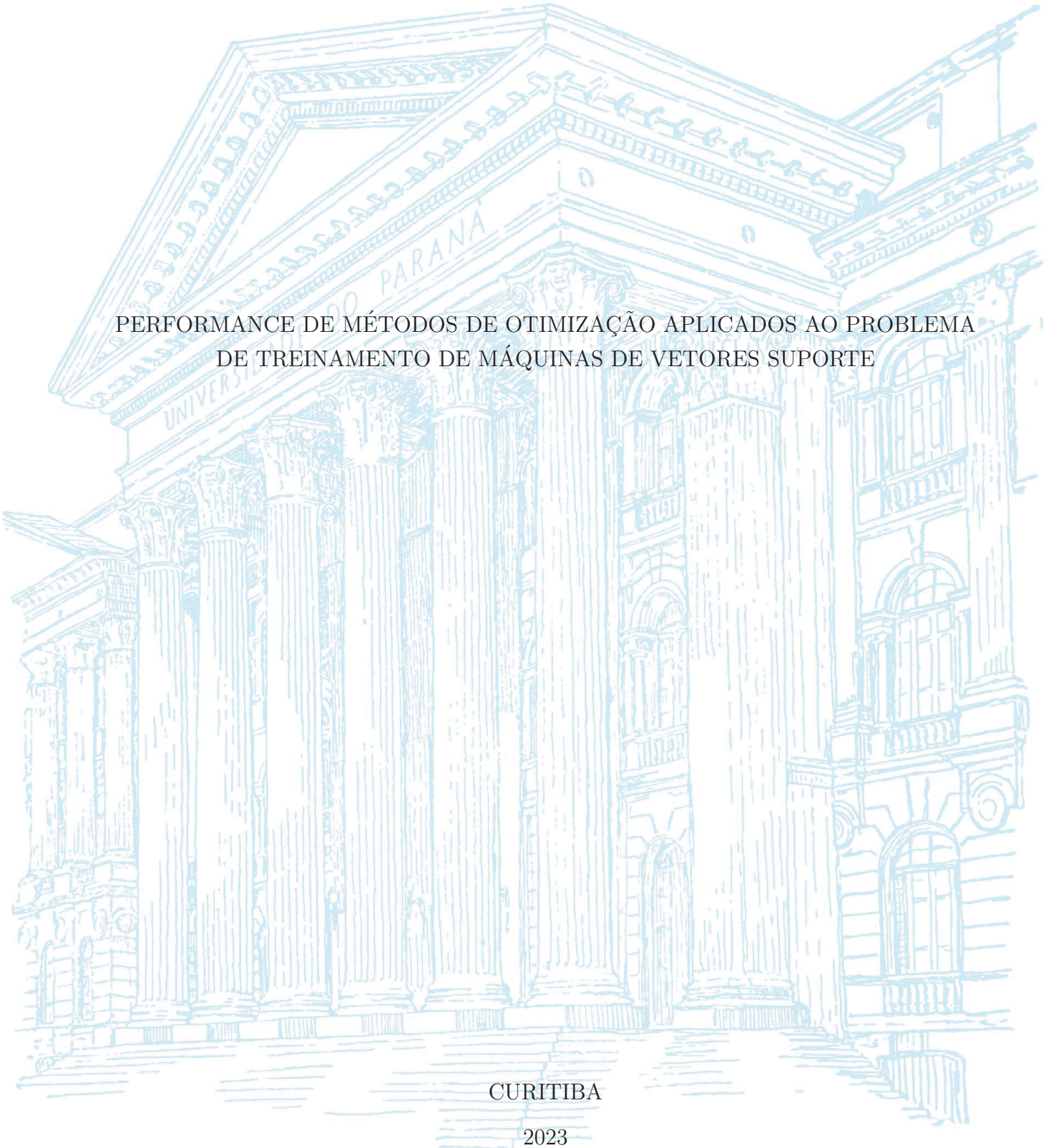
UNIVERSIDADE FEDERAL DO PARANÁ

TIAGO LINO BELLO



PERFORMANCE DE MÉTODOS DE OTIMIZAÇÃO APLICADOS AO PROBLEMA DE TREINAMENTO DE MÁQUINAS DE VETORES SUPORTE

CURITIBA

2023

TIAGO LINO BELLO

PERFORMANCE DE MÉTODOS DE OTIMIZAÇÃO APLICADOS AO PROBLEMA DE TREINAMENTO DE MÁQUINAS DE VETORES SUPORTE

Tese apresentada ao Programa de Pós-Graduação em Métodos Numéricos em Engenharia, área de concentração em Programação Matemática, dos Setores de Ciências Exatas e Tecnologia da Universidade Federal do Paraná, como um dos requisitos parciais para obtenção do título de Doutor em Métodos Numéricos em Engenharia.

Orientador: Dr. Luiz Carlos Matioli
Coorientador: Dr. Lucas Garcia Pedroso

CURITIBA

2023

DADOS INTERNACIONAIS DE CATALOGAÇÃO NA PUBLICAÇÃO (CIP)
UNIVERSIDADE FEDERAL DO PARANÁ
SISTEMA DE BIBLIOTECAS – BIBLIOTECA DE CIÊNCIA E TECNOLOGIA

Bello, Tiago Lino
Performance de métodos de otimização aplicados ao problema de treinamento de máquinas de vetores suporte / Tiago Lino Bello. – Curitiba, 2023.
1 recurso on-line : PDF.

Tese (Doutorado) - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Métodos Numéricos em Engenharia.

Orientador: Luiz Carlos Matioli
Coorientador: Lucas Garcia Pedroso

1. Otimização matemática. 2. Algoritmos. 3. Aprendizado de máquinas. 4. Vetores Suporte. I. Universidade Federal do Paraná. II. Programa de Pós-Graduação em Métodos Numéricos em Engenharia. III. Matioli, Luiz Carlos. IV. Pedroso, Lucas Garcia. V. Título.

Bibliotecário: Elias Barbosa da Silva CRB-9/1894

MINISTÉRIO DA EDUCAÇÃO
SETOR DE CIÊNCIAS EXATAS
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO MÉTODOS NUMÉRICOS EM ENGENHARIA - 40001016030P0

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação MÉTODOS NUMÉRICOS EM ENGENHARIA da Universidade Federal do Paraná foram convocados para realizar a arguição da tese de Doutorado de **TIAGO LINO BELLO** intitulada: **PERFORMANCE DE MÉTODOS DE OTIMIZAÇÃO APLICADOS AO PROBLEMA DE TREINAMENTO DE MÁQUINAS DE VETORES SUPORTE**, sob orientação do Prof. Dr. LUIZ CARLOS MATIOLI, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de doutor está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

Curitiba, 25 de Agosto de 2023.

Assinatura Eletrônica
26/08/2023 15:28:47.0
LUIZ CARLOS MATIOLI
Presidente da Banca Examinadora

Assinatura Eletrônica
28/08/2023 15:13:24.0
CESAR AUGUSTO TACONELI
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
26/08/2023 15:49:32.0
GISLAINE APARECIDA PERIÇARO
Avaliador Externo (UNIVERSIDADE ESTADUAL DO PARANÁ)

Assinatura Eletrônica
26/08/2023 18:13:12.0
PAULO SERGIO MARQUES DOS SANTOS
Avaliador Externo (UNIVERSIDADE FEDERAL DO PIAUí)

Para Rosangela e Augusto Bello,
aqueles que são o meu primeiro tudo.

AGRADECIMENTOS

Gostaria de expressar minha profunda gratidão e apreço a todos que me apoiaram ao longo desta jornada de doutorado.

Gostaria de agradecer de coração aos meus orientadores, Luiz Carlos Matioli e Lucas Garcia Pedroso, pelas orientações, expertise e apoio inabalável. Seus comentários perspicazes, incentivos e disponibilidade constantes foram inestimáveis na formação desta pesquisa e no meu crescimento acadêmico.

Também sou imensamente grato aos membros da banca avaliadora, pelos valiosos *insights*, críticas construtivas e contribuições para este trabalho. Vossas experiências e dedicação desempenharam um papel crucial na melhoria da qualidade e rigor desta pesquisa.

Gostaria de estender meu agradecimento aos meus colegas e amigos. Amigos fora da academia, do PPGMNE, do LEG, Lactec e Bradesco. O apoio, discussões e incentivos foram fundamentais para moldar minhas ideias e expandir meu conhecimento. Em especial, gostaria de agradecer Daniela Miray, pela paciência, conversas, debates, risadas e refeições compartilhadas no R.U..

Gostaria de reconhecer o apoio financeiro fornecido pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior e Conselho Nacional de Desenvolvimento Científico e Tecnológico, sendo fundamentais para viabilizar a conclusão desta pesquisa.

Por fim, quero expressar minha sincera gratidão à minha família pelo amor incondicional. Em especial meu pai, Augusto Bello, e minha mãe, Rosangela Lino Bello. Pelo apoio e compreensão ao longo desta jornada. A crença em mim e incentivos constantes foram a força motriz por trás das minhas conquistas.

A todos que contribuíram, direta ou indiretamente, para esta pesquisa, eu estendo minha sincera apreciação. O apoio e contribuições tiveram um impacto significativo no meu crescimento acadêmico e pessoal.

Muito obrigado.

"It's what you do in the dark
that puts you in the light.
Rule yourself."
— *Campanha publicitária de uma empresa de materiais esportivos, protagonizada por Michael Phelps.*

RESUMO

Neste trabalho, pesquisou-se e implementou-se métodos de Otimização para treinamento em Máquina de Vetores de Suporte. O foco principal consiste em analisar o desempenho de cada um deles e se algum se sai melhor que os demais em termos de tempo computacional e métricas de classificação. Tais métodos de Otimização foram propostos para problemas genéricos de Otimização não Linear restritos, e aqui objetivou-se adaptá-los ao problema de treinamento de Máquina de Vetores Suporte. Dessa forma, foram implementados métodos das classes de projeção, Pontos Interiores, Restrições Ativas, Lagrangiano Aumentados e Filtro. Esses algoritmos foram implementados em MATLAB®, e experimentos numéricos foram conduzidos a partir da aplicação de conjuntos de dados gerados aleatoriamente e de repositórios de Aprendizado de Máquina. Dos experimentos realizados, analisando sob a ótica de qualidade de soluções encontradas, análise de sobrevivência, perfil de desempenho de treinamento, bem como observando métricas de tempo de treinamento, acurácia, *F1 Score* e coeficiente de correlação de Matthews, os resultados indicam que até mesmo uma implementação ingênua do algoritmo de Restrições Ativas baseado em Otimização Sequencial Mínima foi mais eficiente na maioria dos critérios quando comparado aos demais algoritmos implementados.

Palavras-chave: Máquina de Vetores Suporte. Treinamento. Classificação. Algoritmos de otimização. Experimentos numéricos.

ABSTRACT

In this work, Optimization methods were researched and implemented for training Support Vector Machine. The main focus is to analyze the performance of each method and determine if any outperforms the others in terms of computational time and classification metrics. These Optimization methods were originally proposed for generic problems of constrained non-linear Optimization, and the objective here was to adapt them to the problem of Support Vector Machine training. Thus, methods from the classes of projection, Interior Points, Active Sets, Augmented Lagrangian, and Filter method were implemented. These algorithms were implemented in MATLAB®, and numerical experiments were conducted using randomly generated datasets and Machine Learning repositories. From the performed experiments, analyzing from the perspective of quality of solutions found, survival analysis, training performance profile, as well as observing training time, accuracy, F1 Score, and Matthews correlation coefficient, the results indicate that even a naïve implementation of the Active Set algorithm based on Sequential Minimal Optimization was more efficient in most criteria compared to the other implemented algorithms.

Key-words: Support Vector Machine. Training. Classification. Optimization algorithms. Numerical experiments.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

FAL	Método de Filtro combinado com Torrealba et al. (2021)
FN	Falso negativo
FP	Falso positivo
FQP	Método de Filtro combinado com *quadprog*
KKT	Karush-Kuhn-Tucker
KSVM	Método de Torrealba et al. (2021) baseado em Lagrangiano aumentado
KPG	Método de Torrealba et al. (2021) adaptado a um problema de projeção
LIBSVM	Método baseado em SMO com seleção de conjuntos por Fan et al. (2005)
MCC	Coeficiente de Correlação de Matthews
MVP	Método baseado em SMO com seleção de conjuntos por Platt (1998)
PCA	Análise de Componentes Principais
PC1	Primeira componente principal
PC2	Segunda componente principal
PGC	Método de Gradiente Projetado
PGN	Método de Newton Projetado
QP	*Quadprog*
RBF	Função *kernel* Gaussiana
SMO	Otimização Sequencial Mínima
SPGAL	Método de Gradiente Espectral Projetado combinado com Lagrangiano aumentado
SVM	Máquina de Vetores Suporte
TN	Verdadeiro negativo
TP	Verdadeiro positivo

SUMÁRIO

1 INTRODUÇÃO

Com o avanço da tecnologia, a resolução de certos problemas se torna possível graças à maior capacidade de processamento dos computadores. Acompanhando tal progresso, tem-se a produção de uma vasta quantidade de dados em diversas áreas do conhecimento, fenômeno este conhecido popularmente como *Big Data*.

Como descrito em James et al. (2013), com o crescente volume de dados, variedade de dados gerados e a velocidade em que são recebidos e administrados em um determinado processo, o entendimento de métodos que envolvam Matemática e Estatística se tornam essenciais e relevantes. Por meio destes é possível auxiliar os tomadores de decisões a obterem conclusões ou, ainda, obterem previsões de padrões ou característica que o próprio dado possui. Desta forma, estes dados são úteis quando são estudados adequadamente a fim de produzir informações que façam sentido para humanos.

Obviamente, é complexo conhecer quais são as exatas relações que produzem os fenômenos que são observados no mundo. A título de exemplo, como prever: se um indivíduo terá câncer futuramente dados seus hábitos; se um indivíduo comprará um certo produto sabendo que ele comprou outro; a probabilidade de um cliente entrar em inadimplência dada sua situação financeira e assim por diante. Nesta situação, o que é possível ser feito é uma aproximação razoável do evento observado com os dados disponíveis. De forma geral, o objetivo principal da área de estudo de Aprendizagem de Máquina é ajustar um modelo estatístico e matemático que possa replicar o fenômeno real por meio de algoritmos, aprendendo com os dados disponíveis.

A maioria das ferramentas de aprendizado podem ser categorizadas em duas classes: supervisionado e não supervisionado. Um aprendizado supervisionado motiva-se em prever/estimar um valor com base no uso de características de entrada e saída. Neste caso, é possível estimar desempenho e validar o modelo com a variável resposta, seja como regressão – a variável resposta é quantitativa – ou classificação – a variável resposta é qualitativa. Em contrapartida, no aprendizado não supervisionado não existe uma resposta a ser prevista/estimada, objetivando-se descrever relações e padrões apenas com os dados de entrada. Mas, independentemente desta categorização, a performance dos métodos utilizados dependem do problema a ser resolvido. Portanto, não existe consenso em uma abordagem universal que resolve todos os problemas.

No caso de um problema de classificação, o método de Máquina de Vetores Suporte (SVM), do inglês *Support Vector Machine*, tem se mostrado uma das abordagens mais úteis para Aprendizagem de Máquina. Introduzida por Boser, Guyon e Vapnik (1992) e estendida por Cortes e Vapnik (1995), SVM é uma ferramenta útil e efetiva para clas-

sificação de dados de duas classes, seja com disposições lineares ou não lineares entre as classes. Hastie, Tibshirani e Friedman (2009), Bishop (2006), James et al. (2013) e Piccialli e Sciandrone (2018) também descrevem que, motivado pelo que se define *Statistical Learning*, o princípio de treino de um classificador de dados por SVM é que o erro de classificação esperado é minimizado para observações de teste não utilizadas, sendo possível definir bons classificadores de predição. Assim, dado um conjunto de treino para problemas de classificação binário, o método SVM almeja encontrar um hiperplano de máxima margem que separe todas as observações de treino em duas categorias. Desta forma, obtém-se uma função de classificação ajustada aos dados.

Para o caso de classificação binária, o problema de Otimização gerado pelo método de SVM pode ser proposto equivalentemente em duas formas. Basicamente, o problema de treino de SVM irrestrito utiliza uma função de perda – geralmente não diferenciável. Por outro lado, o problema de treino de SVM restrito introduz variáveis de folga e cria restrições, evitando o emprego de função de perda. Essa última torna o problema com restrições lineares de desigualdades e uma função objetivo suave (CHAUHAN; DAHIYA; SHARMA, 2019; PICCIALLI; SCIANDRONE, 2018). Além disso, independentemente dos problemas serem restritos ou não, o problema de treinamento de SVM pode ser reformulado ao aplicar os conceitos da teoria de Dualidade (sobre Dualidade, ver Luenberger, Ye et al. (1984)) e de manipulações algébricas, obtendo um problema de treinamento com boas propriedades no contexto de Otimização.

A partir da formulação do problema de Otimização, seja primal ou dual, a solução do problema fornecerá elementos para ajustar uma função de classificação generalizada ao conjunto de dados treinado. No caso de SVM, esta função de classificação ajustada é baseada apenas em um conjunto de vetores de suporte, que são um subconjunto dos dados de treinamento. Os vetores suporte determinam o classificador, de modo que o modelo obtido se torna o mesmo se observações que não são vetores suporte fossem previamente eliminadas do conjunto de treinamento.

Independentemente da formulação, primal ou dual, o problema de treinamento consiste em resolver um problema de programação convexa de grande escala, cujas dificuldades estão principalmente relacionadas ao número de observações de treinamento, o que leva a um grande número de variáveis. Devido à estrutura, em particular à convexidade, do problema a ser resolvido, algoritmos de Otimização para SVM são necessários para a construção do classificador. Isso é possível porque as propriedades do problema de treinamento de SVM são bem definidas do ponto de vista da Otimização (CHAUHAN; DAHIYA; SHARMA, 2019; PICCIALLI; SCIANDRONE, 2018; CARRIZOSA; MORALES, 2013). Conforme será discutido no decorrer do trabalho, existem vários algoritmos para o resolver o problema de treinamento de SVM, todos possuindo desafios a serem superados.

1.1 OBJETIVO DA PESQUISA

Uma das principais contribuições desse trabalho é a análise e comparação de algoritmos clássicos de Otimização aplicados ao problema de SVM. Para o nosso estudo, define-se o problema de treinamento de SVM a partir da sua formulação dual, devido, principalmente, às propriedades de convexidade. Em poucas palavras, o problema de treinamento dual de SVM binário é um problema de Otimização convexa, sujeito a uma restrição de igualdade e restrições de caixa, que pode ser resolvido por métodos de Otimização Convexa.

No levantamento bibliográfico realizado para a escrita deste texto, sentiu-se falta de trabalhos que comparem a performance de algoritmos clássicos da literatura de Otimização aplicados ao problema de SVM. Aqui, foram implementados métodos baseados em Restrições Ativas, como por exemplo a Otimização Sequencial Mínima (SMO), Gradiente e Newton Projetado, Gradiente Espectral Projetado, Pontos Interiores, Lagrangiano Aumentado e Filtro.

Sendo assim, destaca-se alguns pontos que norteia o desenvolvimento do trabalho, ou seja:

- o comportamento dos métodos de Otimização aplicados ao problema de treinamento de SVM;
- quais categorias de métodos de Otimização se destacam ao serem utilizados para treinar conjunto de dados com comportamentos distintos (densos ou não, classes desbalanceadas ou não);
- a performance computacional dos métodos baseados em Restrições Ativas, mesmo usando uma implementação ingênua, em relação aos demais métodos.

Destaca-se que o objetivo principal do trabalho é analisar e a aplicar de algoritmos de Otimização restritos, elaborados para problemas gerais de Programação não Linear, ao *framework* de SVM, aproveitando as boas propriedades de Otimização que o problema de treinamento possui. Uma vez escolhidos os hiperparâmetros dos algoritmos, a comparação é realizada a partir de testes numéricos, destacando a performance, via tempo computacional de treinamento, e métricas de classificação computadas a partir da matriz de confusão do classificador.

1.2 ESTRUTURA DO TRABALHO

No presente capítulo foi apresentado a essência do problema de SVM, definindo objetivos para a pesquisa.

No Capítulo 2 é descrito o método de SVM, sendo explicado desde sua forma considerando margem rígida até a inclusão de funções *kernel* para determinar um classificador de margem suave. Neste capítulo também são levantados alguns trabalhos publicados sobre o problema de treinamento de SVM.

No Capítulo 3 são descritos algoritmos gerais de Otimização não Linear para solução de problemas com restrições.

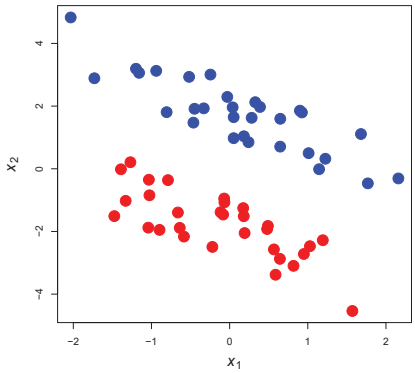
No Capítulo 4 são apresentados experimentos numéricos, com aplicações tanto em dados *benchmark* como gerados aleatoriamente, comparando a performance dos métodos de Otimização clássicos em termos de tempo computacional e métricas de classificação.

No Capítulo 5 é comentada as considerações finais.

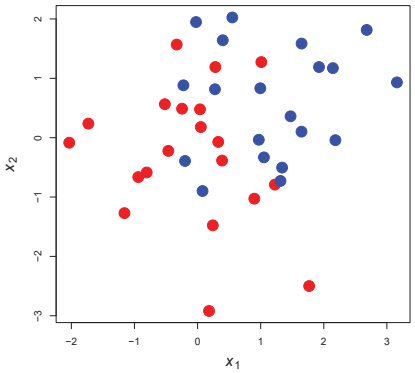
2 MÁQUINA DE VETORES SUPORTE

De acordo com Cristianini, Shawe-Taylor et al. (2000) e Hastie, Tibshirani e Friedman (2009), uma das técnicas de Aprendizagem Supervisionada mais empregadas para classificação de dados/reconhecimento de padrões é a SVM, devido tanto à sua robustez para uma carga elevada de dados de grande dimensão quanto à sua capacidade de generalização. Em um problema de classificação, esse método irá definir um hiperplano ótimo de máxima margem entre determinadas entradas/instâncias, sendo capaz de classificar quaisquer novas entradas em uma de duas saídas/respostas/categorias por meio de uma função de decisão. Como a base de SVM vem de conceitos de Otimização, garante-se formas de definir tal hiperplano e sua unicidade.

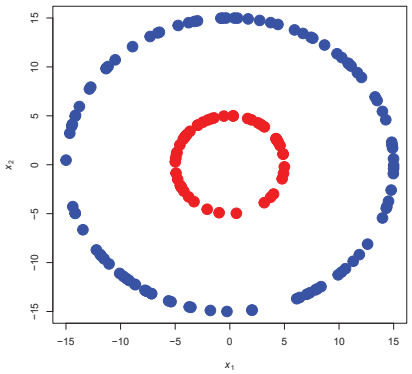
Em SVM, dependendo com que os dados estão distribuídos no espaço de entrada, podemos ou não conseguir um hiperplano ótimo que os separem em duas classes. Em outras palavras, podemos classificar dados que são ou tendem a ser linearmente separáveis e dados que não são linearmente separáveis. A Figura 1 ilustra os casos citados.



(a) Linearmente separáveis.



(b) Linearmente flexível.



(c) Não linear.

FIGURA 1 – Dados possíveis de serem linearmente ou não separáveis.

Para tratar dados perfeitamente separáveis linearmente, Boser, Guyon e Vapnik

(1992) propõem um classificador linear de máxima margem rígida, que determina um hiperplano ótimo de maior margem para dados linearmente separáveis. Esse trabalho é considerado um marco, pois, como descrito em Chauhan, Dahiya e Sharma (2019), muitos o consideram como o principal motivador de SVM. Cortes e Vapnik (1995) descrevem um classificador de máxima margem suave/flexível e um classificador de máxima margem não linear, sendo efetivamente a SVM. Vale destacar que a ideia de SVM foi proposta por Boser, Guyon e Vapnik (1992), mas a formulação comumente utilizada é a de Cortes e Vapnik (1995). No decorrer dos anos, várias alterações foram propostas para problemas de classificação, sendo algumas discutidas neste capítulo.

A organização deste capítulo é realizada como se segue. Inicialmente é destacada a notação utilizada no trabalho. Posteriormente, serão discutidas as formulações de SVM. Para mais detalhes e informações, os seguintes trabalhos são sugeridos: Boser, Guyon e Vapnik (1992), Cortes e Vapnik (1995), Cristianini, Shawe-Taylor et al. (2000), Scholkopf e Smola (2001), Shawe-Taylor e Cristianini (2004), Hastie, Tibshirani e Friedman (2009), Deng, Tian e Zhang (2012), James et al. (2013), Krulikovski (2017), Piccialli e Sciandrone (2018), Chauhan, Dahiya e Sharma (2019).

2.1 NOTAÇÃO ADOTADA

De uma forma geral, as técnicas de aprendizagem supervisonada motivam-se em reconhecer os padrões de determinados eventos a partir de seus registros. Estes registros geralmente são definidos na forma de pares (x, y), onde x representa os atributos (ou dados de entrada) que o problema possui e y a sua respectiva saída/resposta. Assim, uma vez escolhida a técnica de aprendizagem, ajusta-se um modelo a um dado conjunto de registros objetivando a generalizar o comportamento do evento. Todo este processo descrito é denominado como fase de treinamento do modelo.

A fim de verificar a eficiência do classificador ajustado, testam-se novos registros que não foram utilizados durante o seu ajuste, observando se a saída do classificador está correta. Em outras palavras, busca-se, agora, obter estimativas/previsões/saídas para novos dados utilizando uma função de decisão (envolvendo o modelo ajustado). Este segundo processo é conhecido como fase de teste do modelo ajustado.

No decorrer do trabalho, iremos considerar a matriz $X \in \mathbb{R}^{n\times p}$ como a matriz de observações, também denominada como matriz de dados de entrada, sendo que suas linhas representam as n observações extraídas levando em consideração p características/atributos do problema. Assim, o elemento $x_{ij} \in X$ representa o valor da j-ésima variável da observação i, para $j = 1, 2, \ldots, p$ e $i = 1, 2, \ldots, n$. Os índices i e j serão utilizados para representar as observações e variáveis, respectivamente.

Algumas vezes será preciso indicar linhas da matriz X. Para isso, a notação

utilizada será o vetor $x_i \in \mathbb{R}^p$, ou seja,

$$x_i = \begin{pmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ip} \end{pmatrix}.$$

Desta forma, a matriz X poderá ser escrita como

$$X = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{pmatrix}.$$

Será utilizado y_i para indicar a resposta obtida para a i-ésima observação x_i. Assim, para um conjunto de n observações, teremos um vetor $y \in \mathbb{R}^n$ sendo

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}.$$

Ainda, $y_i \in \{-1, 1\}$ devido ao fato de SVM ser fundamentado em classificação binária. Então, quando a resposta for $y_i = 1$, diremos que a observação respectiva x_i pertence à classe positiva. Quando $y_i = -1$, a observação x_i pertencerá à classe negativa.

Assim, um conjunto de treinamento para o ajuste do modelo será da forma

$$\text{TS} = \{(x_i, y_i), x_i \in \mathbb{R}^p, y_i \in \{-1, 1\}, i = 1, 2, \dots, n\}. \tag{2.1}$$

Por fim, os símbolos $\langle \cdot, \cdot \rangle$ e $|| \cdot ||$ indicará o produto interno e a norma Euclidiana, respectivamente. Caso seja necessário descrever símbolos que são semelhantes a estes, os mesmos serão destacados no texto.

2.2 HIPERPLANO SEPARADOR ÓTIMO E SVM LINEAR

Neste primeiro momento, será tratado o conceito de hiperplano separador, em que é possível separar linearmente o conjunto de dados de entrada. Na sequência, serão definidos conceitos imprescindíveis para a descrição da teoria de classificador de margem máxima.

Considere os conjuntos disjuntos X_+ e X_- de pontos em $\mathbb{R}^p$ formados pelos dados de entrada que pertencem às classes positivas e negativas, respectivamente, definidos como

$$X_+ = \{x_i \in X, y_i = 1\} \text{ e } X_- = \{x_i \in X, y_i = -1\}.$$

Para o entendimento do hiperplano separador, formaliza-se a seguir uma definição de conjunto linearmente separável.

Definição 2.1. *Os conjuntos $X_+, X_- \in \mathbb{R}^p$ serão linearmente separáveis quando existirem $w \in \mathbb{R}^p$ e $b \in \mathbb{R}$ tais que: $w^T x + b > 0$, para todo $x \in X_+$; $w^T x + b < 0$, para todo $x \in X_-$.*

De acordo com a Definição 2.1, ao considerar também que X_+ e X_- são conjuntos linearmente separáveis, isto é, existem hiperplanos na forma $H = \{x \in \mathbb{R}^p, w^T x + b = 0\}$ separando os conjuntos X_+ e X_-, sendo $w \in \mathbb{R}^p$ um vetor normal não nulo e $b \in \mathbb{R}$ um escalar, é possível verificar que qualquer um destes hiperplanos H dividirão o espaço $\mathbb{R}^p$ em dois semiespaços, S_+ e S_-, em que

$$S_+ = \{x \in \mathbb{R}^p, w^T x + b \geq 0\} \text{ e } S_- = \{x \in \mathbb{R}^p, w^T x + b \leq 0\}.$$

Desta forma, caso os conjuntos X_+ e X_- sejam finitos e linearmente separáveis por um hiperplano H, existirá um determinado vetor $\bar{w} \in \mathbb{R}^p$ e um escalar $\bar{b} \in \mathbb{R}$ que definem o hiperplano separador de forma que as aplicações de quaisquer vetores $x \in \mathbb{R}^p$ no hiperplano recaem em

$$\begin{aligned} &\bar{w}^T x + \bar{b} \geq 1, \text{ para qualquer vetor } x \in X_+ \text{ e} \\ &\bar{w}^T x + \bar{b} \leq -1, \text{ para qualquer vetor } x \in X_-. \end{aligned}$$

Sabendo que $y_i \in \{-1, 1\}$, podemos reescrever como

$$y_i(\bar{w}^T x_i + \bar{b}) \geq 1, i = 1, 2, \ldots, n. \tag{2.2}$$

Por simplicidade, iremos denotar os elementos do hiperplano separador H, o vetor $\bar{w}$ e escalar $\bar{b}$, apenas como vetor w e escalar b, respectivamente. Assim, temos que os hiperplanos $M_+ = \{x \in \mathbb{R}^p, w^T x + b = 1\}$ e $M_- = \{x \in \mathbb{R}^p, w^T x + b = -1\}$ serão os hiperplanos que definirão as margens que separam os conjuntos X_+ e X_-. A Figura 2 ilustra um exemplo no $\mathbb{R}^2$ da discussão até o momento.

Mesmo que seja definido um hiperplano separador H, deve-se atentar a importância da definição deste hiperplano separador com *margem máxima* entre M_+ e M_-, com o intuito de reduzir a classificação errônea de novos dados. É a partir desta motivação que inicia-se a determinação de um *hiperplano separador ótimo.*

Definição 2.2. *Dados dois conjuntos $X_+, X_- \in \mathbb{R}^p$ linearmente separáveis, o hiperplano separador ótimo é o hiperplano separador $H = \{x \in \mathbb{R}^p, w^T x + b = 0\}$ que possui a máxima margem.*

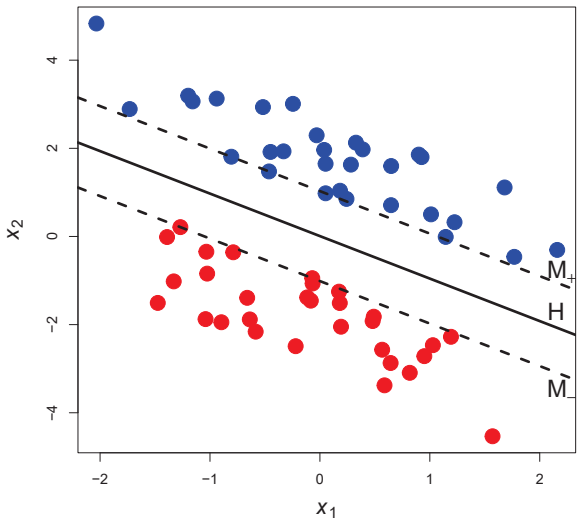
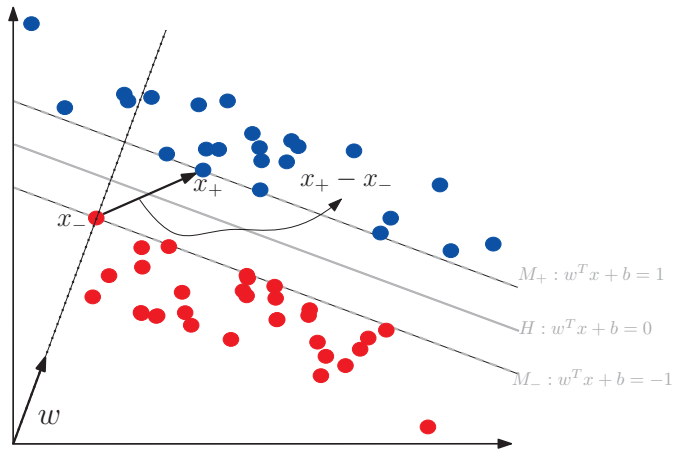


FIGURA 2 – Exemplo de um hiperplano separador.

De acordo com Krulikovski (2017) e Chauhan, Dahiya e Sharma (2019), a margem de um hiperplano separador é a mínima distância d entre pontos de $X_+ \cup X_-$ e o hiperplano separador H. Por esta definição, conseguimos determinar o valor da distância entre os hiperplanos M_+ e M_-. Considerando os vetores $x_+ \in M_+$ e $x_- \in M_-$, conseguimos determinar o valor da distância entre os hiperplanos M_+ e M_- via projeção do vetor $x_+ - x_-$ no vetor w. Este valor é dado por $\frac{2}{||w||}$ (KRULIKOVSKI, 2017). A Figura 3 ilustra a discussão.



$$
\begin{aligned}
d &= ||proj_w(x_+ - x_-)|| \\
&\vdots = \frac{||w^T(x_+ - x_-)w||}{||w||^2} \\
&\vdots = \frac{1}{||w||}(1 - b + 1 + b) \\
d &= \frac{2}{||w||}
\end{aligned}
$$

FIGURA 3 – Distância entre as margens e seu valor.

Desta forma, para obter o maior valor de distância/margem d possível entre M_+ e M_-, deseja-se encontrar o menor valor possível de $||w||$. Logo, um hiperplano separador ótimo pode ser obtido por meio da solução do seguinte problema quadrático convexo,

$$
\begin{aligned}
&\underset{w,b}{\text{minimizar}} && \frac{1}{2}||w||^2 \\
&\text{sujeito a} && y_i(w^T x_i + b) \geq 1, \quad i = 1, 2, \ldots, n.
\end{aligned}
\tag{2.3}
$$

Teorema 2.1. *Se o conjunto de dados de treinamento* (2.1) *é linearmente separável, então o problema* (2.3) *possui um único minimizador global.*

Demonstração. *Ver Teoremas 2.5, Lema 2.6 e Teorema 2.7 de Krulikovski (2017) ou Apêndice A de Piccialli e Sciandrone (2018).* ■

Pelo Teorema 2.1, temos que o problema (2.3) possui solução global e única (w^*, b^*), uma vez que a função objetivo deste problema é convexa e limitada inferiormente pelo conjunto de restrições (não vazio). Desta forma, definido um hiperplano separador otimizado w^* e b^*, podemos classificar uma nova observação $x \in \mathbb{R}^p$ ao empregar uma função de classificação $f : \mathbb{R}^p \to \mathbb{R}$ definida por

$$f(x) = \text{sinal}(w^{*T}x + b^*) = \begin{cases} +1, & \text{se } w^{*T}x + b^* \geq 0 \\ -1, & \text{se } w^{*T}x + b^* < 0 \end{cases}. \tag{2.4}$$

Agora, no caso em que os conjuntos X_+ e X_- não são linearmente separáveis, isto é, os conjuntos X_+ e X_- deixam de ser disjuntos, então as inequações (2.2) não admitem solução. Como consequência, o problema (2.3) não forneceria região viável. Por essa razão, são introduzidas variáveis de folgas $\xi_i \geq 0$, com $i = 1, 2, \ldots, n$, para todas as inequações (2.2)

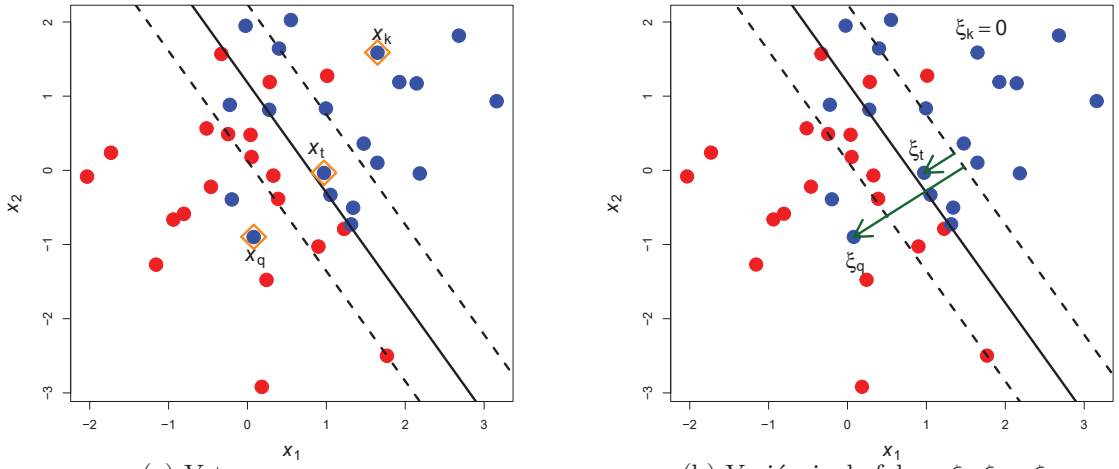
$$y_i(w^T x_i + b) + \xi_i \geq 1, \ i = 1, 2, \ldots, n. \tag{2.5}$$

Ao adicionar esta variável de folga, é possível encontrar um hiperplano que não separe linearmente perfeitamente os dados em duas classes, mas que forneça a melhor separação para a *maioria* dos dados – isto é, permita que ocorram erros de classificação de algumas observações em troca de conseguir classificar corretamente a maioria dos dados.

Observe que para o caso em que a observação x_i não é classificada corretamente, a variável de folga correspondente ξ_i é maior que 1. As variáveis ξ_i correspondentes às observações corretamente classificadas, porém que violam a margem de separação, assumem valores $0 < \xi_i < 1$. No caso em que a observação é classificada corretamente, a variável ξ_i assume valor nulo. Para ilustrar a discussão, a Figura 4 destaca um conjunto de dados não separável linearmente com destaque em 3 observações correspondentes aos casos citados e as suas respectivas variáveis de folga.

Considerando que variáveis de folga ξ_k, ξ_t, ξ_q que permitam o mínimo de violações, como representado na Figura 4, pode-se concluir que:

- a variável ξ_k assumirá o valor nulo, uma vez que o vetor x_k se encontra localizado no semiespaço positivo e distante da margem do hiperplano ótimo;
- a variável ξ_t assumirá um valor entre 0 e 1, pois o vetor x_t encontra-se no semiespaço positivo localizado entre a margem e o hiperplano separador;



(a) Vetores x_k,x_t e x_q. (b) Variáveis de folgas ξ_k,ξ_t e ξ_q.

FIGURA 4 – Comportamento das variáveis de folga num exemplo.

- a variável ξ_q assumirá um valor maior que 1, já que o vetor x_q encontra-se no semiespaço negativo. Assim, x_q estará do lado errado do hiperplano – lembrando que sua classificação é positiva.

Desta forma, o termo $\sum_{i=1}^{n} \xi_i$ é um limitante superior quanto ao erro durante o treinamento do classificador. Logo, torna-se natural acrescentar à função objetivo do problema (2.3) a penalidade $\mathcal{C} \sum_{i=1}^{n} \xi_i$, sendo $\mathcal{C} > 0$ o parâmetro de permissividade de erro. Assim, para o caso em que o conjunto de dados não é linearmente separável, o problema primal quadrático convexo terá a forma

$$\begin{aligned} \underset{w,b,\xi}{\text{minimizar}} \quad & \frac{1}{2}||w||^2 + \mathcal{C}\sum_{i=1}^{n} \xi_i \\ \text{sujeito a} \quad & y_i(w^T x_i + b) + \xi_i \geq 1, \quad i = 1, 2, \ldots, n \\ & \xi_i \geq 0, \qquad\qquad i = 1, 2, \ldots, n. \end{aligned} \tag{2.6}$$

Teorema 2.2. *Se um conjunto de dados de treinamento* (2.1) *for não separável linearmente, então o problema* (2.6) *possui minimizador global.*

Demonstração. *Ver Teorema 2.12 em Krulikovski (2017) ou Teorema 2.3.3 em Deng, Tian e Zhang (2012).* ■

Por razões que serão explicadas no decorrer do capítulo, a formulação dual do problema (2.6) também é considerada. Para mais informações sobre a teoria de Dualidade para problemas de Programação não Linear, sugere-se ao leitor o livro de Luenberger, Ye et al. (1984), bem como o livro de Nocedal e Wright (2006) para condições de otimalidade de Karush-Kuhn-Tucker (KKT).

A função Lagrangiana do problema (2.6), $L : \mathbb{R}^p \times \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, é dada por

$$L(w, b, \xi, \alpha, \beta) = \frac{1}{2}||w||^2 + \mathcal{C}\sum_{i=1}^{n}\xi_i + \sum_{i=1}^{n}\alpha_i(1 - \xi_i - y_i(w^T x_i + b)) - \sum_{i=1}^{n}\beta_i\xi_i,$$

sendo $\alpha \in \mathbb{R}^n_+$ e $\beta \in \mathbb{R}^n_+$, os multiplicadores de Lagrange relacionados aos conjunto de restrições de (2.6). Conforme descrito em Krulikovski (2017), a partir da condição de otimalidade de primeira ordem $\nabla L = 0$, em relação às variáveis primais, obtêm-se as seguintes relações,

$$w = \sum_{i=1}^{n}\alpha_i y_i x_i, \qquad \sum_{i=1}^{n}\alpha_i y_i = 0.$$

Partindo dos conceitos da teoria de Dualidade adotada e manipulações algébricas, a formulação dual do problema (2.6) recai no problema quadrático convexo

$$\begin{aligned} \underset{\alpha}{\text{minimizar}} \quad & \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{n}\alpha_i\alpha_k y_i y_k x_i^T x_k - \sum_{i=1}^{n}\alpha_i \\ \text{sujeito a} \quad & \sum_{i=1}^{n}\alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq \mathcal{C},\ i = 1, 2, \ldots, n \end{aligned} \tag{2.7}$$

no qual $\alpha \in \mathbb{R}^n$ e $\mathcal{C} > 0$. O problema (2.7) possui solução, como descrito no Teorema 2.3.

Teorema 2.3. *Considere o conjunto de dados de treinamento* (2.1)*. O problema* (2.7) *possui minimizador global e gap de Dualidade nulo.*

Demonstração. *Ver Teorema 2.18 de Krulikovski (2017).* ■

Uma vez que o minimizador ótimo α^* do problema (2.7) é determinado, o vetor primal w^* é dado por

$$w^* = \sum_{i=1}^{n}\alpha_i^* y_i x_i.$$

Logo, é perceptível que o vetor w^* depende das observações x_i que correspondem às componentes do vetor de multiplicadores de Lagrange, α_i^*, não nulos. Esta informação fornece a base para introduzir o conceito de *vetor suporte*, ficando clara a nomenclatura *SVM linear*. Vetor suporte é um termo que se refere às observações de treinamento que estão mais próximas da fronteira de decisão, ou seja, o hiperplano que separa as diferentes classes de dados. Esses vetores suporte desempenham um papel fundamental na SVM, pois são eles que determinam a localização e a orientação do hiperplano separador. Os vetores suporte influenciam na definição da margem do classificador e são essenciais para

determinar a capacidade de generalização do modelo. Durante a etapa de treinamento de SVM, o algoritmo seleciona os vetores suporte relevantes e os utiliza para construir o modelo final.

Qualquer observação x_i cujo multiplicador associado α_i^* é não nulo é relevante para a obtenção do classificador linear, pois os vetores suporte serão necessários para definir as margens do classificador. Por sua vez, qualquer observação x_k cujo multiplicador associado α_k^* seja nulo pode ser retirada do conjunto X uma vez que esta não altera o classificador linear – para mais informações, ver Cristianini, Shawe-Taylor et al. (2000) e Shawe-Taylor e Cristianini (2004). Porém, os vetores suporte não serão definidos de uma forma inerente aos dados do problema, porque existe uma dependência da solução do problema dual.

Bishop (2006) e Krulikovski (2017) destacam que uma solução otimizada (w^*, b^*) satisfaz a condição de complementaridade com os multiplicadores α^*. Assim, considerando qualquer vetor suporte x_i com $0 < \alpha_i^* < \mathcal{C}$, temos que

$$y_i((w^*)^T x_i + b^*) - 1 = 0, \tag{2.8}$$

implicando na determinação de um escalar b^* após o cálculo de w^*. Como podem existir mais observações cujos multiplicadores $0 < \alpha_k^* < \mathcal{C}$, é comum computar b^* pela média de todos os valores correspondente das condições de complementaridade (2.8).

Por fim, a função de decisão (2.4) para a SVM linear pode ser adaptada como

$$f(x) = \text{sinal}(w^{*T} x + b^*) = \text{sinal}\left(\sum_{i=1}^{n} \alpha_i^* y_i x_i^T x + b^*\right) = \begin{cases} +1, & \text{se } \sum_{i=1}^{n} \alpha_i^* y_i x_i^T x + b^* \geq 0 \\ -1, & \text{se } \sum_{i=1}^{n} \alpha_i^* y_i x_i^T x + b^* < 0 \end{cases}. \tag{2.9}$$

Comparando a formulação do problema primal (2.6) e o problema dual (2.7), observa-se que o problema dual recai em um caminho conveniente para lidar com as restrições. Outros fatores podem ser destacados, tais como o problema de Otimização dual poder ser descrito em função de produtos internos e a eficiência em ganho de memória da função de classificação. Estas vantagens apresentadas com a formulação dual facilitam a adaptação da SVM para um classificador não linear, tornando a SVM mais versátil para dados com atributos não linearmente relacionados.

2.3 EXTENSÃO PARA O CASO NÃO LINEAR

A ideia empregada na SVM não linear é mapear os dados do espaço original em um outro espaço de dimensão mais alta, denominado espaço de características, com o

objetivo de classificar os dados mapeados no espaço de características utilizando o conceito de hiperplano separador.

Assim, será considerada uma função $\phi : \mathbb{R}^p \to \mathcal{H}$, sendo $\mathcal{H}$ um espaço de Hilbert com dimensão maior que p, para realizar o mapeamento de todas observações $x_i \in \mathbb{R}^p$, $i = 1, 2, \ldots, n$, do espaço original de entrada para um novo espaço de características $\mathcal{H}$.

Então, pode-se pensar que a SVM não linear é o uso de SVM linear no espaço de características (espaço de dimensão mais alta) utilizando a observação transformada $\phi(x_i)$.

A partir do emprego da função de mapeamento ϕ nas observações, a construção do hiperplano separador ótimo, seja para a formulação primal (2.6) ou para a formulação dual (2.7), mantém-se análoga, alterando apenas a observação x_i para $\phi(x_i)$. Porém, não é atrativo o uso do mapeamento ϕ no problema primal (2.6), pois é necessário conhecer a função ϕ e construir o espaço de dimensão mais alta que o original, implicando no aumento do custo computacional do método. Por outro lado, observe que aplicando a ideia de mapeamento nos dados para o problema dual (2.7), temos que:

- o problema (2.7) é substituído pelo seguinte problema,

$$\begin{aligned} \underset{\alpha}{\text{minimizar}} \quad & \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{n}\alpha_i\alpha_k y_i y_k \phi(x_i)^T\phi(x_k) - \sum_{i=1}^{n}\alpha_i \\ \text{sujeito a} \quad & \sum_{i=1}^{n}\alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq \mathcal{C},\ i = 1, 2, \ldots, n; \end{aligned} \tag{2.10}$$

- o vetor ótimo primal w^* é definido como

$$w^* = \sum_{i=1}^{n}\alpha_i^* y_i \phi(x_i);$$

- dado w^* e qualquer vetor suporte satisfazendo $0 < \alpha_i^* < \mathcal{C}$, o escalar b^* pode ser obtido pela seguinte condição de complementaridade,

$$y_i\left(\sum_{k=1}^{n}\alpha_k^* y_k \phi(x_k)^T\phi(x_i) + b^*\right) = 1; \text{ e}$$

- a função de decisão é definida como

$$f(x) = \text{sinal}((w^*)^T\phi(x) + b^*), \tag{2.11}$$

  o que implica que a superfície de separação é um *hiperplano no espaço de características* $\mathcal{H}$ e *uma superfície não linear no espaço original* $\mathbb{R}^p$.

Note que na função objetivo do problema (2.10), existem produtos internos entre as observações x_i e x_k, a saber $\langle \phi(x_i), \phi(x_k) \rangle$. Por ser um produto interno entre dois vetores, o valor resultante indicará o quão próximos estão os vetores. É devido ao aparecimento destes produtos internos que se elabora o *truque do kernel*, uma vez que se pode trocar o referido produto interno por uma função *kernel* – que calculará os produtos internos no espaço de características – aplicada aos mesmos pontos x_i e x_k, evitando a construção explícita de um espaço com dimensão mais alta. É a partir desta situação que a SVM não linear utilizando a formulação dual é mais atrativa em detrimento à formulação primal com o mapeamento.

Para isso, assim como (KRULIKOVSKI, 2017), define-se função *kernel*.

Definição 2.3. *Seja $\mathcal{X} \subset \mathbb{R}^p$. A função $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ é denominada kernel caso exista um mapeamento ϕ do conjunto $\mathcal{X}$ para um espaço de Hilbert $\mathcal{H}$, ou seja, $\phi : \mathcal{X} \to \mathcal{H}$, de forma que*

$$K(a, b) = \langle \phi(a), \phi(b) \rangle,$$

para todo $a, b \in \mathcal{X}$.

Vale ressaltar que a função *kernel* é necessariamente simétrica. Além disso, a partir do Teorema de Mercer garante-se que K é uma função *kernel* se, e somente se, a seguinte matriz $n \times n$,

$$\begin{pmatrix} K(x_1, x_1) & K(x_1, x_2) & \dots & K(x_1, x_n) \\ K(x_2, x_1) & K(x_2, x_2) & \dots & K(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ K(x_n, x_1) & K(x_n, x_2) & \dots & K(x_n, x_n) \end{pmatrix},$$

é semidefinida positiva para qualquer conjunto de treino (2.1). Para mais detalhes acerca desta afirmação, sugere-se a leitura dos trabalhos de Scholkopf e Smola (2001), Shawe-Taylor e Cristianini (2004), Krulikovski (2017) e Piccialli e Sciandrone (2018).

Desta forma, a partir da definição da função *kernel*, o problema (2.10) pode ser reescrito como um problema quadrático convexo como se segue,

$$\begin{aligned} \underset{\alpha}{\text{minimizar}} \quad & \frac{1}{2} \sum_{i=1}^{n} \sum_{k=1}^{n} \alpha_i \alpha_k y_i y_k K(x_i, x_k) - \sum_{i=1}^{n} \alpha_i \\ \text{sujeito a} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq \mathcal{C},\ i = 1, 2, \dots, n \end{aligned}$$

ou na forma matricial,

$$\begin{array}{ll} \underset{\alpha}{\text{minimizar}} & \frac{1}{2}\alpha^T P\alpha - e^T\alpha \\ \text{sujeito a} & y^T\alpha = 0 \\ & 0 \leq \alpha \leq \mathcal{C}, \end{array} \tag{2.12}$$

sendo $P \in \mathbb{R}^{n\times n}$ a matriz Hessiana simétrica, semidefinida positiva e geralmente densa, cujo elementos são computados na forma

$$P_{ik} = y_i y_k K(x_i, x_k),$$

o vetor de uns $e \in \mathbb{R}^n$ e o parâmetro $\mathcal{C} \in \mathbb{R}_+^*$.

Algumas das funções *kernel* mais utilizadas na literatura para SVM, além de outros modelos de Aprendizagem de Máquinas e reconhecimento de padrões, são descritas na Tabela 1. Vale a ressalva de que essas funções respeitam o Teorema de Mercer. Pela Tabela 1, a escolha da função *kernel* como linear faz com que a obtenção dos multiplicadores de Lagrange recaia exatamente no problema (2.7). Além disso, a função *kernel* Gaussiana também é conhecida como *Radial Basis Function* (RBF). Para mais informações, indica-se Scholkopf e Smola (2001).

TABELA 1 – Algumas funções *kernel*.

Nome	**Função** $K(x,z)$	**Parâmetros**
Linear	$x^T z$	-
Polinomial	$(x^T z + p)^d$	grau $d \in \mathbb{N}$, $p \in \mathbb{R}_+$
RBF	$\exp\left(-\sigma\|\|x - z\|\|^2\right)$	$\sigma \in \mathbb{R}_+$
Exponencial	$\exp(\sigma x^T z)$	$\sigma \in \mathbb{R}_+$
Sigmoidal	$\tanh(\kappa x^T z + \theta)$	κ e $\theta < 0$

Por fim, com a definição da função *kernel* podemos descrever a seguinte função de decisão. Vale destacar que esta função de decisão é eficiente em memória, por considerar apenas os vetores suporte na sua construção.

$$\begin{aligned} f(x) &= \text{sinal}((w^*)^T\phi(x) + b^*) \\ &= \text{sinal}\left(\sum_{i=1}^{n} \alpha_i^* y_i K(x_i, x) + b^*\right) \\ &= \begin{cases} +1, & \text{se } \sum\limits_{\alpha_i^*>0} \alpha_i^* y_i K(x_i, x) + b^* \geq 0 \\ -1, & \text{se } \sum\limits_{\alpha_i^*>0} \alpha_i^* y_i K(x_i, x) + b^* < 0 \end{cases}, \end{aligned} \tag{2.13}$$

para qualquer $x \in \mathbb{R}^p$.

2.4 ALGORITMOS PARA TREINAMENTO EM SVM

A partir das publicações de Boser, Guyon e Vapnik (1992) e Cortes e Vapnik (1995), diversas estratégias para treinar um classificador por SVM foram publicadas. Estas estratégias são descritas por meio de algoritmos, os quais dependem de alguns fatores para gerar soluções eficazes.

De acordo com Chauhan, Dahiya e Sharma (2019), pode-se descrever alguns fatores para escolher um algoritmo para SVM. As propriedades da formulação do problema, no âmbito da função objetivo ser convexa ou não, função objetivo ser ou não suave, ser um problema restrito ou irrestrito ou até mesmo resolvido em sua formulação primal ou dual. As propriedades com respeito ao conjunto de dados, relacionadas a um conjunto com mais observações do que características ou vice-versa, um conjunto esparso, entre outras propriedades. E, por fim, o que se espera com o algoritmo, isto é, qual métrica é mais importante para avaliar um algoritmo para treinamento.

Apesar de existirem várias metodologias para resolver o problema (2.10), todas possuem desafios a serem superados. Para o problema de treinamento escolhido para este trabalho, a principal dificuldade é o armazenamento da matriz Hessiana da função objetivo, geralmente densa, para treinamento de grandes conjuntos de dados. Por outro lado, se o problema for de grandes dimensões, metodologias que lidem com essa questão podem ser úteis para gerar resultados satisfatórios. É com base neste argumento que esta pesquisa se fundamenta.

Piccialli e Sciandrone (2018), Chauhan, Dahiya e Sharma (2019) e Carrizosa e Morales (2013) destacam que, tipicamente, o problema de treinamento de SVM (2.10) é resolvido por algoritmos baseados em Pontos Interiores ou Restrições Ativas. De acordo com Carrizosa e Morales (2013) e Scheinberg, Bennett e Parrado-Hernández (2006), os classificadores obtidos via algoritmos de Pontos Interiores são bons em acurácia, isto é, conseguem generalizar o classificador para dados não utilizados no treinamento. Porém, isso ocorre às custas de tempo computacional e memória de armazenamento dos componentes do problema de Otimização. Por outro lado, Piccialli e Sciandrone (2018) destacam que os algoritmos de treinamento desta classe definem uma solução em uma quantidade de iterações quase independente da dimensão do problema. Ferris e Munson (2002), Woodsend e Gondzio (2009), Woodsend e Gondzio (2011) são exemplos de trabalhos cujos algoritmos são baseados em Pontos Interiores aplicados ao problema (2.10) com função *kernel* linear. Trabalhos como o de Serafini, Zanghirati e Zanni (2005) e Cores et al. (2009) aplicam o método de Gradiente Espectral Projetado com busca não monótona para o treinamento de um classificador, enquanto Mangasarian e Musicant (2001) e Niu et al. (2019) aplicam algoritmos envolvendo a função Lagrangiana. Devido ao problema de armazenamento, trabalhos como o de Woodsend e Gondzio (2009) destacam implementações em

paralelo envolvendo o método de Pontos Interiores.

Diferentemente da classe de algoritmos de Pontos Interiores, os da classe de Restrições Ativas tendem a remediar o problema de armazenamento/cache das matrizes ao realizar uma decomposição do problema. Desta forma, estes algoritmos podem ser aplicados a problemas de dimensões altas. De acordo com Scheinberg, Bennett e Parrado-Hernández (2006), Zhang, Wang e Wang (2018), a abordagem de um método de Restrições Ativas parte de uma solução inicial e, iterativamente, se move a pontos viáveis a fim de encontrar uma solução. Estas novas soluções são obtidas a partir da solução do problema dual reduzido, definido a partir de um subconjunto de variáveis denominado conjunto de trabalho (*working set*). Por ser uma estratégia simples de implementar, existem vários trabalhos envolvendo algoritmos de decomposição (ver, por exemplo, Platt (1998), Joachims (1998), Osuna, Freund e Girosit (1997), Keerthi et al. (2001), Hsu e Lin (2002), Fan et al. (2008), Chang e Lin (2011), Glasmachers e Igel (2008), Lucidi et al. (2009), Gonzalez-Lima, Hager e Zhang (2011), Zanghirati e Zanni (2003), Serafini e Zanni (2005)), tendo como principais diferenças a regra de seleção e atualização do conjunto de trabalho.

Em Zhang, Wang e Wang (2018), os autores realizam um levantamento bibliográfico sobre algoritmos de Restrições Ativas, em especial envolvendo estudos e o aprimoramento dos algoritmos $\text{SVM}^{\text{light}}$ e SMO, publicados originalmente por Joachims (1998) e Platt (1998), respectivamente. Em síntese, enquanto $\text{SVM}^{\text{light}}$ decompõe o problema (2.10) em q variáveis para o conjunto de trabalho, sendo q um número par, o algoritmo de SMO realiza a decomposição do problema em apenas duas variáveis. Dentre estas duas estratégias, o algoritmo de SMO não requer um procedimento de Otimização, sendo a solução dos subproblemas decompostos obtidas de forma analítica, além de ser possível provar a convergência do algoritmo, fatores não totalmente satisfeitos pelo $\text{SVM}^{\text{light}}$. Não iremos nos aprofundar nas referências sobre tais algoritmos e as análises de convergência por estarem apresentadas e descritas em Zhang, Wang e Wang (2018) e Piccialli e Sciandrone (2018) e nas referências utilizadas.

Softwares de classificação do estado da arte utilizando SVM incluem alguns dos algoritmos já citados aqui. Porém, atualmente é consenso que, nas referências estudadas, o algoritmo mais difundido de treinamento do problema (2.10) é o LIBSVM, proposto por Fan et al. (2005) e implementado em Chang e Lin (2011). Fan et al. (2005) destaca que este algoritmo é baseado em SMO, e utiliza a estratégia de seleção do conjunto de trabalho com informações de segunda ordem do problema, sendo uma extensão do método de Platt (1998). O SMO é considerado um método eficiente, em termos do custo computacional, e robusto, em termos de classificador gerado, o que são pontos favoráveis para tal difusão. Vale destacar que este algoritmo é aplicado nas ferramentas de SVM em pacotes amplamente utilizados em Aprendizado de Máquina, tais como `scikit-learn` (BUITINCK et al., 2013) e `e1071` (MEYER et al., 2019), o que torna o LIBSVM o

algoritmo referência.

Finalmente, como já destacado, o treinamento SVM para classificação binária pode ser formulado também através do problema primal, com restrições ou não (CHAUHAN; DAHIYA; SHARMA, 2019; PICCIALLI; SCIANDRONE, 2018). Assim, existem outras estratégias já publicadas que também merecem destaques, tais como Sub-gradientes Estocástico (SHALEV-SHWARTZ et al., 2011) e métodos *Semi-Smooth* (FERRIS; MUNSON, 2004; YAN; LI, 2020; YIN; LI, 2019). O presente texto não irá se aprofundar neste tópico por estar fora do escopo do trabalho.

2.5 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Neste capítulo foi apresentado o método de SVM, ilustrando suas ideias e destacando suas formulações. Em um problema de classificação binária, a essência deste método é definir um classificador de máxima margem entre os vetores suporte que são fundamentais para o desenvolvimento dos métodos de SVM, uma vez que são os pontos do espaço amostral mais importantes para construir o classificador. Em um aspecto geral, o método de SVM pode ser considerado bem flexível quanto à entrada de dados (quanto as hipóteses do conjunto de dados para ajuste), entretanto decai em interpretabilidade dos atributos pertinentes ao modelo de classificação à medida em que o problema torna-se não linear.

As formulações apresentadas neste capítulo são de abordagem restrita, considerando variáveis de folga. Também foram descritas as formulações primais e duais, discutindo a aplicação de funções de mapeamento e o efeito do truque do *kernel* para os casos não lineares. Neste trabalho, a pesquisa se baseará na abordagem dual do problema de SVM.

3 DESCRIÇÃO DOS MÉTODOS DE OTIMIZAÇÃO UTILIZADOS

No levantamento bibliográfico realizado durante o desenvolvimento deste material, sentimos falta de trabalhos que comparassem a performance de algoritmos de Otimização, na literatura, aplicados ao problema de treinamento (2.12). Realizar esta comparação quanto à performance destes algoritmos de Otimização, que foram desenvolvidos para resolução de problemas gerais de Programação não Linear, é relevante, pois é importante verificar a competitividade dos algoritmos, isto é, se são capazes de encontrar soluções melhores, mais rápidas ou mais eficientes em relação a outros algoritmos disponíveis. Vale ressaltar que a nossa contribuição é na análise e aplicação de algoritmos de Otimização restritos, com boas propriedades de convergência, ao *framework* de SVM, focando numa implementação eficiente.

Os algoritmos utilizados nas comparações são descritos a seguir. Não entraremos em detalhes específicos acerca da formulação dos métodos, uma vez que são algoritmos consagrados da literatura de Programação não Linear. Os métodos estudados variam de algoritmos de projeções, tais como Gradiente Projetado e Gradiente Espectral Projetado, Restrições Ativas, Lagrangiano Aumentado, Pontos Interiores e Filtro.

3.1 MÉTODOS BASEADOS EM RESTRIÇÕES ATIVAS

De acordo com Luenberger, Ye et al. (1984), a principal ideia de métodos de Restrições Ativas é particionar as restrições do problema de Otimização em dois grupos, sendo um tratado como restrições ativas e outro como não ativas. O livro de Luenberger, Ye et al. (1984) descreve que este método permite reduzir o número de restrições a serem consideradas na resolução do problema, o que pode levar a uma solução mais rápida e eficiente. No entanto, é importante ressaltar que esse método pode não ser a melhor opção em casos em que o conjunto de restrições é muito grande e complexo. Além disso, é importante destacar que o método de Restrições Ativas é um método de Otimização local, ou seja, ele pode não encontrar a solução ótima global.

O método de Restrições Ativas começa com um ponto viável, que satisfaz todas as restrições do problema. Em seguida, é feita uma análise de quais restrições são ativas e não ativas no ponto viável. A partir dessa análise, é gerada uma lista de restrições ativas e não ativas e, assim, é realizada uma redução do problema original para um problema menor, que consiste apenas nas restrições ativas. Como consequência, esta classe de métodos define a cada iteração um subproblema definido para um conjunto das restrições ativas, podendo este ser definido de diversas formas.

O próximo passo recai na resolução do problema menor, uma tarefa mais simples,

uma vez que o número de restrições foi reduzido. A solução encontrada para o problema menor é um ponto viável para o problema original. Se todas as restrições do problema original foram satisfeitas na igualdade da solução encontrada para o problema menor, então essa solução é a solução ótima. Caso contrário, são identificadas novas restrições ativas na solução encontrada para o problema menor e o processo é repetido até que a solução otimizada seja encontrada.

A seguir, o Algoritmo 1 descreve um pseudocódigo simplificado do algoritmo de Restrições Ativas.

Algoritmo 1: Pseudocódigo do método de Restrições Ativas.

1. Defina o ponto viável inicial x^0
2. Enquanto o critério de parada não for satisfeito, faça:
   a. Identifique as restrições ativas e não ativas na solução x^k
   b. Reduza o problema original para um problema menor, considerando apenas as restrições ativas
   c. Resolva o problema menor e organize a solução completa x^{k+1}
   d. Se todas as restrições do problema original foram satisfeitas na igualdade na solução x^{k+1}, então retornar x^{k+1} como a solução ótima
   e. Caso contrário, identifique novas restrições ativas na solução x^{k+1} e atualize o ponto viável inicial para x^{k+1}
3. Retorne como solução o x da última iteração realizada

Outras questões importantes a serem consideradas na aplicação do método de Restrições Ativas é a escolha do ponto viável inicial e a não linearidade das restrições. Em suma, o método de Restrições Ativas é uma técnica poderosa para a resolução de problemas de Otimização com restrições. Ele pode reduzir o número de restrições a serem consideradas e levar a soluções mais rápidas e eficientes. No entanto, é importante ter em mente suas limitações e cuidados necessários na sua aplicação.

No contexto de SVM, os principais algoritmos baseados em métodos de Restrições Ativas são de SVM$^{\text{light}}$ (JOACHIMS, 1998), que define q variáveis, q um número par pré-definido, por meio de um procedimento equivalente a um problema de Otimização Inteira, e SMO (PLATT, 1998), que define o conjunto de trabalho em apenas duas variáveis por meio de uma análise das condições de KKT (reduzidas) do problema de Otimização utilizando apenas informações de primeira ordem. Vale destacar também o trabalho de (FAN et al., 2005), que aplica a mesma ideia do SMO, porém utilizando informações de segunda ordem.

3.2 MÉTODOS BASEADOS EM PROJEÇÃO

A ideia de projeção de direções em métodos de Otimização foi formalizada por Rosen (1961), na qual foi adaptado o método de máxima descida para aplicações a proble-

mas irrestritos a problemas com restrições (de acordo com Luenberger, Ye et al. (1984), lineares de preferência). Desta forma, a direção utilizada por Rosen (1961) foi a do vetor gradiente da função objetivo.

Assim, um método baseado em direções viáveis visa, a cada iteração, provocar um decréscimo no valor da função objetivo mantendo a viabilidade. No trabalho de Rosen (1961), a ideia principal é projetar a direção do vetor gradiente da função objetivo sobre o espaço viável, ou seja, o conjunto de pontos que satisfazem as restrições do problema. Essa projeção é realizada para garantir que o ponto encontrado seja viável e que as restrições sejam satisfeitas, tanto para casos de restrições de igualdade quanto de desigualdade.

Em termos gerais, o método de Gradiente Projetado busca andar o máximo possível na direção sem perder a viabilidade. Este método computa, a cada iteração, uma direção a partir de uma matriz de projeção da direção, que é utilizada com uma busca para encontrar um novo ponto. No caso do método Gradiente Projetado, esta direção é o vetor gradiente, mas esta direção pode ser computada baseada na direção de Newton, de Barzilai-Borwein, ou qualquer outra direção viável. Este processo ocorre até que os critérios de parada sejam satisfeitos. No caso descrito por Rosen (1961) e Luenberger, Ye et al. (1984), este critério pode ser a solução do sistema de KKT do problema de Otimização.

O Algoritmo 2 apresenta um pseudocódigo simplificado de um algoritmo baseado em Gradiente Projetado, generalizado a uma direção de descida.

Algoritmo 2: Pseudocódigo do método baseado em projeção.

1. Defina a solução inicial x^0
2. Inicie o contador de iterações k
3. Enquanto o critério de parada não for satisfeito, faça:
   a. Calcule uma direção d^k e projete esta direção no espaço referente às restrições ativas, obtendo $\hat{d}^k$
   b. Caso $\hat{d}^k$ seja não nula, defina o tamanho do passo α e atualize a solução na forma $x^{k+1} = x^k + \alpha\hat{d}^k$
4. Retorne como solução o x da última iteração realizada

A depender da direção de busca d^k utilizada, pode ser necessária uma correção ou uma salvaguarda a fim de garantir as propriedades de convergência do algoritmo.

Uma das principais vantagens do método de Gradiente Projetado é a sua simplicidade e facilidade de implementação, o que torna esse método uma opção viável mesmo para problemas mais complexos. Além disso, o método também apresenta uma boa eficiência computacional e pode ser utilizado para resolver problemas de grande porte. No entanto, é importante ressaltar que a escolha da solução inicial é fundamental para o sucesso do método. Uma solução inicial inadequada pode levar a problemas de convergência ou a

soluções locais não ótimas.

Outro ponto importante é que o método de Gradiente Projetado é sensível à escolha do tamanho do passo utilizado na atualização da solução. Um tamanho de passo muito pequeno pode levar a um aumento no número de iterações necessárias para atingir a convergência, enquanto um tamanho de passo muito grande pode levar a problemas de convergência ou a soluções sub-ótimas.

Para mais detalhes sobre a análise de convergência, indica-se o artigo de Rosen (1961, Teo. 3) quando a direção adotada for o vetor gradiente. Existem trabalhos que adaptam outros tipos de direções viáveis para o algoritmo de projeções, sendo destacado o trabalho de Bertsekas (1982, Prop. 3) para o caso de utilizar a direção de Newton.

Outro método baseado em projeção e que possui um bom desempenho é o método de Gradiente Espectral Projetado. Proposto por Birgin, Martínez e Raydan (2000), o objetivo do método é encontrar a solução otimizada de um problema de Otimização não linear com restrições, utilizando uma combinação do método do gradiente com o método de projeção. Este algoritmo une o passo espectral de Barzilai-Borwein não-monótono com a ideia do método de gradiente projetado para problemas de Otimização convexos restritos.

O passo espectral de Barzilai-Borwein, uma fórmula fechada para funções quadráticas convexas, fornece o quanto deve ser caminhado na direção de máxima descida assegurando a viabilidade, para formar a direção viável da iteração. A partir dessa iteração, será computado o tamanho do passo e então um novo ponto. No caso de Birgin, Martínez e Raydan (2000), sugere-se uma busca linear não-monótona. Uma vez computado o novo ponto, atualiza-se o passo espectral e repete-se o processo até ser satisfeito um critério de parada.

O método Gradiente Espectral Projetado pode ser encarado como uma extensão do método de Gradiente Projetado, que consiste em atualizar a solução do problema utilizando uma combinação do vetor gradiente da função objetivo e da projeção da solução em um conjunto convexo de restrições. Uma das principais vantagens do método Gradiente Espectral Projetado é a sua capacidade de lidar com problemas de Otimização não linear com restrições complexas e não-convexas. Além disso, o algoritmo apresenta uma boa eficiência computacional e pode ser implementado de maneira simples e direta. No entanto, esse método também apresenta algumas limitações, como a sensibilidade à escolha dos parâmetros do algoritmo, como o tamanho do passo inicial, tolerância para o critério de parada e o número máximo de iterações.

A seguir, o Algoritmo 3 apresenta um pseudocódigo do algoritmo de Gradiente Espectral Projetado baseado em Birgin, Martínez e Raydan (2000).

O método de Gradiente Espectral Projetado possui teoremas que garantem a convergência para um ponto estacionário da função objetivo (ver Birgin, Martínez e Raydan

Algoritmo 3: Pseudocódigo do método de Gradiente Espectral Projetado.

1. Defina a solução inicial x^0
2. Inicie o contador de iterações k
3. Enquanto o critério de parada não for satisfeito, faça:
   a. Calcule o ponto obtido ao dar um passo unitário na direção oposta ao vetor gradiente
   b. Calcule a direção de busca, formada pela diferença entre a projeção no conjunto viável do ponto encontrado no item anterior e o ponto corrente
   c. Compute o tamanho do passo utilizando a direção computada
   d. Atualize a solução e atualize o passo espectral
   e. Atualize o contador de iterações k
4. Retorne como solução o x da última iteração realizada

(2000, Teo. 2.1 e 2.2)). No entanto, este método pode apresentar lentidão em problemas mal-condicionados, o que pode afetar negativamente a sua convergência.

3.3 MÉTODO DE PONTOS INTERIORES

O método de Pontos Interiores foi, inicialmente, desenvolvido para problemas de programação linear e, posteriormente estendido para problemas com função objetivo quadrática e para problemas de Programação não Linear gerais. Neste trabalho, será dado um enfoque ao método de Pontos Interiores para o caso de problemas de Otimização quadrático com restrições lineares.

Este método é um dos mais eficientes e amplamente utilizados para resolver problemas de Otimização convexa. A sua ideia se baseia em encontrar um ponto viável e, em seguida, mover-se em direção ao interior do conjunto de restrições. O método utiliza uma função de barreira para penalizar as soluções que violam as restrições e, ao mesmo tempo, incentivar as soluções próximas às restrições.

Para isso, o método utiliza uma série de iterações que atualizam a solução a cada passo. Cada iteração é composta por duas etapas principais: cálculo da direção de busca e atualização da solução. Na primeira etapa, a direção de busca é determinada pelo cálculo do vetor gradiente da função de barreira e da matriz Hessiana da função objetivo. A direção de busca é a solução do sistema linear resultante. Já na segunda etapa, a solução é atualizada movendo-se na direção de busca determinada na primeira etapa. O tamanho do passo é determinado por meio de variadas técnicas, tal como busca linear.

A seguir, o Algoritmo 4 apresenta um pseudocódigo simplificado do algoritmo de Pontos Interiores.

O método de Pontos Interiores, independente de ser aplicado a problemas de Otimização Linear ou não, possui garantia de convergência, como destacado em Noce-

Algoritmo 4: Pseudocódigo do método de Pontos Interiores.

1. Defina a solução inicial x^0 e o parâmetro de regularização μ
2. Inicie o contador de iterações e a função de barreira ϕ
3. Enquanto o critério de parada não for satisfeito, faça:
   a. Obtenha a direção de busca d resolvendo o sistema linear $(H + \mu * I) * d = -g$, sendo H a matriz Hessiana da função objetivo, g o vetor gradiente, I é a matriz identidade e μ é o parâmetro de controle
   b. Determine o tamanho do passo α_k
   c. Atualize a solução x^{k+1} e a função de barreira ϕ
   d. Atualize o parâmetro de controle μ
   e. Incremente o contador de iterações
4. Retorne como solução o x da última iteração realizada

dal e Wright (2006, Cap. 14, 16 e 19) e Wright (1997). Para problemas de programação quadrática restrita, o método de Pontos Interiores é conhecido por ter uma taxa de convergência quadrática. No entanto, problemas com mau condicionamento podem diminuir a sua taxa de convergência. Para contornar essa limitação, métodos híbridos que combinam o método de Pontos Interiores com outros métodos, como, por exemplo, Gradiente Projetado, foram propostos para melhorar a convergência em problemas de programação quadrática restrita.

O método de Pontos Interiores é bastante robusto e pode ser aplicado a uma ampla variedade de problemas de Otimização Convexa. Além disso, o método é capaz de lidar com problemas com restrições de desigualdade e igualdade, e tem boas propriedades de convergência. Em resumo, o método de Pontos Interiores é um método eficiente e poderoso para resolver problemas de Otimização Convexa, especialmente para problemas com restrições complexas.

3.4 MÉTODO DE LAGRANGIANO AUMENTADO

O método de Lagrangiano Aumentado foi desenvolvido para resolver problemas com restrições de igualdade por Hestenes (1969) e Powell (1969), mas foi generalizado posteriormente para problemas com restrições de desigualdades por Rockafellar (1973). É um método iterativo e baseado em penalização. A ideia é transformar, a cada iteração, o problema restrito original em outro irrestrito ou com restrições simples. Assim, uma sequência de subproblemas é resolvida até que seja satisfeita alguma condição de parada.

O método de Lagrangiano Aumentado utiliza uma função de Lagrangiano para adicionar uma penalidade às violações das restrições do problema. A função de Lagrangiano Aumentado é composta pela soma da função objetivo com as restrições do problema e um parâmetro de penalidade.

A seguir, o Algoritmo 5 apresenta um pseudocódigo do algoritmo de Lagrangiano

Aumentado baseado em Rockafellar (1973).

Algoritmo 5: Pseudocódigo do método Lagrangiano Aumentado.

1. Defina a solução inicial x^0 e o parâmetro de penalização ρ
2. Inicialize o contador de iterações k, a função objetivo e os multiplicadores de Lagrange associado às restrições
3. Enquanto as condições de parada não forem satisfeitas, faça:
   a. Calcule a função Lagrangiana no ponto x^k
   b. Minimize a função Lagrangiana de forma irrestrita para obter x^{k+1}
   c. Atualize os multiplicadores de Lagrange e o parâmetro de penalidade ρ^k
   d. Incremente o contador de iterações k
4. Retorne como solução o x da última iteração realizada

O método de Lagrangiano Aumentado é um método convergente, tendo sua análise estudada em Hestenes (1969), Powell (1969) e Rockafellar (1973). A convergência do método é garantida sob certas condições, como a regularidade do problema. No entanto, a escolha inadequada desse parâmetro pode levar a soluções ineficientes. Além disso, a convergência do método pode ser lenta em algumas situações, especialmente em problemas com restrições muito rígidas ou quando as condições iniciais estão longe da solução ótima.

3.5 MÉTODO DE FILTRO

Introduzido por Fletcher e Leyffer (2002) e estendido por outros autores, este algoritmo visa minimizar o valor da função objetivo e de uma medida de inviabilidade, definida por qualquer função pelo usuário (convexa, de preferência). Assim, cada iteração do método de Filtro tentará melhorar o valor da função objetivo, a medida de inviabilidade, ou ambos, enquanto respeita a seguinte regra: sendo x um ponto viável, para um iterando $r > k$, o iterando x^r deverá ser melhor que o iterando x^k para pelo menos um dos critérios.

Desta forma, o objetivo do método é usar um critério de filtro para verificar se um ponto tentativo deve ser aceito como próximo iterando, visando assim encontrar o melhor ponto viável possível que satisfaça todas as restrições do problema. Logo, este algoritmo consiste em calcular um ponto tentativo, que será aceito como próximo iterando caso este ponto tentativo não seja proibido pelo filtro, visando a cada iteração melhorar suficientemente o valor da função objetivo e de alguma medida de inviabilidade.

Tomando como referência o trabalho de Periçaro, Ribeiro e Karas (2013), ao qual formaliza-se um algoritmo geral de filtro, o procedimento adotado parte de um ponto inicial e, uma vez obtido um ponto tentativo, atualiza-se o filtro corrente a fim de melhorar a qualidade da solução. Como é esperado que ocorra a minimização de ambos os critérios (função objetivo e inviabilidade), forma-se uma região que contém pontos que queremos

evitar, denominada como região proibida. A partir da definição desta região proibida, atualiza-se os filtros a fim de encontrar um ponto tentativo não proibido pelo filtro na iteração corrente.

O Algoritmo 6 apresenta um pseudocódigo generalizado do método de Filtro, baseado em Periçaro, Ribeiro e Karas (2013).

Algoritmo 6: Pseudocódigo generalizado do algoritmo de Filtro.

1. Defina uma solução inicial x^0
2. Inicialize o contador de iterações k
3. Calcule o valor da função objetivo e a medida de inviabilidade avaliada no ponto x^0
4. Enquanto o critério de parada não for atingido, faça:
   a. Construa o filtro corrente com base nos pontos já encontrados
   b. Obtenha um novo ponto iterando x^{k+1} não proibido
   c. Compute os valores da função objetivo e a medida de inviabilidade em x^{k+1}
   d. Se o valor da função objetivo não decrescer em relação ao valor da função objetivo da iteração anterior, atualize o filtro corrente
   e. Incremente o contador de iterações
5. Retorne como solução o x da última iteração realizada

De acordo com Periçaro, Ribeiro e Karas (2013), é possível garantir a convergência do método de Filtro e, caso satisfeitas algumas hipóteses adicionais, garantir a convergência global do método (ver Sessão 3 de Periçaro, Ribeiro e Karas (2013)). Em geral, o método é capaz de encontrar um ponto estacionário com alta precisão em um número relativamente baixo de iterações.

Quanto à obtenção de pontos tentativos, conforme destacado no Algoritmo 6, existem diferentes formas de obtê-lo. Periçaro, Ribeiro e Karas (2013) demonstram em seu trabalho a aplicação e adaptação do algoritmo de Programação Quadrática Sequencial (SQP), provando que o algoritmo de Filtro com os pontos tentativos computados por SQP é convergente (ver Sessão 4 de Periçaro, Ribeiro e Karas (2013)). Neste trabalho, focaremos no uso de SQP para obter um ponto tentativo, conforme descrito em Periçaro, Ribeiro e Karas (2013, Alg. 2), ajustando os passos para obedecer a regra de filtro. Vale destacar que o método SQP realiza aproximações do problema original, tendo que ser solucionados problemas de viabilidade e otimalidade em seu processo.

3.6 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Neste capítulo foram descritos, em essência, métodos de Otimização para problemas convexos que podem ser aplicados a problemas genéricos de Programação Não Linear. Como pode ser observado, são diferentes abordagens de algoritmos citados, cada qual possuindo a sua estratégia. Estes algoritmos serão implementados e utilizados para

resolver o problema de treinamento (2.12) de SVM, realizando uma análise comparativa para algumas métricas pré-fixadas.

4 EXPERIMENTOS NUMÉRICOS

Neste capítulo ilustra-se o desempenho dos algoritmos de Otimização implementados a fim de resolver o problema de treinamento de SVM (2.12) para o comparativo proposto. Todos os resultados computacionais foram obtidos a partir da implementação em MATLAB® versão R2019a em um computador pessoal com processador Intel (R) Core i7-9750H CPU @ 2.60GHz 2.60GHz, 16Gb de RAM e sistema operacional Windows 10. Por possuir muitas informações de resultados, será disponibilizada as implementações e os resultados no seguinte repositório no Github: ⟨https://github.com/tiagobeautiful/SVM-OptPerformance⟩.

Este capítulo é organizado da seguinte forma. A Seção 4.1 destaca os algoritmos de Otimização utilizados no experimento, sendo os hiperparâmetros destes algoritmos descritos no Apêndice A. A Seção 4.2 explica a estrutura do experimento, enquanto a Seção 4.3 descreve as métricas a serem analisadas. Por fim, as Seções 4.4 e 4.5 destacam os resultados dos experimentos.

4.1 SOBRE OS ALGORITMOS DE OTIMIZAÇÃO IMPLEMENTADOS

A Tabela 2 sintetiza os algoritmos implementados, bem como seus respectivos acrônimos utilizados para facilitar a leitura. Em seguida, comentários acerca dos métodos são realizados.

TABELA 2 – Algoritmos implementados e seus acrônimos.

Acrônimos	**Estratégia**	**Implementação baseada em**
FQP	Filtro-SQP	Periçaro, Ribeiro e Karas (2013) com `quadprog`
FAL		Periçaro, Ribeiro e Karas (2013) com Torrealba et al. (2021)
KSVM	Lagrangiano Aumentado	Torrealba et al. (2021, Alg. 1)
KPG		Subproblemas resolvidos por Torrealba et al. (2021, Alg. 1)
PGC	Gradiente Projetado	Luenberger, Ye et al. (1984, p.366)
PGN	Newton Projetado	Luenberger, Ye et al. (1984, p.366), mas com direção de Newton
QP	Pontos Interiores	`quadprog`
SPGAL	Gradiente Espectral Projetado	Birgin, Martínez e Raydan (2000, Alg. 2.1)
MVP	Restrições Ativas	Fan et al. (2005, Apêndice B), com regra de seleção de Platt (1998)
LIBSVM		Fan et al. (2005, Apêndice B)

Para um melhor entendimento dos algoritmos da Tabela 2, realiza-se a seguir alguns comentários sobre as implementações realizadas.

- Para a execução de todos os algoritmos, foi aplicada uma regularização na matriz P do problema (2.12) nos casos em que a ela não era definida positiva. Esta regularização implementada é o acréscimo de um valor na diagonal principal de Q;
- Os algoritmos de projeção PGC e PGN possuem a mesma estrutura, conforme destacado no livro de Luenberger, Ye et al. (1984), porém com a diferença de que PGC

utiliza a direção de máxima descida, enquanto PGN utiliza a direção de Newton. Para computar uma direção ortogonal, foi utilizado o processo de Gram-Schimidt entre a direção e o plano definido pela restrição de igualdade do problema (2.12). Para garantir que a direção computada no método PGN seja sempre de descida, foi necessário criar uma condição acerca do ângulo gerado entre a direção ortogonal computada e o vetor gradiente da função objetivo da iteração, conforme descrito no Apêndice A;

- Dentre as extensões dos estudos envolvendo o método de Lagrangiano Aumentado, o trabalho de Torrealba et al. (2021) o adapta para o problema da mochila não linear, o qual é semelhante ao problema de treinamento de SVM. A proposta aplica o método de Lagrangiano Aumentado com projeção em caixa. O problema irrestrito é resolvido usando a ideia do método de Newton. De uma forma resumida, determina-se por meio das condições de otimalidade uma fórmula fechada para a direção de Newton. Após determinada a solução do subproblema irrestrito, projeta-se o ponto na caixa. Em Torrealba et al. (2021) é analisada a convergência do método e, em suas análises, o método apresenta bons resultados de velocidade e qualidade de soluções, mas os autores não estudam uma aplicação ao problema de SVM. Neste trabalho, o método de Torrealba et al. (2021) para problemas quadráticos restritos será denominado como KSVM;

- Os algoritmos KSVM e KPG utilizam o algoritmo de Torrealba et al. (2021). O algoritmo de Torrealba et al. (2021) resolve dois sistemas lineares antes de seu loop principal, sendo aqui solucionados por uma decomposição de Cholesky;

- O algoritmo KPG computa iterativamente um ponto a partir de uma busca exata na direção de máxima descida, obtendo um ponto $\bar{\alpha}$ e, em seguida, resolve o seguinte problema de minimização

$$\begin{aligned} \underset{\alpha}{\text{minimizar}} \quad & ||\alpha - \bar{\alpha}||^2 \\ \text{sujeito a} \quad & y^T(\alpha - \bar{\alpha}) = 0 \\ & 0 \leq \alpha - \bar{\alpha} \leq \mathcal{C}, \end{aligned} \tag{4.1}$$

  solucionado pelo algoritmo de Torrealba et al. (2021). Assim, os sistemas lineares a serem resolvidos antes do loop principal serão divisões simples, visto que a matriz Hessiana do problema de projeção (4.1) é a identidade (em outras palavras, apenas produtos, somas e subtrações são realizados);

- Os algoritmos FQP e FAL, ambos algoritmos de Filtro-SQP, se diferenciam na solução dos subproblemas de viabilidade e otimalidade que a abordagem possui: enquanto FQP soluciona ambos os subproblemas com a função `quadprog` do MATLAB®, FAL encontra o passo de viabilidade com o algoritmo de Torrealba et al. (2021) e o passo de otimalidade com o `quadprog`;

- O algoritmo SPGAL aplica o método de Gradiente Espectral Projetado a uma minimização de uma função Lagrangiana aumentada do problema (2.12) restrita a caixa, isto é,

$$\begin{array}{ll} \underset{\alpha}{\text{minimizar}} & f_{\text{AL}} = \dfrac{1}{2}\alpha^T P\alpha - e^T\alpha + \lambda y^T\alpha + \dfrac{\rho}{2}||y^T\alpha||^2 \\ \text{sujeito a} & 0 \leq \alpha \leq \mathcal{C}, \end{array} \tag{4.2}$$

  com $\lambda > 0$, $\rho > 0$ e $e \in \mathbb{R}^n$ um vetor de uns. O parâmetro ρ é mantido constante, enquanto λ é atualizado conforme o passo espectral do algoritmo de Birgin, Martínez e Raydan (2000). Neste trabalho, foi utilizada busca exata ao invés de busca linear não-monótona como Birgin, Martínez e Raydan (2000, Alg. 2.1), garantindo que a matriz Q seja sempre definida positiva;

- Os algoritmos MVP e LIBSVM foram implementados de acordo com o pseudocódigo descrito em Fan et al. (2005), com a diferença na obtenção do *working set*: enquanto SMO utiliza o procedimento com informações de primeira ordem do máximo par violador das restrições de KKT proposto por Platt (1998), LIBSVM de Fan et al. (2005) utiliza informações de segunda ordem;

- É de conhecimento dos autores que existe a implementação na linguagem C da ferramenta de Chang e Lin (2011), com o código mais otimizado do que o proposto no pseudocódigo descrito em Fan et al. (2005). Entretanto, optou-se por executar uma implementação própria em MATLAB® nas comparações para evitar distorções nas avaliações das métricas utilizadas.

Por fim, os hiperparâmetros utilizados, bem como os critérios de parada dos algoritmos, são destacados no Apêndice A. Vale salientar que, para todos os algoritmos, foram considerados como critérios de parada (i) o tempo máximo de treinamento, (ii) o número máximo de iterações e (iii) os critérios de parada sugeridos pelos artigos de onde os algoritmos foram extraídos. Neste trabalho, será considerado o termo "algoritmos/métodos convergentes"quando satisfazerem o item (iii).

4.2 ESTRUTURA DO EXPERIMENTO

O treinamento do problema (2.12) foi realizado com os algoritmos descritos, utilizando os conjuntos de dados de duas maneiras: com problemas extraídos de repositórios de Aprendizagem de Máquina, tais como UCI, Statlog e outras fontes[1], e com problemas gerados utilizando a função `make_classification` do pacote `scikit-learn` (BUITINCK et al., 2013). As Seções 4.4 e 4.5 contém mais detalhes acerca dos conjuntos de dados utilizados.

[1] Quando da escrita da tese, todos os conjuntos de dados extraídos dos repositórios citados estão disponíveis em ⟨https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/⟩, acessado em março/2023.

Para cada conjunto de dados, os experimentos foram executados primeiro para obter a solução do problema (2.12) e depois prever os resultados no conjunto de teste. Optou-se por executar os testes com as funções *kernel* linear e RBF. Quanto à escolha dos hiperparâmetros da função *kernel* RBF e o limite superior $\mathcal{C}$ do problema (2.12), foram fixados os valores *default* de acordo com o pacote LIBSVM e `scikit-learn`: γ sendo igual a $\frac{1}{p}$ e $\mathcal{C}$ um vetor de componentes iguais a um. Foi realizado um teste inicial com uma validação cruzada via busca em *grid* dos hiperparâmetros do problema (2.12), mas não foram encontradas grandes diferenças, o que justifica a adoção dos valores supracitados. A Figura 5 ilustra a estrutura do experimento.

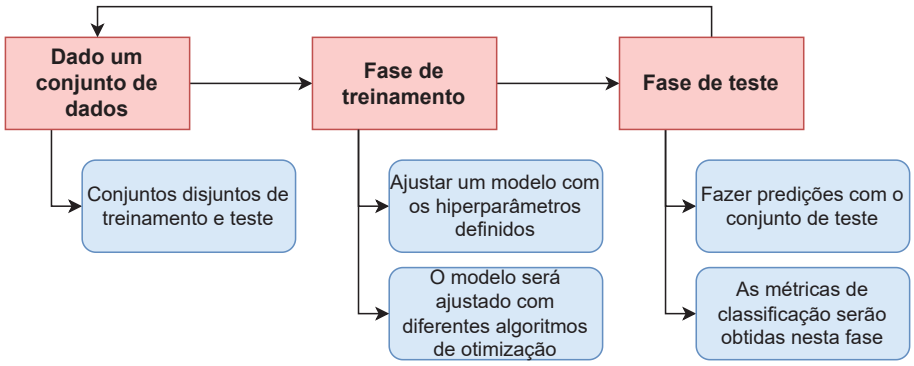


FIGURA 5 – Framework do experimento.

4.3 MÉTRICAS DE PERFORMANCE CONSIDERADAS

Nesta seção é explicitada as métricas analisadas nos experimentos numéricos. Nas análises foram consideradas métricas para avaliar as soluções obtidas pelas algoritmos de otimização implementados, o poder preditivo do classificador ajustado e o perfil de desempenho em relação ao tempo de treinamento do classificador, para assim poder construir um *ranking* das performances obtidas pelos algoritmos implementados.

4.3.1 Comparação das soluções obtidas no problema de treinamento

Para explorar as concordâncias entre as soluções produzidas pelos diferentes métodos nos conjuntos de dados utilizados, gráficos de duas componentes principais, obtidos via método de Análise de Componentes Principais (PCA), foram utilizados. A PCA fornece uma visualização que destaca como as observações em um conjunto de dados se projetam nas duas primeiras componentes principais após a análise de PCA.

A interpretação gráfica dessas projeções ajuda a entender a estrutura e a variabilidade dos dados em um espaço de menor dimensão. De forma geral, a primeira componente principal (PC1) captura a maior quantidade de variância nos dados. Quando a PC1 possui um valor alto para uma observação, isso significa que a observação tem uma grande contribuição na direção da maior variabilidade nos dados. Em outras palavras, a PC1 destaca as principais fontes de variabilidade nos dados. Se dois pontos têm valores de PC1

opostos, significa que eles têm uma alta variabilidade na direção de PC1 e estão distantes um do outro nessa direção. Enquanto isso, a segunda componente principal (PC2) captura a segunda maior quantidade de variância não explicada pela PC1. Portanto, a PC2 representa uma fonte de variabilidade independente de PC1. Quando a segunda componente principal possui um valor alto para uma observação, isso significa que a observação tem uma grande contribuição na direção da segunda maior variabilidade nos dados, que é ortogonal à direção de PC1.

Desta forma, a interpretação gráfica das projeções nas duas primeiras componentes principais pode ajudar a identificar agrupamentos, tendências, ou diferenças nas observações do conjunto de dados. Observações próximas no gráfico têm características similares ou uma variabilidade semelhante nas direções representadas pelas componentes principais, enquanto observações que se afastam no gráfico podem indicar diferenças significativas nas direções representadas pelas componentes principais. A direção de cada componente principal é definida pelas variáveis originais, e os coeficientes associados a cada variável podem fornecer informações sobre quais variáveis contribuem mais para a componente.

4.3.2 Métricas para análise preditiva de um classificador

Para analisar o desempenho dos algoritmos, foi inicialmente construída a matriz de confusão de cada conjunto utilizado. Uma matriz de confusão é uma ferramenta fundamental na avaliação de modelos de classificação, especialmente quando se trata de problemas com duas classes, como positiva e negativa. Ela organiza a contagem de previsões feitas pelo modelo em quatro categorias: verdadeiros positivos (TP) – instâncias corretamente classificadas como positivas, verdadeiros negativos (TN) – instâncias corretamente classificadas como negativas, falsos positivos (FP) – instâncias erroneamente classificadas como positivas, e falsos negativos (FN) – instâncias erroneamente classificadas como negativas. A Tabela 3 exemplifica a discussão.

TABELA 3 – Matriz de confusão binária.

		Previsto	
		Positivo	*Negativo*
Real	*Positivo*	Verdadeiro Positivo	Falso Negativo
	Negativo	Falso Positivo	Verdadeiro Negativo

A matriz de confusão oferece uma visão abrangente do desempenho do modelo, permitindo calcular métricas importantes. Neste trabalho, fixamos as métricas de acurácia, Coeficiente de correlação de Matthews e F1-score, para ajudar avaliar o quão bem o modelo está realizando a tarefa de classificação.

- *Acurácia*: mede a razão de observações classificadas corretamente pelo total de observações do conjunto de dados. É calculada por

$$\text{Acurácia} = \frac{TP + TN}{TP + TN + FP + FN}.$$

A acurácia é uma medida útil para avaliar a capacidade geral do modelo de classificação, mas deve ser usada com cautela, pois pode ser enganosa em casos onde as classes são desbalanceadas;

- *Coeficiente de correlação de Matthews*: introduzido por Matthews (1975), este coeficiente avalia as previsões dos modelos de classificação binária. É definido por

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}}.$$

O MCC penaliza as previsões de FP e FN, e é um avaliador mais robusto que a acurácia se as classes forem desbalanceadas, pois leva em conta as proporções de todos os elementos da matriz de confusão em vez de simplesmente contar o número de instâncias corretamente classificadas. O MCC varia de -1 a 1, onde MCC = 1 indica a predição perfeita e MCC = -1 indica a classificação inversa. Para termos um comparativo com a acurácia, este coeficiente será escalado entre 0 e 1, sendo

$$\text{MCC Escalado} = \frac{\text{MCC} + 1}{2};$$

- *F1 Score*: é uma média harmônica entre as métricas de precisão e do *recall* do modelo. Em poucas palavras, a precisão mede a capacidade do modelo de não rotular erroneamente uma instância como positiva, enquanto o *recall* mede a capacidade do modelo de identificar todas as instâncias positivas. O equilíbrio entre precisão e *recall* é importante para avaliar a efetividade geral do modelo de classificação. Estas métricas são definidas como

$$\text{Precisão} = \frac{TP}{TP + FP}$$

e

$$\text{Recall} = \frac{TP}{TP + FN}.$$

O *F1 Score* é útil quando as classes são desbalanceadas, ou seja, quando uma das classes tem muito mais instâncias do que a outra. O *F1 Score* leva em consideração tanto a precisão quanto o *recall*, tornando-o uma medida mais confiável nessas situações.

$$\textit{F1 Score} = 2\frac{\text{Precisão} * \text{Recall}}{\text{Precisão} + \text{Recall}}.$$

4.3.3 Perfil de desempenho para tempo de execução

Também foi traçado um perfil de desempenho baseado em Dolan e Moré (2002), exibido em escala log2 no eixo das abscissas apenas para o tempo de execução de CPU. Assim, considerando $v_{k,s}$ como um tempo de execução de CPU para um problema $k \in \mathcal{P}$ resolvido por um método $s \in S$, os índices de desempenho usados neste estudo comparativo são definidos por

$$r_{k,s} = \frac{v_{k,s}}{\min\{v_{k,j} : j \in S\}}.$$

Caso o algoritmo s não resolveu o problema k, considere $v_{k,s}$ como infinito. O desempenho geral da avaliação de um determinado solucionador s, $\rho_s : [1, \infty) \to [0, 1]$, é dado por

$$\rho_s(\zeta) = \frac{1}{n_p}\text{card}\{k \in \mathcal{P} : r_{k,s} \leq \zeta\},$$

onde n_p é o número de problemas no conjunto $\mathcal{P}$, sendo ρ_s a porcentagem de problemas que o algoritmo s resolve em ζ vezes o tempo de CPU computado mais eficiente.

4.3.4 Análise de sobrevivência dos algoritmos envolvendo os tempos de execução

A análise de sobrevivência é uma metodologia estatística amplamente utilizada em pesquisa médica, ciências sociais e em muitos outros campos, para investigar e modelar eventos que envolvem a duração de um determinado fenômeno, como o tempo até a ocorrência de um evento adverso, como morte, falha de equipamento ou qualquer outra falha no sistema. Esta abordagem é particularmente útil quando se lida com dados censurados, onde a informação sobre a duração do evento não está disponível para todos os indivíduos ou unidades de observação.

Um dos métodos mais comuns para realizar a análise de sobrevivência é o uso das curvas de Kaplan-Meier. As curvas de Kaplan-Meier apresentam uma representação gráfica das funções de sobrevivência ao longo do tempo, permitindo a visualização das taxas de sobrevivência em diferentes intervalos, isto é, a função de sobrevivência oferece informações valiosas sobre a probabilidade de um evento ocorrer ao longo do tempo.

À medida que o tempo avança, a curva de sobrevivência mostra como essa probabilidade diminui. Ela é especialmente útil para comparar diferentes grupos e avaliar o impacto de variáveis independentes na sobrevivência. Através da análise das curvas de Kaplan-Meier, pesquisadores podem identificar se há diferenças significativas nas taxas de sobrevivência entre grupos e, se necessário, conduzir testes estatísticos, para determinar se essas diferenças são estatisticamente significativas.

Para a pesquisa realizada, as curvas de Kaplan-Meier representa a probabilidade de que a execução do certo algoritmo ainda não tenha terminado em relação ao tempo

de execução. Para construir uma curva de Kaplan-Meier, parte-se de uma função de sobrevivência empírica S em função de um tempo t. Esta função calcula a probabilidade de um indivíduo ou objeto sobreviver além de um ponto específico no tempo t. A função S é definida como

$$S(t) = \prod_{i=1}^{j} \left(1 - \frac{d_i}{n_i}\right),$$

sendo j o número de tempos distintos em que o evento de interesse ocorre, d_i o número de eventos ocorridos no tempo t_i e n_i o número de indivíduos (ou objetos) em risco de sofrer o evento no tempo t_i. A probabilidade de sobrevivência é calculada multiplicando a probabilidade de não ocorrer um evento naquele tempo, considerando todos os tempos anteriores. A curva de Kaplan-Meier é construída ilustrando os valores de S em relação ao tempo t e conectando os pontos.

Sendo assim, nos gráficos a serem ilustrados nessa pesquisa, o eixo das abcissas representa o tempo de execução, enquanto o eixo das ordenadas representa a probabilidade de que a execução do algoritmo está ativa. Conforme o tempo passa, a probabilidade de sobrevivência diminui à medida que as execuções do algoritmo terminam. Logo, é gerada uma curva para cada algoritmo, de acordo com os parâmetros do problema de treinamento (2.12).

Vale destacar que quando as curvas de diferentes algoritmos se interseccionam, há um indicativo de diferenças significativas na eficiência ou tempo de execução entre esses algoritmos. Para evidenciar tais possíveis diferenças, também foi realizado um teste de hipóteses Log-Rank considerando um nível de significância para o teste. O p-valor desse teste indica se há ou não diferenças significativas num comparativo entre pares de algoritmos. Nos testes executados, as hipóteses a serem testadas serão:

$$\begin{cases} H_0: & \text{Não há diferença nas curvas de sobrevivência entre os algoritmos} \\ H_1: & \text{Pelo menos um par de grupos tem diferenças significativas} \end{cases}.$$

Para os testes realizados, o nível de significância adotado foi de 0.05, sendo que p-valores abaixo deste nível sugerem que há diferenças significativas nas curvas de sobrevivência. Para mais informações acerca do teste de hipóteses adotado e/ou análise de sobrevivência, sugere-se a leitura dos trabalhos de Moore (2009), Lee e Wang (2003) e Klein, Moeschberger et al. (2003).

4.3.5 *Ranking* da performance

Ao avaliar o desempenho de diferentes métodos em um cenário de otimização, é comum analisar variadas métricas. A fim de obter um *ranking* da performance dos métodos do teste numérico, que leve em consideração múltiplos critérios e forneça um *ranking* que

reflita o equilíbrio entre esses critérios, optou-se por utilizar uma média ponderada, na qual atribuí-se pesos diferentes a cada critério com base em sua importância relativa.

Para este *ranking*, foi considerado a quantidade de problemas resolvidos, a média das métricas de classificação (Acurácia média, *F1 Scores* e MCC escalado), e também os tempos médios de treinamento do classificador. Optou-se por dar mais peso a métodos que são capazes de resolver um maior número de problemas, pois isso reflete sua capacidade geral de atender a uma variedade de cenários. Para as métricas de classificação, foi considerada os valores médios vista a grande quantidade de problemas a serem resolvidos, além de serem bons indicadores do poder preditivo do classificador gerado. Por outro lado, o tempo de treinamento é um fator que pode ser problemático se não for controlado, especialmente em aplicações em tempo real. Portanto, desejamos penalizar métodos que exigem mais tempo de treinamento.

Para calcular a métrica de *ranking* para um método s, usamos uma média ponderada da seguinte forma:

$$H_s = \frac{\mathrm{P}_A * A_s + \mathrm{P}_D * D_s + \mathrm{P}_E * E_s + \mathrm{P}_F * F_s}{\mathrm{P}_A + \mathrm{P}_D + \mathrm{P}_E + \mathrm{P}_F} - \mathrm{P}_G * G_s, \tag{4.3}$$

sendo H a métrica de ranking, A a quantidade de problemas resolvidos, D a acurácia média para os problemas solucionados, E o *F1 Score* médio para os problemas solucionados, F o MCC escalado médio para os problemas solucionados e G o tempo de treinamento médio para os problemas solucionados. Na equação (4.3), $\mathrm{P}_A, \mathrm{P}_D, \mathrm{P}_E, \mathrm{P}_F$ e P_G são valores referentes ao pesos de problemas resolvidos, acurácia, *F1 Score*, MCC escalado e tempo de treinamento, respectivamente, ao qual podem ser ajustados para refletir a importância relativa desses critérios em sua aplicação específica. A parte chave da equação (4.3) é a penalização do tempo de treinamento, o que significa que métodos com tempos de treinamento mais longos terão uma pontuação final mais baixa. A Tabela 4 destaca os valores dos pesos considerados neste trabalho.

TABELA 4 – Pesos considerados para o cálculo do *ranking* dos métodos.

Nome	Descritivo do peso	Valor
P_A	Quantidade de problemas resolvidos	0.6
P_D	Acurácia média	0.1
P_E	*F1 Score* médio	0.1
P_F	MCC escalado médio	0.1
P_G	Tempo de treinamento médio	0.1

Com base na equação (4.3), é possível obter um *ranking* dos métodos que equilibra eficazmente a capacidade de resolver problemas, o poder preditivo e a penalização pelo tempo de treinamento. Assim, essa abordagem permite uma avaliação mais abrangente

e flexível, levando em consideração critérios de desempenho múltiplos e ponderados de acordo com a prioridade desejada.

4.4 APLICAÇÃO 1: CONJUNTOS DE DADOS GERADOS ALEATORIAMENTE

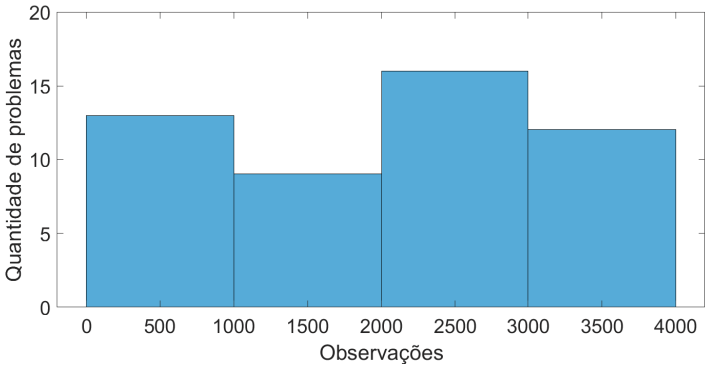
Nesta seção, os conjuntos de dados de duas classes foram gerados a partir da função `make_classification` do pacote `scikit-learn` (BUITINCK et al., 2013). Esta função inicialmente cria um conjunto de dados alocando a cada classe a ser gerada um ou mais grupos de pontos normalmente distribuídos (média zero, desvio padrão um). Além disso, esta função introduz interdependência entre os atributos, bem como diversas formas de ruídos nos dados.

Para os experimentos, foram gerados 50 conjuntos de dados a partir de 50 sementes entre 0 e 100, escolhidas aleatoriamente e salvas. Assim, assegura-se a reprodutibilidade dos conjuntos de dados gerados. Cada conjunto de dados foi construído com base no sorteio da quantidade de observações, em um intervalo de 100 a 5000, quantidade de atributos num intervalo de 2 a 50, com níveis de separação de classe variando no conjunto $\{0.125, 0.25, 0.5, 1, 2, 4, 10\}$. Os valores de níveis de separação representam a relação linear ou não dos atributos com a resposta, sendo que quanto maior for o valor do nível de separação de classes, mais linearmente separável será o conjunto de dados. Quanto aos ruídos a serem introduzidos, não foi permitido nenhum atributo ser repetido e redundante, isto é, as colunas da matriz de dados gerados não poderiam ser definidas a partir de uma combinação linear dos demais atributos.

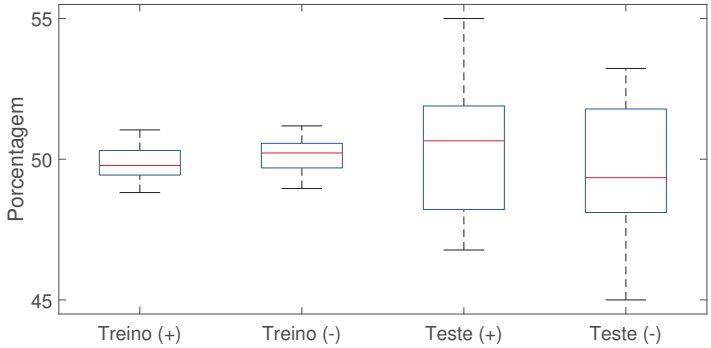
Para gerar os conjuntos de treinamento e teste, os conjuntos de dados foram aleatoriamente divididos na proporção de 80% para treinamento e 20% para teste. Os conjuntos gerados não são esparsos. A Figura 6 destaca o histograma da quantidade de observações dos conjuntos de dados gerados e um boxplot das proporções de classes dos 50 problemas gerados, tanto para conjunto de treinamento como para o teste. Na Figura 6b, os termos "Treino (+)" e "Treino (–)" se remetem a proporção de classes positivas e negativas ao conjunto de treinamento, respectivamente. A interpretação para "Teste (+)"e "Teste (–)"é análoga, porém referente ao conjunto de teste.

A partir da Figura 6a, pode ser observado que mais da metade dos conjuntos gerados, especificamente 28 dos 50 conjuntos, ficaram entre 2000 e 4000 observações. Quanto às proporções de classes, apresentadas na Figura 6, ambos os casos – treinamento e teste – estão equilibrados. Para mais detalhes sobre os conjuntos de dados gerados, encoraja-se o leitor a acessar o repositório criado: ⟨https://github.com/tiagobeautiful/SVM-OptPerformance⟩.

A Figura 7 destaca a quantidade de problemas resolvidos para cada método de Otimização para as funções *kernel* utilizadas nos experimentos. Vale destacar que como



(a) Histograma da quantidade de amostras.



(b) Proporção de classes.

FIGURA 6 – Informações dos conjuntos de dados gerados aleatoriamente.

serão utilizadas duas funções *kernel* diferentes, serão gerados 100 problemas para serem resolvidos aos algoritmos implementados.

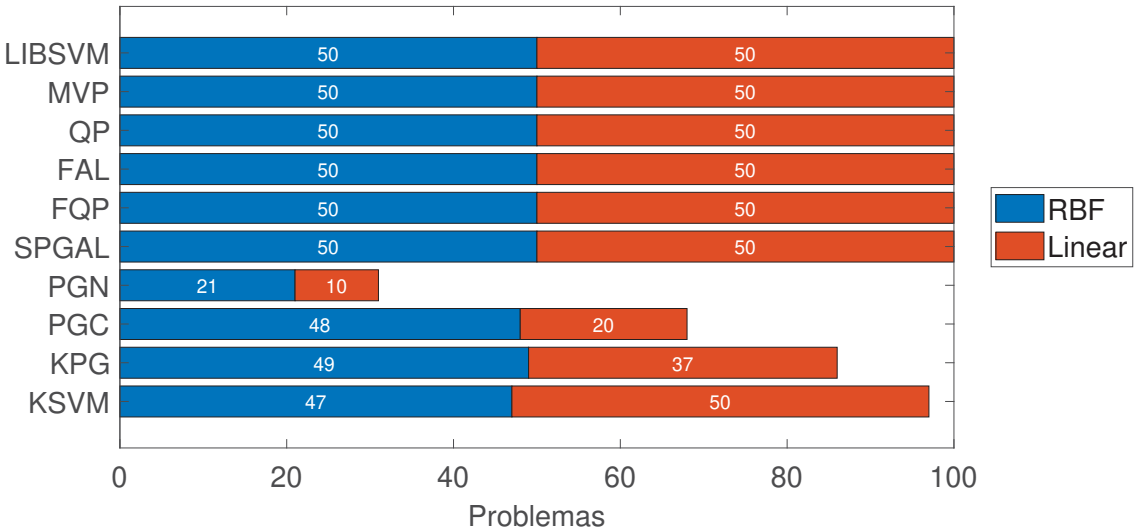


FIGURA 7 – Aplicação 1: Número de problemas solucionados para o conjunto de dados gerados aleatoriamente.

Verifica-se pela Figura 7 que os algoritmos LIBSVM, MVP, QP, FAL, FQP e SPGAL atenderam os critérios de parada imposto para 100% dos problemas gerados. Os algoritmos KSVM e KPG convergiram em 97% e 86% dos problemas simulados, respectivamente, sendo o principal motivo de parada o limite máximo de iterações (3 problemas

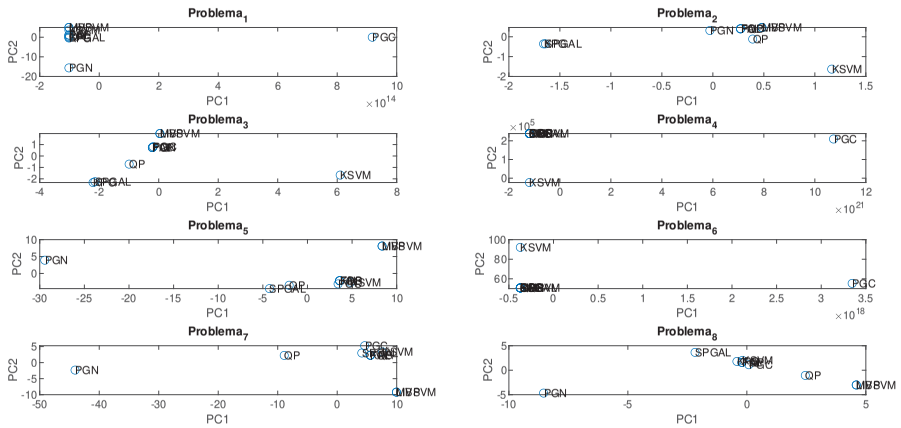
para KSVM e 8 para KPG). O critério de tempo de treinamento foi satisfeito pelo método KPG em 6% dos problemas.

Enquanto isso, os métodos de maior quantidade de problemas não resolvidos foram PGC e PGN, com 68 e 31 problemas resolvidos dos 100 gerados, respectivamente. Os métodos PGC e PGN foram interrompidos em 20% e 46% dos problemas devido ao número máximo de iterações, respectivamente, e em 12% e 23% dos problemas devido a tempo limite de execução.

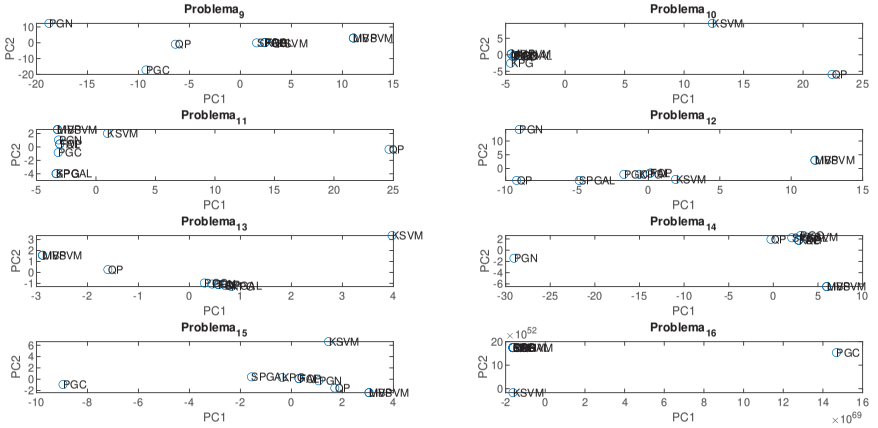
A fim de ilustrar a qualidade das soluções obtidas pelos algoritmos, as Figuras 8-11 destacam gráficos comparativos das duas primeiras componentes principais obtidas num comparativo realizado via PCA. Quando se trata de comparar soluções de algoritmos, um aspecto crucial da PCA é a identificação de agrupamentos no espaço das duas primeiras componentes principais, sendo que cada ponto no gráfico representa uma solução de algoritmos, e a proximidade desses pontos indica similaridades nas características das soluções.

Analisando o caso em que foi utilizado a função *kernel* linear na Aplicação 1, observa-se nos 50 problemas formações de agrupamentos bem destacados visualmente em todos os problemas, com alguns pontos brevemente mais deslocados. De forma geral, os algoritmos que mais se concentraram no mesmo agrupamentos foram MVP, LIBSVM, enquanto FAL, FQP, SPGAL e QP ficaram como pontos satélites aos pontos MVP e LIBSVM. Isso demonstra que as soluções dos algoritmos convergiram para a mesma direção para todos os problemas até satisfazerem os seus respectivos critérios de paradas.

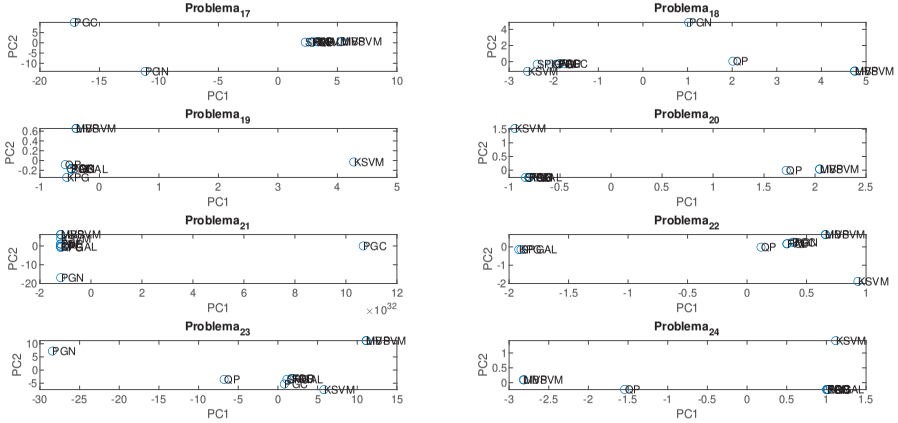
Nas Figuras 8 e 9 também se destacam os algoritmos KSVM, PGC e PGN, por apresentarem as maiores variabilidades nas duas primeiras componentes principais das soluções obtidas em relação aos demais algoritmos. No caso das soluções do algoritmo KSVM é interessante ressaltar que o mesmo atendeu seus critérios de paradas em todos os problemas gerados para o caso em que a função *kernel* linear foi utilizada, tomando direções mais distintas em relação aos algoritmos baseados em Restrições Ativas, Filtro e com projeções.



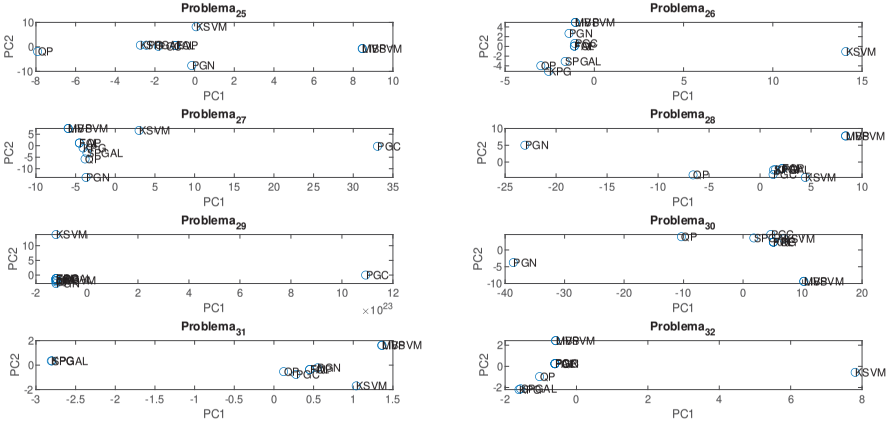
(a) Problemas 1-8.



(b) Problemas 9-16.

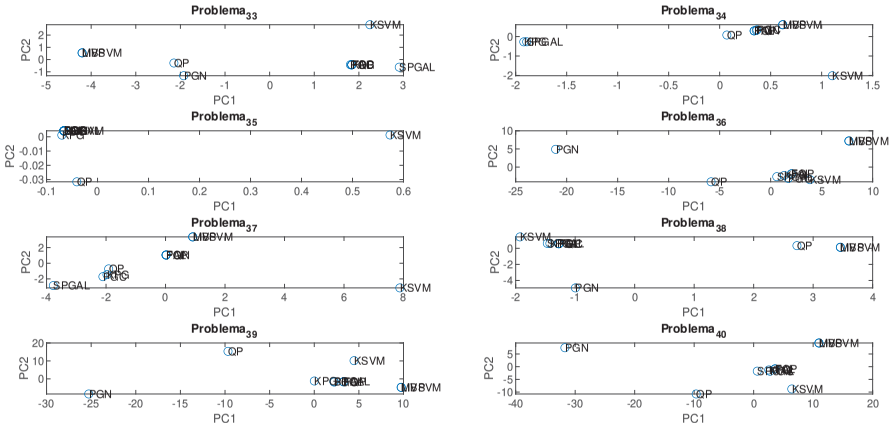


(c) Problemas 17-24.

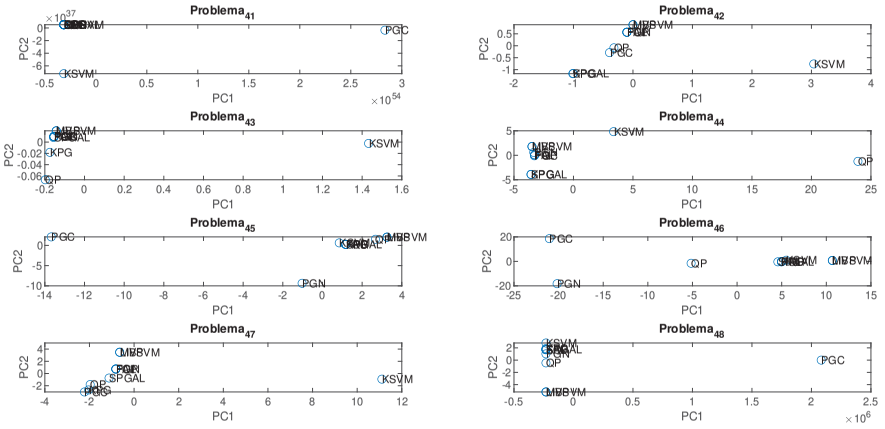


(d) Problemas 25-32.

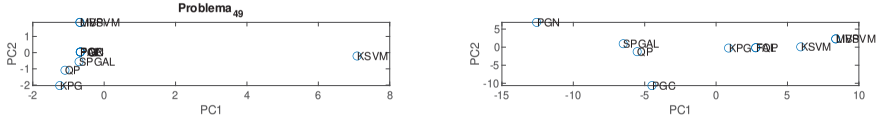
FIGURA 8 – Aplicação 1: Comparativo das componentes principais das soluções aos conjuntos de dados gerados aleatoriamente - Função *kernel* linear (Parte 1).



(a) Problemas 33-40.

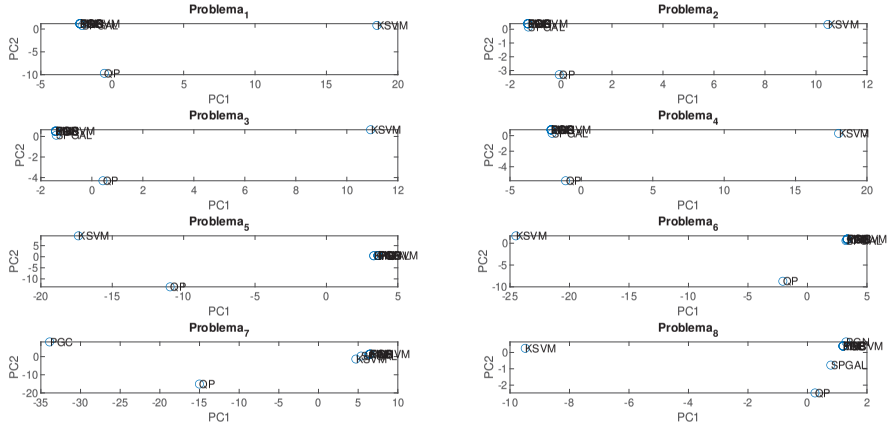


(b) Problemas 41-48.

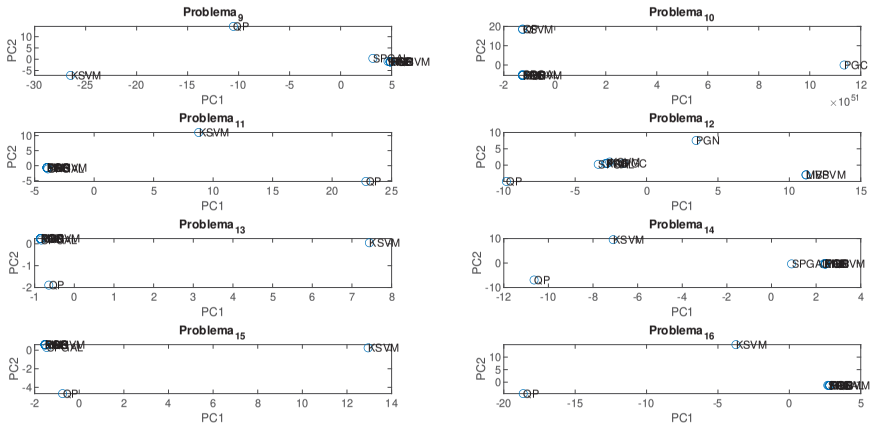


(c) Problemas 49-50.

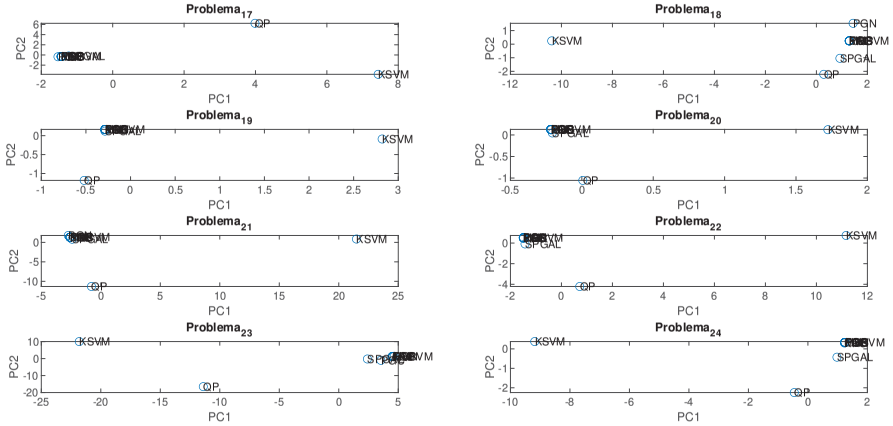
FIGURA 9 – Aplicação 1: Comparativo das componentes principais das soluções aos conjuntos de dados gerados aleatoriamente - Função *kernel* linear (Parte 2).



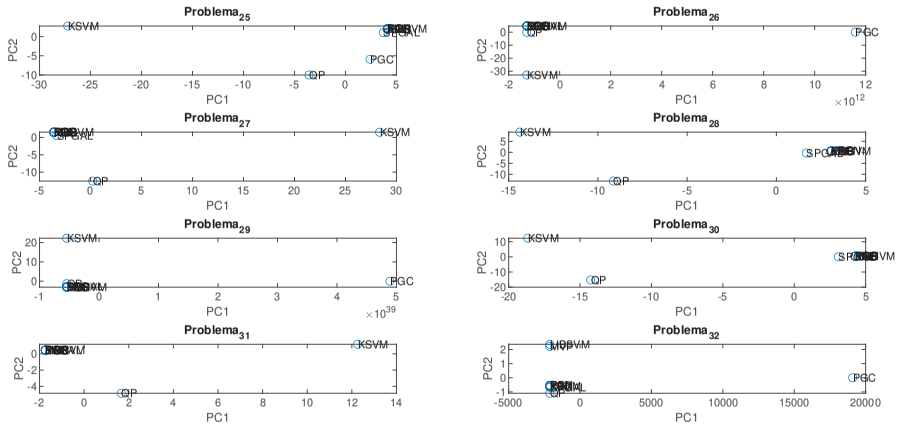
(a) Problemas 1-8.



(b) Problemas 9-16.

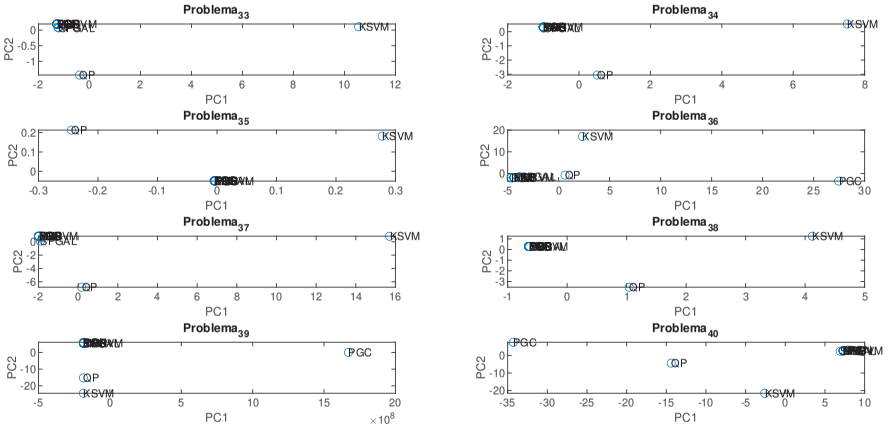


(c) Problemas 17-24.

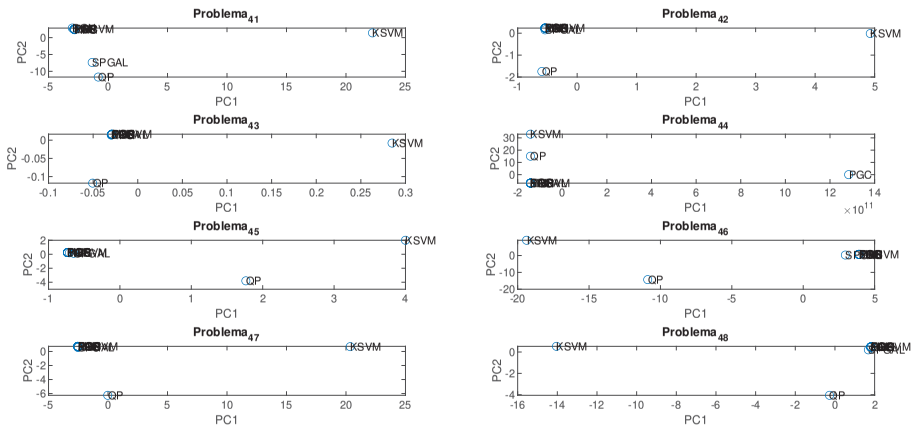


(d) Problemas 25-32.

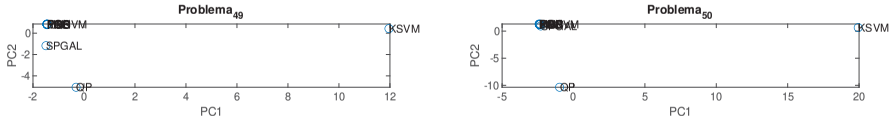
FIGURA 10 – Aplicação 1: Comparativo das componentes principais das soluções aos conjuntos de dados gerados aleatoriamente - Função *kernel* RBF (Parte 1).



(a) Problemas 33-40.



(b) Problemas 41-48.

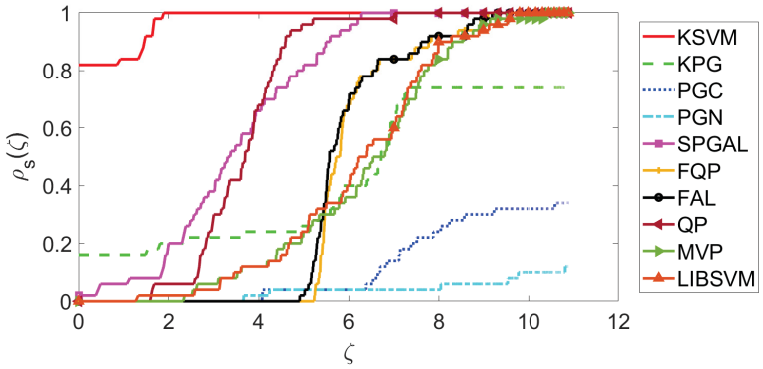


(c) Problemas 49-50.

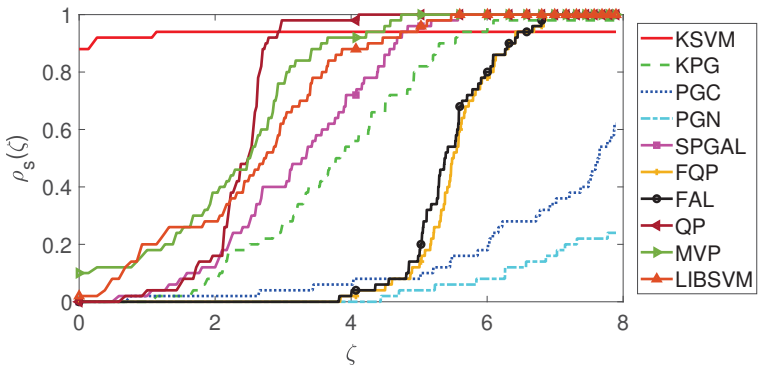
FIGURA 11 – Aplicação 1: Comparativo das componentes principais das soluções aos conjuntos de dados gerados aleatoriamente - Função *kernel* RBF (Parte 2).

Observando o caso em que foi utilizado a função *kernel* RBF na Aplicação 1, ocorreu o mesmo padrão analisado com a função *kernel* linear: formações de agrupamentos bem destacados visualmente em todos os problemas, com os pontos relativos as soluções dos algoritmos KSVM, PGC e PGN apresentando as maiores variabilidades nas duas primeiras componentes principais das soluções obtidas. Novamente, isso destaca como os algoritmos seguiram direções similares para obter um ponto otimizado como solução.

Para destacar a performance quanto ao tempo de treinamento, o perfil de desempenho dos algoritmos em relação ao tempo de CPU para o treinamento do problema é ilustrado na Figura 12, tanto para a função *kernel* linear quanto RBF.



(a) Função *kernel* linear.



(b) Função *kernel* RBF.

FIGURA 12 – Aplicação 1: Performance profile relacionado ao tempo de CPU.

Em questão de tempo de treinamento, a Figura 12 ilustra o fato de que o algoritmo KSVM foi o destaque para ambas as funções *kernel* implementadas. O método KSVM solucionou mais rapidamente 82% dos problemas com função *kernel* linear, seguido pelo algoritmo KPG em 16% dos problemas. Vale destacar que o algoritmo KSVM obteve sucesso para todos os problemas em que a função *kernel* linear foi considerada.

Para a função *kernel* RBF, o método KSVM solucionou mais rapidamente 88% dos problemas, seguido pelo algoritmo MVP com 10% dos problemas solucionados mais rapidamente. Diferentemente do caso com *kernel* linear, o algoritmo KSVM resolveu 47 dos 50 problemas gerados com função *kernel* RBF (94% dos problemas). Pela Figura 12,

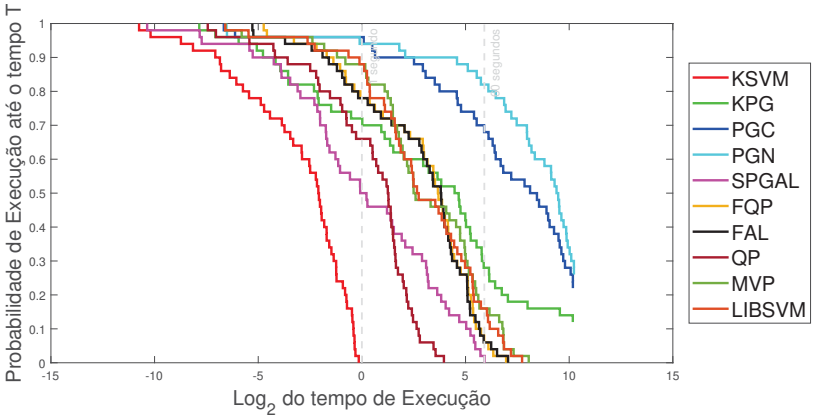
os demais algoritmos não foram competitivos em termos de velocidade de treinamento para os problemas gerados.

Para reforçar os argumentos apresentados, uma análise de sobrevivência dos tempos absolutos para treinamento foram realizados e ilustrados na Figura 13, para ambas as funções *kernel* utilizadas.

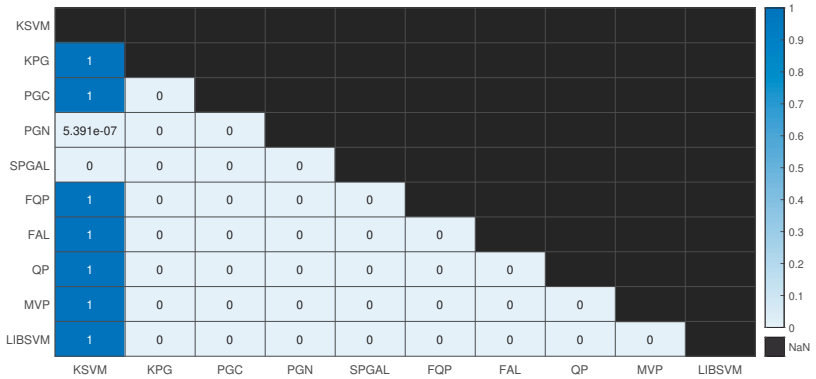
Nas Figuras 13a e 13c, que ilustram as curvas de Kaplan-Meier dos algoritmos em relação a probabilidade de execução da certo tempo, novamente destaca-se que o algoritmo KSVM foi o algoritmo mais veloz em termos de treinamento, seguido dos algoritmos LIBSVM, MVP, SPGAL e QP. Os algoritmos baseados em Filtro obtiveram performances semelhantes, sendo que suas curvas quase se sobrepõem uma a outra, porém indicando um maior tempo de treinamento para obter uma solução. Os algoritmos com a curva de Kaplan-Meier mais deslocadas para a direita neste experimento foram PGC e PGN.

Já as Figuras 13b e 13d, que ilustram os resultados obtidos no teste de hipótese Log-Rank a fim de verificar a existência de diferenças significativas no tempo de treinamento entre dois algoritmos a um p-valor de 0.05, corroboram que o algoritmo KSVM obteve os valores de treinamento mais diferenciados em relação aos demais algoritmos, independente da função *kernel* utilizada no experimento. No caso da função *kernel* linear, os demais algoritmos não apresentaram diferenças estatisticamente significativas entre os valores apresentados, enquanto no caso da função *kernel* RBF houve diferenças no tempo de treinamento nos algoritmos QP, MVP e LIBSVM (além do KSVM) em relação aos demais algoritmos.

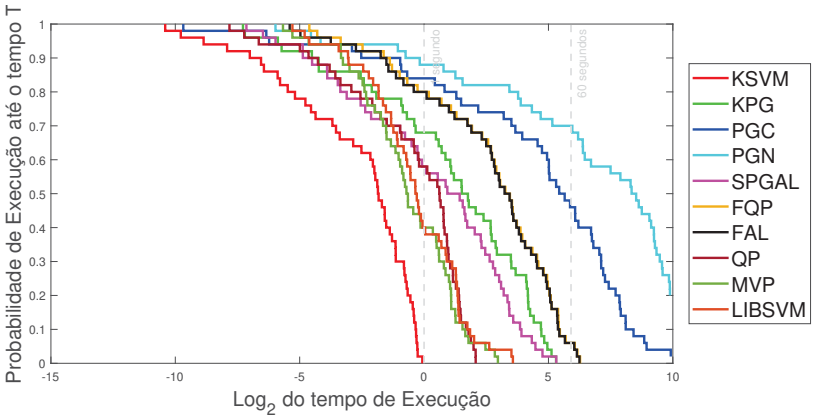
Os resultados quanto às métricas de classificação são ilustrados por boxplots na Figura 14, para a função *kernel* linear, e Figura 15 para o *kernel* RBF. Para auxiliar num comparativo entre as métricas de classificação e tempo de treinamento, gráficos de bolhas são apresentados, sendo que o tamanho da bolha indica o tempo de execução do algoritmos. Vale ressaltar que os valores considerados para o boxplot e gráfico de bolha são somente dos problemas que atingiram o critério de convergência para cada algoritmo. De forma a auxiliar na compreensão dos boxplots e gráficos de bolhas, a Tabela 5 destaca estatísticas descritivas (média e desvio padrão) das métricas de classificação analisadas.



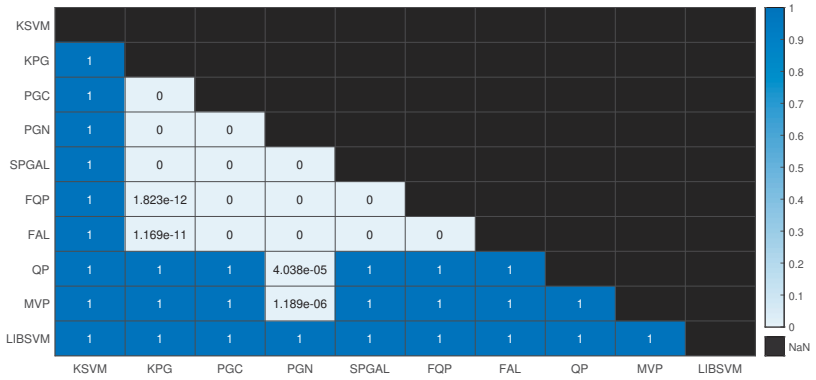
(a) Curvas de Kaplan-Meier - função *kernel* linear.



(b) Mapa de calor do teste Log-Rank - função *kernel* linear.

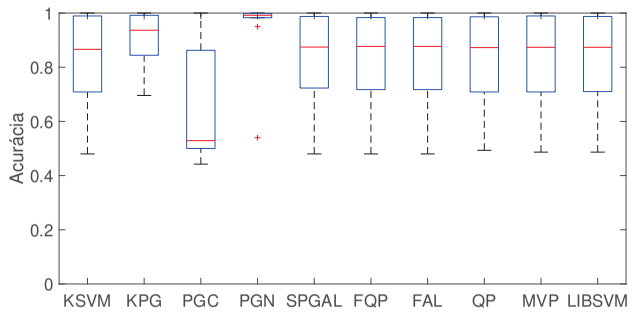


(c) Curvas de Kaplan-Meier - função *kernel* RBF.

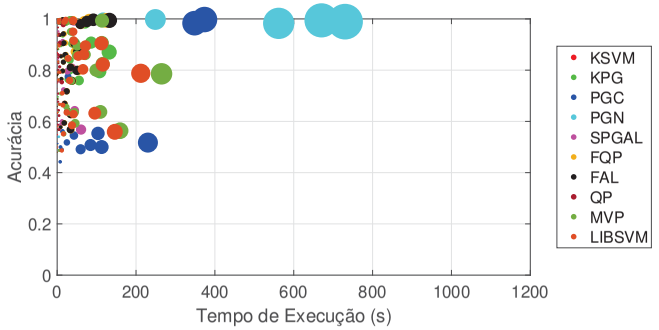


(d) Mapa de calor do teste Log-Rank - função *kernel* RBF.

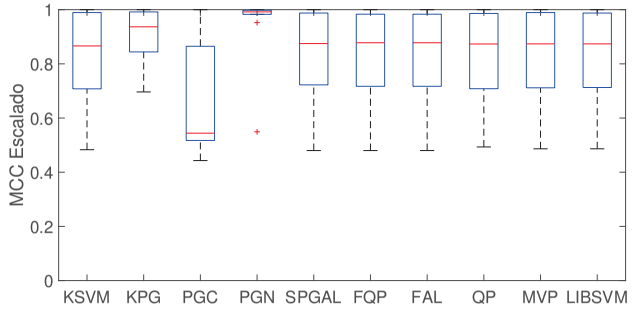
FIGURA 13 – Aplicação 1: Análise de sobrevivência dos algoritmos.



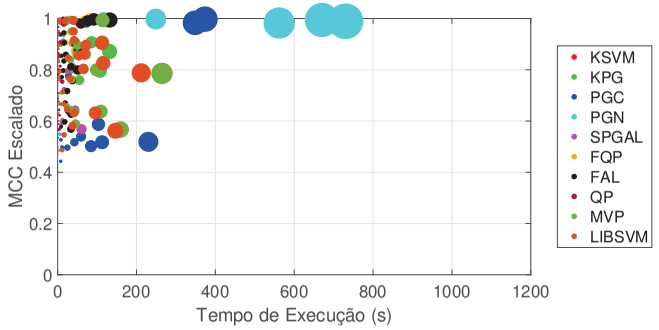
(a) Acurácia.



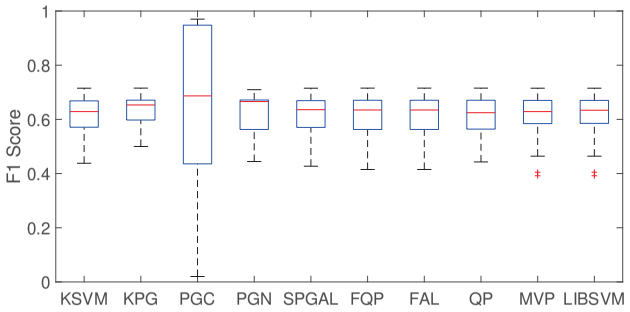
(b) Tempo execução x Acurácia.



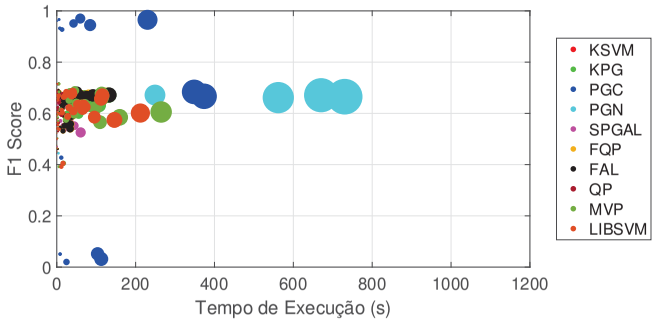
(c) MCC Escalado.



(d) Tempo execução x MCC Escalado.

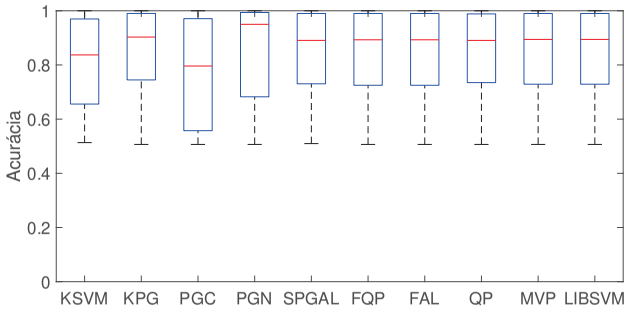


(e) *F1 Score*.

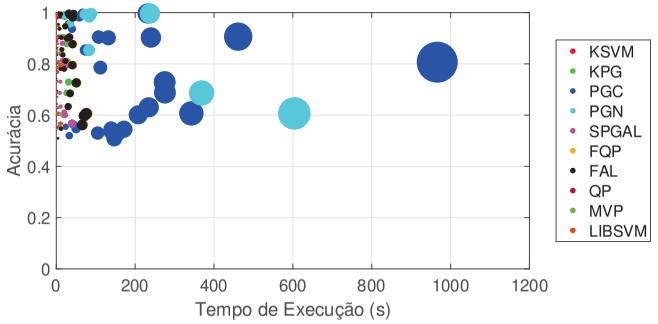


(f) Tempo execução x *F1 Score*.

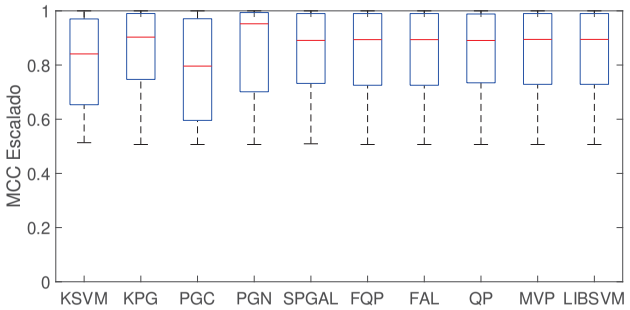
FIGURA 14 – Aplicação 1: Métricas de classificação - função *kernel* linear.



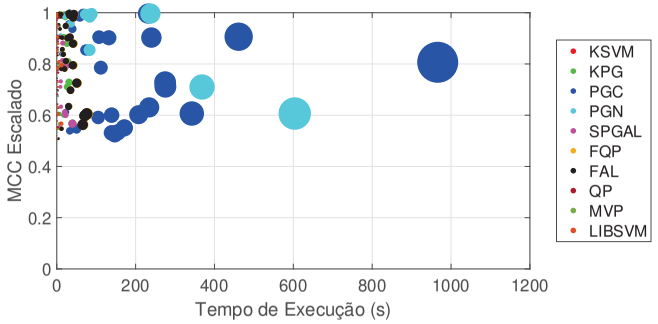
(a) Acurácia.



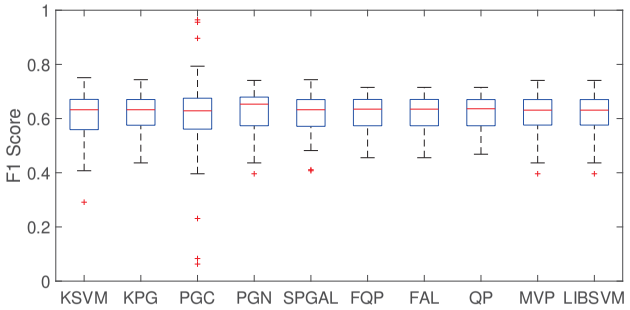
(b) Tempo execução x Acurácia.



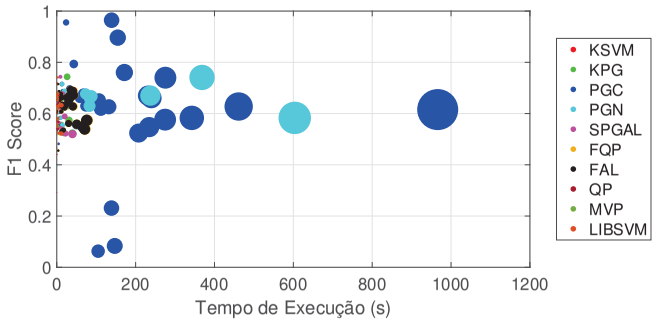
(c) MCC Escalado.



(d) Tempo execução x MCC Escalado.



(e) *F1 Score*.



(f) Tempo execução x *F1 Score*.

FIGURA 15 – Aplicação 1: Métricas de classificação - função *kernel* RBF.

TABELA 5 – Aplicação 1: Média e desvio padrão das métricas obtidas.

Algoritmos	**Função kernel linear**			
	Tempo	*Acurácia*	*MCC escalado*	*F1-Score*
KSVM	0.29 ± 0.27	0.82 ± 0.17	0.82 ± 0.17	0.61 ± 0.07
KPG	22.26 ± 32.89	0.91 ± 0.09	0.91 ± 0.09	0.64 ± 0.05
PGC	72.91 ± 113.68	0.64 ± 0.21	0.65 ± 0.21	0.62 ± 0.35
PGN	233.67 ± 303.31	0.94 ± 0.14	0.94 ± 0.14	0.62 ± 0.09
SPGAL	8.68 ± 15.01	0.83 ± 0.16	0.83 ± 0.16	0.61 ± 0.07
FQP	20.51 ± 24.64	0.83 ± 0.16	0.83 ± 0.16	0.61 ± 0.07
FAL	20.63 ± 26.14	0.83 ± 0.16	0.83 ± 0.16	0.61 ± 0.07
QP	2.88 ± 3.18	0.83 ± 0.16	0.83 ± 0.16	0.61 ± 0.07
MVP	32.68 ± 50.94	0.83 ± 0.16	0.83 ± 0.16	0.62 ± 0.07
LIBSVM	28.18 ± 43.17	0.83 ± 0.16	0.83 ± 0.16	0.62 ± 0.07

Algoritmos	**Função kernel RBF**			
	Tempo	*Acurácia*	*MCC escalado*	*F1-Score*
KSVM	0.32 ± 0.29	0.80 ± 0.16	0.81 ± 0.16	0.61 ± 0.10
KPG	7.95 ± 9.97	0.84 ± 0.16	0.84 ± 0.16	0.62 ± 0.07
PGC	105.89 ± 165.14	0.77 ± 0.19	0.78 ± 0.18	0.61 ± 0.17
PGN	78.17 ± 151.21	0.85 ± 0.18	0.85 ± 0.18	0.62 ± 0.09
SPGAL	5.51 ± 7.99	0.84 ± 0.16	0.84 ± 0.16	0.62 ± 0.07
FQP	17.74 ± 19.91	0.83 ± 0.16	0.84 ± 0.16	0.62 ± 0.07
FAL	17.41 ± 19.60	0.83 ± 0.16	0.84 ± 0.16	0.62 ± 0.07
QP	1.49 ± 1.24	0.84 ± 0.16	0.84 ± 0.16	0.62 ± 0.07
MVP	1.34 ± 1.69	0.84 ± 0.16	0.84 ± 0.16	0.62 ± 0.07
LIBSVM	1.69 ± 2.41	0.84 ± 0.16	0.84 ± 0.16	0.62 ± 0.07

Observando as Figuras 14 e 15, em conjunto com a Tabela 5, os resultados das métricas de classificação são muito semelhantes quando cada método é observado isoladamente, principalmente em relação à métrica *F1 Score*. Isto é um reflexo da proporção de classes dos conjuntos de dados, o que justifica a similaridade das métricas utilizadas. Apesar de ter convergido para uma grande quantidade de problemas, o algoritmo KSVM definiu classificadores cujas métricas (acurácia e MCC uniformizado) estabeleceram uma amplitude interquartil levemente maior quando comparados aos demais métodos, não o tornando competitivo. O algoritmo PGN, que possui uma amplitude interquartil baixa e com uma mediana relativamente similar com os algoritmos implementados para as três métricas ilustradas, não foi considerado com bom desempenho em vista da baixa quantidade de problemas solucionados. Já o algoritmo PGC não apresentou uma boa performance nesta bateria de testes.

Os algoritmos com melhor desempenho em termos da classificação são FQP, FAL, LIBSVM, MVP e QP, não necessariamente nesta ordem, uma vez que são os algoritmos com maior quantidade de problemas para os quais convergiram e os respectivos quartis ilustrados para cada métrica são mais similares entre os algoritmos implementados. De acordo com as análise realizadas quanto ao perfil de desempenho e análise de sobrevivência, ressalta-se que os algoritmos mencionados não foram os melhores em termos de tempo de CPU para treinamento.

Por fim, para realizar um comparativo e um *ranking* da performance dos algoritmos implementados, a Tabela 6 sintetiza um resumo quanto quantidade de problemas resolvidos, médias das métricas de classificação consideradas e tempos de treinamento, e o valor obtido para definir o *ranking* a partir da equação (4.3).

TABELA 6 – Aplicação 1: Resumo das métricas obtidas e *ranking* das performances.

Função kernel Linear									
Algoritmos	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*Ranque*
KSVM	50	0	0	0.82	0.61	0.82	0.29	33.56	1
QP	50	0	0	0.83	0.61	0.83	2.88	33.30	2
SPGAL	50	0	0	0.83	0.61	0.83	8.68	32.72	3
FQP	50	0	0	0.83	0.61	0.83	20.51	31.54	4
FAL	50	0	0	0.83	0.61	0.83	20.63	31.52	5
LIBSVM	50	0	0	0.83	0.62	0.83	28.18	30.77	6
MVP	50	0	0	0.83	0.62	0.83	32.68	30.32	7
KPG	37	7	6	0.91	0.64	0.91	22.26	22.71	8
PGC	20	19	11	0.64	0.62	0.65	72.91	6.26	9
PGN	10	27	13	0.94	0.62	0.94	233.67	-16.42	10
Função kernel RBF									
Algoritmos	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*Ranque*
MVP	50	0	0	0.84	0.62	0.84	1.34	33.45	1
QP	50	0	0	0.84	0.62	0.84	1.49	33.44	2
LIBSVM	50	0	0	0.84	0.62	0.84	1.69	33.42	3
SPGAL	50	0	0	0.84	0.62	0.84	5.51	33.04	4
KPG	49	1	0	0.84	0.62	0.84	7.95	32.13	5
FAL	50	0	0	0.83	0.62	0.84	17.41	31.85	6
FQP	50	0	0	0.83	0.62	0.84	17.74	31.81	7
KSVM	47	3	0	0.80	0.61	0.81	0.32	31.55	8
PGC	48	1	1	0.77	0.61	0.78	105.89	21.65	9
PGN	21	19	10	0.85	0.62	0.85	78.17	6.44	10

Nota: A = Quantidade de problemas resolvidos; B = Quantidade de problemas que atingiram o máximo de iterações imposto; C = Quantidade de problemas que atingiram o limite de tempo; D = Acurácia média para os problemas solucionados; E = *F1 Score* médio para os problemas solucionados; F = MCC escalado médio para os problemas solucionados; G = Tempo de treinamento médio para os problemas solucionados; H = Métrica de *ranking* (4.3).

A partir dos resultados explicitados na Tabela 6 para esta aplicação, pode-se verificar que os métodos baseados em Lagrangiano Aumentado se saíram bem, satisfazendo os critérios de paradas impostos. Os métodos baseados em projeção não conseguiram atender os critérios de parada implementados, com exceção de SPGAL. Como discutido anteriormente, os métodos baseado em Filtro e SPGAL também tiveram resultados satisfatórios, tendo boas métricas de classificação às custas de um alto custo computacional no treinamento do classificador. Dentre os algoritmos implementados, os métodos baseados em Restrições Ativas (LIBSVM e MVP) e Pontos Interiores foram os que apresentaram um melhor equilíbrio entre o tempo de treinamento e as métricas de classificação.

4.5 APLICAÇÃO 2: CONJUNTOS DE DADOS *BENCHMARK*

Os problemas tratados nesta seção foram extraídos de repositórios de Aprendizagem de Máquina, tais como UCI, Statlog e outras fontes, sendo problemas comums para verificar a performance dos métodos. Os conjuntos de dados selecionados estão sintetizados na Tabela 7, sendo descritos a quantidade de instâncias para treino e teste do modelo, quantidade de atributos e a densidade, dada pela razão entre elementos não nulos e $n \times p$. A proporção de classe de conjuntos de treinamento e teste também é exibida, sendo utilizado os termos "Treino (+)" e "Treino (–)" para indicar a proporção de classes positivas e negativas ao conjunto de treinamento, respectivamente. A interpretação para "Teste (+)"e "Teste (–)"é análoga, porém referente ao conjunto de teste. Para o caso em que um conjunto de dados não possui um conjunto de teste predefinido, separou-se

aleatoriamente o conjunto de dados original em 80% de treinamento e 20% de teste.

TABELA 7 – Informação dos conjuntos de dados *benchmark*.

Dataset	n_{treino}	n_{teste}	p	**Densidade** (%)	**Proporção de classes** (%)			
					Treino (+)	*Treino (-)*	*Teste (+)*	*Teste (-)*
leukemia	38	34	7129	100.00	71.05	28.95	58.82	41.18
duke	36	8	7129	100.00	58.33	41.67	25.00	75.00
colon.cancer	50	12	2000	100.00	30.00	70.00	58.33	41.67
liver.disorders	145	200	5	100.00	37.93	62.07	50.00	50.00
sonar	167	41	60	99.99	46.71	53.29	46.34	53.66
heart	216	54	13	74.75	42.13	57.87	53.70	46.30
heart.scale	216	54	13	96.15	45.37	54.63	40.74	59.26
breast.cancer	547	136	10	100.00	36.20	63.80	30.15	69.85
breast.cancer.scale	547	136	10	100.00	34.37	65.63	37.50	62.50
australian	552	138	14	79.65	44.02	55.98	46.38	53.62
australian.scale	552	138	14	87.42	44.20	55.80	45.65	54.35
diabetes	615	153	8	87.54	63.25	36.75	72.55	27.45
diabetes.scale	615	153	8	99.82	66.02	33.98	61.44	38.56
fourclass	690	172	2	100.00	34.78	65.22	38.95	61.05
fourclass.scale	690	172	2	99.57	36.09	63.91	33.72	66.28
madelon	2000	600	500	100.00	50.00	50.00	50.00	50.00
german.numer	800	200	24	74.97	30.38	69.63	28.50	71.50
german.numer.scale	800	200	24	95.91	27.88	72.13	38.50	61.50
splice	1000	2175	60	100.00	51.70	48.30	52.00	48.00
svmguide3	1243	41	22	80.50	23.81	76.19	100.00	0.00
svmguide1	3089	4000	4	99.70	64.75	35.25	50.00	50.00
mushrooms	6500	1624	112	18.75	51.68	48.32	52.28	47.72
a1a	1605	30956	123	11.65	24.61	75.39	24.05	75.95
a2a	2265	30296	123	11.65	25.25	74.75	23.99	76.01
a3a	3185	29376	123	11.37	24.27	75.73	24.06	75.94
a4a	4781	27780	123	11.37	24.85	75.15	23.95	76.05
a5a	6414	26147	123	11.37	24.46	75.54	23.99	76.01
a6a	11220	21341	123	11.37	23.99	76.01	24.13	75.87
w1a	2477	47272	300	3.82	2.91	97.09	2.98	97.02
w2a	3470	46279	300	3.88	3.08	96.92	2.96	97.04
w3a	4912	44837	300	3.88	2.91	97.09	2.98	97.02
w4a	7366	42383	300	3.89	2.93	97.07	2.98	97.02

A partir da Tabela 7 é possível observar que a maior parte dos 32 conjuntos de dados possuem alta densidade, o que significa que não são esparsos. Nos casos em que o conjunto de dados possui dados faltantes, foi considerado como valor zero e não foi realizado nenhum tratamento estatístico para os conjuntos de dados esparsos. Outro ponto é o fato do não balanceamento de classes nos dados, tanto para o treinamento do modelo quanto para a fase de teste. Estas informações são ilustradas na Figura 16, sendo que os termos "Treino (+)" e "Treino (–)" se remetem a proporção de classes positivas e negativas ao conjunto de treinamento, respectivamente. A interpretação para "Teste (+)"e "Teste (–)"é análoga, porém referente ao conjunto de teste.

Outra observação deve ser realizada quanto aos conjuntos de dados presentes na Tabela 7. Os conjuntos de dados utilizados na bateria de teste não sofrem nenhum tratamento estatístico da parte dos autores, isto é, são os valores originários da fonte em que os dados foram extraídos.

A Figura 17 destaca a quantidade de problemas resolvidos por cada método de Otimização para as funções *kernel* utilizadas nos experimentos. Observa-se que os métodos de Otimização não atingiram os critérios de paradas escolhidos para os 64 problemas gerados, sendo interrompidos pelo tempo limite de treinamento ou atingiram o limite

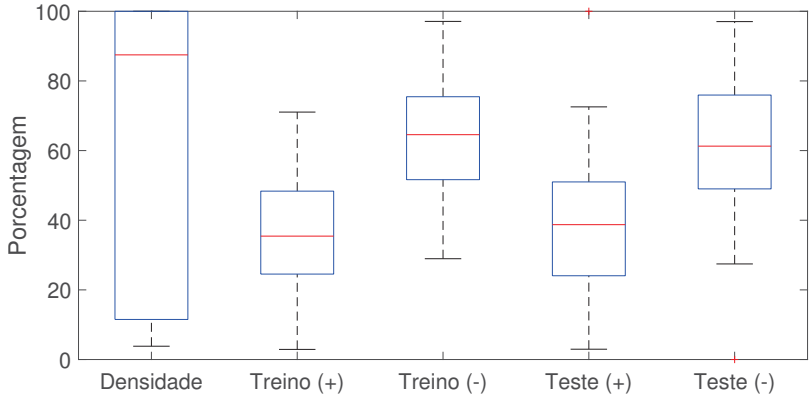


FIGURA 16 – Boxplot da densidade e proporção de classes treino-teste apresentada na Tabela 7.

máximo de iterações.

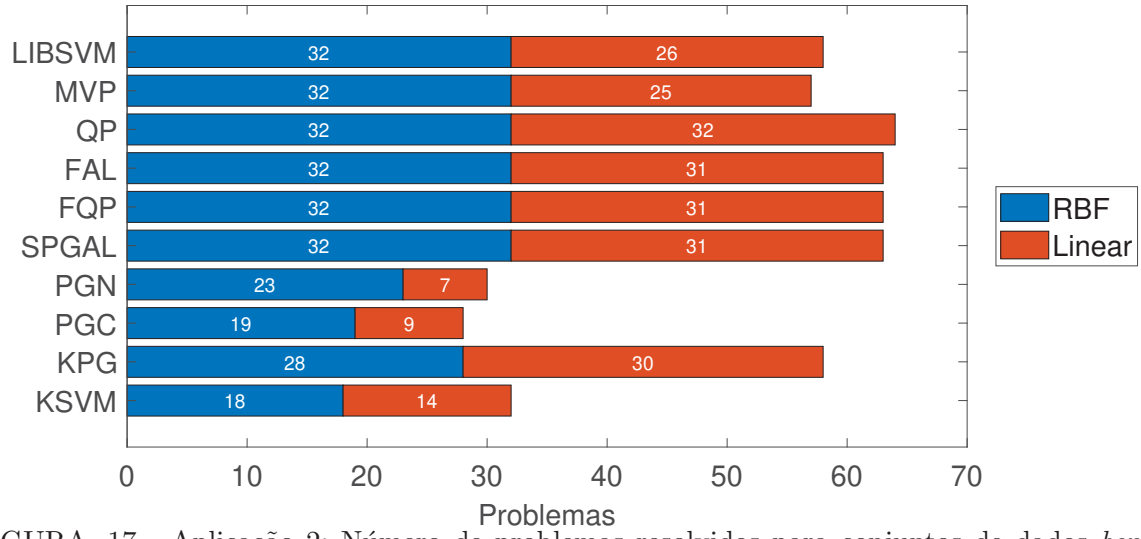


FIGURA 17 – Aplicação 2: Número de problemas resolvidos para conjuntos de dados *benchmark*.

Verifica-se pela Figura 17 que os algoritmos KPG, SPGAL, FQP, FAL, QP, MVP e LIBSVM atenderam os seus critérios de parada em 90.6%, 98.4%, 98.4%, 98.4%, 100%, 89.1% e 90.6% dos problemas, respectivamente. A principal causa de não satisfazer o critério de parada de treinamento implementado é o limite máximo de iterações, com o algoritmo MVP parando sua execução para 9.4% dos problemas (6 problemas), enquanto que os métodos KPG, FQP, FAL e LIBSVM ficam com porcentagens abaixo de 9% para este caso (5 problemas). O método SPGAL não sofreu interrupções devido ao máximo de iterações. O critério de tempo máximo de treinamento foi satisfeito pelos métodos KPG, SPGAL, FQP, FAL, MVP e LIBSVM em 1.6% dos problemas (1 problema), porém não são necessariamente para os mesmos problemas em cada algoritmo citado.

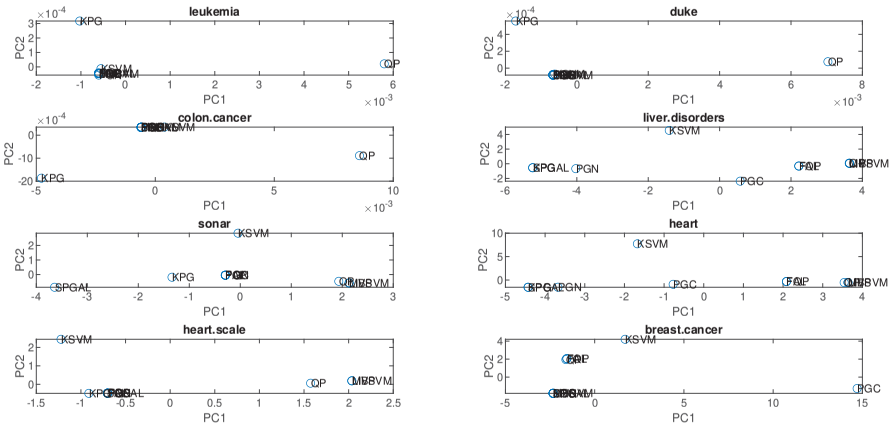
Enquanto isso, a Figura 17 ilustra que os métodos de pior desempenho foram KSVM, PGC e PGN, com 32, 28 e 30 problemas resolvidos dos 64 gerados, respectivamente. O método KSVM foi interrompido em 32 problemas devido ao máximo de iterações,

o que representa 50% dos problemas que não atenderam o critério de parada utilizado. Os métodos PGC e PGN foram interrompidos em 40.6% e 28.1% dos problemas devido ao máximo de iterações, respectivamente, e em 15.6% e 25% dos problemas devido a tempo limite de execução.

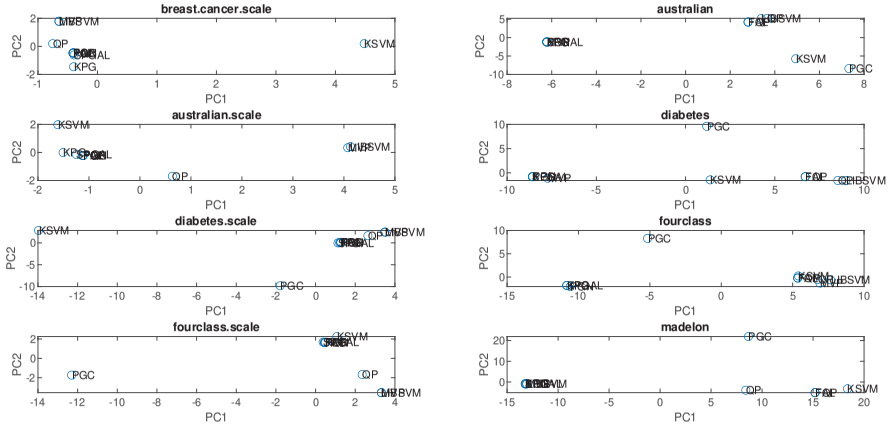
Com o objetivo de evidenciar a qualidade das soluções produzidas por diferentes algoritmos, as Figuras 18-19 apresentam gráficos comparativos das duas primeiras componentes principais, obtidos por meio da análise comparativa utilizando PCA. Os resultados obtidos seguiram o mesmo padrão obtido na Aplicação 1, em ambos os casos de função *kernel* consideradas.

Nesta aplicação, independentemente da função de *kernel*, pelas Figuras 18-19 observa-se a formação de agrupamentos distintamente visíveis nos 32 problemas, com alguns pontos apresentando deslocamentos ligeiramente maiores. Os algoritmos que mais se agruparam foram MVP e LIBSVM, enquanto FAL, FQP, SPGAL e QP ficaram como pontos periféricos aos agrupamentos de MVP e LIBSVM. Isso evidencia que as soluções dos algoritmos tomaram a mesma direção para todos os problemas até alcançarem seus critérios de parada respectivos.

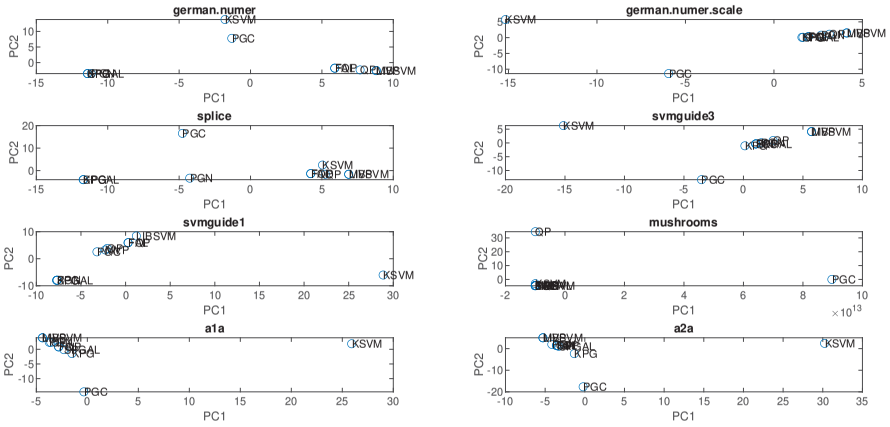
Nas Figuras 18 e 19, também destacam-se os algoritmos KSVM, PGC e PGN, devido às maiores variabilidades nas duas primeiras componentes principais em comparação com os outros algoritmos. É de se ressaltar que tais algoritmos não atenderam os critérios de paradas impostos, adotando direções mais distintas em relação aos algoritmos baseados em Restrições Ativas, Filtro e projeções.



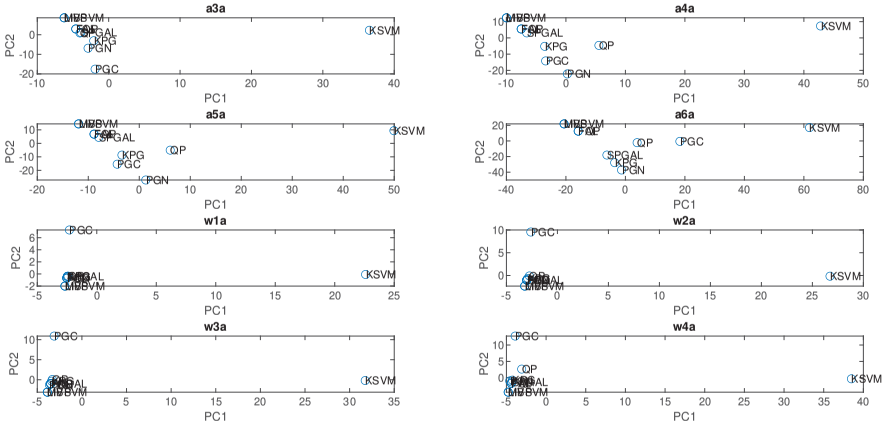
(a) Conjunto de dados: *leukemia*, *duke*, *colon.cancer*, *liver.disorders*,*sonar*, *heart*, *heart.scale* e *breast.cancer*.



(b) Conjunto de dados: *breast.cancer.scale*, *australian*, *australian.scale*,*diabetes*, *diabetes.scale*, *fourclass*, *fourclass.scale* e *madelon*.

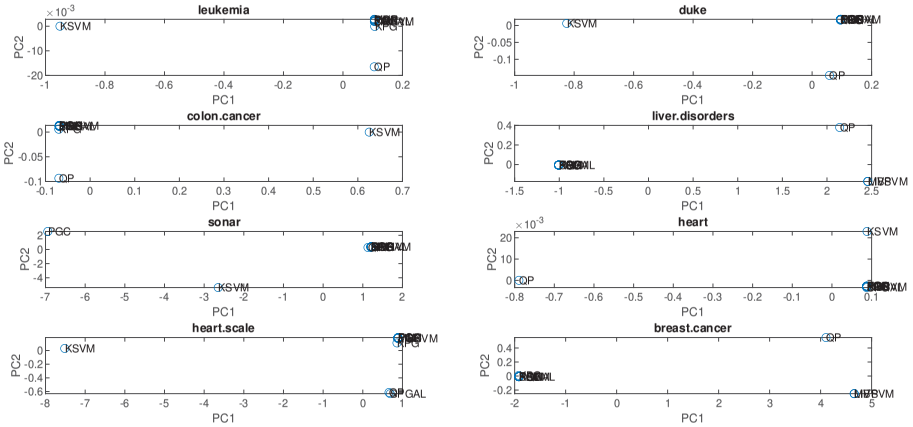


(c) Conjunto de dados: *german.numer*, *german.numer.scale*, *splice*, *svmguide1*, *mushrooms*, *a1a* e *a2a*.

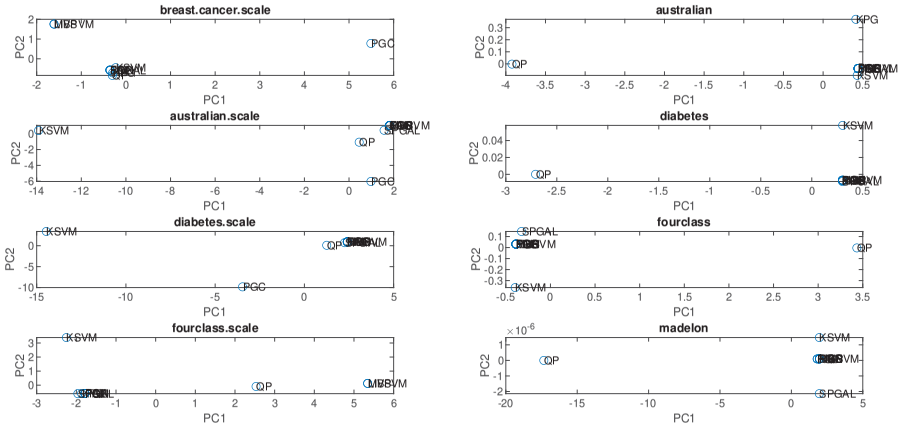


(d) Conjunto de dados: *a3a*, *a4a*, *a5a*, *a6a*, *w1a*, *w2a*, *w3a* e *w4a*.

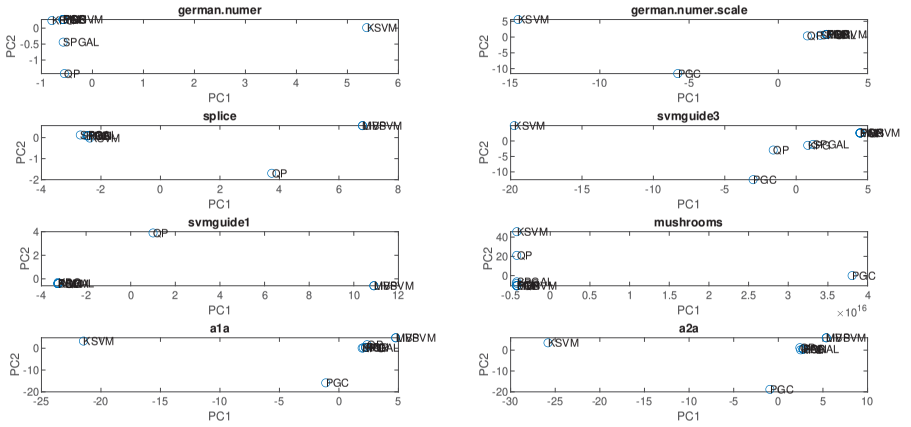
FIGURA 18 – Aplicação 2: Comparativo das componentes principais das soluções aos conjuntos de dados gerados aleatoriamente - Função *kernel* linear.



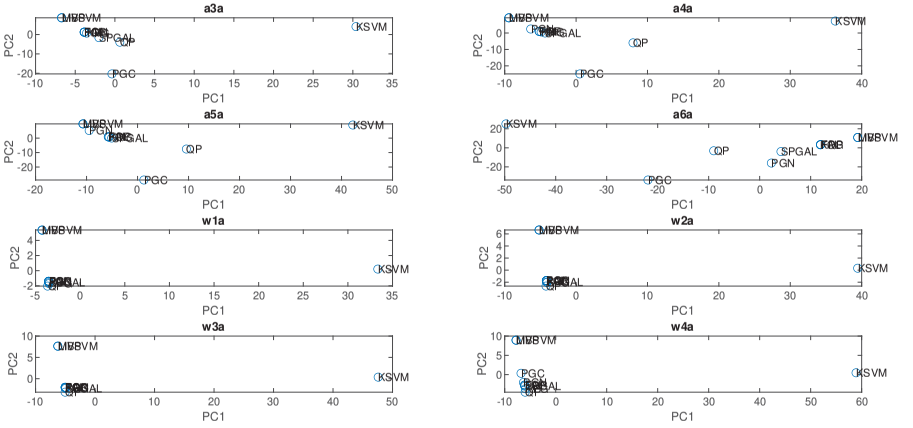
(a) Conjunto de dados: *leukemia*, *duke*, *colon.cancer*, *liver.disorders*,*sonar*, *heart*, *heart.scale* e *breast.cancer*.



(b) Conjunto de dados: *breast.cancer.scale*, *australian*, *australian.scale*,*diabetes*, *diabetes.scale*, *fourclass*, *fourclass.scale* e *madelon*.



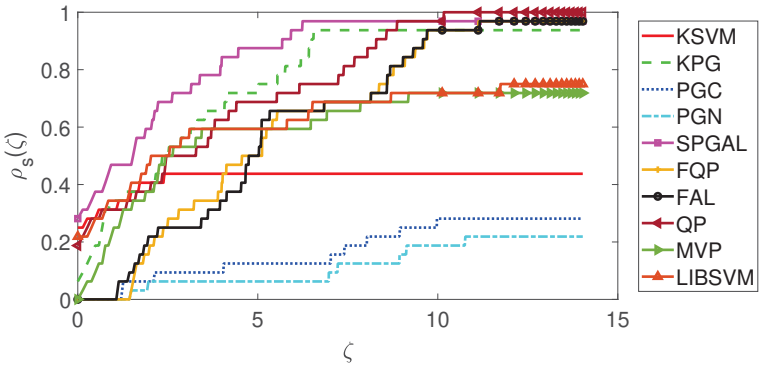
(c) Conjunto de dados: *german.numer*, *german.numer.scale*, *splice*, *svmguide1*, *mushrooms*, *a1a* e *a2a*.



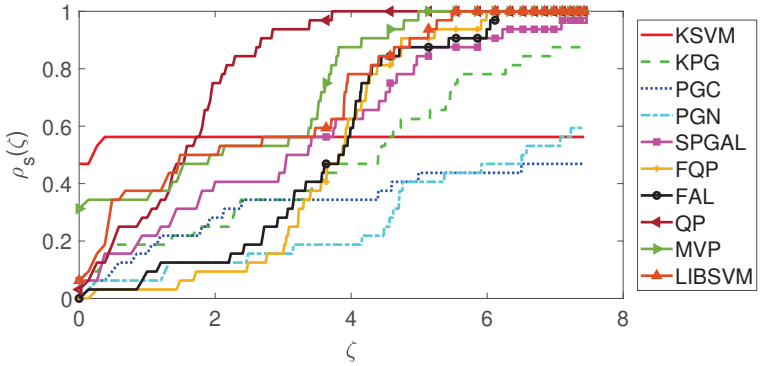
(d) Conjunto de dados: *a3a*, *a4a*, *a5a*, *a6a*, *w1a*, *w2a*, *w3a* e *w4a*.

FIGURA 19 – Aplicação 2: Comparativo das componentes principais das soluções aos conjuntos de dados gerados aleatoriamente - Função *kernel* RBF.

O perfil de desempenho dos algoritmos em relação ao tempo de CPU para o treinamento do problema é ilustrado na Figura 20, tanto para a função *kernel* linear quanto RBF.



(a) Função *kernel* linear.



(b) Função *kernel* RBF.

FIGURA 20 – Aplicação 2: Performance profile relacionado ao tempo de CPU.

Em questão de tempo de treinamento, a Figura 20 ilustra o fato que os algoritmos SPGAL, MVP e LIBSVM são os algoritmos de melhor tempo de treinamento para ambas as funções *kernel* implementadas. O algoritmo SPGAL solucionou mais rapidamente 28.1% os problemas com função *kernel* linear, seguido de LIBSVM e QP (28.1% e 18.7%, respectivamente). Já o método MVP solucionou mais rapidamente 31.5% dos problemas com a função *kernel* RBF. Vale destacar também que o método KSVM possui uma alta taxa de problemas resolvidos mais rapidamente, com 46.8% dos problemas com função *kernel* RBF e 25% com função *kernel* linear, mas com a ressalva de não atingir os critérios de parada para metade dos 64 problemas gerados. Os métodos FQP e FAL, apesar de terem convergido em 98.4% dos problemas gerados, não foram velozes na obtenção de uma solução. Já o método QP, representado pela função `quadprog` do MATLAB®, não foi veloz no comparativo nos problemas gerados com a função *kernel* RBF, mas foi um dos mais rápidos em relação à função *kernel* linear. Isto se deve aos precondicionadores que a função `quadprog` possui implementados.

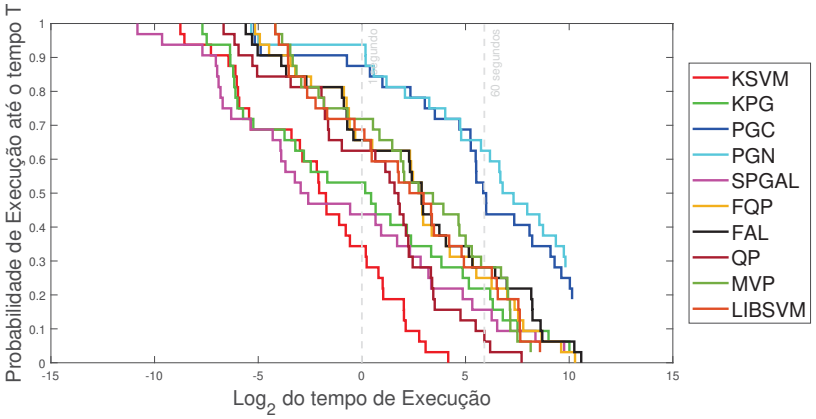
Para reforçar os argumentos apresentados, uma análise de sobrevivência dos tempos absolutos para treinamento foram realizados e ilustrados na Figura 21, para ambas

as funções *kernel* utilizadas.

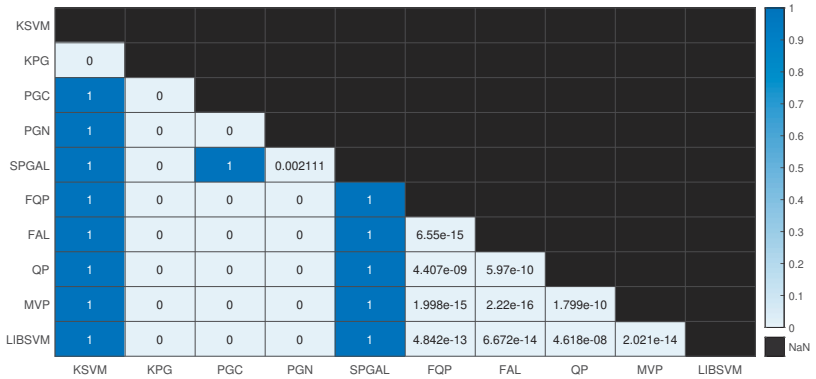
Nas Figuras 21a e 21c, que ilustram as curvas de Kaplan-Meier dos algoritmos em relação a probabilidade de execução da certo tempo, novamente destaca-se que os algoritmo SPGAL e KSVM foram os algoritmos mais velozes em termos de treinamento, seguido dos algoritmos LIBSVM, MVP e QP. Os algoritmos baseados em Filtro obtiveram performances semelhantes, sendo que suas curvas quase se sobrepõem uma a outra, porém indicando um maior tempo de treinamento para obter uma solução. Os algoritmos com a curva de Kaplan-Meier mais deslocadas para a direita neste experimento foram PGC e PGN.

Vale destacar que existem curvas de diferentes algoritmos se interseccionando nas Figuras 21a e 21c, dando um indicativo de diferenças significativas na eficiência ou tempo de execução entre esses algoritmos. As Figuras 13b e 13d, que ilustram os resultados obtidos no teste de hipótese Log-Rank para destacar a existência de diferenças significativas no tempo de treinamento entre dois algoritmos a um p-valor de 0.05, corroboram que os algoritmos KSVM e SPGAL obtiveram os valores de treinamento mais diferenciados em relação aos demais algoritmos, para o caso da função *kernel* linear, e que os algoritmos QP, MVP e LIBSVM (além do KSVM) obtiveram diferenças em relação aos demais algoritmos no caso de uso da função *kernel* RBF. Os demais algoritmos não apresentaram diferenças estatisticamente significativas entre os valores apresentados.

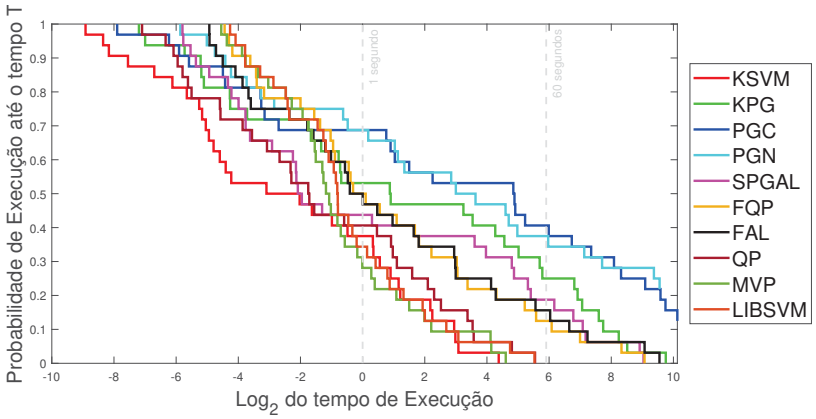
Os resultados quanto às métricas de classificação são ilustrados por boxplots na Figura 22, para a função *kernel* linear, e Figura 23 para o *kernel* RBF. Para auxiliar num comparativo entre as métricas de classificação e tempo de treinamento, gráficos de bolhas são apresentados, sendo que o tamanho da bolha indica o tempo de execução do algoritmos. Vale ressaltar que os valores considerados para o boxplot e gráfico de bolha são somente dos problemas que atingiram o critério de convergência para cada algoritmo. Para auxiliar no entendimento dos boxplots e gráficos de bolhas, a Tabela 8 destaca estatísticas descritivas (média e desvio padrão) das métricas de classificação obtidas.



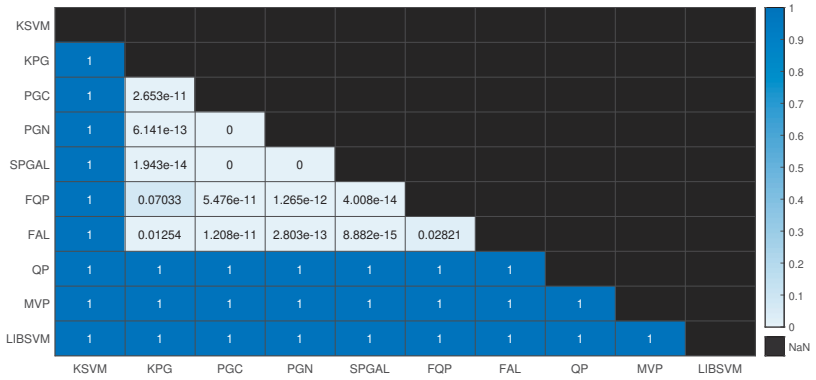
(a) Curvas de Kaplan-Meier - função *kernel* linear.



(b) Mapa de calor do teste Log-Rank - função *kernel* linear.

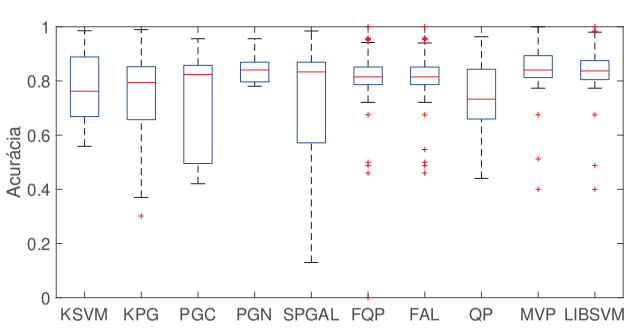


(c) Curvas de Kaplan-Meier - função *kernel* RBF.

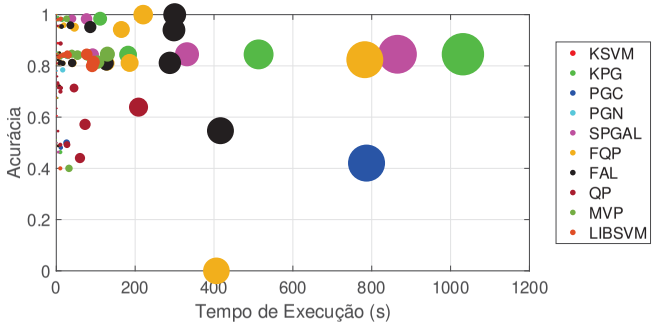


(d) Mapa de calor do teste Log-Rank - função *kernel* RBF.

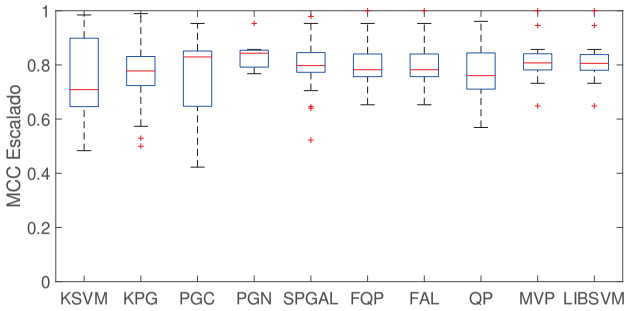
FIGURA 21 – Aplicação 2: Análise de sobrevivência dos algoritmos.



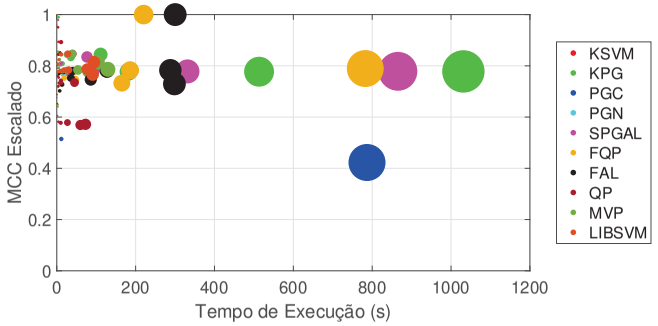
(a) Acurácia.



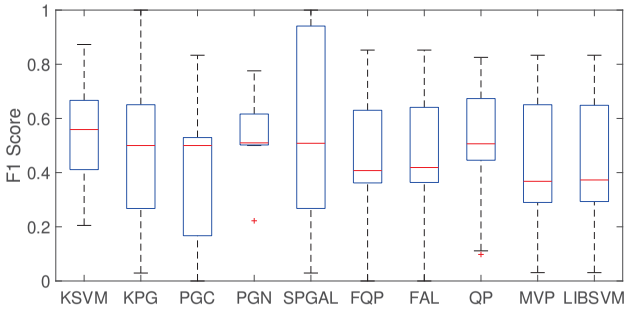
(b) Tempo execução x Acurácia.



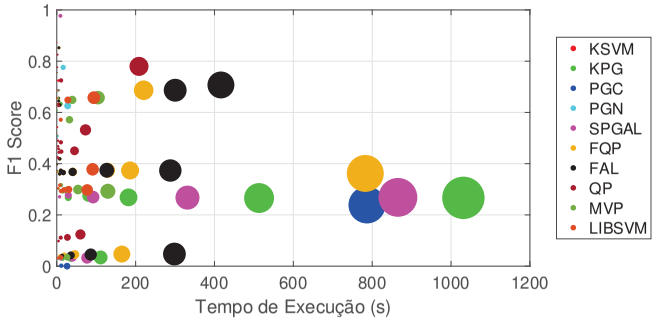
(c) MCC Escalado.



(d) Tempo execução x MCC Escalado.

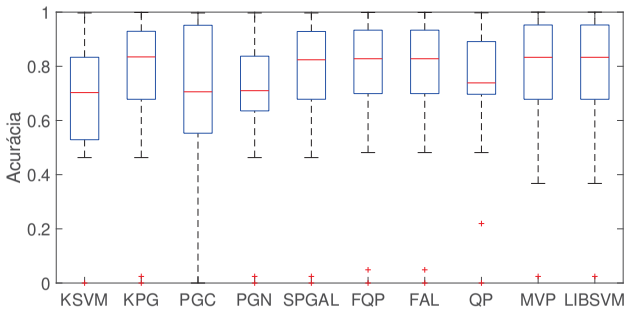


(e) *F1 Score*.

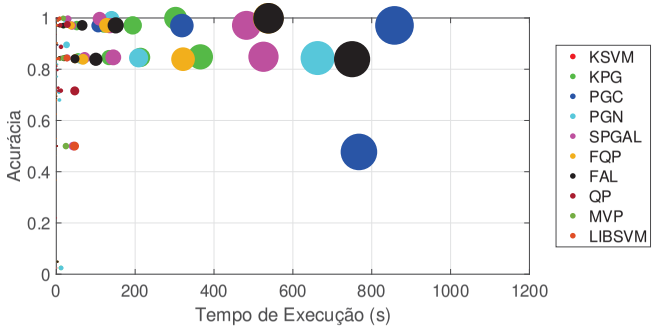


(f) Tempo execução x *F1 Score*.

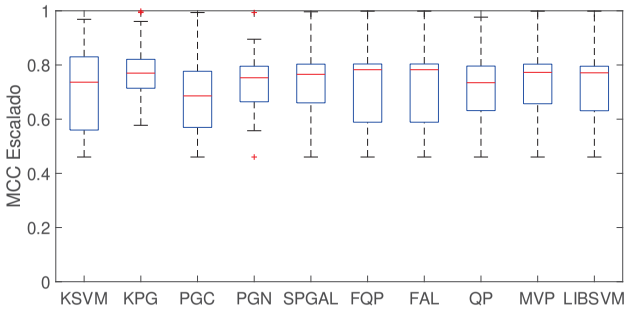
FIGURA 22 – Aplicação 2: Métricas de classificação - função *kernel* linear.



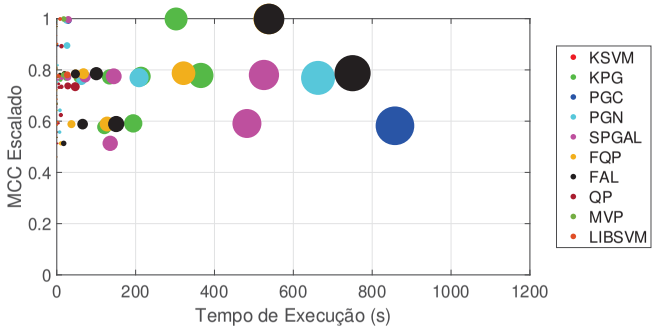
(a) Acurácia.



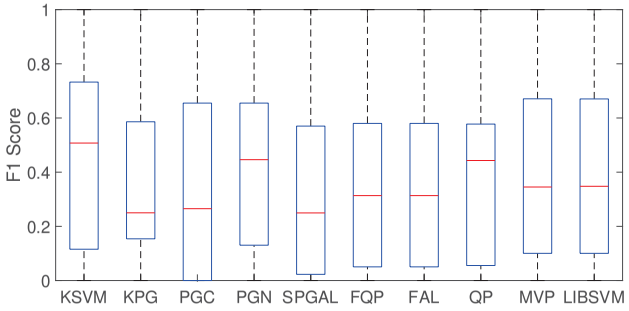
(b) Tempo execução x Acurácia.



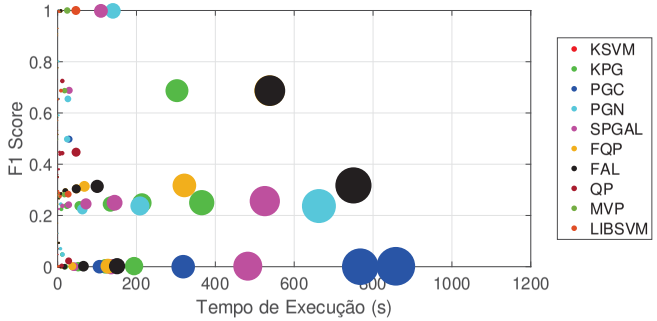
(c) MCC Escalado.



(d) Tempo execução x MCC Escalado.



(e) *F1 Score*.



(f) Tempo execução x *F1 Score*.

FIGURA 23 – Aplicação 2: Métricas de classificação - função *kernel* RBF.

TABELA 8 – Aplicação 2: Média e desvio padrão das métricas obtidas.

Algoritmos	**Função kernel linear**			
	Tempo	*Acurácia*	*MCC escalado*	*F1-Score*
KSVM	0.47 ± 1.14	0.77 ± 0.13	0.76 ± 0.15	0.55 ± 0.19
KPG	67.46 ± 207.17	0.74 ± 0.20	0.77 ± 0.12	0.47 ± 0.30
PGC	92.04 ± 260.69	0.72 ± 0.20	0.75 ± 0.18	0.38 ± 0.28
PGN	6.97 ± 10.76	0.85 ± 0.06	0.84 ± 0.06	0.53 ± 0.17
SPGAL	47.56 ± 163.86	0.74 ± 0.22	0.79 ± 0.10	0.53 ± 0.35
FQP	106.99 ± 264.78	0.78 ± 0.19	0.80 ± 0.07	0.44 ± 0.24
FAL	142.62 ± 349.02	0.80 ± 0.14	0.80 ± 0.07	0.45 ± 0.24
QP	15.56 ± 39.36	0.74 ± 0.14	0.76 ± 0.11	0.51 ± 0.20
MVP	18.53 ± 33.19	0.83 ± 0.14	0.81 ± 0.07	0.42 ± 0.24
LIBSVM	15.23 ± 27.95	0.83 ± 0.14	0.81 ± 0.07	0.41 ± 0.24
Algoritmos	**Função kernel RBF**			
	Tempo	*Acurácia*	*MCC escalado*	*F1-Score*
KSVM	0.21 ± 0.53	0.68 ± 0.23	0.71 ± 0.16	0.48 ± 0.36
KPG	53.79 ± 99.39	0.75 ± 0.25	0.78 ± 0.12	0.39 ± 0.32
PGC	109.88 ± 259.16	0.70 ± 0.25	0.70 ± 0.16	0.38 ± 0.40
PGN	50.51 ± 142.88	0.68 ± 0.25	0.73 ± 0.13	0.44 ± 0.35
SPGAL	51.35 ± 125.23	0.75 ± 0.24	0.74 ± 0.14	0.34 ± 0.32
FQP	37.73 ± 109.55	0.76 ± 0.24	0.74 ± 0.14	0.37 ± 0.31
FAL	54.01 ± 160.60	0.76 ± 0.24	0.74 ± 0.14	0.37 ± 0.31
QP	4.27 ± 9.62	0.74 ± 0.22	0.73 ± 0.13	0.40 ± 0.30
MVP	2.62 ± 5.80	0.77 ± 0.22	0.75 ± 0.13	0.43 ± 0.35
LIBSVM	3.51 ± 9.21	0.77 ± 0.22	0.74 ± 0.13	0.43 ± 0.35

Observando as Figuras 22 e 23, em conjunto com a Tabela 5, e considerando o número de problemas resolvidos, o algoritmo PGN, que possui uma amplitude interquartil baixa e com uma mediana relativamente similar com os algoritmos implementados para as três métricas ilustradas, não foi considerado com bom desempenho em vista da baixa quantidade de problemas solucionados. Já os algoritmos KSVM, KPG e PGC não apresentaram uma boa performance nesta bateria de testes, além da baixa quantidade de problemas resolvidos nesta aplicação.

Os algoritmos com melhor desempenho em termos da classificação são FQP, FAL, LIBSVM, MVP e QP, não necessariamente nesta ordem, uma vez que os respectivos quartis ilustrados para cada métrica são mais competitivos entre os algoritmos implementados, além de serem os algoritmos com maior quantidade de problemas que convergiram. Observa-se que FQP e FAL possuem quartis muitos semelhantes nas três métricas analisadas, com valores de medianas ligeiramente maiores quando comparados com os algoritmos MVP e LIBSVM com funções *kernel* linear e RBF.

Outra observação pertinente é a maior variabilidade de *F1 Score*, que resultou em maiores amplitudes interquartis para ambas as funções *kernel* nesta aplicação, bem como medianas inferiores a 0.6, quando comparado com os resultados apresentados na Seção 4.4. Isto é reflexo do desbalanceamento das classes para treinamento e teste dos conjuntos de dados.

Apesar de ter convergido para uma grande quantidade de problemas, o algoritmo SPGAL definiu classificadores com uma maior amplitude interquartil em termos de acurácia e *F1 Score*, não o tornando competitivo com o FQP, FAL, MVP, LIBSVM e QP com função *kernel* linear. Porém, é de se ressaltar que o método SPGAL pode ser

considerado competitivo quando observados apenas os resultados da função *kernel* RBF.

TABELA 9 – Aplicação 2: Resumo das métricas obtidas e *ranking* das performances.

Função kernel Linear									
Algoritmos	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*Ranque*
QP	32	0	0	0.74	0.51	0.76	15.56	20.00	1
SPGAL	31	0	1	0.74	0.53	0.79	47.56	16.14	2
LIBSVM	26	5	1	0.83	0.41	0.81	15.23	16.04	3
MVP	25	6	1	0.83	0.42	0.81	18.53	15.04	4
KPG	30	1	1	0.74	0.47	0.77	67.46	13.47	5
FQP	31	0	1	0.78	0.44	0.80	106.99	10.19	6
KSVM	14	18	0	0.77	0.55	0.76	0.47	9.52	7
FAL	31	0	1	0.80	0.45	0.80	142.62	6.63	8
PGN	7	16	9	0.85	0.53	0.84	6.97	4.22	9
PGC	9	17	6	0.72	0.38	0.75	92.04	-3.00	10
Função kernel RBF									
Algoritmos	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*Ranque*
MVP	32	0	0	0.77	0.43	0.75	2.62	21.29	1
LIBSVM	32	0	0	0.77	0.43	0.74	3.51	21.20	2
QP	32	0	0	0.74	0.40	0.73	4.27	21.11	3
FQP	32	0	0	0.76	0.37	0.74	37.73	17.77	4
SPGAL	32	0	0	0.75	0.34	0.74	51.35	16.40	5
FAL	32	0	0	0.76	0.37	0.74	54.01	16.14	6
KPG	28	4	0	0.75	0.39	0.78	53.79	13.50	7
KSVM	18	14	0	0.68	0.48	0.71	0.21	12.19	8
PGN	23	2	7	0.68	0.44	0.73	50.51	10.49	9
PGC	19	9	4	0.70	0.38	0.70	109.88	1.88	10

Nota: A = Quantidade de problemas resolvidos; B = Quantidade de problemas que atingiram o máximo de iterações imposto; C = Quantidade de problemas que atingiram o limite de tempo; D = Acurácia média para os problemas solucionados; E = *F1 Score* médio para os problemas solucionados; F = MCC escalado médio para os problemas solucionados; G = Tempo de treinamento médio para os problemas solucionados; H = Métrica de *ranking* (4.3).

A partir dos resultados explicitados na Tabela 9 para esta aplicação, com os resultados obtidos nesta aplicação, os métodos baseados em projeção não conseguiram atender os critérios de parada implementados, com exceção do SPGAL. Os métodos baseados em Lagrangiano Aumentado não atenderam os critérios de paradas impostos, tendo também uma baixa quantidade de problemas solucionados. Analogamente aos resultados da Aplicação 1 na Seção 4.4, os métodos baseados em Filtro, Pontos Interiores e SPGAL também resultaram em boas métricas de classificação às custas de um alto custo computacional no treinamento do classificador. Novamente, os métodos baseados em Restrições Ativas apresentaram um melhor equilíbrio, com baixos valores de tempo de treinamento e significativos valores para as métricas de classificação analisadas.

4.6 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Foram apresentados, neste capítulo, os experimentos numéricos envolvendo o treinamento de um classificador por SVM aplicando métodos de Otimização clássicos. Considerou-se duas aplicações, sendo uma baseada em conjuntos de dados gerados aleatoriamente e outra baseada em conjuntos de dados *benchmark* extraídos de repositórios de Aprendizagem de Máquina. Os resultados obtidos foram discutidos e analisados sob o ponto de vista de performance de treinamento e capacidade de generalização dos modelos gerados, e o que ficou evidenciado é um melhor desempenho dos métodos SPGAL, FQP, FAL, QP, MVP e LIBSVM (não necessariamente nesta ordem). Entretanto, vale a

ressalva de que o tempo de treinamento avaliado para grande parte dos métodos testados foi a principal desvantagem em relação aos métodos baseados em Restrições Ativas.

Para os experimentos numéricos conduzidos, foram implementadas funções *kernel* linear e RBF a fim de desenvolver modelos de classificação que permitem modelar as relações entre os dados de entrada (lineares ou não). Nos resultados obtidos, a principal diferença do uso destas duas funções foi quanto ao custo computacional para o treinamento. Esses resultados evidenciam a importância de escolher cuidadosamente o método de Otimização mais adequado para o treinamento de SVM, considerando tanto a natureza dos dados quanto as metas de desempenho esperadas.

Os métodos de Otimização, destacados no Capítulo 3, foram analisados e implementados para resolver o problema de treinamento de SVM. Tais métodos possuem suas especificidades, necessitando a calibração de seus parâmetros e os critérios de parada impostos. Assim, suas performances são totalmente dependentes de seus ajustes.

Nos testes executados, é possível observar que os métodos implementados que atingiram os critérios de parada impostos possuíram um desempenho competitivo entre si, com exceção dos métodos de projeção (Gradiente e Newton). De forma geral, foi identificada a falta de atendimento aos critérios de parada dos métodos de Otimização considerados, o que pode ser atribuído a diferentes fatores. Um deles pode ser o tipo do critério de parada, bem como a rigidez da tolerância estabelecida no critério de parada. Ao relaxar essa tolerância, é possível que alguns métodos tenham conseguido alcançar os critérios desejados. Isso ocorre porque a diminuição da exigência de precisão na convergência pode permitir que os métodos alcancem soluções aproximadas satisfatórias. No entanto, é importante encontrar um equilíbrio entre a flexibilização da tolerância e a garantia de resultados de qualidade.

No problema para o treinamento de SVM, outro fator que pode influenciar a não convergência de alguns métodos de Otimização é a presença de sistemas lineares a serem resolvidos durante o seu processo iterativo. Se esses sistemas lineares são mal-condicionados, isto é, possuem números de condição elevados, isso pode dificultar a obtenção de soluções precisas. A má condição de um sistema linear pode levar a problemas numéricos, como erros de arredondamento e instabilidade numérica, que impactam a capacidade do método de encontrar uma solução aceitável. Portanto, é necessário considerar a natureza dos sistemas lineares envolvidos e utilizar métodos adequados para lidar com essas situações desafiadoras.

5 CONSIDERAÇÕES FINAIS DA TESE

O abordagem de SVM é um método de Aprendizado de Máquina supervisionado utilizado tanto para regressão como para classificação de dados, conforme apresentado neste trabalho. O objetivo da SVM é encontrar um hiperplano que separe de forma otimizada as observações de duas classes em um espaço de características de alta dimensão. Para isso, a SVM maximiza a margem entre o hiperplano de separação e as amostras mais próximas de cada classe, que são chamadas de vetores de suporte. A SVM pode lidar com problemas de classificação não lineares através do uso de funções *kernel*, que mapeiam as amostras para um espaço de características de alta dimensão em que se espera que possam ser linearmente separáveis. O método SVM é amplamente utilizado em áreas como reconhecimento de padrões, processamento de imagem e bioinformática, entre outras.

Neste trabalho, foi proposta uma comparação entre diferentes métodos de Otimização aplicados ao problema de treinamento de SVM. Os métodos implementados seguem de referências clássicas da literatura de Otimização, formulados para problemas mais gerais, que abrangem abordagens de métodos direcionais, Pontos Interiores, Restrições Ativas e Filtro. Alguns destes métodos implementados emulam os métodos de Otimização difundidos, com a diferença de que todos os métodos foram implementados pelo autor em uma mesma linguagem de programação. Para executar uma comparação básica, os métodos implementados foram comparados em termos de métricas de classificação, a saber, acurácia, coeficiente de correlação de Matthews, *F1 Score* e tempo de CPU para treinamento. Os problemas utilizados foram desde conjuntos de dados aleatórios até dados *benchmark* extraídos de repositórios reconhecidos em Aprendizagem de Máquinas, com variadas características.

Quanto aos resultados obtidos, os métodos envolvendo Lagrangiano aumentado se saíram bem quando as observações dos conjuntos de dados possuíam classes balanceadas, sendo velozes ao atender os critérios de paradas definidos. Os métodos de projeção não tiveram uma boa performance com hiperparâmetros dos métodos fixados, com exceção do método de Gradiente Espectral Projetado. De forma geral, os métodos envolvendo Filtro e Pontos Interiores atingiram boas métricas de classificação, mas foram mais lentos ao realizarem o treinamento. Os métodos baseados em Restrições Ativas, emulando Platt (1998) e Fan et al. (2005), possuem um maior equilíbrio entre velocidade de treinamento e as métricas de classificação. Assim, os resultados indicam que a implementação básica de algoritmos baseados em SMO são mais eficientes que as demais classes de Otimização aqui implementadas.

REFERÊNCIAS

BERTSEKAS, D. P. Projected newton methods for optimization problems with simple constraints. *SIAM Journal on control and Optimization*, SIAM, v. 20, n. 2, p. 221–246, 1982.

BIRGIN, E. G.; MARTÍNEZ, J. M.; RAYDAN, M. Nonmonotone spectral projected gradient methods on convex sets. *SIAM Journal on Optimization*, SIAM, v. 10, n. 4, p. 1196–1211, 2000. DOI: ⟨https://doi.org/10.1137/S1052623497330963⟩.

BISHOP, C. M. *Pattern recognition and machine learning*. [S.l.]: Springer, 2006.

BOSER, B. E.; GUYON, I. M.; VAPNIK, V. N. A training algorithm for optimal margin classifiers. In: *Proceedings of the fifth annual workshop on Computational learning theory*. [S.l.: s.n.], 1992. p. 144–152. DOI: ⟨https://doi.org/10.1145/130385.130401⟩.

BUITINCK, L.; LOUPPE, G.; BLONDEL, M.; PEDREGOSA, F.; MUELLER, A.; GRISEL, O.; NICULAE, V.; PRETTENHOFER, P.; GRAMFORT, A.; GROBLER, J.; LAYTON, R.; VANDERPLAS, J.; JOLY, A.; HOLT, B.; VAROQUAUX, G. API design for machine learning software: experiences from the scikit-learn project. In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. [S.l.: s.n.], 2013. p. 108–122. ArXiv: ⟨arxiv.org/abs/1309.0238⟩.

CARRIZOSA, E.; MORALES, D. R. Supervised classification and mathematical optimization. *Computers & Operations Research*, Elsevier, v. 40, n. 1, p. 150–165, 2013. DOI: ⟨https://doi.org/10.1016/j.cor.2012.05.015⟩.

CHANG, C.-C.; LIN, C.-J. Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, Acm New York, NY, USA, v. 2, n. 3, p. 1–27, 2011. DOI: ⟨https://doi.org/10.1145/1961189.1961199⟩.

CHAUHAN, V. K.; DAHIYA, K.; SHARMA, A. Problem formulations and solvers in linear svm: a review. *Artificial Intelligence Review*, Springer, v. 52, n. 2, p. 803–855, 2019. DOI: ⟨https://doi.org/10.1007/s10462-018-9614-6⟩.

CORES, D.; ESCALANTE, R.; GONZÁLEZ-LIMA, M.; JIMENEZ, O. On the use of the spectral projected gradient method for support vector machines. *Computational & Applied Mathematics*, SciELO Brasil, v. 28, p. 327–364, 2009. DOI: ⟨https://doi.org/10.1590/S1807-03022009000300005⟩.

CORTES, C.; VAPNIK, V. N. Support-vector networks. *Machine learning*, Springer, v. 20, n. 3, p. 273–297, 1995. DOI: ⟨https://doi.org/10.1007/BF00994018⟩.

CRISTIANINI, N.; SHAWE-TAYLOR, J. et al. *An introduction to support vector machines and other kernel-based learning methods*. [S.l.]: Cambridge University press, 2000.

DENG, N.; TIAN, Y.; ZHANG, C. *Support Vector Machines - Optimization based theory, algorithms, and extensions*. [S.l.]: Chapman & Hall/CRC Press, 2012.

DOLAN, E. D.; MORÉ, J. J. Benchmarking optimization software with performance profiles. *Mathematical programming*, Springer, v. 91, n. 2, p. 201–213, 2002. DOI: ⟨https://doi.org/10.1007/s101070100263⟩.

FAN, R.-E.; CHANG, K.-W.; HSIEH, C.-J.; WANG, X.-R.; LIN, C.-J. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, v. 9, n. Aug, p. 1871–1874, 2008.

FAN, R.-E.; CHEN, P.-H.; LIN, C.-J.; JOACHIMS, T. Working set selection using second order information for training support vector machines. *Journal of Machine Learning Research*, v. 6, n. 12, 2005.

FERRIS, M. C.; MUNSON, T. S. Interior-point methods for massive support vector machines. *SIAM Journal on Optimization*, SIAM, v. 13, n. 3, p. 783–804, 2002. DOI: ⟨https://doi.org/10.1137/S1052623400374379⟩.

FERRIS, M. C.; MUNSON, T. S. Semismooth support vector machines. *Mathematical Programming*, Springer, v. 101, n. 1, p. 185–204, 2004. DOI: ⟨https://doi.org/10.1007/s10107-004-0541-8⟩.

FLETCHER, R.; LEYFFER, S. Nonlinear programming without a penalty function. *Mathematical programming*, Springer, v. 91, n. 2, p. 239–269, 2002. DOI: ⟨https://doi.org/10.1007/s101070100244⟩.

GLASMACHERS, T.; IGEL, C. Second-order smo improves svm online and active learning. *Neural Computation*, MIT Press, v. 20, n. 2, p. 374–382, 2008. DOI: ⟨https://doi.org/10.1162/neco.2007.10-06-354⟩.

GONZALEZ-LIMA, M. D.; HAGER, W. W.; ZHANG, H. An affine-scaling interior-point method for continuous knapsack constraints with application to support vector machines. *SIAM Journal on Optimization*, SIAM, v. 21, n. 1, p. 361–390, 2011. DOI: ⟨https://doi.org/10.1137/090766255⟩.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The elements of statistical learning: data mining, inference, and prediction*. [S.l.]: Springer Science & Business Media, 2009.

HESTENES, M. R. Multiplier and gradient methods. *Journal of optimization theory and applications*, Springer, v. 4, n. 5, p. 303–320, 1969. DOI: ⟨https://doi.org/10.1007/BF00927673⟩.

HSU, C.-W.; LIN, C.-J. A simple decomposition method for support vector machines. *Machine Learning*, Springer, v. 46, n. 1, p. 291–314, 2002. DOI: ⟨https://doi.org/10.1023/A:1012427100071⟩.

JAMES, G.; WITTEN, D.; HASTIE, T.; TIBSHIRANI, R. *An introduction to statistical learning*. [S.l.]: Springer, 2013. v. 112. DOI: ⟨doi.org/10.1007/978-1-4614-7138-7⟩.

JOACHIMS, T. *Making large-scale SVM learning practical*. [S.l.], 1998.

KEERTHI, S. S.; SHEVADE, S. K.; BHATTACHARYYA, C.; MURTHY, K. R. K. Improvements to platt's smo algorithm for svm classifier design. *Neural computation*, MIT Press, v. 13, n. 3, p. 637–649, 2001. DOI: ⟨https://doi.org/10.1162/089976601300014493⟩.

KLEIN, J. P.; MOESCHBERGER, M. L. et al. *Survival analysis: techniques for censored and truncated data*. [S.l.]: Springer, 2003. v. 1230.

KRULIKOVSKI, E. H. M. *Análise teórica de máquinas de vetores suporte e aplicação a classificação de caracteres*. Dissertação (Mestrado) — Universidade Federal do Paraná, 2017.

LEE, E. T.; WANG, J. *Statistical methods for survival data analysis*. [S.l.]: John Wiley & Sons, 2003. v. 476.

LUCIDI, S.; PALAGI, L.; RISI, A.; SCIANDRONE, M. A convergent hybrid decomposition algorithm model for svm training. *IEEE transactions on neural networks*, IEEE, v. 20, n. 6, p. 1055–1060, 2009. DOI: ⟨https://doi.org/10.1109/TNN.2009.2020908⟩.

LUENBERGER, D. G.; YE, Y. et al. *Linear and nonlinear programming*. [S.l.]: Springer, 1984. v. 2.

MANGASARIAN, O. L.; MUSICANT, D. R. Lagrangian support vector machines. *Journal of Machine Learning Research*, v. 1, n. Mar, p. 161–177, 2001.

MATTHEWS, B. W. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, Elsevier, v. 405, n. 2, p. 442–451, 1975. DOI: ⟨https://doi.org/10.1016/0005-2795(75)90109-9⟩.

MEYER, D.; DIMITRIADOU, E.; HORNIK, K.; WEINGESSEL, A.; LEISCH, F. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. [S.l.], 2019. R package version 1.7-3. Disponível em: ⟨https://CRAN.R-project.org/package=e1071⟩.

MOORE, D. S. *Introduction to the Practice of Statistics*. [S.l.]: WH Freeman and company, 2009.

NIU, D.; WANG, C.; TANG, P.; WANG, Q.; SONG, E. A sparse semismooth newton based augmented lagrangian method for large-scale support vector machines. *arXiv preprint arXiv:1910.01312*, 2019. DOI: ⟨https://doi.org/10.48550/arXiv.1910.01312⟩.

NOCEDAL, J.; WRIGHT, S. *Numerical optimization*. [S.l.]: Springer Science & Business Media, 2006.

OSUNA, E.; FREUND, R.; GIROSIT, F. Training support vector machines: an application to face detection. In: IEEE. *Proceedings of IEEE computer society conference on computer vision and pattern recognition*. [S.l.], 1997. p. 130–136. DOI: ⟨https://doi.org/10.1109/CVPR.1997.609310⟩.

PERIÇARO, G. A.; RIBEIRO, A. A.; KARAS, E. W. Global convergence of a general filter algorithm based on an efficiency condition of the step. *Applied Mathematics and Computation*, Elsevier, v. 219, n. 17, p. 9581–9597, 2013. DOI: ⟨https://doi.org/10.1016/j.amc.2013.03.012⟩.

PICCIALLI, V.; SCIANDRONE, M. Nonlinear optimization and support vector machines. *4OR*, Springer, v. 16, n. 2, p. 111–149, 2018. DOI: ⟨https://doi.org/10.1007/s10288-018-0378-2⟩.

PLATT, J. *Sequential minimal optimization: A fast algorithm for training support vector machines*. [S.l.], 1998.

POWELL, M. J. A method for nonlinear constraints in minimization problems. *Optimization*, Academic Press, p. 283–298, 1969.

ROCKAFELLAR, R. T. The multiplier method of hestenes and powell applied to convex programming. *Journal of Optimization Theory and applications*, Springer, v. 12, n. 6, p. 555–562, 1973. DOI: ⟨doi.org/10.1007/BF00934777⟩.

ROSEN, J. B. The gradient projection method for nonlinear programming. part ii. nonlinear constraints. *Journal of the Society for Industrial and Applied Mathematics*, SIAM, v. 9, n. 4, p. 514–532, 1961. DOI: ⟨https://doi.org/10.1137/0109044⟩.

SCHEINBERG, K.; BENNETT, K. P.; PARRADO-HERNÁNDEZ, E. An efficient implementation of an active set method for svms. *Journal of Machine Learning Research*, v. 7, n. 10, 2006.

SCHOLKOPF, B.; SMOLA, A. J. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. [S.l.]: MIT press, 2001.

SERAFINI, T.; ZANGHIRATI, G.; ZANNI, L. Gradient projection methods for quadratic programs and applications in training support vector machines. *Optimization Methods and Software*, Taylor & Francis, v. 20, n. 2-3, p. 353–378, 2005. DOI: ⟨https://doi.org/10.1080/10556780512331318182⟩.

SERAFINI, T.; ZANNI, L. On the working set selection in gradient projection-based decomposition techniques for support vector machines. *Optimization Methods and Software*, Taylor & Francis, v. 20, n. 4-5, p. 583–596, 2005. DOI: ⟨https://doi.org/10.1080/10556780500140714⟩.

SHALEV-SHWARTZ, S.; SINGER, Y.; SREBRO, N.; COTTER, A. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, Springer, v. 127, n. 1, p. 3–30, 2011. DOI: ⟨https://doi.org/10.1007/s10107-010-0420-4⟩.

SHAWE-TAYLOR, J.; CRISTIANINI, N. *Kernel methods for pattern analysis*. [S.l.]: Cambridge University press, 2004.

TORREALBA, E.; SILVA, J.; MATIOLI, L.; KOLOSSOSKI, O.; SANTOS, P. Augmented lagrangian algorithms for solving the continuous nonlinear resource allocation problem. *European Journal of Operational Research*, Elsevier, 2021. DOI: ⟨https://doi.org/10.1016/j.ejor.2021.11.027⟩.

WOODSEND, K.; GONDZIO, J. Hybrid mpi/openmp parallel linear support vector machine training. *Journal of Machine Learning Research*, JMLR. org, v. 10, p. 1937–1953, 2009.

WOODSEND, K.; GONDZIO, J. Exploiting separability in large-scale linear support vector machine training. *Computational Optimization and Applications*, Springer, v. 49, n. 2, p. 241–269, 2011. DOI: ⟨https://doi.org/10.1007/s10589-009-9296-8⟩.

WRIGHT, S. J. *Primal-dual interior-point methods*. [S.l.]: SIAM, 1997.

YAN, Y.; LI, Q. An efficient augmented lagrangian method for support vector machine. *Optimization Methods and Software*, Springer, 2020. DOI: ⟨https://doi.org/10.1080/10556788.2020.1734002⟩.

YIN, J.; LI, Q. A semismooth newton method for support vector classification and regression. *Computational Optimization and Applications*, Springer, v. 73, n. 2, p. 477–508, 2019. DOI: ⟨https://doi.org/10.1007/s10589-019-00075-z⟩.

ZANGHIRATI, G.; ZANNI, L. A parallel solver for large quadratic programs in training support vector machines. *Parallel computing*, Elsevier, v. 29, n. 4, p. 535–551, 2003. DOI: ⟨https://doi.org/10.1016/S0167-8191(03)00021-8⟩.

ZHANG, Q.; WANG, D.; WANG, Y. Convergence of decomposition methods for support vector machines. *Neurocomputing*, Elsevier, v. 317, p. 179–187, 2018. DOI: ⟨https://doi.org/10.1016/j.neucom.2018.08.030⟩.

ANEXO A – HIPERPARÂMETROS DOS MÉTODOS DE OTIMIZAÇÃO ANALISADOS

Os hiperparâmetros utilizados, bem como os critérios de parada dos algoritmos, são destacados a seguir. Os valores fixados foram definidos a partir de pré-testes com alguns dos conjuntos de dados aqui trabalhados. É de se ressaltar que as notações que serão apresentadas são referentes aos algoritmos propostos pelos trabalhos apresentados na Tabela 2.

- Para todos os algoritmos foram considerados como critério de parada (i) o tempo máximo de treinamento (20 minutos), (ii) o máximo de iterações (definido para cada algoritmo) e (iii) os critérios de parada sugeridos pelos artigos dos quais algoritmos foram extraídos;

- algoritmo de Periçaro, Ribeiro e Karas (2013, Alg. 1), implementado em FQP e FAL: Considere $x \in \mathbb{R}^n$ a variável de decisão do problema (2.10). A função de medida de inviabilidade é definida como $h : \mathbb{R}^n \to \mathbb{R}_+$

$$h(x) = |y^T x - 10^{-8}|,$$

onde $|\cdot|$ é o valor absoluto. O critério de parada adotado é

$$\|P_{\mathcal{L}_k}(x^{k+1} - \nabla f(x^{k+1})) - x^k\|_\infty \leq 10^{-5} \text{ e } h(x^{k+1}) \leq 10^{-5},$$

com $P_{\mathcal{L}_k}$ sendo a projeção ortogonal no conjunto linearizado $\mathcal{L}_k$, ou se satisfeitos os seguintes critérios derivados das condições de KKT de otimalidade,

$$\|\nabla f(x^{k+1}) + A^T\lambda\| \leq 10^{-5} \text{ e } \|\lambda_I \circ c_I\|_\infty \leq \delta \text{ and } h(\alpha^{k+1}) \leq 10^{-5},$$

onde $\circ$ é o produto Hadamard, A é a matriz Jacobiana das restrições do problema (2.10), λ e λ_I os multiplicadores de Lagrange da restrição de igualdade e caixa do problema (2.10), respectivamente. Para o algoritmo de Filtro, o máximo de iterações foi fixado em 50.

- algoritmo SQP adaptado para o Filtro, de Periçaro, Ribeiro e Karas (2013, Alg. 2), implementado em FQP e FAL: a constante da região proibida é $\alpha = 0.1$, o ponto inicial é $x^0 = 0$, a matriz Hessiana do modelo quadrático como a própria matriz Q. Quanto ao algoritmo SQP, o raio inicial da região de confiança é definido como $\Delta_0 = 10^6 \cdot \min\{\|\nabla f(x^0)\|, h(x^0)\}$, com ∇f sendo o vetor gradiente da função objetivo do problema (2.10), $\xi = 0.8$, $\eta = 0.01$, $c_p = 10^{-4}$ e $\gamma = 0.5$. Para o algoritmo de

SQP não foi fixado o máximo de iterações, mas um limite de tempo de execução em 30 minutos.

Para o caso do algoritmo FQP implementado, que utiliza a função `quadprog` para resolver os subproblemas de viabilidade e otimalidade gerados pelo algoritmo SQP, foi definido o algoritmo de Pontos Interiores com uma tolerância de 10^{-5} para convergência. Já para o algoritmo FAL, aplicou-se o `quadprog` para resolver o subproblema de otimalidade nas mesmas condições que FQP, mas com o algoritmo Torrealba et al. (2021) para resolver o problema de viabilidade com os seguintes parâmetros: o multiplicador de Lagrange inicial $\lambda^0 = 1$, o parâmetro de penalidade $r = 1$ fixo, constante de atualização $\beta = 1$ e $k_{\max} = 6000$;

- algoritmo de Torrealba et al. (2021, Alg. 1), implementado em KSVM e KPG: O multiplicador de Lagrange inicial foi fixado em $\lambda^0 = 1$, o parâmetro de penalidade $r^k = 1$ para todo k, a constante de atualização $\beta = 1$ e o limite máximo de iteração $k_{\max} = 100000$. Os critérios de convergência deste algoritmo são

$$|y^T x^{k+1}| \leq 10^{-5} \text{ e } \|x^{k+1} - x^k\| \leq 10^{-5};$$

- algoritmo de Luenberger, Ye et al. (1984), implementado em PGC: por opção do autor, não foi construída a matriz linhas das restrições ativas A_q. Ao invés disso, armazenou-se os índices das restrições ativas. Para a iteração k, a direção d, definida pela projeção ortogonal entre o vetor gradiente ∇f da função objetivo do problema (2.10) e o plano Ω da restrição de igualdade do problema (2.10), é obtida a partir da aplicação do processo de Gram-Schmidt na forma

$$d = \text{proj}_{\Omega}(\nabla f(x^k)) = \nabla f(x^k) - \frac{y^T \nabla f(x^k)}{y^T y} y.$$

O novo ponto é obtido via busca exata, ou seja

$$\alpha = -\frac{d^T \nabla f}{d^T Q d},$$

observando se cada coordenada do novo ponto está na caixa, projetando-o caso não esteja. O critério de convergência adotado foi o mesmo descrito em Luenberger, Ye et al. (1984).

No caso de PGN, a única diferença recai em como se considera a direção. Ao contrário de PGC, que considera o vetor gradiente de cada iteração, o PGN considerá a direção

$$d_N = x^* - x^k,$$

com x^* sendo a solução do sistema linear gerado pela equação de Newton, $Qx^* - e = 0$, com $e \in \mathbb{R}^n$ um vetor de uns. Para garantir que a direção sempre seja de descida, foi implementada uma salvaguarda para observar o ângulo entre a direção computada e o vetor gradiente da função objetivo,

$$\nabla f(x^k)^T d_N \geq -10^{-8},$$

adotando a direção de máxima descida caso satisfeita. Quanto à obtenção de um novo ponto e o critério de parada, o algoritmo PGN segue o mesmo processo que PGC;

- algoritmo de Birgin, Martínez e Raydan (2000), implementado para SPGAL: Por utilizar a função Lagrangiana aumentada, considerou-se como multiplicador de Lagrange inicial $\lambda_{AL}^0 = 1$ e o parâmetro de penalidade ρ constante igual a 1. O multiplicador de Lagrange é atualizado da forma

$$\lambda_{\mathrm{AL}}^{k+1} = \lambda_{\mathrm{AL}}^k + \rho y^T x^k.$$

A direção foi computada considerando o vetor gradiente da função objetivo do problema (4.2) e o passo espectral λ calculado conforme o passo Barzilai-Borwein, partindo de $\lambda^0 = 1$. Ao contrário de Birgin, Martínez e Raydan (2000), que computa um novo ponto a partir de uma busca não monótona, neste trabalho foi implementada uma busca exata para obter um novo ponto, com o comprimento do passo obtido a partir da função objetivo f_{AL} do problema (4.2). As salvaguardas do passo espectral λ^k foram $\lambda_{\min} = 10^{-30}$ e $\lambda_{\max} = 10^{30}$. O limite máximo de iterações considerado foi de $k_{\max} = 100000$. Os critérios de convergência deste algoritmo são

$$|y^T x^{k+1}| \leq 10^{-5} \text{ e } \|P(x^k - \lambda^k \nabla f_{\mathrm{LA}}) - x^k\| \leq 10^{-2};$$

- algoritmo de Fan et al. (2005), implementado para MVP e LIBSVM: O limite máximo de iterações considerado foi de $k_{\max} = 1000000$, e a tolerância para convergência foi 10^{-3}, tanto para MVP quanto para LIBSVM.