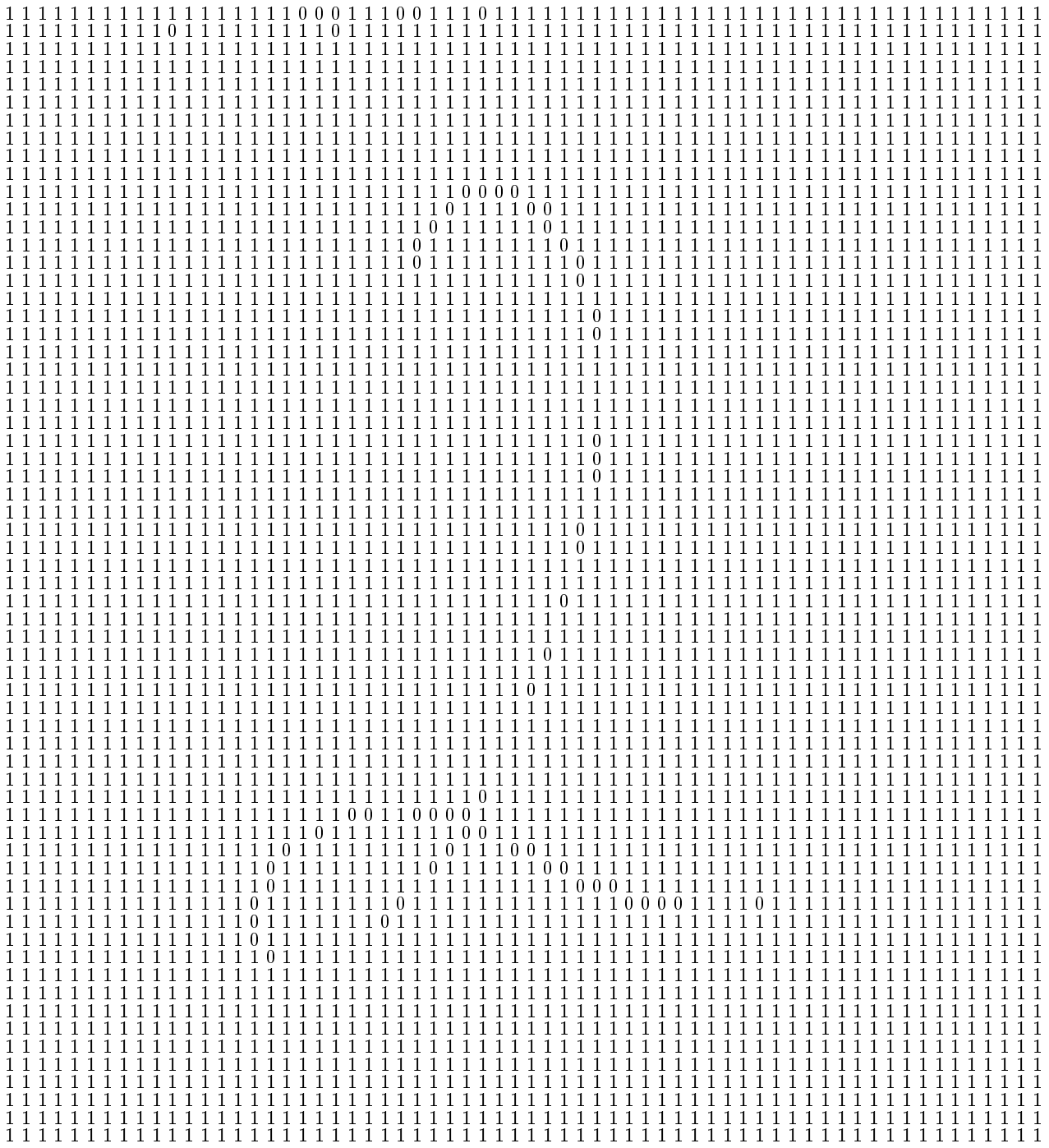


Figura 6.9: Matriz binária $M_b$ resultante da transformação.

### 6.2.2 Resultados e discussões

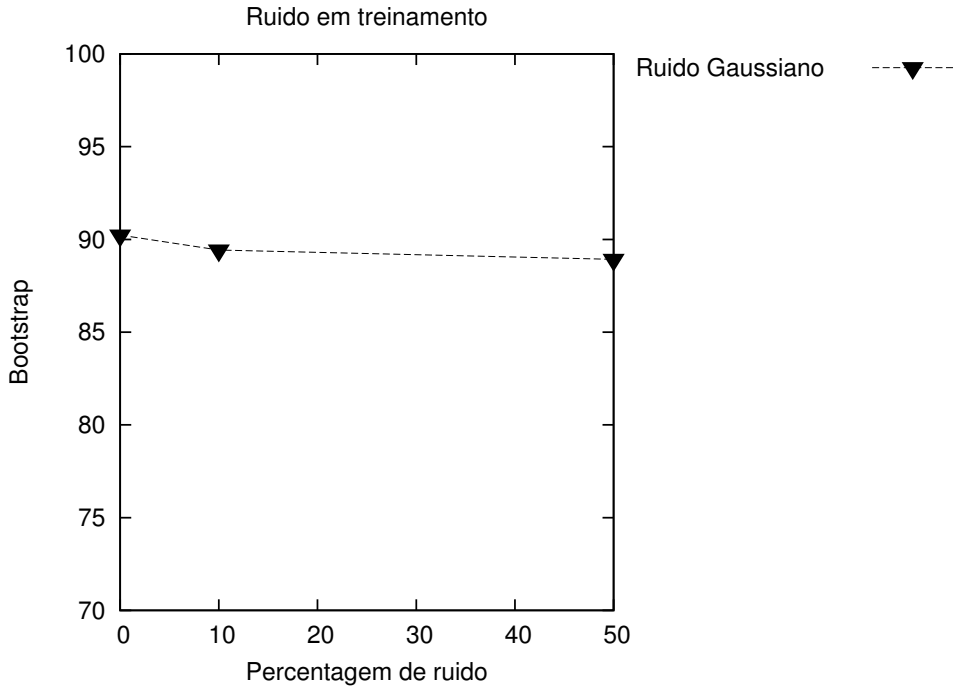
O banco de dados foi gerado após repetido o processo de extração de características para todas as 150 imagens. Este problema foi submetido aos tipos de ruído A, B, C e D, descritos na Seção 6.1, porém o ruído não é introduzido da mesma forma que nos exemplos anteriores. O método de seleção de quais exemplos são alterados é o mesmo, mas o ruído é introduzido às imagens e não ao aos vetores de características dos bancos de dados, como feito em (YAZDI; EFFATI; SABERI, 2007). A introdução de ruído às imagens, por sua vez, ocorre de maneira semelhante à descrita na Seção 6.1, assumindo que os bancos de dados dos exemplos anteriores são matrizes de características e que as imagens do problema atual estão em forma de matriz binária conforme a Figura 6.9. Portanto, para cada característica $x_j$, após selecionados os exemplos a serem alterados, esses terão a característica $j$ alterada com probabilidade 0,5, ou seja, se o valor atual for 0 é atribuído 1 e vice-versa. Os resultados obtidos são mostrados na Tabela 6.7.

Tabela 6.7: Valores de validação cruzada obtidos com SVM+DE e SVM+DE+LS para o problema reconhecimento de dígitos escritos à mão.

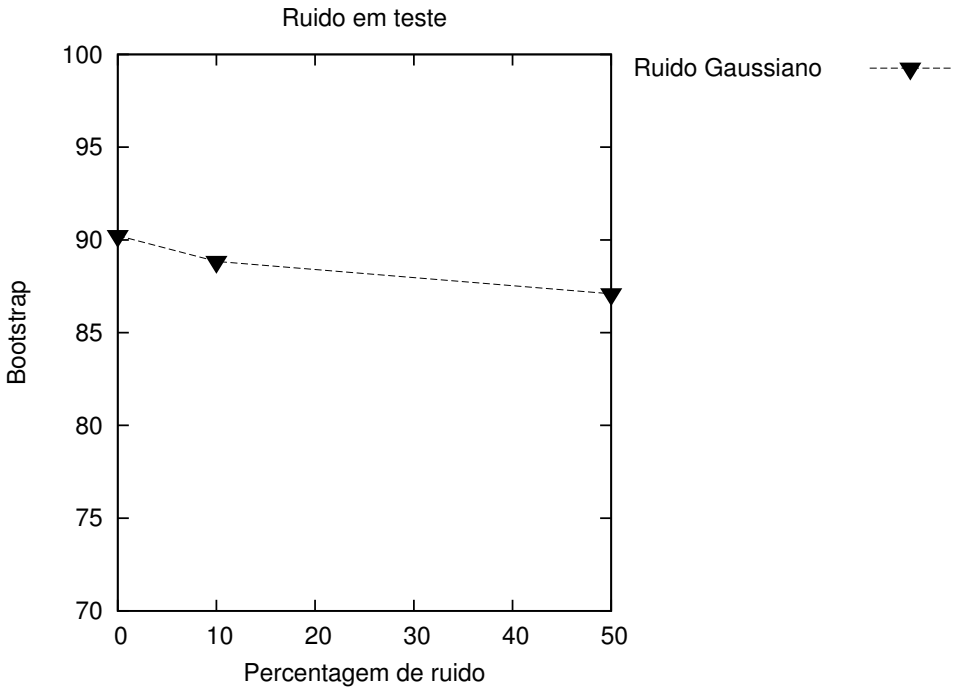
| no. carac | algoritmo | sem ruido | tipo A | | tipo B | | tipo C | | tipo D | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 10% | 50% | 10% | 50% | 10% | 50% | 10% | 50% |
| 2 | SVM+DE | 61.8 | 63.6 | 62.6 | 62.8 | 63.1 | 53.7 | 70.4 | 58.8 | 64.9 |
| | SVM+DE+LS | **89.8** | **87.2** | **87.9** | **87.0** | **88.7** | **91.0** | **88.9** | **90.3** | **89.2** |
| 4 | SVM+DE | 80.2 | 77.8 | 77.3 | 66.5 | 82.5 | 73.8 | 73.2 | 76.4 | 79.2 |
| | SVM+DE+LS | **86.3** | **81.8** | **92.6** | **90.0** | **88.1** | **90.3** | **90.2** | **90.7** | **90.2** |
| 8 | SVM+DE | 88.0 | 85.9 | 84.7 | 88.1 | 82.8 | 85.9 | 86.8 | 88.0 | 88.9 |
| | SVM+DE+LS | **89.7** | **89.1** | **89.6** | **90.0** | **88.6** | **89.3** | **90.6** | **88.5** | **91.1** |

Para o problema do reconhecimento de dígitos escritos à mão, a Tabela 6.7 mostra a comparação entre SVM+DE e SVM+DE+LS, em que a versão com busca local obteve melhores resultados em todos os experimentos. Nos casos em que o banco de dados foi gerado com números de características diferentes, o algoritmo com busca local teve o melhor desempenho em comparação com versão sem busca local. O banco de dados com

2 características foi gerado utilizando o algoritmo ZCZ, ou seja, foi utilizada apenas a distância dos pontos em relação aos centróides de duas zonas, pois o algoritmo ICZ-ZCZ com número de zonas $N_z = 1$ resultaria em duas características idênticas, já que o centróide da imagem coincide com o centróide da zona. Para gerar o banco de dados com 4 e 8 características, foi usado o algoritmo ICZ-ZCZ com 2 e 4 zonas, respectivamente. Quando o banco de dados do problema de classificação de dígitos manuscritos foi gerado com apenas 2 características, o desempenho do classificador otimizado pelo algoritmo sem busca local foi bem inferior. Este fato é observado nos experimentos do tipo C, em que a diferença entre os algoritmos chega a pouco mais de 40% para o banco com 2 características e 18% para o banco com 4 características. Há ainda o caso particular do experimento tipo B, em que para 10% de ruído introduzido a diferença entre os dois algotimos é de pouco mais de 26% e com a introdução de ruído de 50% essa diferença cai para menos de 1%. Fato repetido para o banco com 2 características nos experimentos tipo C e D. A tabela 6.7 mostra que a introdução dos tipos de ruído não diminuíram significativamente a acurácia para a versão com busca local. Pelo contrário, a acurácia aumentou em relação ao caso quando as imagens não foram perturbadas com ruído. A exceção desta afirmativa é o caso do banco com 4 características, em que a versão sem busca local somente no experimento tipo B com 50% de ruído a acurácia não diminuiu, e a acurácia do classificador gerado pelo algoritmo com busca local no experimento tipo A com 10% de ruído caiu 5% em relação ao experimento sem ruído. Os gráficos mostrados na Figura 6.10 mostram a estabilidade da acurácia após a introdução dos ruídos Gaussianos (para 8 características).

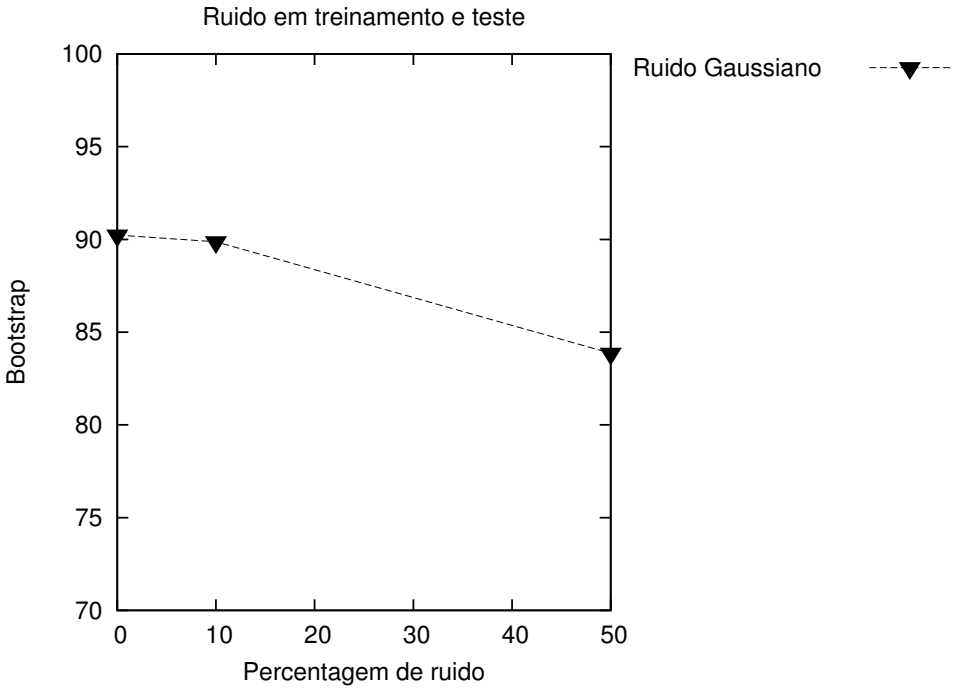



(a) Ruídos A (ruído no conjunto de treinamento).

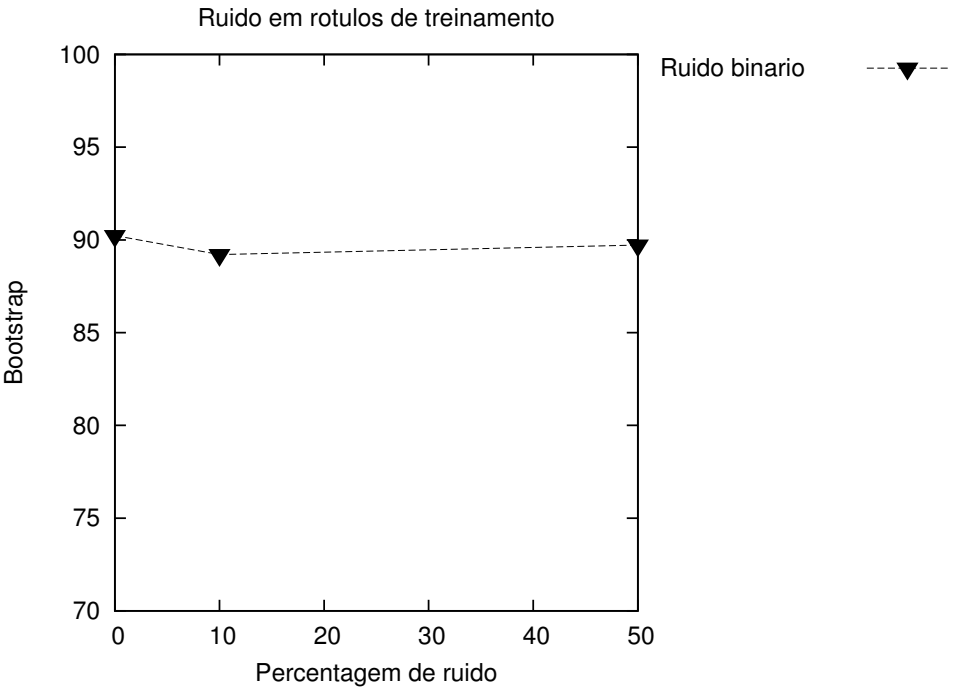



(b) Ruídos B (ruído no conjunto de validação).

Figura 6.10: Valores de bootstrap da função objetivo com ruído crescente para o problema do reconhecimento de dígitos escritos à mão.




(c) Ruídos C (ruído nos conjuntos de treinamento/validação).




(d) Ruído D (ruído binário nos rótulos de treinamento).

Figura 6.10: Valores de bootstrap da função objetivo com ruído crescente para o problema do reconhecimento de dígitos escritos à mão.

# Capítulo 7

# Conclusões

Neste trabalho foi proposto o uso de Evolução Diferencial hibridizada com busca local para otimizar os parâmetros da máquina de vetores suporte para classificar dados. O comportamento dos classificadores foi avaliado enquanto ruídos foram introduzidos em diferentes partes dos dados, ruído Gaussiano no treinamento, na validação, treinamento/validação e variáveis ruidosas no treinamento, validação e treinamento/validação, respectivamente denotados por ruídos A, B, C, D, E, F e G.

Analisando os resultados para o problema do coração, em termos de tipo de ruído, a deterioração foi notada com 10% de ruído, quando os dados de teste foram alterados. Isto é, todos os tipos de ruído que afetaram o conjunto de validação sofreram deterioração: ruídos B, C, F. A única exceção foi ruído G, que manteve o mesmo nível de acurácia. Embora a deterioração de desempenho tenha sido maior com 10% de ruído, quando os dados de validação foram perturbados, o mesmo não ocorreu com um maior percentual de ruído. Com o aumento do percentual de ruído para 50%, os ruídos que afetam o conjunto de dados de treinamento foram os mais deteriorados: ruído A e ruído C. Apesar de os ruídos D, E e G afetarem os dados de treinamento, eles não demonstraram a mesma deterioração. Ao contrário, com 10% de ruído eles apresentaram

pouca deterioração ou mantiveram o desempenho; com 50% de ruído eles apresentaram melhores níveis de desempenho comparados ao conjunto de dados original. O mesmo ocorreu com o ruído D, que aumentou a acurácia para 10% e 50% de ruído.

Os resultados para o problema do câncer de mama foram diferentes. Ambos dados de treinamento e validação, quando perturbados com ruído apresentaram deterioração de desempenho. Desta vez, o ruído D não foi uma exceção e apresentou uma queda de acurácia. A deterioração para ruídos A e B foi a mesma para ambas as percentagens de ruído. Para este conjunto de dados, o ruído D mostrou deterioração de desempenho, mas ainda foi menor que todos os outros tipos, enquanto ruído C teve a maior deterioração. Ruído E apenas teve uma queda de desempenho com 50% de ruído. Para ruídos F e G, eles não apresentaram nenhuma queda expressiva de desempenho, mantendo a acurácia. Apenas ruído F teve uma pequena queda com 10% de ruído, mas com 50% de ruído seu nível de acurácia aumentou novamente permanecendo um pouco abaixo do nível de desempenho dos dados originais.

O problema da íris, dividido em 3 sub-problemas, irisc0, irisc1 e irisc2, seguiram o mesmo padrão observado para o problema do câncer de mama. Ruídos A e B tiveram deterioração de desempenho mas ruído C teve a maior. Ruídos D, E, F e G mantiveram o mesmo desempenho. Com 10% de ruído, a deterioração, quando observada, foi pequena com quase nenhuma alteração de desempenho. Mas à medida que a percentagem de ruído introduzido aumentou para 50% a diferença entre as acurácias obtidas pelo algoritmo SVM+DE e SVM+DE+LS cresceu, sendo que o algoritmo com busca local, para essa porcentagem de ruído, obteve melhores resultados.

Para o problema do reconhecimento dos dígitos escritos à mão, a deterioração de desempenho aconteceu para o experimento com ruídos A e B, sendo que a maior diferença de desempenho na versão do algoritmo sem e com busca local ocorreu no caso do banco gerado com 2 características. Para o estudo de caso 2, é possível afirmar que o

algoritmo proposto SVM+DE+LS melhorou o desempenho dos classificadores gerados.

Juntamente com a avaliação de eficácia do método, foi possível verificar a sensibilidade da técnica de SVM ao ruído em cada um dos bancos de dados. Pode-se perceber que a deterioração de desempenho do método com taxas crescentes de ruído depende não só do tipo de banco de dados utilizado, mas também das taxas de ruído aplicadas aos dados. A queda no desempenho é proporcional ao aumento da taxa de ruído, porém a taxa de deterioração varia de acordo com o banco de dados, alguns sofrem mais que outros devido às suas características, como número de características e distribuição de exemplos de cada classe. De modo geral, a introdução da busca local resultou em uma pequena melhora de desempenho de classificação em relação aos classificadores otimizados por SVM+DE puro. Como discutido na Seção 6.1.2, a melhora ocorreu para os problemas do câncer de mama, íris e reconhecimento de dígitos escritos à mão, para os dois primeiros pequena proporcionalmente ao acréscimo de tempo computacional gerado pela busca local, para o último a melhora foi significativa para o caso com 2 características, devido à diferença em relação ao algoritmo sem busca local.

Em um problema de classificação no mundo real, os dados podem estar corrompidos e sujeitos a ruído. Este trabalho dá uma perspectiva do que esperar quando se adotar SVM+DE para classificar dados ruídosos. Análises preliminares indicam que o desempenho de SVM+DE depende do tipo e da percentagem do ruído e as características dos dados. Para este trabalho, optou-se por analisar e comparar a influência de ruído na classificação de dados utilizando os algoritmos SVM+DE e SVM+DE+TS apenas, pois uma comparação com os resultados de ruído Gaussiano (NETTLETON; ORRIOLS-PUIG; FORNELLS, 2010) e variáveis ruidosas (LUUKKA; LAMPINEN, 2011) deve ser feita sob as mesmas configurações de experimentos. Para trabalhos futuros, pode-se comparar nosso método com outras abordagens estudadas em (NETTLETON; ORRIOLS-PUIG; FORNELLS, 2010) e analisar os motivos que fizeram com

que em alguns casos a acurácia melhorasse com o aumento da porcentagem de ruído introduzido aos dados. Outra possibilidade de trabalho futuro, a técnica SVM fundamentada matematicamente por Vapnik (1995) possui base estatística que a torna robusta, além de ter se tornado uma técnica amplamente utilizada (LUTS et al., 2010; WU et al., 2007; SAPANKEVYCH; SANKAR, 2009). Para tratar as incertezas introduzidas, pode-se utilizar técnicas nebulosas (inglês: fuzzy) em conjunto com SVM para agrupamento de dados (eliminando ruído) (BANERJEE, 2009), e classificação, respectivamente; ou ainda para descrever os dados em linguagem natural (YU; LI, 2008), facilmente interpretável por humanos, diferentemente do modelo matemático do SVM. Como o custo computacional do SVM é alto, o algoritmo pode ser paralelizado. C-CUDA é uma linguagem de programação que tem sido bastante utilizada para programar algoritmos paralelizáveis para unidades de processamento gráfico (inglês: Graphics Processing Unit - GPU), que possuem em sua arquitetura os mecanismos necessários à paralelização (NVIDIA, 2007). Alguns algoritmos de otimização já foram paralelizados em C-CUDA, como DE (VERONESE; KROHLING, 2010). A paralelização do SVM é também uma possibilidade para trabalho futuro.

# Referências Bibliográficas


BANERJEE, A. Robust fuzzy clustering as a multi-objective optimization procedure. In: *Proceedings of the 28th North American Fuzzy Information Processing Society Annual Conference (NAFIPS 2009)*. [S.l.: s.n.], 2009. p. 1 –6.

BASTOS-FILHO, C. J. A.; CARACIOLO, M. P.; MIRANDA, P. B. C.; CARVALHO, D. F. Multi-ring Particle Swarm Optimization. In: *10th Brazilian Symposium on Neural Networks, 2008. SBRN '08.* [S.l.: s.n.], 2008. p. 111 –116.

BHATTACHARYYA, C. Robust classification of noisy data using second order cone programming approach. In: *Proceedings of International Conference on Intelligent Sensing and Information Processing, 2004.* [S.l.: s.n.], 2004. p. 433 – 438.

BYEON, B.; RASHEED, K. Simultaneously Removing Noise and Selecting Relevant Features for High Dimensional Noisy Data. In: *Proceedings of the 7th International Conference on Machine Learning and Applications, 2008.* [S.l.: s.n.], 2008. p. 147 –152.

DAS, S.; ABRAHAM, A.; CHAKRABORTY, U. K.; KONAR, A. Differential evolution using a neighborhood-based mutation operator. *IEEE Transactions on Evolutionary Computation*, v. 13, n. 3, p. 526 –553, June 2009.

DAS, S.; SUGANTHAN, P. Differential Evolution: A Survey of the State-of-the-Art. *IEEE Transactions on Evolutionary Computation*, v. 15, n. 1, p. 4 –31, February 2011.

DU, P.; PENG, J.; TERLAKY, T. Self-adaptive support vector machines: modelling and experiments. *Computational Management Science*, v. 6, n. 1, p. 41–51, 2009.

EBERHART, R. C.; SHI, Y. *Computational Intelligence: Concepts to Implementations.* [S.l.]: Morgan Kaufmann, 2007.

FRANK, A.; ASUNCION, A. *UCI Machine Learning Repository.* 2010. Disponível em: <http://archive.ics.uci.edu/ml/>.

GLOVER, F.; LAGUNA, M. Tabu search. In: PARDALOS, P. M.; RESENDE, M. G. C. (Ed.). *Handbook of Applied Optimization.* [S.l.]: Oxford University Press, 2002. p. 194–208.

GUNN, S. R. *Support Vector Machines for Classification and Regression.* [S.l.], May 1998.

HERTZ, A.; TAILLARD, E.; WERRA, D. D. *A tutorial on tabu search.* [S.l.], 1995.

HUANG, C.; WANG, C. A GA-based feature selection and parameters optimization for support vector machines. *Expert Systems with Applications*, v. 31, n. 2, p. 231–240, 2006.

JUN, S.; JIAN, L. A combination of differential evolution and support vector machine for rainstorm forecast. In: *Third International Symposium on Intelligent Information Technology Application, 2009. IITA 2009.* [S.l.: s.n.], 2009. v. 3, p. 392 –395.

KAWANO, S.; OKUMURA, D.; TAMURA, H.; TANAKA, H.; TANNO, K. Online learning method using support vector machine for surface-electromyogram recognition. *International Symposium on Artificial Life and Robotics*, v. 13, p. 483–487, February 2009.

KHOSHGOFTAAR, T. M.; HULSE, J. V.; NAPOLITANO, A. Comparing boosting and bagging techniques with noisy and imbalanced data. *IEEE Transactions on*

*Systems, Man and Cybernetics, Part A: Systems and Humans*, v. 41, n. 3, p. 552 –568, 2011.

KIM, H.; PANG, S.; JE, H.; KIM, D.; BANG, S. Y. Constructing support vector machine ensemble. *Pattern Recognition*, v. 36, n. 12, p. 2757–2767, 2003.

KROHLING, R. A. Comunicado privado. *PPGI/UFES*, 2011.

KROHLING, R. A.; COELHO, L. dos S. PSO-E: Particle Swarm with Exponential Distribution. In: *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2006*. [S.l.: s.n.], 2006. p. 1428 –1433.

LI, J. A combination of DE and SVM with feature selection for road icing forecast. In: *Proceedings of the 2nd international Asia conference on Informatics in control, automation and robotics*. Piscataway, NJ, USA: IEEE Press, 2010. (CAR'10, v. 2), p. 509–512.

LIN, S.; YING, K.; CHEN, S.; LEE, Z. Particle swarm optimization for parameter determination and feature selection of support vector machines. *Expert Systems with Applications*, v. 35, p. 1817–1824, 2008.

LLERENA, N. E. M. *Ensembles na classificação relacional*. Dissertação (Mestrado) — Universidade de São Paulo, São Carlos, Setembro 2011. Instituto de Ciências Matemáticas e de Computação - ICMC.

LUTS, J.; OJEDA, F.; VAN DE PLAS, R.; DE MOOR, B.; VAN HUFFEL, S.; SUYKENS, J. A. K. A tutorial on support vector machine-based methods for classification problems in chemometrics. *Analytica Chimica Acta*, v. 665, n. 2, p. 129–145, April 2010.

LUUKKA, P.; LAMPINEN, J. Differential Evolution Classifier in Noisy Settings and with Interacting Variables. *Applied Soft Computing*, v. 11, n. 1, p. 891–899, 2011.

MASHINCHI, M. H.; ORGUN, M. A.; PEDRYCZ, W. Hybrid optimization with improved tabu search. *Applied Soft Computing*, v. 11, p. 1993–2006, 2011.

NELDER, J. A.; MEAD, R. A simplex method for function minimization. *The Computer Journal*, v. 7, n. 4, p. 308–313, 1965.

NETTLETON, D. F.; ORRIOLS-PUIG, A.; FORNELLS, A. A study of the effect of different types of noise on the precision of supervised learning techniques. *Artificial Intelligence Review*, v. 33, n. 4, p. 275–306, April 2010.

NVIDIA. *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*. [S.l.]: NVIDIA Corporation, 2007.

PAQUET, U.; ENGELBRECHT, A. Training support vector machines with particle swarms. In: *Proceedings of the International Joint Conference on Neural Networks, 2003*. [S.l.: s.n.], 2003. v. 2, p. 1593 – 1598 vol.2.

RAJASHEKARARADHYA, S. V. Efficient zone based feature extration algorithm for handwritten numeral recognition of four popular south indian scripts. *Journal of Theoretical and Applied Information Technology*, JATIT, v. 4, n. 12, p. 1171–1181, 2008.

RÜBESAM, A. *Estimação Não Paramétrica Aplicada a Problemas de Classificação via Bagging and Boosting*. Dissertação (Mestrado) — Unicamp, São Paulo, Fevereiro 2004.

SAPANKEVYCH, N.; SANKAR, R. Time Series Prediction Using Support Vector Machines: A Survey. *IEEE Computational Intelligence Magazine*, v. 4, n. 2, p. 24 –38, may 2009.

SCHAPIRE, R. E. The strength of weak learnability. *Machine Learning*, v. 5, p. 197–227, 1990.

SEGUNDO, G. A. S.; KROHLING, R. A.; COSME, R. C. Constrained min-max optimization problems: A novel approach with co-evolutionary differential evolution. In: *CEC'11*. [S.l.: s.n.], 2011. p. 1–7.

SEMOLINI, R. *Support Vector Machines, Inferência Transdutiva e o Problema de Classificação*. Dissertação (Mestrado) — Universidade Estadual de Campinas, Dezembro 2002. Faculdade de Engenharia Elétrica e de Computação, Departamento de Engenharia de Computação e Automação Industrial.

SILVA, E. K. *Evolução Diferencial para Problemas de Otimização Restrita*. Dissertação (Mestrado) — Laboratório Nacional de Computação Científica, Março 2009. Programa de Pós Graduação em Modelagem Computacional.

SOUZA, M. B. *Rede de aprendizado supervisionado como método de auxílio na detecção do ceratocone*. Tese (Doutorado) — Universidade de São Paulo, 2011. Faculdade de Medicina.

STORN, R. Differential Evolution Design of an IIR-Filter with Requirements for Magnitude and Group Delay. In: *Proceedings of IEEE International Conference on Evolutionary Computation ICEC 96*. [S.l.: s.n.], 1995. p. 268–273. ICSI.

STORN, R.; PRICE, K. Differential Evolution – A simple and efficient heuristic for global optimization over Continuous Spaces. *Journal of Global Optimization*, v. 11, p. 341–359, 1997.

SUN, Y.; YANG, G.; WANG, L.; SHI, Y.; WU, Y. Gas load prediction based on DE-SVM algorithm. In: *Proceedings of the 2nd International Conference on Future Computer and Communication (ICFCC)*. [S.l.: s.n.], 2010. v. 1, p. 155–158.

VAPNIK, V.; GOLOWICH, S. E.; SMOLA, S. Support Vector Method for Function Approximation, Regression Estimation, and Signal Processing. In: *Advances in Neural Information Processing Systems 9*. [S.l.]: MIT Press, 1996. p. 281–287.

VAPNIK, V. N. *The nature of statistical learning theory*. New York, NY, USA: Springer-Verlag, 1995.

VERONESE, L. P.; KROHLING, R. A. Differential evolution algorithm on the gpu with c-cuda. In: *Proceedings of 2010 IEEE Congress on Evolutionary Computation (CEC)*. [S.l.: s.n.], 2010. p. 1 –7.

WANG, S.; MATHEW, A.; CHEN, Y.; XI, L.; MA, L.; LEE, J. Empirical analysis of support vector machine ensemble classifiers. *Expert Systems with Applications*, v. 36, p. 6466–6476, April 2009.

WU, C. H.; TZENG, G. H.; GOO, Y. J.; FANG, W. C. A real-valued genetic algorithm to optimize the parameters of support vector machine for predicting bankruptcy. *Expert Systems with Applications*, v. 32, n. 2, p. 397–408, February 2007.

WU, T.; DUCHATEAU, J.; MARTENS, J.; VAN COMPERNOLLE, D. Feature subset selection for improved native accent identification. *Speech Communication*, Elsevier, v. 52, p. 83–98, February 2010.

YAZDI, H. S.; EFATI, S.; SABERI, Z. The probabilistic constraints in the support vector machine. *Applied Mathematics and Computation*, p. 467–479, June 2007.

YAZDI, H. S.; EFFATI, S.; SABERI, Z. The probabilistic constraints in the support vector machine. *Applied Mathematics and Computation*, v. 194, n. 2, p. 467–479, 2007.

YU, W.; LI, X. On-line fuzzy modeling via clustering and support vector machines. *Information Sciences*, v. 178, p. 4264–4279, November 2008.

ZHANG, T.; RAMAKRISHNAN, R.; LIVNY, M. BIRCH: A New Data Clustering Algorithm and Its Applications. *Data Mining and Knowledge Discovery*, v. 1, p. 141–182, 1997.

ZOUBIR, A.; BOASHASH, B. The bootstrap and its application in signal processing. *IEEE Signal Processing Magazine*, v. 15, n. 1, p. 56 –76, jan 1998.

# Trabalhos publicados e submetidos pelo autor

- COSME, R. C.; KROHLING, R. A. . Reinforcement Learning Hybridized with Differential Evolution. In: II WCI, 2010, São Bernardo. II Workshop on Computational Intelligence, 2010.

- COSME, R. C.; KROHLING, R. A. . Support Vector Machines applied to noisy data classification using differential evolution with local search. In: WSC´16, 2011. World Conference on Soft Computing in Industrial Applications (WSC-2011).

- SEGUNDO, G. A. S.; KROHLING, R. A.; COSME, R. C.. Constrained Min-Max Optimization Problems: A Novel Approach with Co-Evolutionary Differential Evolution. In: CEC´11, 2011, Flórida. IEEE Congress on Evolutionary Computation, 2011. p. 1-7.

- SEGUNDO, G. A. S.; KROHLING, R. A.; COSME, Solving Constrained Min-Max Optimization Problems Using Differential Evolution. Submitted to Expert Systems with Applications, Elsevier, submitted 2012.