Daniel Guerreiro e Silva

Uso de Aprendizado de Máquina para estimar esforço de execução de testes funcionais

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Engenharia de Computação.

Orientador: Prof. Dr. Mario Jino

Banca Examinadora:
Prof. Dr. Romis Ribeiro de Faissol Attux - DCA/FEEC/UNICAMP
Profa. Dra. Silvia Regina Vergilio - DINF/UFPR

Campinas, SP
2009

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

Si38u

Silva, Daniel Guerreiro e
Uso de aprendizado de máquina para estimar esforço de execução de testes funcionais / Daniel Guerreiro e Silva. --Campinas, SP: [s.n.], 2009.

Orientador: Mario Jino.
Dissertação de Mestrado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Software. 2. Variaveis (Matemática). 3. Aprendizado de computador. 4. Variáveis latentes. I. Jino, Mario. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês: Using machine learning to estimate execution effort of functional tests
Palavras-chave em Inglês: software, Variables (Mathematics), Machine learning, Latent variables
Área de concentração: Engenharia de Computação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: Romis Ribeiro de Faissol Attux, Silvia Regina Vergilio
Data da defesa: 30/11/2009
Programa de Pós Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Daniel Guerreiro e Silva

Data da Defesa: 30 de novembro de 2009

Título da Tese: "Uso de Aprendizado de Máquina para Estimar Esforço de Execução de Testes Funcionais"

Prof. Dr. Mario Jino (Presidente): _____

Profa. Dra. Silvia Regina Vergilio: _____

Prof. Dr. Romis Ribeiro de Faissol Attux: _____

Resumo

O planejamento das atividades de teste tem papel essencial para qualquer equipe independente de testes que realize testes de diferentes sistemas de *software*, desenvolvidos por diferentes equipes de desenvolvimento. Dado que o esforço empreendido no processo de testes pode chegar até a metade do esforço total de desenvolvimento de um sistema, estimar adequadamente o esforço de testes pode evitar custos desnecessários e contribuir para a boa qualidade dos produtos.

Para superar este desafio, ferramentas de aprendizado de máquina têm sido usadas em pesquisa para estimar esforço e para solucionar outros problemas de engenharia de *software*, principalmente porque eles constituem uma classe de problemas complexos com muitas limitações à sua solução por abordagens matemáticas clássicas.

Este trabalho estuda a aplicação das ferramentas de aprendizado de máquina — redes neurais artificiais e máquinas de vetor de suporte — e de ferramentas de seleção de variáveis na solução do problema de estimar esforço de execução de testes funcionais. Um estudo do processo de execução de testes é desenvolvido e são conduzidos experimentos em duas bases de dados reais com o objetivo de propor uma metodologia adequada para abordar sistematicamente o problema, tanto em termos de qualidade de resultados como em praticidade de uso.

As principais contribuições deste trabalho são: a proposta de realizar a seleção de variáveis para a síntese da base de dados; a adoção de um modelo de rede neural treinada por uma função custo assimétrica; e um estudo comparativo de desempenho dos modelos preditores.

Palavras-chave: teste de *software*, esforço, estimativa, seleção de variáveis, aprendizado de máquina, redes neurais artificiais, máquina de vetor de suporte, teste funcional.

Abstract

Planning and scheduling of testing activities play a key role for any independent test team that performs tests for different software systems, produced by different development teams. Since the effort that is applied in the test process can amount to up to half of the total effort of software development, adequate estimation of test effort can prevent unnecessary costs and improve the quality of delivered products.

To overcome this challenge, machine learning tools have been used in research to estimate effort and to solve other software engineering problems, mainly because they constitute a class of complex problems with many limitations to their solution by classical mathematical approaches.

This work studies the application of machine learning tools — artificial neural networks and support vector machines — and variable selection tools to solve the problem of estimating the execution effort of functional tests. An analysis of the test execution process is done and experiments are performed with two real databases aimed at proposing a suitable methodology to systematically tackle this problem, considering both the quality of results and ease of application.

The main contributions of this work are: the proposal of applying variable selection for database synthesis; the adoption of an artificial neural network trained with an asymmetric cost function; and a comparative study of performance with the predictive models.

Keywords: software testing, effort, estimate, variable selection, machine learning, artificial neural networks, support vector machines, functional testing.

"Sentir é criar. Sentir é pensar sem ideias,
e por isso sentir é compreender,
visto que o universo não tem ideias."
Fernando Pessoa

Aos meus pais:
Alberto e Marta.
E ao meu irmão:
Rodrigo.

Agradecimentos

Este período de mestrado marcou-me por grandes mudanças profissionais e pessoais, por isso gostaria de agradecer a uma série de pessoas que me ajudaram e me transformaram.

Agradeço ao meu orientador, professor Mario Jino, pela simpática e respeitosa orientação, e pelos preciosos conselhos e ideias.

Aos professores Fernando Von Zuben e Romis Attux, pelas valiosas orientações e críticas para a elaboração dos experimentos.

Aos colegas do projeto HARPIA, pelo ótimo convívio e aprendizado durante os dois anos de projeto. Em especial a Bruno Teixeira de Abreu, pelos conselhos e pela ajuda imprescindível na redação desta dissertação e dos artigos oriundos deste mestrado.

Aos amigos da *Unicamp Swimming Society Reloaded*, pela maravilhosa convivência e fiel amizade que vai muito além dos treinos de natação.

Aos grandes amigos Rubens de Figueiredo Camargo e Daniel Takata Gomes, pela grande ajuda prestada nos fundamentos de Matemática e Estatística deste trabalho, e pela inspiração e companheirismo que me oferecem na vida acadêmica.

Aos amigos que lealmente dividem comigo angústias, alegrias, tristezas, ideias, bobagens e pensamentos há vários anos, estejam longe ou perto: Raphael Garrido, Guilherme Thó, Luiz Sérgio Utino, Henrique Concon, Felipe Belussi, Marcelo Julião, Thiago Barros, Janaína Giovanetti e todos os demais Muetas; Flávia Marquitti, Lígia Marçola, Rodrigo Palma, Júlio César Berselli e Pedro Nilton Berselli.

Aos meus avós, tios e primos, pelo amor que sempre me ofereceram.

À minha namorada Rafaela, pela compreensão, cumplicidade, alegria, paciência e amor acima de tudo.

Aos meus pais, Alberto e Marta, e ao meu irmão, Rodrigo, pelo incondicional amor e apoio em todos os aspectos e de todas as formas, durante minha vida toda. Tudo é graças a vocês.

A Deus, pelas privilegiadas oportunidades dadas ao longo da minha vida e pelo amparo nos momentos mais difíceis.

Sumário

Lista de Figuras

Lista de Tabelas

Glossário

AM - Aprendizado de Máquina.

CFS - Do inglês *Correlation-based Feature Selection*.

CICM - Do inglês *Cognitive Information Complexity Measure*.

COCOMO - Do inglês *Constructive Cost Model*.

CT - Caso de Teste.

FPA - Do inglês *Function Point Analysis*.

IA - Inteligência Artificial.

KKT - Karush-Kuhn-Tucker.

LOC - Do inglês *Lines of Code*.

MARE - Do inglês *Mean Absolute Relative Error*.

MLP - Do inglês *Multi Layer Perceptron*, ou *Perceptron* de Múltiplas Camadas.

MLPP - Rede MLP ponderada.

MSE - Do inglês *Mean-Squared Error*, ou erro quadrático médio.

MVS - Máquina de Vetor de Suporte.

OCR - Do inglês *Optical Character Recognition*.

PUK - Do inglês *Pearson VII Universal Kernel*.

RBF - Do inglês *Radial Basis Function*.

RNA - Rede Neural Artificial.

RUP - Do inglês *Rational Unified Process*.

SMO - Do inglês *Sequential Minimal Optimization*.

SVR - Do inglês *Support Vector Regression*.

TI - Tecnologia da Informação.

UCP - Do inglês *Use Case Points*.

Lista de Símbolos

x_i	-	Sinal na entrada da sinapse i em um neurônio
w_{kj}	-	J-ésimo peso sináptico do neurônio k
$\varphi(.)$	-	Função de ativação de um neurônio
b_k	-	Valor de *bias* do neurônio k
v_k	-	Saída do combinador linear do neurônio k
y_k	-	Valor de saída do neurônio k
$y_k(t)$	-	Valor de saída computada pelo neurônio k, no instante t
$d_k(t)$	-	Saída esperada para o neurônio k, no instante t
N	-	Número de neurônios na camada de saída da rede
M	-	Número de pares entrada-saída apresentados no treinamento da rede
$J(t)$	-	Função de soma dos erros quadráticos
J_{av}	-	Função de erro quadrático médio
$\mathbf{x_l}$	-	L-ésimo vetor de dados de entrada
y_l	-	L-ésimo valor de saída esperado
ϵ	-	Valor de erro permitido
$f(\mathbf{x})$	-	Função de aproximação de uma máquina de vetor de suporte
$\mathbf{w}$	-	Vetor de coeficientes lineares de $f(\mathbf{x})$
b	-	Valor de *bias* de $f(\mathbf{x})$
ξ_i	-	I-ésima variável de folga
ξ_i^*	-	I-ésima variável de folga que faz par com ξ_i
C	-	Constante de compromisso da função de otimização
$\lvert\xi\rvert_\epsilon$	-	Função de perda insensível a ϵ
α_i	-	I-ésimo multiplicador de Lagrange
α_i^*	-	I-ésimo multiplicador de Lagrange que faz par com α_i
$\Phi(\mathbf{x})$	-	Mapeamento não-linear para um espaço de características F
$K(\mathbf{x}, \mathbf{x}')$	-	Núcleo de Produto Interno entre $\mathbf{x}$ e $\mathbf{x}'$
$R[f]$	-	Funcional de risco esperado
$R_{emp}[f]$	-	Funcional de risco empírico
$R_{reg}[f]$	-	Funcional de risco regularizado
λ	-	Constante de regularização
$c(\mathbf{x}, y, f(\mathbf{x}))$	-	Função de penalidade para os erros de estimativas
X	-	Conjunto das variáveis de entrada
Y	-	Conjunto das variáveis de saída

S_i	-	Conjunto das variáveis de entrada X, excluindo a variável X_i
$P(Y\|X_i, S_i)$	-	Função distribuição de probabilidade de Y, dada a ocorrência de X_i e S_i
$P(Y\|S_i)$	-	Função distribuição de probabilidade de Y, dada a ocorrência de S_i
N_S	-	Nota atribuída pela heurística do filtro CFS ao subconjunto S de variáveis
r_{XY}	-	Coeficiente de correlação linear de Pearson
$p(e, \alpha, \beta)$	-	Função de erro ponderado
$J_p(t, \alpha, \beta)$	-	Função de soma dos erros quadráticos ponderados
$J_{p_{av}}$	-	Função média da soma de erros quadráticos ponderados
δ_k	-	Função de sensibilidade da rede em cada neurônio k
$C(d_k, y_k)$	-	Função custo genérica para uma rede neural artificial
MARE	-	Média do erro relativo absoluto
PRED$(0{,}25)$	-	Percentual de estimativas dentro de 25%
PRED$(-0{,}25)$	-	Percentual de estimativas dentro de -25%
E_{rel}	-	Medida de erro relativo do esforço

Trabalhos Publicados Pelo Autor

1. Silva, D. G.; Abreu, B. T. & Jino, M.. "A Simple Approach for Estimation of Execution Effort of Functional Test Cases". *International Conference on Software Testing Verification and Validation, 2009* (ICST '09), Denver, Colorado, Estados Unidos, pg. 289-298, Abril 2009.

2. Silva, D. G.; Jino, M. & Abreu, B. T.. "Machine learning methods and asymmetric cost function to estimate execution effort of software testing". Aprovado para *International Conference on Software Testing Verification and Validation, 2010* (ICST '10), Paris, França, Abril 2010.

Capítulo 1

Introdução

1.1 Motivação

Este trabalho teve seu início no projeto HARPIA — Convênio da Receita Federal do Brasil com a Unicamp e o Instituto Tecnológico de Aeronáutica —, onde o autor trabalhou nas funções de arquiteto e analista na equipe independente de testes que foi montada para atender as necessidades do projeto.

No trabalho básico de uma equipe independente de testes, testadores executam um determinado conjunto de casos de teste[1], previamente planejado e projetado de acordo com a especificação do *software* e com os requisitos da equipe de desenvolvimento. O objetivo final é fornecer as informações de falhas à equipe de desenvolvimento.

Assim, é necessária uma gerência adequada da equipe independente de testes para que seu gerente tome decisões quanto ao término dos ciclos de testes e ao número de testadores necessários. O planejamento correto do processo de testes é uma tarefa crucial, que impacta o cronograma de desenvolvimento de todos os sistemas de *software* que estão sob os cuidados da equipe de testes.

Enquanto as atividades como planejamento, projeto e elaboração de relatórios de testes podem ser realizadas em paralelo com as atividades de desenvolvimento, a atividade de execução de testes obriga os desenvolvedores a aguardar os resultados do teste; por isso é vital haver um planejamento adequado da fase de execução dos testes para impactar o mínimo possível o desenvolvimento.

Há muitos estudos de técnicas para estimar esforço e custo em desenvolvimento de *software*, como, por exemplo, Pontos por Caso de Uso [41] e COCOMO [8]. Essas técnicas são importantes para o planejamento adequado de projetos de *software*, levando à entrega do produto dentro do prazo e com um nível aceitável de falhas de baixa severidade [37].

[1] Um caso de teste (CT) consiste da descrição do estado do *software* previamente ao teste (estado inicial), de uma sequência de entradas de teste, e de uma relação de resultados esperados para cada entrada ou conjunto de dados de entrada de teste [6].

Uma pesquisa conduzida na Austrália, com 65 empresas, mostrou que aproximadamente 68% delas possuiam equipes independentes de teste [63], o que atesta a importância crescente desta atividade dentro do processo de *software* e o aumento na demanda por ferramentas de suporte à gerência do processo de testes.

Destaca-se também, no cenário atual, a bem-sucedida empreitada que as técnicas de Inteligência Artificial (IA) têm realizado na solução de problemas de engenharia de *software*, como geração de dados para testes [65], agrupamento de componentes [17], e estimativa de esforço e custo [16, 23, 64, 98]. São meta-heurísticas que funcionam bem quando há restrições complexas, inter-relacionadas e que competem entre si [18].

Entretanto, diferentemente da pesquisa em estimativas para desenvolvimento, a pesquisa para estimativa de esforço em testes não possui muitos estudos, especialmente com o uso de técnicas de IA. Particularmente, novas perspectivas são necessárias para estimar esforço da execução de testes funcionais, preferencialmente por meio de um método simples, robusto e que não dependa de avaliações subjetivas para compor sua base de conhecimento, pois as propostas até então apresentadas possuem algumas limitações como a dependência de avaliações subjetivas [3, 98] e a ausência de validação em ambiente real [49].

1.2 Escopo

Partindo do princípio de uso de dados de um determinado problema para sintetizar um modelo que o descreva, este trabalho estuda o uso de técnicas de Aprendizado de Máquina (em inglês, *Machine Learning*) e de seleção de variáveis para estimar o esforço da execução de testes funcionais. As propostas de aprendizado de máquina consideradas neste trabalho são Redes Neurais Artificiais e Máquinas de Vetor de Suporte; para seleção de variáveis, os métodos por envoltório e filtro são usados.

O problema de estimar esforço tem natureza complexa e é limitada a possibilidade de solução por abordagens matemáticas clássicas. Neste contexto, a escolha do modelo de Redes Neurais Artificiais é adequada porque sua capacidade de adaptação, de aprendizado e de tolerância a ruídos permite que seja usado em problemas desse tipo. Quanto às Máquinas de Vetor de Suporte, elas apresentam características semelhantes às das redes neurais, mas, por outro lado, possuem uma maior automação dos parâmetros de ajuste e não sofrem do problema de aumento da complexidade em relação à dimensão dos dados; portanto podem também ter um bom desempenho na solução do problema.

1.2.1 Esforço em Testes

O processo de teste envolve diversas atividades como o planejamento de testes, o projeto de testes e a execução de testes. Além disso, há diversas técnicas de testes que podem ser aplicadas de forma separada ou em conjunto, como os testes estruturais, testes baseados em defeitos e os testes funcionais [61].

O foco desta dissertação está no problema de estimar esforço da atividade de execução de testes funcionais. Escolheu-se a atividade de execução devido às razões já citadas na Seção 1.1, e a técnica de teste funcional porque é uma abordagem de teste comum em equipes independentes de testes e cuja realização é imprescindível para avaliar a qualidade e adequação do *software* aos requisitos funcionais definidos pelo cliente. Além disso, o escopo está em equipes independentes de teste que realizam vários ciclos de testes para sistemas de *software* distintos ou não, sem uso de automação.

1.2.2 Redes Neurais Artificiais

As Redes Neurais Artificiais são uma proposta consolidada na solução dos mais diversos problemas de classificação e regressão [33]. Além disso, elas também são aplicadas, por exemplo, em problemas de memória associativa [36] e de agrupamento de dados [45].

É dado foco neste trabalho ao *Perceptron* de Múltiplas Camadas, com o treinamento por meio do algoritmo de retropropagação, tanto com a tradicional função objetivo de soma dos erros quadráticos quanto com uma função assimétrica de erro quadrático. A meta é que a rede crie assim um mapeamento não linear entre dados do processo de testes conhecidos previamente à execução e o esforço efetivamente realizado.

1.2.3 Máquinas de Vetor de Suporte

As Máquinas de Vetor de Suporte são uma proposta recente de técnica de aprendizado de máquina que vem apresentando resultados positivos para diversos problemas de classificação e regressão[2]. Desenvolvidas a partir da teoria de aprendizado estatístico, ou Teoria VAPNIK-CHERVONENKIS (VC) [89], elas têm um número reduzido de parâmetros para ajuste em comparação com redes neurais artificiais, e são robustas ao aumento da dimensão do problema.

Recentes trabalhos utilizando regressão por vetor de suporte para estimar esforço em desenvolvimento [64] e em testes de *software* [98] mostraram que o método tem bom funcionamento para problemas com poucos dados. Portanto, o objetivo é aplicar a regressão por vetor de suporte aos poucos dados disponíveis, para avaliar e comparar o desempenho com o obtido pelo uso de redes neurais

[2]Também chamada neste caso de Regressão por Vetor de Suporte.

artificiais.

1.2.4 Seleção de Variáveis

Simplicidade de uso é um dos principais requisitos para se propor um método para estimar esforço que tenha boa aceitação. As técnicas de seleção de variáveis têm o propósito de determinar o subconjunto de variáveis, entre todas que estão disponíveis para o problema, que traz o melhor desempenho para o modelo preditor[3] e que ainda reduz, se possível, o número de variáveis em uso.

Dois métodos de seleção de variáveis são bastante adotados: (1) a abordagem por filtro, que realiza a seleção de forma independente do modelo preditor que será aplicado posteriormente; e (2) a abordagem por envoltório, que realiza a seleção tendo em vista o desempenho de um modelo preditor previamente escolhido. As duas abordagens são aplicadas para avaliar os benefícios que podem trazer à solução do problema.

1.3 Objetivos

Este trabalho tem os seguintes objetivos:

1. Estudar e comparar a aplicação de redes neurais artificiais e máquinas de vetor de suporte, bem como o uso de seleção de variáveis, no âmbito do problema de estimar esforço de execução de testes, para duas bases de dados distintas.

2. Propor um método sistemático para estimar esforço da execução de testes que seja livre de avaliações subjetivas e viável para adoção por equipes de teste de *software*.

1.4 Metodologia

A metodologia adotada consiste em uma sequência de experimentos feitos sobre duas bases de dados reais que, embora não tenham garantidamente uma representatividade suficiente de todos os sistemas de *software* possíveis de serem testados, fornecem subsídios para se propor uma forma promissora para que equipes de teste de *software* consigam estimar o esforço adequadamente. Com a realização destes experimentos, tenta-se responder às seguintes questões:

1. Qual é a melhor abordagem para selecionar variáveis, nos dois estudos de caso?

2. Qual é o melhor modelo para predizer esforço, nos dois estudos de caso?

[3] Um modelo preditor consiste em um modelo capaz de realizar previsões através de dados passados.

3. O melhor modelo nos dois estudos de caso apresenta um desempenho bom para os padrões da literatura?

Estas questões estão vinculadas diretamente aos objetivos do trabalho, de forma que as respostas obtidas servem para comparar as técnicas consideradas e para estabelecer um método de estimar esforço de execução de testes.

É importante frisar que o princípio de abordagem do problema, via técnicas de aprendizado de máquina, implica que o foco está somente nos dados históricos disponíveis para treinamento dos modelos. Embora seja feito um estudo sobre o processo de testes para determinação das melhores variáveis para compor a base, não há nenhum conhecimento prévio inserido na lógica de cada modelo preditor utilizado.

1.5 Contribuições da dissertação

As contribuições deste trabalho são as seguintes:

1. Aplicação de redes neurais artificiais para estimar esforço em testes, considerando tanto a configuração tradicional de uma rede neural artificial com função custo simétrica, como a proposta com uma função custo assimétrica.

2. Aplicação de técnicas de seleção de variáveis ao problema de estimar esforço.

3. Comparação das técnicas de Redes Neurais Artificiais e Máquinas de Vetor de Suporte em combinação com métodos de seleção de variáveis para modelar o problema em questão.

4. Estudo do problema de se estimar esforço em testes pelo uso exclusivo de variáveis quantitativas e isentas de avaliações subjetivas.

1.6 Organização da dissertação

O Capítulo 2 apresenta conceitos básicos de aprendizado de máquina e dá ênfase a duas técnicas: redes neurais artificiais e máquinas de vetor de suporte. O Capítulo 3 introduz o estudo de seleção de variáveis, descrevendo o propósito e características de duas técnicas: filtro e envoltório.

O Capítulo 4 detalha o problema de estimar esforço em teste de *software*, enfatizando os trabalhos já realizados sobre o assunto, a análise crítica das atividades básicas do processo de teste e os fatores que podem ter relação com o esforço. No Capítulo 5 descrevem-se os experimentos realizados com as técnicas de aprendizado de máquina e a seleção de variáveis bem como os resultados obtidos.

O Capítulo 6 apresenta as conclusões do trabalho, a avaliação dos resultados quanto aos objetivos, e trabalhos futuros a serem considerados.

Capítulo 2

Aprendizado de máquina: Redes Neurais Artificiais e Máquinas de Vetor de Suporte

Este capítulo apresenta conceitos introdutórios sobre Redes Neurais Artificiais (RNAs) e Máquinas de Vetor de Suporte (MVS). Quanto à primeira técnica, é apresentada a sua definição, características, propriedades e tipos de aplicação. Aborda-se também o modelo *perceptron* de múltiplas camadas, que é aplicado ao problema de estimar esforço. Para a segunda técnica, é descrito seu uso particular para solução de problemas de regressão (Regressão por Vetor de Suporte). São apresentados o histórico de pesquisas na área, o princípio de funcionamento, algoritmos de treinamento, aplicações, e exemplos de funções núcleo utilizadas. As principais referências para este capítulo são os trabalhos de HAYKIN [33] e SMOLA & SCHÖLKOPF [78].

2.1 Aprendizado de Máquina

Existem técnicas ou algoritmos computacionais que possuem a capacidade de melhorar o seu próprio desempenho, com base em algum critério pré-estabelecido, a partir de experiências às quais são submetidos. Define-se assim que eles possuem a capacidade de *aprender*, e por isso se encaixam no contexto da teoria de "Aprendizado de Máquina" (em inglês, *Machine Learning*) [58].

A área de Aprendizado de Máquina (AM) é bastante multidisciplinar, com resultados vindos das áreas de Inteligência Artificial, Estatística, Probabilidade, Teoria de Complexidade Computacional, Teoria de Controle, Teoria da Informação, Filosofia, Psicologia, Sociologia, Neurociência, Biologia, entre outras. Neste trabalho consideram-se duas técnicas de aprendizado de máquina: Redes Neurais Artificiais, que têm origem na Neurociência; e Máquinas de Vetor de Suporte, que foram criadas a partir da Teoria de Aprendizado Estatístico.

Há três fatores que devem ser definidos para se tratar um problema por meio de uma técnica

de aprendizado de máquina: (1) as tarefas que devem ser realizadas; (2) a medida de desempenho na realização das tarefas, que se buscará otimizar; e (3) a fonte de experiência para o aprendizado. Por exemplo, em um problema de reconhecimento de palavras manuscritas, estes três fatores seriam definidos assim:

- *Tarefa*: reconhecer e classificar as palavras manuscritas segundo um conjunto de imagens.
- *Medida de desempenho*: percentual de palavras classificadas corretamente.
- *Fonte de experiência*: banco de dados com palavras manuscritas já classificadas.

Tanto a tarefa a ser realizada quanto a medida de desempenho são dependentes e muitas vezes específicas do problema a ser tratado. Embora a experiência de aprendizado também seja dependente do problema, ela pode ser classificada segundo diferentes formas, que são chamadas de "Paradigmas de Aprendizagem". São três os principais paradigmas existentes: aprendizado supervisionado, aprendizado por reforço e aprendizado não-supervisionado.

2.1.1 Aprendizado supervisionado

O aprendizado supervisionado, como o próprio nome já diz, consiste no aprendizado em que há a presença de um "supervisor" que controla o processo. Ou seja, o modelo ou máquina de aprendizado é treinado tendo como base um conjunto de pares estímulo-resposta (ou entrada-saída), de forma que, para cada estímulo, há a informação de como deveria ser seu comportamento. Dado o estímulo, o supervisor pode realizar o ajuste dos parâmetros livres da rede com base na medida de erro entre a saída computada pela rede e a saída esperada, objetivando minimizar este erro a cada iteração.

Como exemplos de processos de aprendizado supervisionado há os algoritmos para treinamento de máquinas de vetor de suporte, e o algoritmo de retropropagação (em inglês, *backpropagation*) [93] para treinamento de redes neurais artificiais.

2.1.2 Aprendizado por reforço

No aprendizado por reforço, o processo de ajuste dos parâmetros é feito pela interação contínua com o ambiente para minimizar (ou maximizar) um determinado índice de desempenho. Assim, não há um supervisor indicando a saída esperada a cada estímulo fornecido como entrada, mas sim uma espécie de "crítico" que atribui uma nota para a resposta da máquina de aprendizado ao estímulo. O objetivo assim é alcançar o nível máximo de sucesso no seu funcionamento com base no índice estabelecido.

2.1.3 Aprendizado não-supervisionado

O aprendizado não-supervisionado não possui "supervisor" ou "crítico" para monitorar o processo de aprendizado, o que se pode considerar um processo de auto-organização. Assim, a máquina ajusta seus parâmetros por meio de regularidades estatísticas dos estímulos fornecidos, formando representações internas das entradas e criando classificações novas de maneira automática [5].

O mapa auto-organizável de KOHONEN [45] é um exemplo de máquina que utiliza um algoritmo de treinamento baseado em aprendizado não-supervisionado. O princípio básico de funcionamento deste algoritmo consiste na competição entre neurônios quando apresentados aos padrões de entrada, com somente um vencedor para cada padrão apresentado. Desta maneira, ao longo da execução do treinamento os neurônios vão se especializando e passam a identificar conjuntos de padrões similares, o que faz com que assimilem as regularidades estatísticas dos dados.

2.2 Redes Neurais Artificiais

2.2.1 Histórico

A pesquisa em RNAs foi inicialmente desenvolvida devido à motivação dos pesquisadores em compreender como funcionam o cérebro humano e sua unidade básica, o neurônio.

O trabalho pioneiro no sentido de tentar modelar o funcionamento do neurônio foi o de MCCULLOCH & PITTS [53] em 1943, que propõe um modelo formal de neurônio que segue um funcionamento binário. Os autores também demonstram que, dado um número suficiente desses neurônios artificiais e com conexões sinápticas ajustadas apropriadamente e funcionando de forma síncrona, uma rede com essas características poderia realizar, a princípio, a computação de qualquer função computável.

Quinze anos depois, ROSENBLATT propôs o *perceptron* [70], uma arquitetura de rede neural semelhante à das RNAs atuais e que tinha convergência garantida no seu treinamento para classificação de dados linearmente separáveis. Até o final da década de 60, dados os trabalhos que corroboraram a proposta do *perceptron*, acreditava-se que as redes neurais poderiam realizar qualquer tipo de computação. Entretanto, em 1969 MINSKY & PAPERT [57] demonstraram matematicamente que havia limites para o que os *perceptrons* de camada única poderiam realizar, e ainda acrescentaram não haver motivos para crer que os *perceptrons* de múltiplas camadas pudessem superar estes limites.

A publicação desse livro e a dificuldade tecnológica na época contribuíram para frear o desenvolvimento das redes neurais artificiais. A ausência de um algoritmo de treinamento eficiente limitava o uso dos *perceptrons* de múltiplas camadas, o que foi somente solucionado com a proposta de RUMELHART *et al.* [71], em 1986, de uso do algoritmo de retropropagação no treinamento de *perceptrons*

de múltiplas camadas.

Antes disso, em 1982, as possibilidades de aplicações das RNAs se ampliaram com os trabalhos de KOHONEN sobre mapas auto-organizáveis [45] e com o trabalho de HOPFIELD, que criou uma rede neural recorrente para uso como memória associativa [36]. Assim, a década de 80 foi marcada pela retomada das pesquisas em redes neurais artificiais e a sua consolidação como ferramenta de larga aplicação em problemas como: classificação, agrupamento e regressão de dados; identificação de sistemas; otimização combinatória; controle de sistemas; criação de memória associativa, entre outros.

Ao longo dos anos de pesquisa e estudos sobre RNAs, percebeu-se que a forma de computar do cérebro humano é diferente daquela realizada pelos computadores digitais convencionais. O cérebro é, de fato, um sistema de processamento de informação altamente complexo, não-linear, paralelo e conectado [33].

2.2.2 Definição

Redes Neurais Artificiais (RNAs) são sistemas de processamento de informação massivamente paralelos e distribuídos, compostos por unidades de processamento simples (neurônios), que têm a propriedade intrínseca de armazenar conhecimento por experiência e assim torná-lo disponível para uso [33].

O conhecimento de uma RNA fica armazenado nos padrões de conexão entre os neurônios e nas intensidades das ligações entre eles. A conexão é denominada sinapse, e sua intensidade é descrita por um peso sináptico. A aquisição de conhecimento ocorre por um processo de aprendizado, o que significa ajustar, via um mecanismo de apresentação de estímulos ambientais, os pesos sinápticos das conexões entre neurônios de forma que a rede se comporte de acordo com o desejado. No caso das redes mais tradicionais, os parâmetros são exclusivamente os pesos sinápticos, enquanto a arquitetura da rede já está pré-estabelecida.

O aprendizado é interpretado então como um processo de condicionamento da rede neural, capaz de reproduzir em computador associações entre estímulos e respostas, algo que o cérebro realiza de forma muito eficaz [45]. Seu objetivo final é criar um modelo implícito do fenômeno ao qual foi submetido via estímulos ambientais.

O uso de RNAs oferece diversas propriedades úteis como: não-linearidade, mapeamento de entrada-saída, adaptatividade e tolerância a falhas. Além disso, as possibilidades de arquitetura, tipo de neurônio e estratégia de aprendizado criam múltiplas abordagens de RNAs, com distintas aplicações como aproximação de funções, predição de séries temporais, classificação de padrões, memória associativa, agrupamento de dados, etc..

O neurônio é a unidade elementar de uma RNA. O diagrama de blocos da Figura 2.1 apresenta o

modelo mais comum de um neurônio, no qual são identificados quatro elementos básicos:

1. Um conjunto de sinapses, cada uma caracterizada por um peso, de forma que o sinal x_i na entrada da sinapse i conectada ao neurônio k é multiplicado pelo peso sináptico w_{ki}.

2. Um somador para as entradas ponderadas pelos respectivos pesos sinápticos, o que dá origem a uma combinação linear.

3. Uma função de ativação, que tem os objetivos de limitar a amplitude da saída do neurônio e introduzir não-linearidade no modelo. Ela pode ser de vários tipos: linear, degrau, função de base radial, sigmoidal, etc..

4. Um valor de *bias*, denominado b_k, que incrementa ou diminui o valor de entrada na função de ativação.

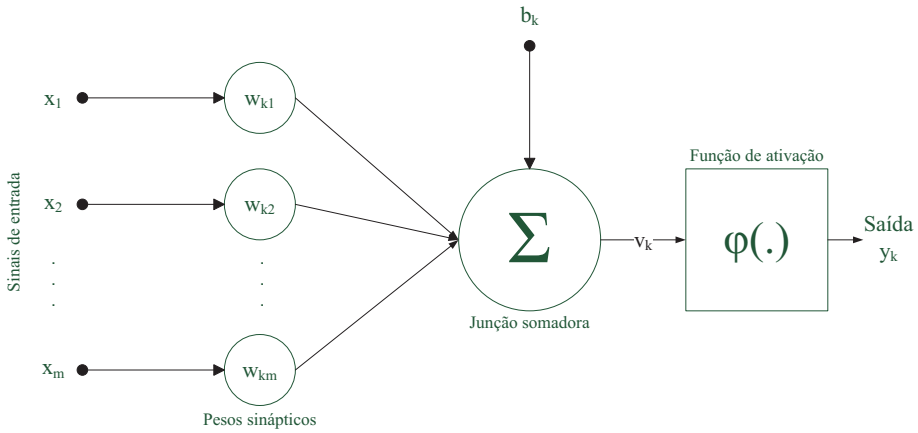

Fig. 2.1: Modelo genérico de um neurônio.

Matematicamente, o modelo de neurônio é representado pela Equação 2.1, onde $x_1, x_2, ..., x_m$ são os sinais de entrada, $w_{k1}, w_{k2}, ..., w_{km}$ são os pesos sinápticos do neurônio k, v_k é a saída da combinação linear dos sinais de entrada, b_k é o valor de *bias*, $\varphi(.)$ é a função de ativação e y_k é a saída do neurônio.

$$y_k = \varphi(v_k) = \varphi(\sum_{j=1}^{m} w_{kj}x_j + b_k) \tag{2.1}$$

2.2.3 Arquiteturas de uma RNA

Existem arquiteturas padronizadas de RNAs destinadas a resolver certas classes de problemas, sendo que cada arquitetura determina a forma como os elementos da rede (neurônios) se interligam.

De forma análoga ao que ocorre no cérebro humano, as RNAs também estão dispostas em camadas de neurônios, de forma que se pode ter uma camada de entrada, uma ou mais camadas intermediárias (ou nenhuma), e uma camada de saída. Basicamente, há então três classes de arquiteturas de RNAs: rede *feedforward* de uma única camada, rede *feedforward* de múltiplas camadas e rede recorrente.

Rede *feedforward* de uma única camada

Esta é a arquitetura mais simples de uma rede neural: consiste de uma camada de entrada e de uma camada de saída. A camada de entrada consiste de nós receptores que simplesmente replicam os dados de entrada para a camada de saída, a qual é composta pelos neurônios que realizarão a computação. Os dados são projetados somente da camada de entrada para a de saída, o que justifica o nome *feedforward*. Embora haja duas camadas, só há efetivamente computação em uma camada. A Figura 2.2 ilustra um exemplo de rede neural com esta arquitetura.

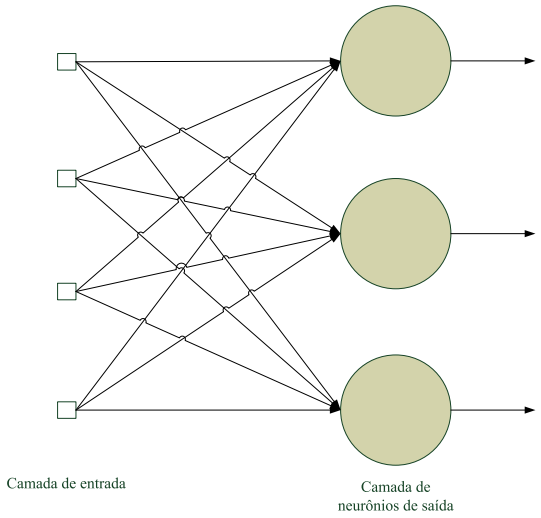

Fig. 2.2: Exemplo de uma rede neural *feedforward* de uma única camada, com 3 neurônios na camada de saída.

Rede *feedforward* de múltiplas camadas

Esta é a arquitetura de rede neural em que, além da camada de entrada dos dados e da camada de neurônios de saída, há uma ou mais camadas intermediárias (ou ocultas) de neurônios. Assim, a saída de uma camada intermediária é utilizada como entrada para a seguinte, conforme mostra o exemplo da Figura 2.3.

A adição das camadas ocultas em uma arquitetura de rede neural é útil porque aumenta a capacidade da rede de extrair informações estatísticas de maior ordem a partir dos dados [33] e, assim,

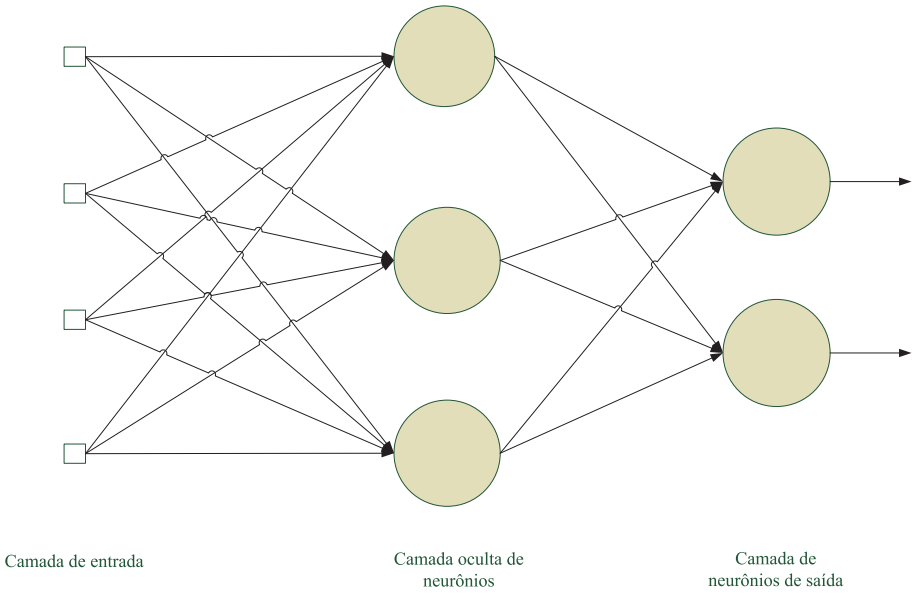

Fig. 2.3: Exemplo de uma rede neural *feedforward* de múltiplas camadas, com 3 neurônios na camada oculta.

aumenta seu poder de processamento.

A redes do tipo *Perceptron* de Múltiplas Camadas (em inglês, *Multi Layer Perceptron*, MLP) [71] são o exemplo mais conhecido de RNAs com a arquitetura de múltiplas camadas.

Rede recorrente

Uma RNA recorrente distingue-se das demais arquiteturas de rede simplesmente pelo fato de que ela possui ao menos um laço de realimentação. Por exemplo, pode-se ter uma rede neural com duas camadas de neurônios, sendo que o único neurônio da camada de saída tem sua saída ligada às entradas de todos os neurônios da camada anterior, como ilustra a Figura 2.4.

A existência de laços de realimentação interfere consideravelmente na aprendizagem e no desempenho da rede. Pressupondo que a rede seja composta de neurônios não-lineares, isto implica que o funcionamento dela é descrito por um comportamento dinâmico não-linear.

A redes de HOPFIELD [36], utilizadas principalmente como memória associativa e para solução de problemas de otimização combinatória, são um exemplo clássico de redes recorrentes.

2.2.4 O *Perceptron* de Múltiplas Camadas

A rede neural MLP pode ser considerada uma generalização do *perceptron* proposto por ROSENBLATT [70]. Ela é uma rede de múltiplas camadas que teve sua disseminação após RUMELHART &

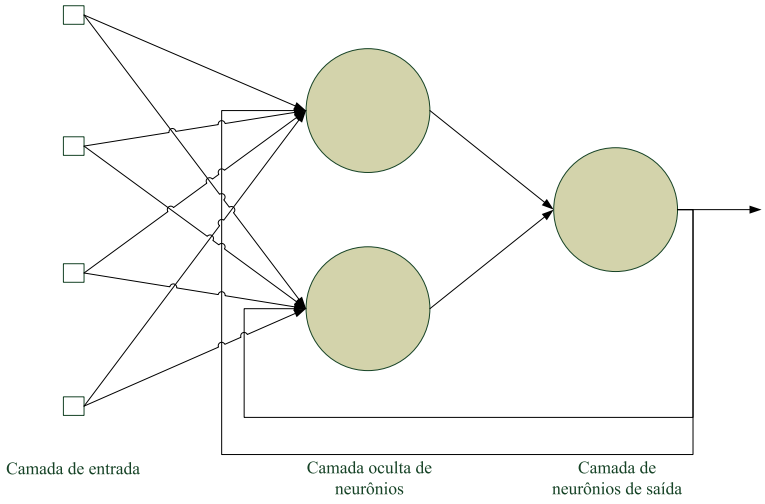

Fig. 2.4: Exemplo de uma rede neural recorrente.

MCCLELLAND [71] apresentarem a proposta de cálculo do gradiente para o treinamento pelo algoritmo de retropropagação [93].

Do ponto de vista arquitetônico, a rede MLP apresenta uma camada de entrada, composta somente por neurônios sensoriais (que propagam os sinais de entrada); uma ou mais camadas ocultas, compostas por neurônios com função de ativação do tipo sigmoidal (ex.: função logística ou tangente hiperbólica); e uma camada de saída composta por neurônios que realizam a combinação linear dos sinais das camadas intermediárias, propagando o resultado para uma função de ativação linear ou não-linear. Outra característica importante é que a rede possui alto grau de conectividade entre seus neurônios.

A Figura 2.5 apresenta um exemplo de rede MLP com duas camadas ocultas. A rede possui n valores de entrada $(x_1, x_2, ..., x_n)$ e produz como resposta m valores de saída $(y_1, y_2, ..., y_m)$, sendo que os valores +1 como entrada em cada camada de neurônios estão ligados a pesos sinápticos que fazem o papel de *bias*.

A rede MLP é uma ferramenta adequada para realizar mapeamentos não-lineares de entrada com a saída, de natureza geral. Esta propriedade pode ser vista como aquela que a torna aplicável a uma vasta gama de problemas, tais como problemas de aproximação de funções, classificação de padrões, e predição de séries temporais.

Neste contexto, em 1989, GEORGE CYBENKO demonstrou rigorosamente que uma rede MLP com uma única camada intermediária é suficiente para aproximar uniformemente qualquer função contínua que esteja definida num hipercubo unitário [20]. Vale ressaltar que este é um teorema de existência, o qual afirma, em outras palavras, que com um conjunto de dados de treinamento suficientemente significativo existe um *perceptron* de múltiplas camadas, com uma única camada interme-

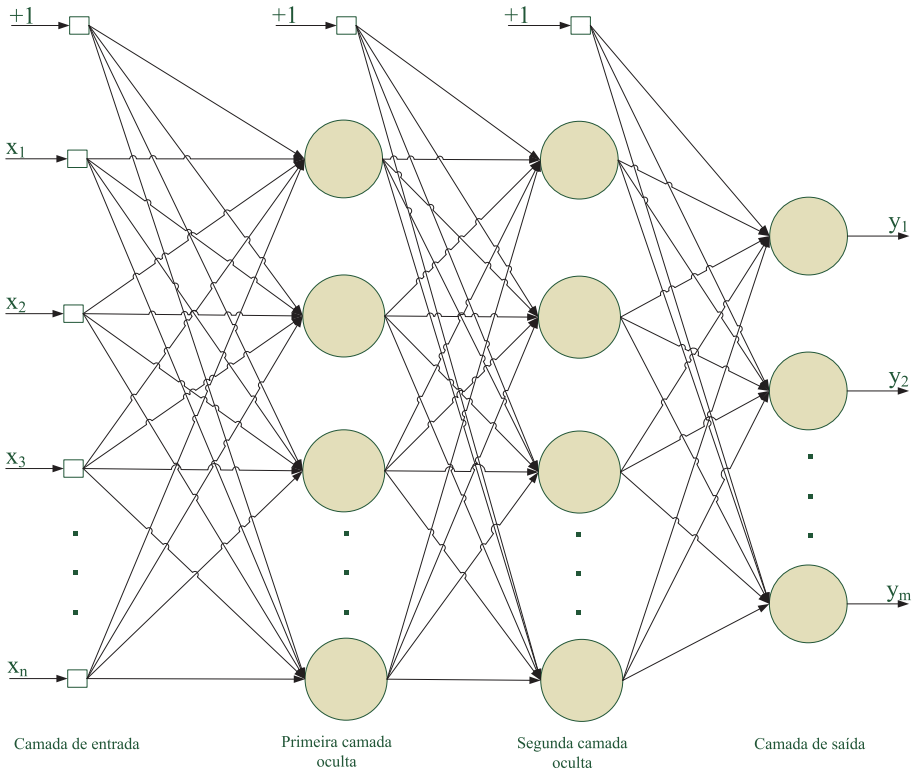

Fig. 2.5: Exemplo da arquitetura de uma rede MLP com duas camadas ocultas.

diária, que aproxima a função com um erro arbitrário. Assim, o teorema não diz como construir esta configuração de forma que também se tente maximizar a capacidade de generalização [7].

Na prática, há um número finito de amostras (o teorema supõe infinitas amostras) para realizar o treinamento da rede. Isto implica que existem infinitas configurações de redes neurais que aproximam igualmente os estímulos contidos no conjunto de treinamento. Escolher qualquer uma dessas configurações com um nível de erro aceitável não é suficiente porque também é necessário avaliar a capacidade que a rede tem de estimar a partir de dados que não foram apresentados durante o treinamento.

Tal propriedade é chamada de capacidade de generalização, e uma maneira prática e amplamente adotada para obter uma estimativa dessa propriedade é por meio da técnica de validação cruzada, que abrange essencialmente os seguintes passos:

1. Dividir os dados em dois conjuntos: (1) o conjunto de treinamento e (2) o conjunto de validação;

2. Efetuar o treinamento do modelo utilizando os dados pertencentes ao conjunto de treinamento;

3. Utilizar o modelo treinado para fazer estimativas com os dados pertencentes ao conjunto de validação;

4. Calcular o erro das estimativas feitas para o conjunto de validação, o que fornece um indicador da capacidade de generalização do modelo.

Sendo assim, a configuração mais adequada é aquela cuja aproximação tem o menor erro nas estimativas para o conjunto de validação.

Treinamento de uma rede MLP

O treinamento de uma rede MLP pode ser tratado como um problema de otimização não-linear irrestrita, no qual se realiza uma busca no espaço dos parâmetros (pesos) pelo mínimo global da função de erro quadrático médio entre a saída da rede e a saída desejada. Para encontrar uma solução factível deste problema, é necessário definir os seguintes itens:

- *Dados de treinamento*: consiste no conjunto de N amostras de pares entrada-saída, podendo tanto a entrada quanto a saída ter qualquer dimensão.

- *Função custo*: a função custo para o problema do aprendizado de uma rede MLP tradicionalmente é o erro quadrático médio (em inglês, *Mean-Squared Error*, MSE). A Equação 2.2 define o erro quadrático instantâneo $J(t)$, onde N é o número de neurônios na camada de saída da rede, $d_k(t)$ é a saída esperada para o neurônio k, dada a entrada fornecida no instante t, e $y_k(t)$ é a saída computada pelo neurônio k, dada a entrada fornecida no instante t. Já a Equação 2.3 apresenta o cálculo do erro quadrático médio, somando $J(t)$ para todos os pares entrada-saída apresentados e fazendo a divisão pelo tamanho M deste conjunto.

$$J(t) = \frac{1}{2}\sum_{k=1}^{N}[d_k(t) - y_k(t)]^2 \tag{2.2}$$

$$J_{av} = \frac{1}{M}\sum_{t=1}^{M} J(t) \tag{2.3}$$

- *Algoritmo de otimização*: há diversos métodos iterativos para otimização não-linear irrestrita, sendo o mais conhecido o método do gradiente, de primeira ordem. Há também os métodos de segunda ordem, atualmente considerados os mais eficientes no treinamento de RNAs, como o de Levenberg-Marquardt [30] e o do Gradiente Conjugado [27]. Seja o método de primeira ou segunda ordem, é necessário o uso do algoritmo de retropropagação para obtenção do gradiente.

- *Método de ajuste dos pesos*: os pesos da rede podem ser ajustados à medida que cada par entrada-saída é apresentado ou os ajustes referentes a cada amostra vão sendo acumulados até

que todas tenham sido apresentadas. A primeira forma é chamada de atualização local, padrão-a-padrão ou *online*; a segunda forma é chamada de atualização em lote, *offline* ou em batelada.

- *Critério de parada*: há diversos critérios de parada para o treinamento como: limite máximo de apresentações do conjunto de dados (épocas), valor mínimo da função custo e critérios de parada com base no erro de generalização.

O algoritmo de retropropagação é usado comumente no treinamento de redes MLP [93], e foi proposto para resolver o problema de cálculo do gradiente da função custo em relação aos pesos das camadas ocultas da rede. Ele pode ser dividido em duas etapas:

1. *Propagação positiva do sinal de entrada*: neste passo os pesos da rede estão fixos e o sinal de entrada é propagado pela rede, de forma que seja computado um valor de saída.

2. *Retropropagação do erro*: neste passo calcula-se o erro do valor de saída em relação ao desejado e este erro é propagado em sentido oposto ao sinal de entrada, o que dá origem ao conceito de retropropagação do erro.

Como se sabe o erro somente na camada de saída da rede, o algoritmo utiliza a regra da cadeia para obter as relações dos valores de gradiente de uma camada em função da camada seguinte. Ao executá-lo, inicia-se calculando o valor de gradiente da camada de saída, que será usado para cálculo do gradiente na penúltima camada e, assim, sucessivamente até chegar aos pesos da primeira camada. O uso do método do gradiente com o algoritmo de retropropagação é semelhante ao algoritmo *Least Mean Squares* (LMS) [94], uma vez que ambos utilizam o mesmo princípio de ajustar os pesos no sentido contrário ao vetor gradiente do erro, o que faz tender à minimização do erro entre a saída da rede e a saída desejada.

Uma rede neural deve ser treinada de forma a aprender adequadamente a partir dos dados de treinamento e a generalizar apropriadamente no futuro, ou seja, responder a novos estímulos com uma margem de erro aceitável. Deve-se evitar a situação de sobreajuste: a rede neural chega ao fim do treinamento com um baixo erro sobre os dados apresentados, mas, ao ser testada em outros dados, não consegue atingir um desempenho satisfatório porque foi treinada excessivamente.

Uma técnica para tentar garantir esta capacidade de generalização da rede e evitar o sobreajuste é a validação cruzada [82, 83], em que os dados disponíveis são divididos ao menos num conjunto de treinamento e num conjunto de validação, sendo o último usado para medir a capacidade de generalização da rede ao longo do processo. É possível também separar os dados em uma terceira parte denominada conjunto de testes, que pode ser utilizada para medir o desempenho da rede depois que ela foi treinada.

A capacidade de generalização é medida simplesmente pelo cálculo da função de erro quadrático médio sobre as amostras do conjunto de validação, na rede treinada. Desta forma, pode-se, por exemplo, determinar como critério de parada do treinamento o momento em que o erro sobre o conjunto de validação aumentar k vezes consecutivas e utilizar o ajuste da rede na época[1] anterior ao início deste processo de aumento. Outra possibilidade é realizar o treinamento por um número fixo de N épocas, armazenando os pesos da rede em cada iteração e, após o término, considerar os vetores de pesos da época em que o erro de validação foi mínimo.

2.2.5 Aplicações de RNAs em estimativa de esforço

As RNAs possuem um vasto campo de aplicação em problemas de diversas naturezas, como:

- Problemas de classificação de dados: reconhecimento de faces, reconhecimento óptico de caracteres.
- Problemas de regressão de dados: aproximação de funções [91], identificação de sistemas, predição de séries temporais [28], controle de processos.
- Problemas de agrupamento de dados: mineração de dados e textos, síntese de informação.
- Problemas de otimização combinatória: problemas de caixeiro viajante, problemas de roteamento de veículos.

O uso de RNAs também tem sido frequente nas pesquisas para estimar esforço de desenvolvimento de *software*, principalmente porque o problema de estimar esforço é de natureza complexa e é limitada a possibilidade de solução por abordagens matemáticas clássicas. Neste contexto se encaixa a RNA porque sua capacidade de adaptação, de aprendizado e de tolerância a ruídos permite que seja usada em problemas deste tipo. Trabalhos como o de FINNIE *et al.* [26] e DOLADO & FERNÁNDEZ [23] comparam RNAs com outras técnicas como Regressão Linear, Programação Genética e Raciocínio Baseado em Casos; já o trabalho de SHUKLA [76] propõe o uso de RNA treinada por algoritmo genético sobre uma base de dados com 39 atributos, para estimar o esforço de desenvolvimento.

Vale aqui ressaltar que as RNAs podem ser enquadradas na área de Computação Natural, onde se encontra uma coleção de teorias, conceitos e algoritmos que compartilham do mesmo nicho de aplicação: problemas de natureza complexa, difíceis de serem modelados matematicamente e/ou terem todas suas variáveis descobertas, ou que têm soluções por métodos tradicionais inviáveis de serem realizadas em tempo razoável [14].

[1] Época é o nome dado à apresentação do conjunto completo de dados de treinamento a uma RNA. Usualmente o treinamento consiste de várias épocas.

2.3 Máquinas de Vetor de Suporte

2.3.1 Histórico

Durante a década de 60, VAPNIK *et al.* desenvolveram o algoritmo *Generalized Portrait* [87, 86]. Esta técnica originou, aproximadamente 30 anos depois, a Máquina de Vetor de Suporte (MVS), que pode ser considerada uma generalização não-linear do método antecessor.

Foi a partir de 1992 que a MVS teve sua forma atual desenvolvida por VAPNIK e seus colegas dos laboratórios *Bell*, com uma forte orientação para a solução de problemas da indústria. Ela se baseia na teoria de aprendizado estatístico ou *Teoria Vapnik-Chervonenkis* (VC) [89], que foi elaborada entre as décadas de setenta e noventa por VAPNIK & CHERVONENKIS e, de maneira resumida, define propriedades para que máquinas de aprendizado consigam generalizar o comportamento para dados não conhecidos. Seu princípio fundamental, a ser explicado nas próximas seções, é chamado de minimização do risco estrutural.

A aplicação de uma MVS concentrou-se, a princípio, em problemas de classificação de padrões como OCR (em inglês, *Optical Character Recognition*). Porém, rapidamente apresentou resultados competitivos com as técnicas até então vigentes, tanto para OCR quanto para outros problemas de reconhecimento de padrões [73], além de problemas de regressão de dados [52].

Desta forma, Máquinas de Vetor de Suporte tornaram-se objeto de intensa e ativa pesquisa. Há trabalhos introdutórios como o de SMOLA & SCHÖLKOPF para o uso de MVS para regressão [78], bem como para o uso em problemas de classificação [11, 50]. Além disso, aplicações de MVS em problemas de engenharia de *software* já são encontradas, como nos trabalhos de OLIVEIRA [64] e ZHU *et al.* [98].

2.3.2 Princípio Básico de Funcionamento

Aqui é dado enfoque ao uso de MVS para o problema de regressão de dados, que pode ser chamada também de Regressão por Vetor de Suporte (em inglês, *Support Vector Regression*, SVR), uma vez que a técnica é utilizada desta forma na metodologia proposta nesta dissertação.

Seja o conjunto de dados para treinamento $\{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), ..., (\mathbf{x_l}, y_l)\} \subset X \times \mathbb{R}$, onde X consiste no espaço dos dados de entrada (por exemplo $X = \mathbb{R}^d$). Considerando o método ϵ-SV [89], o objetivo então é encontrar um mapeamento $f(\mathbf{x})$ que tenha um erro de valor máximo igual a ϵ, para cada valor y_i esperado. Ao mesmo tempo, a função deve ser a mais plana possível.

Por razões didáticas, considere inicialmente o caso das funções lineares, que podem ser descritas da seguinte forma:

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b \quad \text{com} \quad \mathbf{w} \in X, \quad b \in \mathbb{R} \tag{2.4}$$

onde $\langle ., . \rangle$ indica o produto interno em X. A ideia equivalente a tornar uma função qualquer a mais plana possível seria, para o caso especial da Equação 2.4, minimizar o valor de $\mathbf{w}$, o que pode ser alcançado pela minimização do quadrado de sua norma, $||\mathbf{w}||^2$. Assim, chega-se a um problema de otimização convexa descrito como:

$$\begin{aligned} \text{minimizar} \quad & \frac{1}{2}||\mathbf{w}||^2 \\ \text{sujeito a} \quad & \begin{cases} y_i - \langle \mathbf{w}, \mathbf{x_i} \rangle - b \leq \epsilon \\ \langle \mathbf{w}, \mathbf{x_i} \rangle + b - y_i \leq \epsilon \end{cases} \end{aligned} \tag{2.5}$$

Há um porém nesta modelagem: o problema descrito na Equação 2.5 supõe que existe uma função que aproxima todos os pares entrada-saída com um erro máximo ϵ, e assim ele é factível; mas nem sempre existe uma função que atende às restrições do problema. Para contornar tal situação, são adicionados dois conjuntos de variáveis de folga não-negativas $\{\xi_i\}_{i=1}^{l}$ e $\{\xi_i^*\}_{i=1}^{l}$ para compreender os dados para os quais a função não consegue aproximar com erro máximo ϵ. Tem-se então o problema de otimização definitivo para o caso linear:

$$\begin{aligned} \text{minimizar} \quad & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \\ \text{sujeito a} \quad & \begin{cases} y_i - \langle \mathbf{w}, \mathbf{x_i} \rangle - b \leq \epsilon + \xi_i \\ \langle \mathbf{w}, \mathbf{x_i} \rangle + b - y_i \leq \epsilon + \xi_i^* \\ \xi_i, \quad \xi_i^* \geq 0 \end{cases} \end{aligned} \tag{2.6}$$

A constante $C > 0$ estabelece um compromisso entre o grau de achatamento da função e o grau de tolerância aos erros maiores que ϵ. Vale ressaltar também que as variáveis de folga ξ_i e ξ_i^* descrevem a função de perda insensível a ϵ, definida por:

$$|\xi|_\epsilon := \begin{cases} 0 & \text{se} \quad |\xi| \leq \epsilon \\ |\xi| - \epsilon & \text{caso contrário} \end{cases} \tag{2.7}$$

A Equação 2.6 é um problema de otimização com função objetivo convexa e restrições lineares, e pode ser chamado de *problema primordial*. Este tipo de problema pode ser resolvido pela técnica dos multiplicadores de Lagrange, no qual primeiro se constrói a função Lagrangiana, dependente de $\mathbf{w}$, b e dos multiplicadores, para ter como resultado do problema de otimização restrito o ponto

de sela da função. Para maiores detalhes desse procedimento, consulte os trabalhos de SMOLA & SCHÖLKOPF [78] e de HAYKIN [33].

Ao realizar o cálculo das derivadas em relação às variáveis da função Lagrangiana, igualando-as à zero, e fazendo a substituição dos resultados nas equações do problema primordial, chega-se ao *problema dual* de otimização:

$$\begin{aligned} \text{maximizar} \quad & \begin{cases} -\frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle \mathbf{x_i}, \mathbf{x_j}\rangle \\ -\epsilon\sum_{i=1}^{l}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{l} y_i(\alpha_i - \alpha_i^*) \end{cases} \\ \text{sujeito a} \quad & \sum_{i=1}^{l}(\alpha_i - \alpha_i^*) = 0 \quad \text{e} \quad \alpha_i, \alpha_i^* \in [0, C] \end{aligned} \tag{2.8}$$

onde $\alpha_i, \alpha_i^*, \alpha_j, \alpha_j^*$ são os multiplicadores de Lagrange. É possível também reescrever $\mathbf{w}$ e $f(\mathbf{x})$, ficando:

$$\mathbf{w} = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)\mathbf{x_i} \quad \text{e} \quad f(\mathbf{x}) = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)\langle \mathbf{x_i}, \mathbf{x}\rangle + b \tag{2.9}$$

A Equação 2.9 é a conhecida *expansão por vetor de suporte*: pode-se escrever o parâmetro $\mathbf{w}$ como uma combinação linear dos dados de treinamento. Nesta combinação, os valores de $\mathbf{x_i}$ que possuem coeficientes diferentes de zero são chamados de *vetores de suporte*.

Resta agora definir como é calculado o valor de b. Utiliza-se para isso as condições de KARUSH-KUHN-TUCKER (KKT) [42, 48], que determinam que no ponto da solução do problema o produto entre as variáveis duais e as restrições deve ser igual a zero. São estabelecidos assim limitantes para o valor de b:

$$\begin{aligned} \max\{-\epsilon + y_i - \langle \mathbf{w}, \mathbf{x_i}\rangle | \alpha_i < C \quad \text{ou} \quad \alpha_i^* > 0\} \leq b \leq \\ \min\{-\epsilon + y_i - \langle \mathbf{w}, \mathbf{x_i}\rangle | \alpha_i > 0 \quad \text{ou} \quad \alpha_i^* < C\} \end{aligned} \tag{2.10}$$

Caso algum $\alpha_i^* \in (0, C)$, as desigualdades descritas na Equação 2.10 se tornam igualdades e o valor de b é estabelecido.

2.3.3 Núcleo de Produto Interno

Passa-se agora a descrever o funcionamento de uma Máquina de Vetor de Suporte não-linear. Uma forma de construção da MVS não-linear é fazer, por exemplo, um mapeamento não-linear Φ

dos dados de treinamento para um *espaço de características* F. Feito este pré-processamento, pode-se aplicar a técnica padrão apresentada na Seção 2.3.2.

Entretanto, calcular o mapeamento para as amostras de treinamento é inviável, do ponto de vista de custo computacional, para dados de maior dimensão [78]. É necessário então a definição do *Núcleo de Produto Interno*, também conhecido como função *Kernel*:

Definição 2.1 (Núcleo de Produto Interno)**.**

$$K(\mathbf{x}, \mathbf{x}') := \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$$

onde $\mathbf{x}$ *e* $\mathbf{x}'$ *são vetores do espaço de entrada* X *e* $\Phi : X \to F$ *é um mapeamento não-linear para o espaço de características* F*.*

Pode-se constatar que K recebe então dois vetores do espaço de entrada e computa o produto interno no espaço de características. Para melhorar o entendimento da definição, considere o Exemplo 2.1.

Exemplo 2.1 (Núcleo polinomial de segunda ordem)**.** *Considere o mapeamento* $\Phi : \mathbb{R}^2 \to \mathbb{R}^3$*, com*

$$\Phi(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

Tem-se que:

$$\langle \Phi(x_1, x_2), \Phi(x_1', x_2') \rangle = \langle (x_1^2, \sqrt{2}x_1x_2, x_2^2), (x_1'^2, \sqrt{2}x_1'x_2', x_2'^2) \rangle = \langle \mathbf{x}, \mathbf{x}' \rangle^2$$

$$\Rightarrow K(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle^2 \quad \textit{é uma função núcleo de produto interno.}$$

Em grande parte dos casos, o mapeamento do espaço de características assume formas muito complexas; assim, é mais simples computar o núcleo sem conhecer explicitamente o mapeamento, o que significa no Exemplo 2.1 calcular $\langle \mathbf{x}, \mathbf{x}' \rangle^2$ ao invés de $\langle \Phi(x_1, x_2), \Phi(x_1', x_2') \rangle$. Aí está a utilidade da função núcleo de produto interno: usar a simplicidade do cálculo, aliada à capacidade de representar mapeamentos não-lineares de elevada dimensão.

Uma vez que o problema de otimização da MVS, definido nas Equações 2.8 e 2.9, depende somente do produto interno entre os dados de entrada $\mathbf{x_i}$, se é escolhida uma função K que é um núcleo de produto interno, é possível fazer a reformulação do problema de otimização para o descrito a seguir:

$$\begin{aligned} \text{maximizar} \quad & \begin{cases} -\frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)K(\mathbf{x_i}, \mathbf{x_j}) \\ -\epsilon\sum_{i=1}^{l}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{l} y_i(\alpha_i - \alpha_i^*) \end{cases} \\ \text{sujeito a} \quad & \sum_{i=1}^{l}(\alpha_i - \alpha_i^*) = 0 \quad \text{e} \quad \alpha_i, \alpha_i^* \in [0, C] \end{aligned} \tag{2.11}$$

É possível também reescrever a Equação 2.9, que fica:

$$\mathbf{w} = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)\Phi(\mathbf{x_i}) \quad \text{e} \quad f(\mathbf{x}) = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)K(\mathbf{x_i}, \mathbf{x}) + b \tag{2.12}$$

Chega-se, finalmente, à formulação final do método de treinamento de uma máquina de vetor de suporte não-linear, por meio de um problema de programação quadrática.

Condições para existência do Núcleo

O Teorema de MERCER [56] estabelece as condições para que uma função seja um núcleo de produto interno.

Teorema 2.1 (Mercer)**.** *Se a condição*

$$\int_b^a \int_b^a K(\mathbf{x}, \mathbf{x}')f(\mathbf{x})f(\mathbf{x}')d\mathbf{x}d\mathbf{x}' \geq 0$$

é válida para todo $f(.)$ *para o qual*

$$\int_b^a f^2(\mathbf{x})d\mathbf{x} < \infty$$

então $K(\mathbf{x}, \mathbf{x}')$ *é um núcleo simétrico, contínuo e definido para* $\mathbf{x} \in [a, b]$ *e* $\mathbf{x}' \in [a, b]$.

É importante ressaltar que o teorema não descreve como encontrar um núcleo de produto interno, mas sim a condição necessária e suficiente para que uma função seja considerada um núcleo. A Tabela 2.1 apresenta então quatro exemplos de funções que satisfazem o Teorema 2.1 e, portanto, são núcleos de produto interno, podendo ser utilizados em máquinas de vetor de suporte não lineares.

Os núcleos do tipo polinomial e função de base radial (em inglês, *Radial Basis Function*, RBF) satisfazem o Teorema de Mercer para quaisquer valores de parâmetros, que são ajustados *a priori* pelo usuário. Entretanto, o núcleo do tipo tangente hiperbólica só satisfaz o teorema para alguns valores de parâmetros, embora seu uso venha ocorrendo com sucesso [78]. O núcleo do tipo Pearson VII é também conhecido em inglês como *Pearson VII Universal Kernel* (PUK), e foi proposto no trabalho recente de ÜSTÜN *et al.* [84] como uma alternativa genérica para os núcleos do tipo linear, polinomial e função de base radial.

Tab. 2.1: Exemplos de núcleo de produto interno.

Função	Fórmula	Observações
Polinomial	$K(\mathbf{x}, \mathbf{x}') = (\langle \mathbf{x}, \mathbf{x}' \rangle + c)^p$	$p \in \mathbb{N}, c \geq 0$
Tangente hiperbólica	$K(\mathbf{x}, \mathbf{x}') = \tanh(\vartheta + \kappa \langle \mathbf{x}, \mathbf{x}' \rangle)$	O Teorema de Mercer é satisfeito somente para determinados valores de ϑ e κ
Função de base radial	$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}\right)$	
Pearson VII	$K(\mathbf{x}, \mathbf{x}') = \left[1 + \left(\frac{2\sqrt{\|\mathbf{x}-\mathbf{x}'\|^2}\sqrt{2^{(1/\omega)}-1}}{\sigma}\right)^2\right]^{-\omega}$	

Portanto, a escolha do tipo de núcleo a ser usado é uma decisão de projeto, que depende do problema ao qual a MVS está sendo aplicada. Do ponto de vista de uso, o núcleo do tipo função de base radial é aquele que mais tem sido utilizado, com bons resultados em diversos problemas [64, 51, 59].

2.3.4 Visão Geral

Nas seções anteriores é descrito o princípio de funcionamento de uma máquina de vetor de suporte, é definido o problema de otimização restrito a ser resolvido para realizar seu treinamento, e é apresentado o uso do conceito de núcleo de produto interno com o objetivo de estender a MVS para que realize mapeamentos não-lineares de entrada-saída. Chega-se assim à visão geral de uma MVS pela Figura 2.6, que apresenta a arquitetura da máquina.

Inicialmente, faz-se o núcleo do produto interno do vetor de entrada $\mathbf{x}$, de dimensão n, com cada um dos m vetores de suporte $\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_m}$. A saída de cada operação de núcleo de produto interno é então multiplicada pelo peso $\beta_i = (\alpha_i - \alpha_i^*)$ e feita sua soma, incluindo aí o valor de b. É importante ressaltar a diferença dos termos $x_1, x_2, ..., x_n$, que são os componentes do vetor de entrada $\mathbf{x}$, para os termos $\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_m}$, que são os vetores de suporte definidos após o treinamento da MVS.

Ainda considerando a Figura 2.6, pode-se notar a semelhança da arquitetura de uma máquina de vetor de suporte com uma rede neural artificial. Se é implementada uma MVS com um núcleo do tipo tangente hiperbólica tem-se uma máquina equivalente a um *Perceptron* de múltiplas camadas (com uma camada oculta); por outro lado, se é escolhido um núcleo do tipo função de base radial, tem-se uma máquina equivalente a uma rede neural de função de base radial. Porém, há algumas importantes distinções entre as duas propostas de máquinas de aprendizado [33]:

1. A teoria para construção de uma MVS não necessita de heurísticas frequentemente usadas no projeto de RNAs:

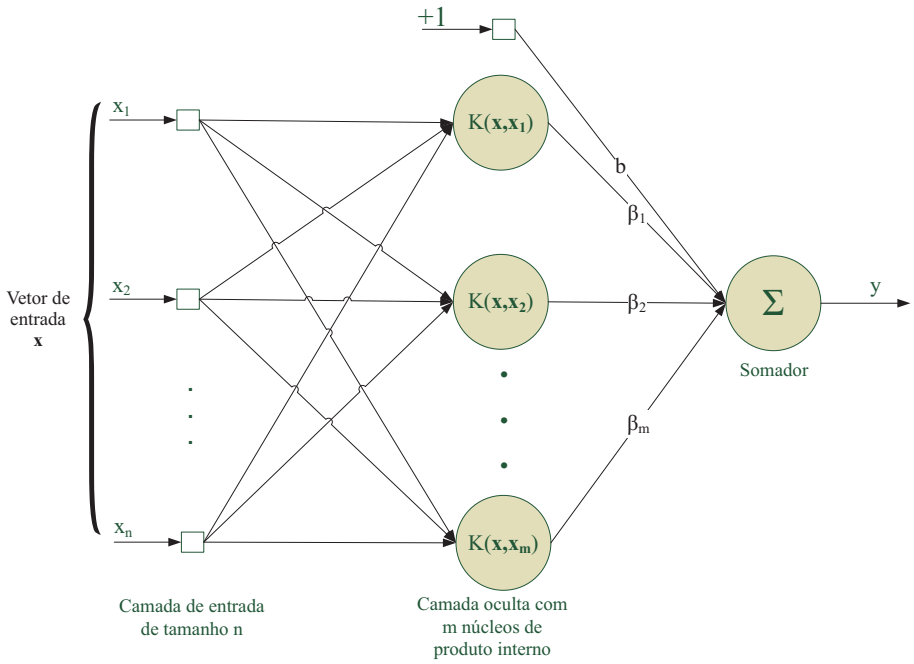

Fig. 2.6: Arquitetura da máquina de vetor de suporte.

- A MVS com núcleo função de base radial tem o número de funções de base radial e seus centros determinados automaticamente pelo número de vetores de suporte e seu valores, respectivamente.
- A MVS com núcleo tangente hiperbólica tem os elementos análogos ao número de neurônios ocultos e seus vetores de peso determinados automaticamente pelo número de vetores de suporte e seus valores, respectivamente.

2. No treinamento de uma RNA, a complexidade do modelo é controlada mantendo-se o número de neurônios ocultos pequeno. Já numa MVS a complexidade do modelo é controlada independentemente da dimensão, graças à abordagem do problema de otimização restrito utilizando sua forma dual e o conceito de núcleo de produto interno.

3. Todo problema de otimização convexa e restrita (como o treinamento de uma MVS) tem um único mínimo, portanto uma solução única. Assim, a MVS não sofre do problema de convergir para um mínimo local no treinamento, como ocorre com as RNAs.

4. Os meios para incorporar conhecimento prévio do problema em questão são mais escassos na MVS, em comparação com as RNAs.

5. Não há controle sobre o número de amostras de treinamento usadas para serem vetores de suporte na MVS, enquanto o número de neurônios pode ser controlado no projeto das RNAs.

2.3.5 Minimização do Risco

Até o momento, tratou-se de explicar, em linhas gerais, o funcionamento de uma MVS para problemas de regressão, do tipo ϵ-SV, sem preocupação quanto à ideia que originou o método. Mas, como já citado, a MVS tem como princípio fundamental o de minimização do risco estrutural, advindo da teoria de aprendizado estatístico. Esta seção, então, introduz este fundamento matemático que também se relaciona com outros métodos de estimação de funções.

Seja novamente um conjunto de dados para treinamento $X := \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), ..., (\mathbf{x_l}, y_l)\} \subset X \times \mathbb{R}$, o qual supõe-se serem dados independentes e identicamente distribuídos a partir de alguma distribuição de probabilidade $P(\mathbf{x}, y)$. O nosso objetivo então é encontrar uma função aproximadora dos dados que minimize o risco esperado [88]

$$R[f] = \int c(\mathbf{x}, y, f(\mathbf{x}))dP(\mathbf{x}, y) \tag{2.13}$$

para os dados de treinamento X, onde $c(\mathbf{x}, y, f(\mathbf{x}))$ é uma função custo que determina a penalidade dada para os erros de estimativa. Entretanto, não é possível conhecer a distribuição $P(\mathbf{x}, y)$ e por isso só se conta com os dados de treinamento para realizar a minimização de $R[f]$. Assim, recorre-se a uma aproximação da função de risco, denominada *Funcional de Risco Empírico*, dada pela equação:

$$R_{emp}[f] = \frac{1}{l}\sum_{i=1}^{l} c(\mathbf{x}, y, f(\mathbf{x})). \tag{2.14}$$

Feito isso, há o problema de busca da função f_0, pertencente a uma classe de funções H, que minimiza $R_{emp}[f]$. Mas esta abordagem é incompleta, porque pode-se correr o risco da classe H conter funções que, em situações onde os dados de treinamento são mais escassos, realizam um sobreajuste e assim perdem a capacidade de generalização. Uma forma de evitar que a função se ajuste demais aos dados é fazê-la ser a mais plana possível; como visto na Seção 2.3.2, isto pode ser feito adicionando o termo $||\mathbf{w}||^2$, que leva ao *Funcional de Risco Regularizado* [88]:

$$R_{reg}[f] = R_{emp}[f] + \frac{\lambda}{2}||\mathbf{w}||^2 \quad , \quad \lambda > 0. \tag{2.15}$$

O termo λ é chamado de *constante de regularização*. A configuração padrão da MVS para regressão, citada na Seção 2.3.2, é utilizar como função custo a função de perda insensível a ϵ:

$$c(\mathbf{x}, y, f(\mathbf{x})) = |y - f(\mathbf{x})|_\epsilon. \tag{2.16}$$

De fato, pode-se demonstrar que minimizar a Equação 2.15 com a função custo dada na Equação 2.16 é equivalente a minimizar a Equação 2.6, com $C = 1/(\lambda l)$.

O raciocínio empregado permite que sejam escolhidos outros tipos de função custo, de maneira

a tentar aquela que melhor se adequa ao problema a ser resolvido. É possível utilizar o princípio de *máxima verossimilhança* para, dadas as funções custo mais comuns, determinar o modelo de densidade de probabilidade do ruído associado aos dados [78], tendo assim mais informação para escolha do modelo de função.

Porém, deve-se tomar cuidado para que a função escolhida mantenha a convexidade do problema de otimização, uma vez que isto é a garantia da existência de um único mínimo como solução [27].

2.3.6 Algoritmos de treinamento

O treinamento de uma máquina de vetor de suporte é um problema de programação quadrática que só pode ser resolvido analiticamente para casos onde a quantidade de dados de treinamento é pequena, uma vez que a complexidade da solução analítica é de ordem N^3, onde N é o número de vetores de suporte [11].

Desta forma, existem pacotes matemáticos que resolvem por métodos numéricos problemas de programação quadrática, e que assim podem ser utilizados para o treinamento de uma MVS. Pode-se citar como exemplos os pacotes LOQO [85], MINOS [60] e OSL [95].

Porém, estes pacotes são também inadequados para os problemas de aprendizado de máquina onde o número de dados de treinamento é razoavelmente grande, porque o cálculo dos valores da função núcleo cresce de forma quadrática com o número de amostras. Torna-se, assim, inviável armazenar os valores em memória.

Para contornar estas limitações, há diversos trabalhos que tentam simplificar o processo de treinamento usando a esparsidade do problema (o fato de que a maioria dos coeficientes de Lagrange possui valor nulo) e a convexidade da função a ser otimizada. O trabalho pioneiro no sentido de decompor o problema de otimização em subproblemas menores é o de VAPNIK [88], chamado de técnica de *Chunking*. Sua proposta consiste em remover as linhas e colunas da matriz Núcleo (que armazena os valores da função K para todos os dados de treinamento) que tenham valores α_i nulos, mantendo no treinamento somente os dados que são vetores de suporte e aqueles que violam as condições KKT. O algoritmo termina quando todos os vetores de suporte da matriz satisfazem as condições KKT.

Um exemplo que leva a idéia de *Chunking* ao extremo é o algoritmo *Sequential Minimal Optimization* (SMO) [67]. O seu funcionamento consiste em selecionar subconjuntos de tamanho dois e realizar o treinamento com respeito a eles, repetindo a operação até que haja convergência para o extremo da função custo. Demonstrou-se na proposta que o algoritmo possui boas propriedades de convergência e é fácil de implementar, principalmente porque o fato de utilizar um subconjunto de tamanho dois permite que o subproblema de otimização seja resolvido de maneira analítica.

Há diversas implementações do algoritmo SMO: adaptações para regressão do tipo ϵ-SV [79], critérios distintos para seleção de subconjuntos [24] e diferentes formas para cálculo do valor da

constante b. Este trabalho utiliza a implementação em MATLAB© do pacote LIBSVM [15], que se baseia no algoritmo SMO para regressão do tipo ϵ-SV.

2.3.7 Aplicações

As máquinas de vetor de suporte têm encontrado aplicações em diversos problemas de regressão e classificação de dados, com desempenho comparável ou até superior ao de outras técnicas. Uma importante propriedade que lhe provê desempenho alto nas aplicações é que o uso de núcleos de produto interno permite a construção de espaços de características de alta dimensão, podendo ser até infinita, de forma tratável do ponto de vista computacional. Por isso uma MVS não sofre da "maldição" da dimensionalidade: o alto número de parâmetros dos dados de entrada não torna o problema intratável computacionalmente, e também não causa sobreajuste [11]. Além disso, ela requer um número relativamente baixo de parâmetros para ajuste no treinamento, em comparação com outras heurísticas.

São exemplos de aplicações de MVS em problemas de classificação: trabalhos para identificação de faces em imagens [34], reconhecimento de números a partir de dígitos manuscritos [21], identificação de genes [51, 99], etc..

Para problemas de regressão, destacam-se trabalhos em predições de séries temporais [59], reconstrução de sistemas caóticos [52], estimativa de esforço de projeto de *software* [64] e estimativa de esforço de execução de testes [98].

2.3.8 Limitações

Embora a máquina de vetor de suporte possua resultados extremamente bem-sucedidos em diversas aplicações, há limitações na sua concepção, que ainda são passíveis de melhoria:

- A forma como incorporar conhecimento prévio ao treinamento de uma MVS ainda é uma questão aberta: embora já existam trabalhos que propõem modificações [10], ainda não está claro como incorporar de maneira eficiente conhecimento prévio para problemas de regressão.

- Mesmo com diversas propostas de algoritmos de treinamento com redução de conjunto, como o SMO, o desempenho de uma MVS ainda é insatisfatório para realizar treinamento e predição em bases de dados da ordem de milhões de amostras, como no caso dos problemas de *data mining*.

- Ainda há uma dependência do utilizador da MVS para definir seus parâmetros antes do treinamento, como o tipo de função núcleo e suas constantes, a constante C, a constante ϵ no caso da regressão ϵ-SV, etc..

- De forma análoga às RNAs, a forma de aquisição e armazenamento do conhecimento não é, na maioria das vezes, interpretável. Existem outras técnicas de aprendizado de máquina, como as Árvores de Decisão, que constroem estruturas de representação do conhecimento passíveis de interpretação.

2.4 Síntese

Neste capítulo são apresentados conceitos introdutórios sobre aprendizado de máquina, como a sua definição e os três tipos de aprendizado que uma máquina pode desenvolver: (1) o aprendizado supervisionado, (2) o aprendizado por reforço, e (3) o aprendizado não-supervisionado. São detalhadas duas técnicas de aprendizado de máquina utilizadas nos experimentos deste trabalho: Redes Neurais Artificiais e Máquinas de Vetor de Suporte.

As redes neurais artificiais, em especial o *Perceptron* de Múltiplas Camadas, são bastante utilizadas na aproximação não-linear de funções para a solução de diversos problemas. O algoritmo de treinamento mais conhecido e utilizado para este tipo de rede é o algoritmo de retropropagação.

As Máquinas de Vetor de Suporte baseiam-se no princípio de minimização do risco estrutural e utilizam uma classe de funções denominadas de Núcleo de Produto Interno para também realizar aproximações não-lineares de funções. Recentemente, elas têm apresentado resultados promissores em vários estudos e são mais fáceis para ajustar os seus parâmetros, em comparação com as redes neurais artificiais.

O capítulo seguinte apresenta uma introdução à teoria de seleção de variáveis e detalha duas abordagens comumente utilizadas: o método de seleção por filtro e o método de seleção por envoltório. Estas duas técnicas são consideradas na parte experimental deste trabalho.

Capítulo 3

Seleção de variáveis

Este capítulo contém uma introdução à teoria de seleção de variáveis, contemplando os principais métodos existentes, as características de cada um, as formas de aplicação, vantagens e desvantagens. Dá-se ênfase aos dois métodos utilizados na parte experimental desta dissertação, que são a técnica por envoltório e a técnica por filtro. O conteúdo deste capítulo é baseado principalmente nos trabalhos de GUYON & ELISSEF [29], HALL [32] e VILLANUEVA [90].

3.1 Introdução

Na última década, o desenvolvimento dos algoritmos de aprendizado de máquina aliado ao considerável aumento de capacidade computacional disponível permitiu que problemas de larga escala, tanto em termos de número de amostras de dados como em número de variáveis, passassem a ser tratados. Tanto na indústria como na academia, o estudo e desenvolvimento de soluções para mineração de dados (em inglês, *data mining*) se tornaram possíveis graças a estas mudanças. Assim, é comum hoje encontrar trabalhos que exploram espaços de dados com centenas e até milhares de variáveis.

Neste contexto, novas técnicas surgiram com o objetivo de solucionar o problema da presença de variáveis irrelevantes e redundantes em conjuntos de dados, de forma a melhorar o desempenho dos algoritmos de aprendizado. Mesmo em cenários nos quais o número de dados é pequeno, qualquer ruído provocado por informação desnecessária pode comprometer os resultados; por isso o processo de seleção de variáveis também pode ser útil nestes casos.

Para ilustração, considere o exemplo de uma ferramenta de suporte à decisão para uma empresa do setor de fornecimento de energia. Ela contém uma base de dados com milhões de clientes e, para cada cliente, milhares de parâmetros referentes a informações como: endereço, média de consumo, horários de consumo maior, histórico de pagamento de faturas, etc.. O desejo da empresa é criar uma ferramenta que, a partir desta base de dados, possa indicar o risco de inadimplência de um

determinado cliente. Assim, uma prévia seleção das variáveis pertinentes a este problema implicará certamente em ganho de desempenho tanto em termos de custo computacional como em acurácia no funcionamento.

Vale ressaltar aqui que há uma distinção entre os termos "variável" e "característica": enquanto o primeiro significa as informações "cruas" que uma certa amostra dos dados possui, o segundo significa uma variável construída com base nas variáveis de entrada originais. Há muitos trabalhos que utilizam sem distinção estes dois termos; aqui considera-se somente o problema de seleção de variáveis, uma vez que nesta dissertação não há construção de características. De toda forma, com exceção das técnicas baseadas em funções núcleo de produto interno, selecionar variáveis ou características leva aos mesmos métodos.

Há vários benefícios do uso de seleção de variáveis: facilita-se o entendimento e a visualização de dados, reduz-se a necessidade de armazenamento e coleta de informação, e é reduzido o tempo de treinamento e operação das máquinas de aprendizado.

3.2 Fatores importantes

O procedimento de seleção de variáveis consiste em descobrir um subconjunto de variáveis da base de dados que fornece um melhor, possivelmente ótimo, resultado para o problema de aprendizado de máquina em questão. Seja um conjunto de dados com N variáveis de entrada, todas pertencentes ao conjunto dos números reais, e uma variável de saída também real: aplicar um método de seleção de variáveis sobre os dados resultará em um subconjunto de dados com M variáveis, tal que $M \leq N$. Desta forma, o problema de mapeamento dos dados de entrada para a saída passa de $\mathbb{R}^N \rightarrow \mathbb{R}$ para $\mathbb{R}^M \rightarrow \mathbb{R}$.

Este é um problema de busca cujo espaço de entrada consiste de todas as combinações possíveis das variáveis, em que para executar uma busca exaustiva será necessária a avaliação de $2^N - 1$ subconjuntos. Este crescimento exponencial no tamanho do problema em relação ao número de variáveis obriga que haja uso de heurísticas de busca para torná-lo tratável.

Além disso, há dois conceitos importantes para se determinar o que seria um subconjunto ótimo de variáveis, que são a *relevância* e a *redundância* de uma variável [97].

3.2.1 Relevância

A relevância de uma variável está relacionada à importância que ela tem para solução do problema. Esta medida de importância pode ser feita de diversas formas, como medidas de correlação no caso da abordagem por filtro ou medidas de decréscimo no erro quadrático médio no caso da

abordagem por envoltório (ambas as abordagens são descritas nas próximas seções).

JOHN *et al.* apresentaram em 1994 [38] definições formais para relevância, definindo três classes: relevância forte, fraca e irrelevância. Considerando X como o conjunto das variáveis de entrada, X_i como uma variável, $S_i = X - \{X_i\}$ como o conjunto de todas as variáveis de entrada, exceto a variável X_i, e Y como o conjunto das variáveis de saída, tem-se as três classes formalizadas a seguir.

Definição 3.1 (Relevância forte)**.** *Uma variável X_i é considerada fortemente relevante se e somente se*

$$P(Y|X_i, S_i) \neq P(Y|S_i).$$

Definição 3.2 (Relevância fraca)**.** *Uma variável X_i é considerada fracamente relevante se e somente se*

$$P(Y|X_i, S_i) = P(Y|S_i), \quad e$$
$$\exists S'_i \subset S_i, \quad \textit{tal que} \quad P(Y|X_i, S'_i) \neq P(Y|S'_i).$$

Definição 3.3 (Irrelevância)**.** *Uma variável X_i é irrelevante se e somente se*

$$\forall S'_i \subseteq S_i, \quad P(Y|X_i, S'_i) = P(Y|S'_i).$$

Desta forma, uma variável fortemente relevante é sempre necessária para um subconjunto ótimo de variáveis, enquanto a variável fracamente relevante pode ser importante para o subconjunto ótimo em determinadas condições. Já a variável irrelevante não é necessária ao subconjunto em nenhuma situação. Resta então utilizar o conceito de redundância para tentar determinar qual é o subconjunto ótimo.

3.2.2 Redundância

Embora exista uma definição mais precisa para redundância, que no caso usa o conceito de *"Markov Blanket"* para determinar se a variável é redundante ou não [46], considera-se neste trabalho a definição de que duas variáveis são redundantes se seus valores são completamente correlacionados. A escolha se deu pelo fato de que esta é a definição mais comum nos trabalhos sobre seleção de variáveis e também a mais simples de se compreender e utilizar.

Portanto, para o subconjunto ótimo, a melhor escolha de variáveis fracamente relevantes é por aquelas que não são redundantes. O subconjunto ótimo de variáveis deve assim ser composto, como ilustra o diagrama de Venn na Figura 3.1, pelos elementos de B mais os elementos de C.

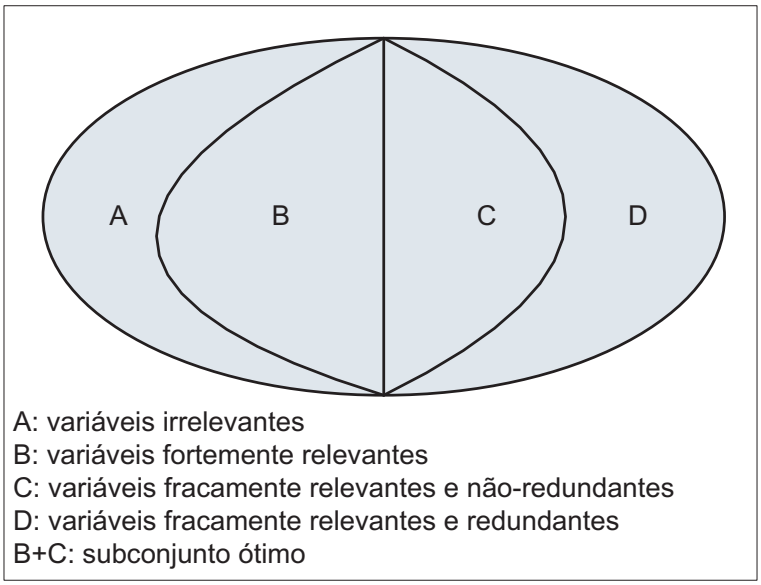

Fig. 3.1: Definição do subconjunto ótimo de variáveis.

3.3 Métodos de seleção de variáveis

Um método de seleção de variáveis consiste em um problema de busca. Dado este contexto, as propostas mais utilizadas possuem uma estrutura de funcionamento semelhante, a qual é sucintamente descrita no Algoritmo 3.3.1.

Algoritmo 3.3.1: Estrutura geral de um algoritmo de seleção de variáveis.

entrada: X, Y
saída : S_f

$max \leftarrow 0$;
repeat
  $S_i \leftarrow busca(X)$;
  $nota \leftarrow avalia(S_i, Y)$;
  if $nota > max$ **then**
    $max \leftarrow nota$;
    $S_f \leftarrow S_i$;
  end
until $criterioParada() = TRUE$;

Este pseudo-código tem como entrada o conjunto das variáveis de entrada X e das variáveis de saída Y, e a saída da rotina será um subconjunto de variáveis $S_f \subseteq X$. Para chegar a esta saída, realiza-se um procedimento iterativo: inicia-se com a busca por possíveis subconjuntos candidatos, em que o candidato escolhido em cada iteração (S_i) é avaliado por um critério que indique o poder

preditivo de S_i em relação ao conjunto Y, ficando armazenado sempre aquele subconjunto com o máximo desempenho, até que se atinja o critério de parada da busca. De forma resumida, pode-se dizer então que as técnicas de seleção de variáveis possuem três importantes sub-atividades que as distinguem:

1. Heurística de busca;
2. Critério de avaliação do subconjunto de variáveis;
3. Critério de parada da busca.

A classificação dos diferentes métodos se dá pela maneira como cada um avalia o subconjunto (Sub-atividade 2); afinal, esta é a atividade chave que define de fato quais variáveis são escolhidas e quais são preteridas. Até alguns anos atrás esta classificação era feita somente por métodos livres de modelo e métodos baseados em modelo. A primeira forma consiste na realização do processo de seleção em uma etapa anterior à síntese do modelo classificador ou regressor do problema, enquanto, na segunda forma, a tarefa de seleção de variáveis e a síntese do modelo ocorrem de forma interdependente. Descrevendo de outro jeito, métodos livres de modelo fazem a avaliação dos subconjuntos de variáveis candidatos à seleção sem usar um algoritmo de aprendizado de máquina, enquanto os métodos baseados em modelo dele fazem uso na avaliação.

Em 1997, o trabalho de KOHAVI & JOHN [44] propôs aumentar a especificidade desta taxonomia, levando assim a três classes de técnicas de seleção de variáveis: método por filtro, por envoltório e embutido. Cada uma delas tem suas características, vantagens, desvantagens e escopo principal de aplicação, que são descritos a seguir. Considera-se à parte das classes de métodos as heurísticas de busca, que são tratadas em uma seção exclusiva.

3.3.1 Filtro

O método por filtro corresponde às técnicas de seleção livres de modelo, ou seja, não se baseiam no desempenho dos possíveis subconjuntos dentro de um determinado modelo. Assim, um algoritmo de filtro utiliza medidas de associação e correlação entre as variáveis para escolher um subconjunto vencedor. Caso seja um problema de aprendizado supervisionado, é feito o cálculo das medidas das variáveis de entrada com as variáveis de saída. Se for aprendizado não-supervisionado, como um problema de agrupamento, as medidas são feitas somente entre as variáveis de entrada.

A Figura 3.2 apresenta um fluxograma simples para ilustrar de forma mais clara como se dá o processo de seleção de variáveis utilizando a abordagem por filtro. Percebe-se que, partindo do conjunto inicial de variáveis disponíveis, a seleção das melhores pelo filtro e a síntese do modelo são

atividades totalmente independentes, onde o resultado do filtro serve como entrada para a atividade final.

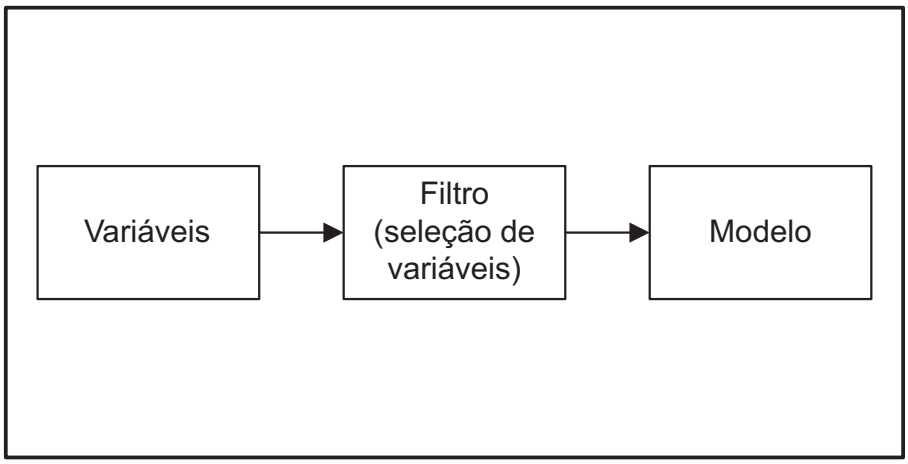

Fig. 3.2: Processo de seleção de variáveis pela abordagem por filtro.

O bom funcionamento de um filtro depende de uma medida robusta de associação ou correlação, seja linear ou não-linear. Além disso, é necessário um bom algoritmo de busca que consiga escapar de mínimos locais. Dadas estas restrições, as principais vantagens e justificativas para uso dos filtros são:

- Maior rapidez no uso, em comparação com a abordagem por envoltório.
- Algumas implementações de filtros conseguem fornecer uma seleção genérica de variáveis, sem dependência do modelo em uso posteriormente.
- Podem ser usados como uma etapa de pré-processamento para reduzir a dimensionalidade do problema e evitar sobreajuste.

Porém, a independência de um modelo implica também na principal desvantagem do método por filtro: pode-se não alcançar o critério de otimalidade e o resultado esperado pode não acontecer ao usar o modelo. Desta forma, recomenda-se seu uso em problemas com bases de dados contendo elevado número de amostras e com alta dimensão, uma vez que neste caso a utilização de outras abordagens pode ser inviável devido ao alto esforço computacional que será necessário.

Além dos filtros que utilizam medidas de auto-correlação linear ou não-linear, como o que se descreve a seguir e que é utilizado neste trabalho, filtros que utilizam medidas de informação mútua e entropia são também usados em diversas aplicações [46].

O filtro CFS

Este trabalho utiliza o filtro denominado *"Correlation-based Feature Selection"* (CFS), proposto por HALL [31]. Este procedimento seleciona o melhor subconjunto de variáveis por meio de uma heurística que atribui nota a cada um dos subconjuntos examinados no espaço de buscas. Considerando um problema de classificação, a hipótese na qual se baseia esta heurística é a de que bons subconjuntos contêm variáveis altamente correlacionadas com a classe dos dados, mas que não são correlacionadas entre si. A Equação 3.1 formaliza a meta da heurística:

$$N_s = \frac{k \cdot \overline{r_{cf}}}{\sqrt{k + k \cdot (k-1) \cdot \overline{r_{ff}}}} \tag{3.1}$$

onde N_s é a nota atribuída pela heurística ao subconjunto S que contém k variáveis, $\overline{r_{cf}}$ é a média dos valores de correlação de cada variável com a classe, e $\overline{r_{ff}}$ é a média dos valores de correlação das variáveis entre si. A Equação 3.1 pode ser vista também como tendo no numerador o poder de predição do subconjunto (relevância), enquanto no denominador o grau de redundância entre as variáveis deste grupo. A nota a ser escolhida no processo de busca deve ser a maior possível.

As medidas de correlação são calculadas de maneira global, considerando todas as amostras de dados. Porém, há casos onde variáveis que não são relevantes globalmente possuem relevância local, ou seja, para um determinado subconjunto das amostras a variável é importante na determinação dos valores de saída. Por isso o método também utiliza uma heurística de seleção de variáveis que são localmente relevantes; consiste em analisar as variáveis que ficaram de fora após a busca ter ocorrido, introduzindo no subconjunto aquelas que possuem uma correlação com a variável de saída maior que o máximo da correlação dela com qualquer outra variável já selecionada.

Dado o contexto deste trabalho, que é o de solucionar um problema de regressão com dados numéricos e contínuos, o método propõe que o cálculo dos valores de correlação para chegar à nota pela Equação 3.1 seja feito usando o coeficiente de correlação linear de Pearson, dado por:

$$r_{XY} = \frac{\sum(x - \overline{x})(y - \overline{y})}{n\sigma_x\sigma_y} \tag{3.2}$$

onde X e Y são variáveis contínuas, n é o número de amostras das duas variáveis, $\overline{x}$ e $\overline{y}$ são seus valores médios, e σ_x e σ_y são os respectivos valores de desvio padrão.

3.3.2 Envoltório

O método por envoltório (em inglês, *wrapper*), cuja ideia é ilustrada na Figura 3.3, é uma abordagem dependente de modelo que foi popularizada principalmente graças ao trabalho de KOHAVI & JOHN, em 1997 [44]. Seu princípio de funcionamento consiste primeiro na escolha de um modelo

preditor ajustado para os dados; este é então usado como uma caixa-preta para fornecer uma medida de desempenho de cada subconjunto de variáveis candidato que vai sendo percorrido ao longo da busca. Normalmente, esta medida que deve ser maximizada consiste de alguma métrica que indique o desempenho do subconjunto na predição.

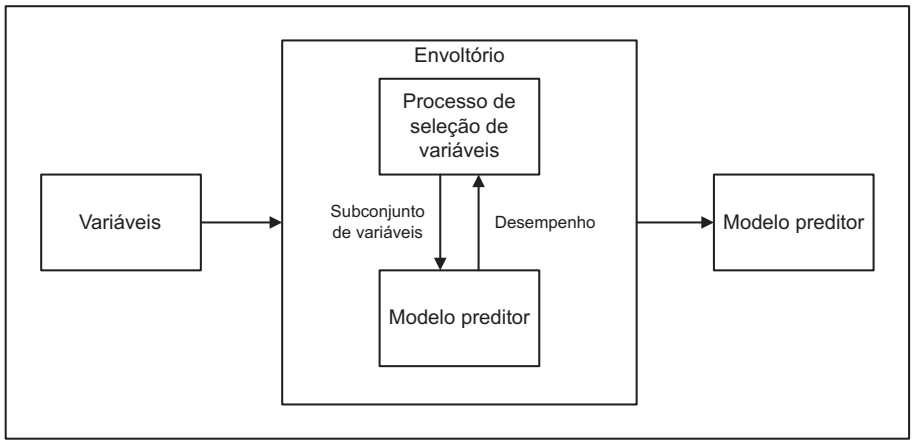

Fig. 3.3: Seleção de variáveis pelo método por Envoltório.

Uma vez que existe esta interação do processo de seleção com um modelo ajustado, a principal vantagem do método por envoltório é que a qualidade do subconjunto selecionado tende a ser superior ao resultado obtido pelo método de filtro. Além disso, o uso do modelo preditor como uma caixa-preta torna o método simples e universal para aplicação.

Por outro lado, o método pode ser computacionalmente mais custoso em termos de tempo e memória devido ao fato de que é necessário realizar o treinamento do modelo preditor para cada subconjunto candidato de variáveis. Recomenda-se seu uso então em problemas de número reduzido de amostras, como séries temporais curtas, alguns problemas de bioinformática, mineração de dados e engenharia de *software*.

Na questão da implementação é preciso então tomar as seguintes decisões para usar a técnica:

1. Como explorar o espaço de busca dos possíveis subconjuntos de variáveis.

2. Como avaliar o desempenho na predição de uma máquina de aprendizado para guiar a busca.

3. Qual máquina de aprendizado utilizar.

O Item 1 é descrito na Seção 3.4, o Item 3 depende do problema que está sendo tratado e de qual modelo melhor se adequa para uma solução. Para o Item 2, a heurística mais popular é a de múltiplas repetições da validação cruzada em 5 partições (em inglês, *five-fold cross-validation*) [44].

A validação cruzada em k partições consiste em dividir o conjunto de treinamento em k partições disjuntas, realizar o treinamento da máquina de aprendizado com $k-1$ partições e medir o desempenho da predição na partição que ficou de fora. Esse procedimento é repetido k vezes, em que a cada vez uma partição é separada para o papel de conjunto de teste e ao final tira-se a média dos k valores de desempenho para tomá-la como medida final.

A Figura 3.4 ilustra o uso desta proposta para avaliar o desempenho de um subconjunto de variáveis no método por envoltório, no caso utilizando validação cruzada em 3 partições. Três modelos passam pelo treinamento, cada um com uma das 3 combinações possíveis de partições duas a duas, sendo que são consideradas somente as variáveis que fazem parte naquele momento do subconjunto candidato à seleção. Feito o treinamento, utilizam-se os parâmetros (representados na figura pela letra P) determinados para executar novamente o preditor, agora com o conjunto de teste (a partição que ficou de fora). Chega-se então às três medidas de desempenho, de onde se tira a média final.

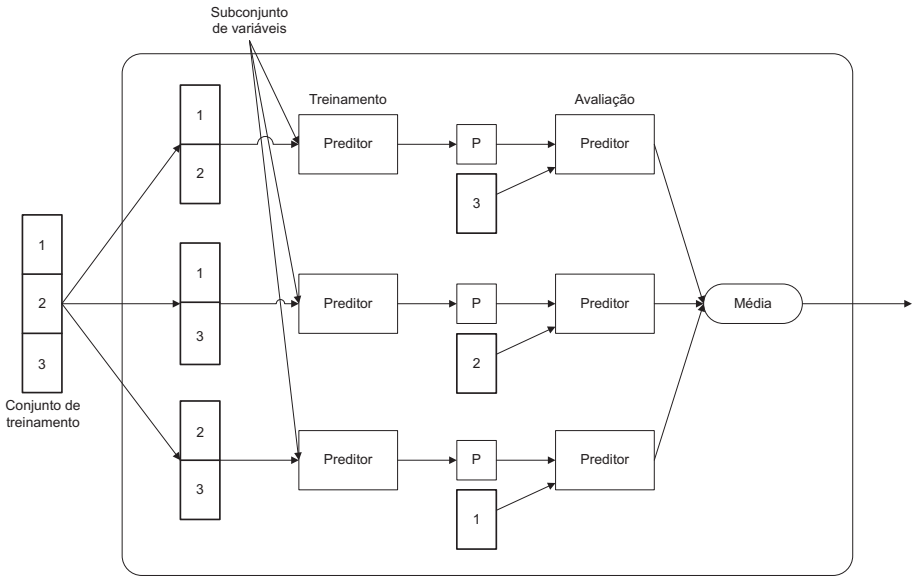

Fig. 3.4: Método de validação cruzada em três partições para estimar desempenho.

A validação cruzada em cinco partições é repetida várias vezes, pela proposta de KOHAVI & JOHN, com o objetivo de contornar o problema de mínimos locais em alguns algoritmos de treinamento de modelos. Por exemplo, no caso de uma rede neural artificial é possível que o treinamento convirja para diferentes mínimos, dependendo de como foram iniciados os pesos da rede. Por isso, fazer a validação cruzada somente uma vez e avaliar o seu resultado pode não representar de forma fiel o seu desempenho.

3.3.3 Embutido

Como o próprio nome diz, os métodos de seleção de variáveis Embutidos (em inglês, *Embedded*) são aqueles em que o processo de seleção faz parte da própria síntese do modelo. A Figura 3.5 ilustra esta classe de métodos.

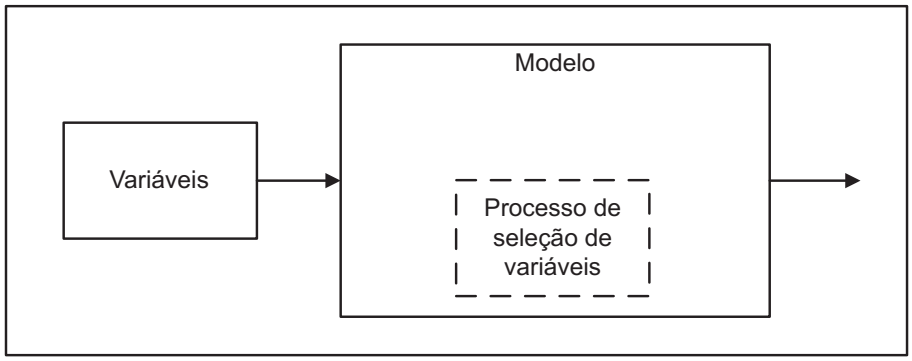

Fig. 3.5: Método embutido de seleção de variáveis.

Em comparação aos métodos por envoltório, defende-se que os métodos embutidos são mais eficientes em diversos aspectos [29], como: eles fazem melhor uso dos dados de treinamento, pois não precisam fazer a divisão dos dados em um conjunto para treinamento e outro para validação; e a sua velocidade é superior, porque não repetem o treinamento do modelo para cada subconjunto candidato de variáveis.

Como exemplos de métodos embutidos há o algoritmo por árvore de decisão CART [9] e máquinas de vetor de suporte em que a seleção das variáveis sai como resultado junto à resolução do problema de programação quadrática [69].

3.4 Estratégias de busca para seleção de variáveis

Como descrito na Seção 3.2, a busca exaustiva no espaço de subconjuntos de variáveis tem custo computacional de ordem exponencial em relação ao número de variáveis. É necessário então utilizar alguma heurística para tornar o processo mais rápido; são descritas a seguir algumas abordagens para isto.

3.4.1 Seleção Progressiva e Eliminação Regressiva

A ideia de funcionamento da busca por seleção progressiva (em inglês, *forward selection*) consiste em começar com um subconjunto vazio de variáveis e incluir uma variável de cada vez, considerando em cada iteração a inclusão que trouxe o melhor desempenho (ou menor medida de erro). É uma

heurística de busca gulosa[1], que possui complexidade computacional de ordem $O(n^2)$, onde n é o número total de variáveis.

Para melhor entendimento, considere o exemplo na Figura 3.6. Ela apresenta o gráfico da medida de erro *versus* o subconjunto de melhor desempenho a cada inclusão de variável, ao executar o método por envoltório para um conjunto de dados com 11 variáveis.

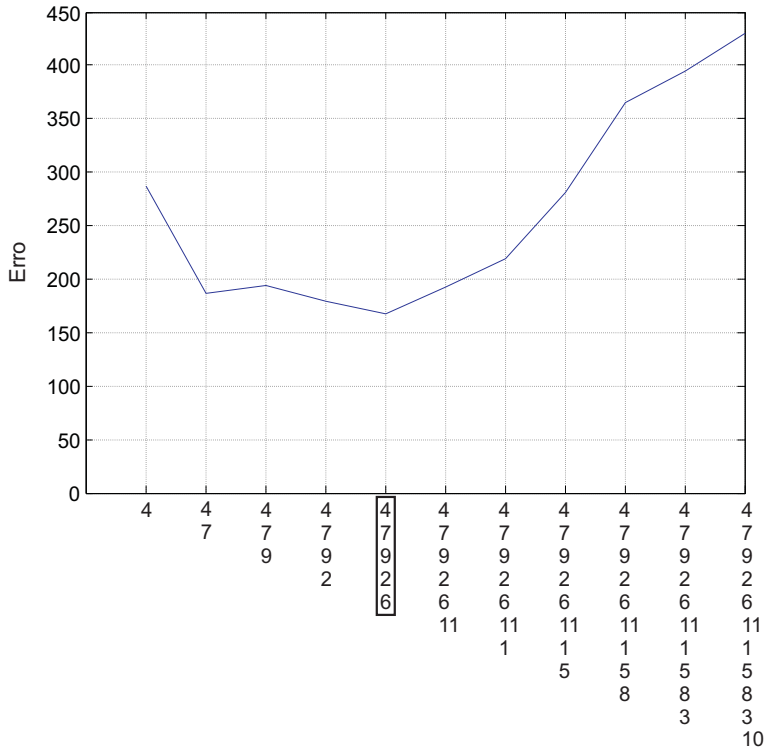

Fig. 3.6: Exemplo de busca por seleção progressiva, com n=11.

O processo de busca começa com um subconjunto S vazio; são avaliados então os 11 subconjuntos possíveis ao incluir em S uma variável, calculando a medida de erro. O melhor subconjunto, ou seja, aquele que apresentou o menor erro, foi $S = \{4\}$. Passa-se então à segunda iteração da busca, e são avaliados os 10 subconjuntos possíveis constituídos pela variável 4 e mais uma variável que ainda não foi escolhida, chegando como resultado ao subconjunto $S = \{4, 7\}$. O processo então continua, agora com a avaliação de 9 possíveis subconjuntos, depois 8, 7, 6, ..., até um subconjunto final que nada mais é que o próprio conjunto de todas as variáveis. Neste ponto, o algoritmo termina e é escolhido como resposta aquele subconjunto que teve a menor medida de erro, que no exemplo foi $S = \{4, 7, 9, 2, 6\}$.

No processo todo de busca, considerando que, para cada subconjunto candidato é feita uma ava-

[1]O termo guloso vem do fato de que o algoritmo não revisita decisões passadas para incluir ou excluir variáveis e assim tomar novas decisões.

liação, tem-se ao final $n(n+1)/2$ avaliações. No exemplo da Figura 3.6, para $n = 11$ houve 66 avaliações. Vale ressaltar que, no caso do método por envoltório descrito na Seção 3.3.2, o número total de treinamentos do modelo preditor compreende o número de avaliações multiplicado pelo número de repetições da validação cruzada e pelo número de partições. Supondo que no exemplo dado repetia-se 20 vezes a validação cruzada em 5 partições, chega-se ao valor final de $20 \times 5 \times 66 = 6600$ treinamentos do modelo.

Ainda sobre a Figura 3.6, vale destacar que, caso a curva da figura apresente redução nos valores a cada iteração, o que significa melhor desempenho do modelo, tem-se o caso onde todas as variáveis disponíveis inicialmente são importantes e portanto nenhuma delas deve ser descartada na criação do modelo preditor.

A heurística de eliminação regressiva (em inglês, *backward elimination*) também é um algoritmo guloso de busca e que tem a ideia central semelhante à técnica de seleção progressiva. A diferença está no fato de que, em vez de partir do subconjunto de variáveis vazio, o algoritmo começa com o subconjunto contendo todas as variáveis candidatas e a cada iteração é eliminada aquela variável que deixa como subconjunto resultante o de melhor desempenho (menor erro).

Comparando-se as duas estratégias [44], tem-se que a seleção progressiva possui maior probabilidade de encontrar um subconjunto de tamanho menor do que a eliminação regressiva, trazendo assim melhor redução dimensional do problema. Por outro lado, a eliminação regressiva tem, em teoria, maior facilidade para capturar variáveis que em conjunto têm relevância com as variáveis de saída. Uma vez que normalmente a prioridade no uso de seleção de variáveis é reduzir a dimensão do problema, utiliza-se a seleção progressiva.

Também é possível combinar as duas estratégias; por exemplo, realizando numa iteração a inserção de duas variáveis e na iteração seguinte a eliminação de uma variável [90].

3.4.2 Outras estratégias de busca

Embora as estratégias vistas na Seção 3.4.1 tenham complexidade computacional inferior à da busca exaustiva e resultem em bom ganho de tempo, o uso delas em problemas cuja dimensão é muito elevada acaba se tornando impraticável. Por isso, outras estratégias de busca foram propostas.

A busca *Best-First* [72] é uma estratégia não-gulosa, ou seja, ela permite retornar no caminho de busca. Da mesma forma que as estratégias de eliminação regressiva e seleção progressiva, ela vai caminhando pelo espaço de busca inserindo ou removendo variáveis do subconjunto, uma a uma. Porém, se é percebido que o subconjunto em formação não está melhorando de desempenho após um número pré-definido de iterações, o algoritmo retorna para um subconjunto anterior com um melhor desempenho e dali continua sua busca até atingir um determinado critério de parada.

Outra possibilidade de busca é por algoritmos genéticos [35]. Para a seleção de variáveis haveria

uma população inicial de subconjuntos candidatos escolhidos aleatoriamente; a partir deles operadores genéticos são aplicados iterativamente, promovendo uma evolução na população para que sobrevivam somente os subconjuntos mais adaptados que, no caso, seriam os melhor avaliados quanto ao problema a ser modelado.

3.5 Síntese

Este capítulo apresenta uma introdução aos métodos de seleção de variáveis. Dado o aumento da dimensão e quantidade dos dados presentes nos problemas em que máquinas de aprendizado podem atuar, as técnicas de seleção de variáveis surgiram com o intuito de eliminar informações desnecessárias e que podem prejudicar o desempenho das soluções elaboradas. Os métodos de seleção por filtro e envoltório são utilizados na parte experimental do trabalho e por isso são detalhados.

A técnica de filtro baseia-se no princípio de selecionar o subconjunto de variáveis independentemente do modelo preditor que será posteriormente utilizado; medidas de correlação linear, como para o caso do filtro CFS, e outras medidas e propriedades matemáticas dos dados são consideradas nos algoritmos de filtro para fazer a escolha do subconjunto.

O método por envoltório utiliza o princípio de selecionar o subconjunto de variáveis de acordo com um modelo preditor pré-determinado. Devido a isso, seu tempo de execução costuma ser maior do que o de métodos por filtro; porém, o subconjunto escolhido pelo envoltório possui maior chance de levar o modelo preditor a soluções melhores do que o subconjunto selecionado pelo filtro.

O capítulo seguinte realiza o estudo teórico sobre o problema em foco neste trabalho, que é a predição de esforço de execução de testes funcionais. Há uma revisão dos trabalhos relacionados ao assunto, uma análise do processo de execução de testes funcionais e do impacto das estimativas de esforço, encerrando com o planejamento das atividades experimentais.

Capítulo 4

Esforço de execução de testes

Este capítulo apresenta uma análise sobre o problema de estimar esforço de execução de testes, em especial testes funcionais. Primeiro é feita uma revisão sobre os trabalhos já publicados sobre estimativa de esforço para o processo de *software* e exclusivamente para testes; a seguir é discutido em maiores detalhes o processo de execução de testes funcionais, dando ênfase aos fatores que podem afetar direta ou indiretamente o esforço. Finalmente, discute-se quais experimentos são necessários para o estudo das técnicas de AM e seleção de variáveis na solução do problema.

4.1 Trabalhos relacionados

Dado que há uma forte relação do processo de desenvolvimento de *software* com o processo de testes, afinal o segundo está contido no primeiro, é importante que se analise não somente os trabalhos sobre esforço em testes, mas também aqueles sobre esforço da etapa de desenvolvimento ou até de todo o processo de *software*. Não suficiente, muitos trabalhos da área de testes tiveram suas propostas inspiradas em estudos para estimar o esforço do desenvolvimento de um *software*. Por isso, nas próximas duas seções ambos os problemas são contemplados.

4.1.1 Esforço em processo de *software*

Um dos primeiros métodos para estimar esforço no desenvolvimento completo de um *software* é a Análise por Pontos de Função (em inglês, *Function Point Analysis*, FPA), proposta por ALBRECHT [1]. Ela estima a complexidade do *software* em estágios iniciais do processo com base no número de entradas, saídas, requisições, arquivos e interfaces do sistema, incluindo também fatores técnicos. Esta análise resulta em uma medida da complexidade do sistema, chamada número de Pontos de Função. O usuário deste método necessita de dados históricos para poder estabelecer o número ne-

cessário de linhas de código (em inglês, *Lines of Code*, LOC) para implementar um ponto de função, bem como a relação entre esforço em pessoa-hora e LOC.

A relação entre esforço e LOC foi abordada no trabalho de BOEHM, que propôs o modelo COCOMO (em inglês, *Constructive Cost Model*) [8]. Este modelo consiste basicamente em uma série de equações não-lineares para estimar o esforço para os tipos mais comuns de *software* na época, utilizando como variável independente o número de LOCs a serem entregues ao cliente e, assim, é complementar ao modelo FPA.

Ao final do século XX, os avanços nas práticas de engenharia de *software* e a consolidação dos projetos de *software* orientado a objetos encorajaram a pesquisa em novos métodos de estimativa, como o modelo de Pontos por Caso de Uso (em inglês, *Use Case Points*, UCP), elaborado por KARNER [41]. Esta proposta é baseada nas ideias do modelo FPA mas ela muda a perspectiva de complexidade do sistema das funções de código para os casos de uso modelados. Valores são atribuídos de acordo com as características de cada caso de uso, como os atores e número de interações; estas características são então ponderadas por fatores técnicos a fim de resultar em um número de pontos por caso de uso. De forma similar ao método FPA, o usuário do método UCP precisa definir uma razão entre um ponto por caso de uso e o esforço necessário para implementá-lo.

Outro caminho tomado na pesquisa em estimativa de esforço foi o do uso de técnicas de aprendizado de máquina para sintetizar um modelo preditor, por meio dos dados históricos disponíveis. Destacam-se nesse sentido alguns trabalhos, em ordem cronológica:

- FINNIE *et al.* [26], em 1997, aplicam redes neurais artificiais, modelos de regressão e a técnica chamada Raciocínio Baseado em Casos (em inglês, *Case-Based Reasoning*);
- DOLADO & FERNÁNDEZ [23], em 1998, comparam programação genética, redes neurais artificias e regressão linear;
- SHUKLA [76], em 2000, propõe o uso de redes neurais artificiais treinadas por algoritmos genéticos para estimar esforço a partir de 39 variáveis disponíveis;
- SHAN *et al.* [75], em 2002, também utilizam programação genética;
- OLIVEIRA [64], em 2006, aplica regressão por vetor de suporte para um experimento de estimar esforço sobre a base de dados pública da Agência Espacial Norte-Americana.

Caso o leitor deseje se aprofundar mais na literatura disponível, recomenda-se o trabalho de JØRGENSEN & SHEPPERD [39], que faz uma ampla revisão dos trabalhos publicados sobre custo de desenvolvimento de *software* até 2007.

4.1.2 Esforço de teste de *software*

A proposta de KARNER [41] para estimar esforço em desenvolvimento inspirou o modelo proposto por NAGESWARAN para esforço de teste de *software* [62], cujo método é bastante abrangente e utiliza pesos e notas que incluem as características dos casos de uso para computar o número de pontos por caso de uso. Diferentemente da proposta de KARNER, o resultado do modelo de NAGESWARAN é proporcional ao esforço requerido para as atividades de teste, ao invés das atividades de desenvolvimento.

Há duas críticas ao modelo de NAGESWARAN: o método tem a limitação de que o esforço previsto cobre todo o processo de testes, sem discriminar as etapas; além disso, há notas e pesos que precisam ser definidos no modelo pelo gerente de testes, o que torna a tarefa de estimar subjetiva demais; sua proposta teve modificações sugeridas por ALMEIDA *et al.* [2]. Em um estudo preliminar a este trabalho de mestrado, SILVA *et al.* [77] propõem o uso de uma métrica de eficiência para realizar as estimativas de esforço de execução de testes funcionais.

Por meio de uma estratégia diferente, ARANHA & BORBA [3] usam o conceito de complexidade de caso de teste (definido como Pontos de Execução) para apresentar um modelo para estimar o esforço da execução de testes. É proposta a avaliação de cada caso de teste segundo um grupo de características, as quais são avaliadas com pesos específicos dados por especialistas da equipe de testes. O cálculo da complexidade dos casos de testes permite estabelecer uma relação, também baseada no histórico de dados, entre os Pontos de Execução e o esforço de fato realizado em cada ciclo de testes. Os resultados experimentais do modelo demonstraram uma precisão muito boa; porém, além de também depender de avaliações subjetivas por especialistas, o método requer que os casos de testes sejam projetados em uma linguagem natural controlada [74], o que pode ser inviável para organizações de pequeno porte.

Apesar da inexistência de resultados experimentais, KUSHWAHA & MISRA [49] defendem o uso de uma métrica diferente de complexidade, denominada *Cognitive Information Complexity Measure* (CICM), como uma ferramenta apropriada para estimar esforço. Essa métrica tenta mensurar a quantidade de informação contida no código e a complexidade que tal informação possui. Assumindo a hipótese de que sistemas mais complexos, do ponto de vista de informação, requerem mais esforço para testes, a métrica CICM seria adequada para uso na tarefa de estimar esforço e teria um desempenho superior se comparada à tradicional métrica de complexidade ciclomática[1].

Ao estabelecer uma analogia da dinâmica ecológica de presa e predador com a dinâmica de defeitos e testadores, CALZOLARI *et al.* [12] abordam o problema de esforço de manutenção de *software* propondo a adaptação do modelo LOTKA-VOLTERRA para esta realidade. São avaliadas duas varia-

[1] A complexidade ciclomática é uma métrica de *software* que proporciona uma medida da complexidade lógica de um programa [80].

ções, uma linear e outra não-linear, e pode-se observar que o modelo não-linear apresenta nos estudos de casos os melhores desempenhos tanto para mapeamento da dinâmica real como para prever esforço de teste a partir de dados iniciais.

Baseado neste trabalho, STIKKEL [81] propõe a utilização do modelo dinâmico de LOTKA-VOLTERRA linearizado para relacionar o esforço de testes de integração com o número acumulado de falhas reveladas. Desta maneira, é possível fazer um acompanhamento do processo de teste e o gerente pode calcular o tempo restante necessário para término do teste, supondo que o processo atingiu naquele momento o máximo de eficiência na descoberta de defeitos. A limitação tanto desta proposta como da realizada por CALZOLARI *et al.* reside justamente no fato de que somente é possível utilizar os modelos para fazer uma predição quando o processo já está em andamento, não havendo possibilidade de realizar uma estimativa prévia do esforço para executar o ciclo completo de testes.

Com o foco no esforço de teste unitário, MCGREGOR & SRINIVAS [54] propõem uma métrica de testabilidade de uma classe de código orientado a objeto, afirmando que há uma relação direta da testabilidade com o esforço necessário para testá-la, porém não realizam nenhuma validação com dados de projetos de *software* reais.

ZHU *et al.* [98] apresentam uma abordagem distinta das demais, propondo o uso de um algoritmo de aprendizado de máquina — Regressão por Vetor de Suporte com função núcleo do tipo PUK — sobre uma base de dados para sintetizar um modelo preditor do esforço de execução de testes. A base contém três variáveis que descrevem o número de casos de testes, a complexidade de execução de cada teste e o nível dos testadores em relação à equipe. Há um porém: as variáveis referentes à complexidade dos testes e nível dos testadores são definidas de forma que se depende de avaliações subjetivas. Não obstante, o autor não realiza um experimento comparativo utilizando a mesma base de dados com outros modelos de aprendizado de máquina, assim nada garante que um modelo diferente não possa apresentar um desempenho ainda melhor.

De maneira geral, nota-se como limitações dos trabalhos relacionados, especificamente sobre esforço de teste: a dependência de avaliações subjetivas, a necessidade de especificação dos testes em linguagem controlada, a ausência de validação em ambiente real e a ausência de estudos comparativos de modelos de aprendizado de máquina. Para evitar esses problemas, este trabalho elabora um método que tem os seguintes requisitos:

- Utilizar somente variáveis objetivas.
- Não requisitar a modificação dos casos de testes.
- Observar o desempenho por meio de estudos de casos com sistemas de *software* reais.
- Comparar o desempenho de diferentes modelos preditores.

4.2 O processo de execução de testes funcionais

Há três fases importantes no processo de testes, no contexto do problema deste trabalho: projeto de testes, execução de testes e análise dos resultados de teste. É importante frisar aqui que a análise feita parte do processo de testes utilizado pelo projeto HARPIA, que se baseia no modelo RUP (em inglês, *Rational Unified Process*) [47] e no qual foi tomado o cuidado de considerar somente as atividades que são comuns à maioria dos processos de testes. A Figura 4.1 apresenta uma visão geral das etapas principais, descritas em maiores detalhes a seguir.

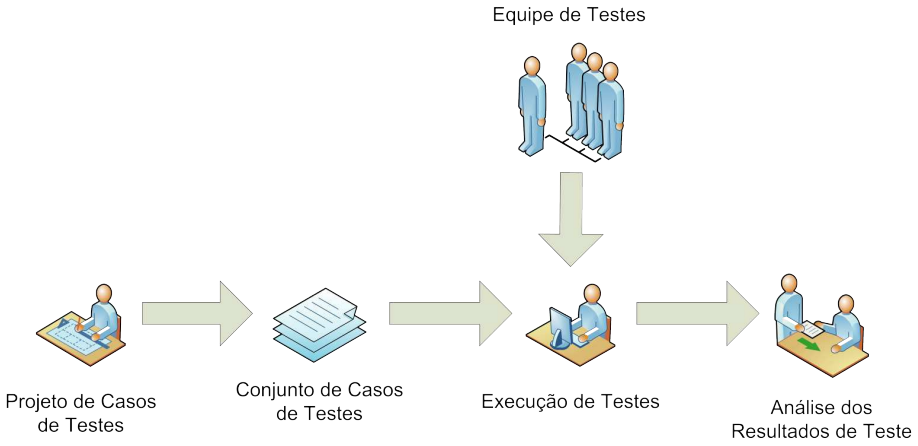

Fig. 4.1: Fluxograma das atividades principais de teste.

Um caso de teste (CT) consiste da descrição do estado do *software* previamente ao teste (estado inicial), de uma sequência de entradas de teste, e de uma relação de resultados esperados para cada entrada ou conjunto de dados de entrada de teste [6]. Considera-se também, neste trabalho, como parte de um caso de teste as operações necessárias para realizar o teste (*script* de teste). Estas operações contemplam a interação do usuário com a interface do sistema, bem como a manipulação de dados. Assim, em outras palavras, um caso de teste deve contemplar os *passos* que devem ser executados, os *dados de entrada* e os *dados esperados* como saída da sua execução.

A fase de projeto dos casos de testes consiste em especificar os casos de testes. A maneira como os casos de testes são criados depende do tipo, nível e dos critérios de teste que estão sendo utilizados. No caso dos testes funcionais, os CTs são criados com base nos requisitos funcionais do sistema a ser testado, e outras informações como as regras de negócios. O produto final desta etapa é o Conjunto de Casos de Testes.

Os CTs são então atribuídos aos testadores segundo critérios específicos do gerente de testes. Usualmente, ele atribui aos testadores mais experientes CTs que demandam maior atenção, que são mais complexos ou prioritários, deixando os CTs mais simples e de menor importância para os testadores menos experientes ou menos capacitados.

Inicia-se então a execução do Conjunto de Casos de Testes, o que corresponde a um Ciclo de Testes. Um testador que tem um determinado caso de teste a ser realizado executa passo a passo as ações descritas, manualmente e como se fosse um usuário comum do sistema. Estas ações podem variar em termos de complexidade, como um passo que determina que o testador submeta um arquivo e aguarde o sinal de *OK* do sistema, ou um passo em que o testador deve executar uma consulta SQL no banco de dados da aplicação, a fim de verificar alguma regra de negócio. Em geral, quanto maior o *número de passos* de um CT, maior será o tempo que o testador demora para executá-lo.

A complexidade dos passos de um caso de teste pode ter relação com as características do sistema que está sendo testado: o tipo de interface com o usuário, o número de funcionalidades do sistema e os relacionamentos entre elas, os tipos de dados que são manipulados, o tempo de resposta, etc.. Consequentemente, a *complexidade dos casos de teste* está relacionada ao esforço que será exigido para realizar testes do sistema. Não obstante, a *complexidade do sistema* também pode ter relação com o esforço nos testes [25].

Ao longo da execução dos passos, o testador encontra falhas que devem ser registradas. O processo de registro da falha pode ser feito após o término da execução do caso de teste; mas, de toda maneira, o testador necessita guardar, no momento em que ocorreu a falha, informações que lhe permitam registrá-la posteriormente. Portanto, o *número de falhas* encontradas afeta o tempo de execução do caso de teste.

Terminada a execução de um CT, o testador passa à execução de outro, até que a equipe termine o conjunto de testes selecionados para execução no ciclo. A *experiência* do testador com o sistema cresce (até certo limite) à medida que executa CTs que tenham similaridades e também quando volta a testar o mesmo sistema em um novo ciclo de testes.

Testadores possuem experiência e habilidade distintas na execução de testes. Assim, a *composição* da equipe por testadores mais ou menos experientes influi na eficiência do trabalho, bem como no esforço requerido para completá-lo.

Encerrada a etapa de execução de testes, passa-se à análise dos resultados e elaboração do relatório de testes para a equipe de desenvolvimento. Com os resultados em mãos, os desenvolvedores podem realizar correções no sistema e enviar uma nova versão para outro ciclo de testes. Este novo ciclo pode conter novos casos de testes verificando funcionalidades recém-implementadas, ou pode repetir a execução dos CTs antigos. Configura-se assim um processo iterativo entre o desenvolvimento e os testes, como pode ser visto na Figura 4.2, retirada do livro de McGregor & Sykes [55].

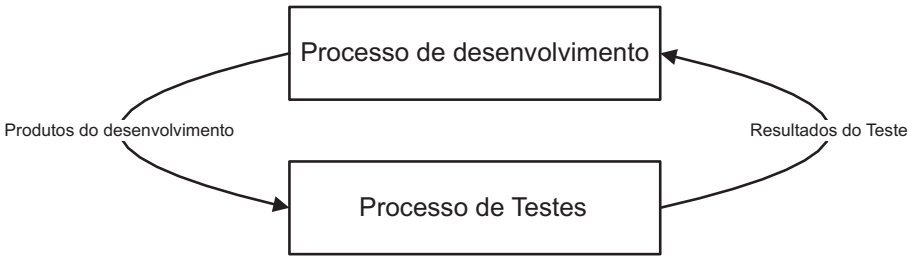

Fig. 4.2: Interação entre o processo de desenvolvimento e o processo de testes.

4.3 Impacto da estimativa de esforço no processo de execução de testes

Além de entender quais são os principais fatores que determinam o esforço, é importante compreender o que acarreta uma estimativa acima ou abaixo do que realmente ocorreu. Há dois cenários possíveis:

1. O gerente de testes prevê um esforço da execução que ao final percebe-se ser mais do que o necessário.

2. O gerente de testes prevê um esforço da execução que ao final percebe-se ser menos do que o necessário.

Não são encontrados trabalhos sobre o assunto, ou seja, abordando o esforço em testes. No entanto, pode-se utilizar a experiência da equipe de testes do projeto HARPIA e um artigo publicado por JØRGENSEN & SJØBERG em 2001 [40], cujo escopo foi todo o processo de *software*, para levantar algumas hipóteses.

Dado o primeiro cenário, em que ocorre uma estimativa pessimista (ou superestimada), dois estudos de caso apresentados no trabalho de JØRGENSEN & SJØBERG, compostos por depoimentos de desenvolvedores e gerentes de duas organizações distintas, apontam que o tempo que sobra quando o trabalho termina antes é aproveitado para efetuar melhorias no produto de *software*. De maneira análoga, na realização de testes no projeto HARPIA isto também ocorria: se um ciclo terminava em um tempo menor do que o previsto, este tempo era aproveitado para projetar e executar mais casos de testes funcionais e/ou outros tipos de testes, como testes de usabilidade e desempenho. Ambos os comportamentos observados, seja nos estudos de caso ou no projeto HARPIA, seguem o princípio de PARKINSON: "o trabalho é expandido de tal forma a preencher o tempo disponível para sua conclusão" [66].

Para o cenário de uma estimativa otimista, o artigo mostra nos dois estudos que a solução de *software* é simplificada, requisitos de qualidade são ignorados e/ou o esforço de testes é reduzido

para entrega do produto no tempo determinado. Ao executar testes no HARPIA e se deparar com a mesma situação, a medida mais comum tomada pelo gerente de testes para que a equipe encerrasse a execução no prazo era priorizar os casos de testes das funcionalidades mais importantes do sistema em detrimento de outras funções.

O artigo ainda mostra que, nos projetos estudados, as estimativas para as etapas de especificação de requisitos, modelagem, gerência e revisão eram normalmente cumpridas. Por outro lado, as etapas de implementação e testes eram frequentemente subestimadas. Isto pode indicar uma situação na qual o projeto corta atividades das etapas iniciais para se manter no cronograma, às custas de retrabalho nas etapas de implementação e testes.

Fica claro então que, enquanto superestimar causa normalmente trabalho extra para que a equipe de testes ocupe o tempo que ainda resta e, consequentemente, melhora-se a análise do sistema, subestimar aumenta o risco de não descobrir falhas importantes devido à priorização de CTs, podendo comprometer a qualidade do *software* a ser entregue.

Estimar de forma pessimista ou otimista acarreta prejuízos, sem dúvida, mas, ao se tomar como prioridade a entrega do produto final do processo com boa qualidade, é fato que subestimar causa maiores estragos do que superestimar. De forma geral, pode-se concluir que é mais prudente estimar um esforço maior do que o necessário porque subestimar aumenta os custos para terminar o ciclo de testes no prazo acordado, promove tensão e *stress* na equipe de testadores envolvida no trabalho e aumenta o risco de entregar resultados de baixa qualidade [68]. Tal constatação pode ser levada em conta na síntese dos modelos preditores do esforço.

4.4 Escolha dos modelos de aprendizado de máquina para o problema

O uso das técnicas de aprendizado de máquina é recomendado, entre outras circunstâncias, quando é difícil descrever todas as variáveis de um problema e, assim, modelá-lo adequadamente. O processo de testes pode ser considerado um problema deste tipo, uma vez que a intervenção humana aumenta consideravelmente a complexidade do problema, no sentido de não ser possível conhecer com exatidão todos os fatores que determinam o esforço realizado pelos testadores.

Redes Neurais Artificiais do tipo Perceptron de Múltiplas Camadas e Regressão por Vetor de Suporte do tipo ϵ-SV com núcleo RBF são as duas técnicas de aprendizado de máquina escolhidas para este trabalho. Além delas, é incluído também nos experimentos um modelo de regressão linear múltipla por mínimos quadrados [92]. A escolha de cada proposta foi baseada nas seguintes razões:

- Rede MLP

  - Histórico de bons resultados alcançados em trabalhos anteriores (vide Seção 4.1).
  - É uma abordagem clássica para criar mapeamentos não-lineares de entrada-saída em problemas complexos.

- SVR com núcleo RBF
  - Possui robustez a problemas de alta dimensão (vide Seção 2.3.7 do Capítulo 2).
  - Possui alta capacidade de generalização, sem necessitar de conhecimento prévio do problema, graças à construção do seu modelo matemático de aproximação.
  - Trabalhos recentes indicam resultados promissores em estimativa de esforço (vide Seção 4.1).
  - O núcleo RBF é o mais adotado e possui bons resultados relatados na literatura.

- Regressão Linear
  - Facilidade de uso e interpretação do modelo.
  - Estabelecer uma comparação entre o seu desempenho e o desempenho das máquinas que realizam aproximações não-lineares.
  - Avaliar se é de fato necessário um modelo não-linear para abordar o problema de estimar esforço.

4.4.1 Função custo assimétrica para a MLP

Conforme visto no Capítulo 2, o treinamento de uma rede MLP pelo algoritmo de retropropagação consiste em solucionar um problema de otimização não-linear irrestrito, com uma função custo que comumente é a média da soma dos erros quadráticos (definido na Equação 2.3). A função de soma dos erros quadráticos é par, e, portanto, simétrica em relação ao eixo y; ou seja, valores negativos ou positivos do erro são ponderados da mesma maneira.

Entretanto, na Seção 4.3 é mostrado que o impacto de subestimar esforço da execução de testes é potencialmente maior do que no caso de superestimar esforço, sob a perspectiva de qualidade do produto. É desejável, portanto, que o modelo de máquina de aprendizado a ser sintetizado "prefira" superestimar a subestimar. Para que isso aconteça, propõe-se a utilização de uma função de erro quadrático assimétrica para o treinamento da rede MLP, de forma que o peso de uma subestimativa seja maior do que o de uma superestimativa no cômputo da função custo. Para solucionar o problema de otimização o algoritmo de treinamento terá então que buscar parâmetros que possuam um compromisso entre evitar erros absolutos grandes e também evitar erros onde a saída da rede é menor do que a saída esperada.

Considere novamente o problema de treinamento de uma rede neural MLP, onde N é o número de neurônios na camada de saída da rede, $d_k(t)$ é a saída esperada para o neurônio k, dada a entrada fornecida no instante t, $y_k(t)$ é a saída computada pelo neurônio k, dada a entrada fornecida no instante t, e M é o número de amostras a serem apresentadas. As Definições 4.1 e 4.2 então apresentam a nova função de erro e seu respectivo valor médio.

Definição 4.1. *Função soma de erro quadrático ponderado.*

Dada a função $p : \mathbb{R}^3 \to \mathbb{R}$ *definida por*

$$p(e, \alpha, \beta) = \begin{cases} \alpha e^2 & \textit{se} \quad e > 0 \\ \beta e^2 & \textit{se} \quad e \leq 0 \end{cases}$$

tem-se que a soma dos erros quadráticos ponderados é

$$J_p(t, \alpha, \beta) = \frac{1}{2} \sum_{k=1}^{N} p(d_k(t) - y_k(t), \alpha, \beta).$$

Definição 4.2. *Função média da soma de erros quadráticos ponderados.*

$$J_{p_{av}} = \frac{1}{M} \sum_{t=1}^{M} J_p(t, \alpha, \beta)$$

Os parâmetros α e β devem ser constantes e determinam os pesos do erro se ele for positivo ou negativo, respectivamente. Para que se tenha uma função custo que dá maior peso às subestimativas é necessário que $\alpha > \beta$. Por exemplo, se são escolhidos $\alpha = 1{,}5$ e $\beta = 1$, tem-se uma função p que calcula o quadrado do erro se ele for negativo (superestimar), e o quadrado do erro acrescido em 50% se ele for positivo (subestimar). A Figura 4.3 apresenta o gráfico desta configuração.

Para utilizar a função custo proposta ainda é preciso realizar uma adaptação no algoritmo de retropropagação, originalmente implementado para ser usado com a função soma de erro quadrático. Para isso toma-se como base o trabalho de CRONE [19], que propõe uma generalização na fórmula do algoritmo de retropropagação a fim de poder ser utilizado com qualquer função custo que seja diferenciável (analiticamente ou numericamente). Assumindo que daqui em diante é considerado o cálculo do erro para uma amostra no instante t como uma aproximação do valor médio, este símbolo é suprimido da Equação 2.2 com o objetivo de simplificação, que assim fica

$$J = \frac{1}{2} \sum_{k=1}^{N} [d_k - y_k]^2. \tag{4.1}$$

Utilizando esta função, calcula-se a função *sensibilidade* da rede em cada neurônio k de cada camada pela regra

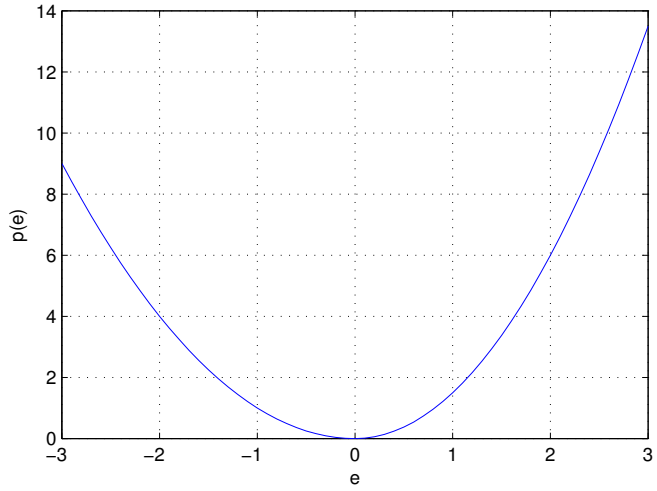

Fig. 4.3: Função $p(e, \alpha, \beta)$ para $\alpha = 1{,}5$ e $\beta = 1$.

$$\delta_k = \begin{cases} (d_k - y_k)\varphi'_k(v_k) & \text{se o neurônio } k \text{ está na camada de saída} \\ \varphi'_k(v_k) \sum_i \delta_i w_{ki} & \text{se o neurônio } k \text{ está numa camada oculta} \end{cases} \tag{4.2}$$

onde $\varphi'_k(v_k)$ corresponde à derivada primeira da função de ativação do neurônio k em relação à entrada líquida[2] v_k, δ_i corresponde aos valores de sensibilidade para os neurônios da camada subseqüente, e w_{ki} consiste no peso sináptico entre o neurônio k e o neurônio i da camada subseqüente.

CRONE então adota uma generalização da regra, partindo de uma função custo genérica

$$E = C(d_k, y_k), \tag{4.3}$$

para modificar a Equação 4.2, ainda em conformidade com o algoritmo de RUMELHART *et al.* [71], chegando à regra geral para cálculo da sensibilidade:

$$\delta_k = \begin{cases} \frac{\partial C(d_k, y_k)}{\partial y_k} \varphi'_k(v_k) & \text{se o neurônio } k \text{ está na camada de saída} \\ \varphi'_k(v_k) \sum_i \delta_i w_{ki} & \text{se o neurônio } k \text{ está numa camada oculta} \end{cases} \tag{4.4}$$

No caso da função soma de erro quadrático ponderado, o termo de derivada parcial da função C apresentada na Equação 4.4 é obtido analiticamente. Se α, β forem considerados constantes, é possível suprimi-los e assim começar por

$$\frac{\partial J_p}{\partial y_k} = \frac{1}{2} \sum_{k=1}^{N} \frac{\partial p(d_k - y_k)}{\partial y_k},$$

[2] A entrada líquida de um neurônio é o resultado da combinação linear dos sinais de entrada com os respectivos pesos.

a partir daí pode-se chamar $e = d_k - y_k$. Aplicando a regra da cadeia, chega-se a

$$\frac{\partial J_p}{\partial y_k} = \frac{1}{2}\sum_{k=1}^{N}\frac{\partial p(e)}{\partial e}\frac{\partial e}{\partial y_k} = -\frac{1}{2}\sum_{k=1}^{N}\frac{\partial p(e)}{\partial e}. \tag{4.5}$$

Deve-se então calcular $\frac{\partial p(e)}{\partial e}, \forall e$, em duas etapas:

$$\text{Se} \quad e > 0 \rightarrow p(e) = \alpha e^2 \rightarrow \frac{\partial p(e)}{\partial e} = 2\alpha e,$$

$$\text{Se} \quad e \leq 0 \rightarrow p(e) = \beta e^2 \rightarrow \frac{\partial p(e)}{\partial e} = 2\beta e,$$

e assim se determina a fórmula final:

$$\frac{\partial p(e)}{\partial e} = \begin{cases} 2\alpha e & \text{se} \quad e > 0 \\ 2\beta e & \text{se} \quad e \leq 0 \end{cases}. \tag{4.6}$$

Usando as Equações 4.4, 4.5 e 4.6, é possível implementar o algoritmo de retropropagação para treinar uma rede MLP com a função custo de média da soma dos erros quadráticos ponderados, a qual passa a ser chamada de rede MLP ponderada (MLPP). Tanto esta configuração como a configuração tradicional são avaliadas como modelos preditores para o problema. Portanto, incluindo o modelo de regressão linear e a SVR, há quatro máquinas de aprendizado em estudo.

4.5 Planejamento dos experimentos

Dado o processo de execução de testes funcionais de um sistema e os quatro modelos preditores escolhidos, a abordagem experimental para o problema consiste em uma sequência de atividades, descritas a seguir, cujo objetivo é responder às questões levantadas na Seção 1.4 do Capítulo 1, e assim chegar a uma metodologia adequada para estimar esforço de execução. Essas atividades com os modelos são feitas sobre duas bases de dados reais que, embora não sejam representativas de todos os sistemas de *software*, podem fornecer indicações para se propor uma forma de estimar o esforço adequadamente.

4.5.1 Coleta de dados

A análise do processo de execução de testes, feita na Seção 4.2, permite identificar quatro dimensões na determinação do esforço de executar um ciclo de testes, ilustradas na Figura 4.4: a complexidade dos CTs; a complexidade do sistema de *software* que está sendo testado; a capacidade

da equipe de testes, que equivale à experiência profissional mais a competência para o trabalho; e a probabilidade de se provocar falhas.

Existem outros fatores que influenciam o esforço de execução dos testes, como o tipo de interface com o usuário, o tipo de arquitetura e de ambiente do *software*, o critério de testes adotado, etc.. Entretanto, decidiu-se considerar exclusivamente as quatro dimensões citadas como os principais determinantes de esforço, tendo em conta que o escopo deste trabalho contempla apenas testes funcionais executados manualmente.

Fig. 4.4: Fatores de influência no esforço de execução de testes.

Para que se forme uma base de dados promissora para criação dos modelos preditores, é recomendado obter métricas que contemplem estas quatro dimensões. Alguns exemplos de métricas a considerar, para cada dimensão, são mostrados na Tabela 4.1.

Valores de variáveis que contemplem estas quatro dimensões devem ser coletados para formar as bases de dados de treinamento dos modelos. Estão disponíveis duas coleções de dados reais: a primeira compreende os dados do projeto HARPIA mais os provenientes de uma empresa especializada em testes, SOFIST, fundada por ex-membros da equipe de testes do HARPIA; a segunda fonte consiste dos dados fornecidos pelo pesquisador Xiaochun Zhu, originários do departamento de TI (Tecnologia da Informação) de uma organização chinesa de serviços financeiros. Passam a ser chamadas, no restante do trabalho, respectivamente, de Base 1 e Base 2.

Naturalmente há restrições quanto à factibilidade de coletar dados do maior número de possível de variáveis relativas a cada dimensão, devido ao impacto que pode ser provocado no processo de teste das organizações fornecedoras dos dados.

Por fim, note que é tratado aqui da predição do esforço de um ciclo de teste como unidade fundamental; por isso as variáveis são medidas com valores referentes a cada ciclo, o qual pode ser de regressão, de teste de novas funcionalidades ou de combinação dos dois casos anteriores. Por exem-

Tab. 4.1: Exemplos de métricas para cada dimensão de influência no esforço.

Complexidade dos CTs	Complexidade do Sistema	Probabilidade de Falhas	Capacidade da Equipe de Testes
1. Número total de CTs	1. Número de Casos de Uso	1. Valor esperado de falhas a encontrar no sistema, com base em algum modelo probabilístico	1. Número de testadores envolvidos
2. Número de passos de CTs	2. Número de linhas de código	2. Estágio de desenvolvimento do sistema: versão alfa, beta, final, etc..	2. Experiência, em número de dias na organização, dos testadores envolvidos
3. Número total de dados de entrada submetidos	3. Número de classes do sistema	3. Experiência na área, em anos, dos testadores envolvidos	3. Uso de ferramentas de suporte ao teste
4. Número total de dados de saída a verificar	4. Número de funções do sistema	4. Porcentual dos testadores da equipe que já testaram o sistema	
5. Classificação de complexidade do CT	5. Número de pontos por Caso de Uso		
	6. Complexidade Ciclomática do código		

plo, um registro da medida da variável "Número total de CTs" é o número de casos de testes a serem executados para o ciclo N do sistema A. Se houver n ciclos disponíveis nos dados históricos, haverá então n valores para a variável.

4.5.2 Seleção de variáveis

Embora a etapa de coleta dos dados realize, com base na análise do processo, uma escolha do maior número de métricas que em princípio possuam relação com o esforço, pode haver informação irrelevante ou redundante entre elas. Assim, é necessário aplicar um procedimento de seleção de variáveis [29] com o propósito de determinar quais são as variáveis que levarão ao melhor desempenho das máquinas de aprendizagem.

Refletindo de forma mais cuidadosa, percebe-se que uma etapa funciona em complemento à outra, conforme ilustra a Figura 4.5. Na atividade de coleta de dados tenta-se obter o maior número de métricas que, de acordo com a análise especializada, tem influência no esforço; porém, é muito provável que entre as métricas escolhidas haja informação irrelevante ou repetida para o modelo preditor, que assim teria um desempenho melhor se essa informação fosse descartada por meio da atividade de seleção de variáveis. A tentativa de realizar uma inspeção humana das melhores métricas já na própria etapa de coleta de dados teria o grande risco de ignorar variáveis que poderiam ser úteis para determinado modelo; afinal, a complexidade matemática dos modelos preditores torna impossível

prever quais informações serão necessárias para treinar bem cada máquina para o problema.

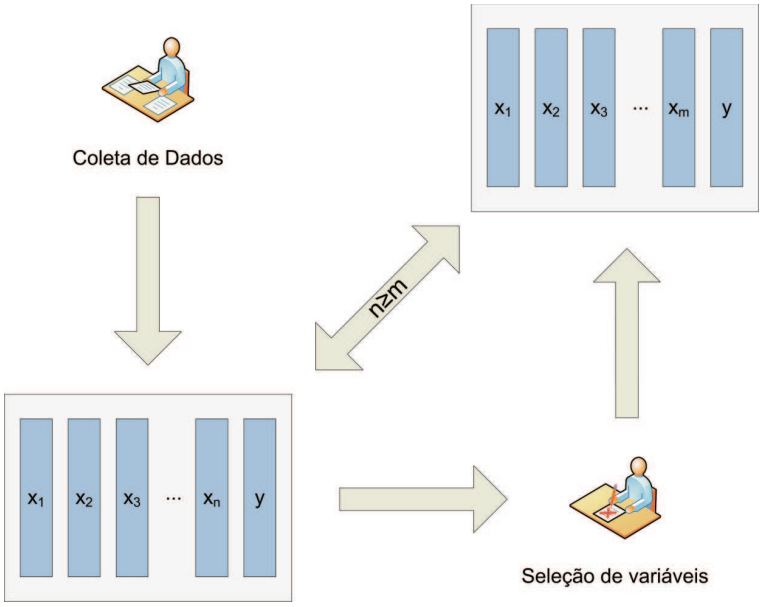

Fig. 4.5: Complementaridade entre as etapas de coleta de dados e seleção de variáveis.

Como foi visto no Capítulo 3, há diversos métodos de seleção de variáveis, sendo os dois mais comuns os métodos por envoltório e o método por filtro. Este experimento, portanto, consiste na aplicação do método por envoltório (implementado conforme a proposta de KOHAVI & JOHN [44]) para cada um dos quatro modelos de máquina de aprendizado avaliados, mais a aplicação do filtro CFS, sobre as duas bases de dados.

A escolha dos métodos foi feita por serem os mais tradicionais na literatura; a escolha visa também avaliar se um método de seleção independente de modelo pode ser tão ou mais competente do que o método dependente do modelo, mesmo que com um número pequeno de dados nas bases (em que normalmente se recomenda o método por envoltório). Ao final da seleção haverá cinco subconjuntos de variáveis escolhidas para cada base de dados, relacionados na Tabela 4.2.

Tab. 4.2: Propostas de seleção de variáveis a serem estudadas.

Método	**Máquina**	**Subconjunto**
Envoltório	SVR	env-SVR
	MLP tradicional	env-MLP
	MLP ponderada	env-MLPP
	Reg. Linear	env-Linear
Filtro CFS	N/A	filtro-CFS

O resultado esperado deste experimento é estabelecer uma comparação entre as soluções obtidas em cada proposta, em termos de variáveis escolhidas e desempenho apresentado.

4.5.3 Comparação de modelos

Dados os subconjuntos de variáveis determinados pelos métodos de seleção, deve-se estabelecer experimentalmente qual configuração "máquina de aprendizagem e subconjunto de variáveis" possui o melhor desempenho. Para cada modelo preditor há duas possibilidades: treiná-lo com as variáveis selecionadas pelo filtro ou com as variáveis selecionadas pelo envoltório. Há então oito configurações distintas para observar.

Para estabelecer o experimento, é preciso levar em conta que o cenário do problema é bastante limitado quanto ao número de amostras disponíveis: há apenas 30 amostras disponíveis na Base 1, enquanto há 70 amostras na Base 2. Tal fato é reflexo da dificuldade de se obter dados em organizações de *software*, principalmente quando se tratam de métricas exclusivas do processo de testes. Há barreiras como restrições de sigilo, ausência de processos bem definidos que viabilizem coleta de dados [80], e dificuldades em termos de intercâmbio de informações no meio acadêmico e deste com o meio industrial. No caso deste trabalho foram contatados mais de 10 pesquisadores com trabalhos publicados sobre estimativa de esforço; somente um contato foi bem sucedido quanto à obtenção de dados.

Corre-se assim o risco de que os dados disponíveis não possuam representatividade suficiente para a obtenção de conclusões gerais a respeito do problema. Esta realidade já faz com que diversas propostas e estudos sobre estimativa de esforço sejam abordagens locais, ou seja, os métodos propostos realizam ajustes e calibragens dos modelos de forma que eles funcionem adequadamente para os dados e a realidade da organização em estudo [43]; nesses casos deve-se considerar como contribuição a metodologia utilizada para se chegar à solução.

Para contornar as limitações do número de dados pequeno e do risco de conclusões indevidas, KIRSOPP & SHEPPERD [43] propõem realizar medidas do experimento repetidas várias vezes. Tomadas estas considerações, o experimento ilustrado de forma geral na Figura 4.6 pode ser descrito pelos seguintes passos:

1. Realizar um particionamento aleatório dos dados, 80% para o treinamento do modelo e 20% para validação.

2. Para o mesmo particionamento, treinar as oito configurações possíveis de modelos e registrar o desempenho no conjunto de validação.

3. Repetir mil vezes os passos um e dois.

Há diversas métricas que podem ser utilizadas como medida de desempenho dos modelos; as duas mais adotadas na literatura são a média do erro relativo absoluto (em inglês, *mean absolute relative error*, MARE) e o porcentual de estimativas dentro de 25% de erro relativo absoluto.

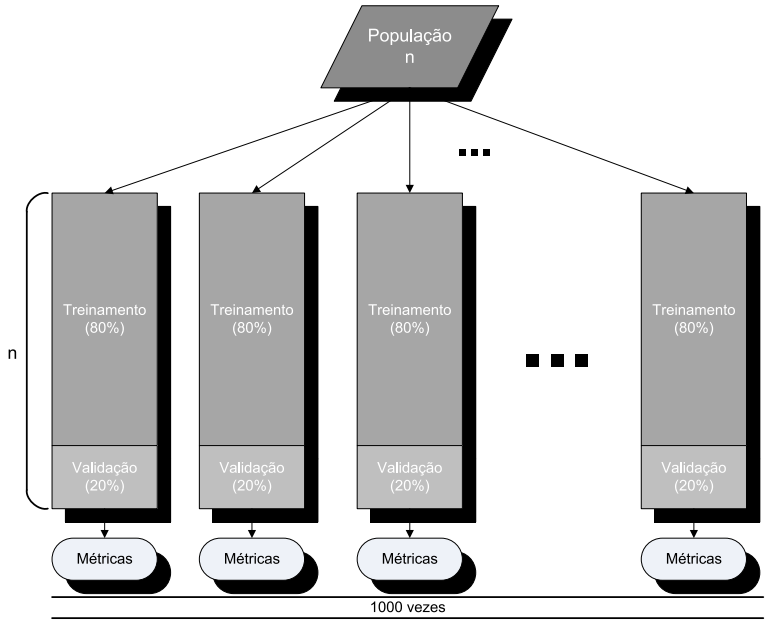

Fig. 4.6: Experimento de comparação dos modelos preditores em uma base de dados.

Definição 4.3. *Média do erro relativo absoluto.*

Seja

$$ARE_i = \left| \frac{d_i - y_i}{d_i} \right|.$$

A média do erro relativo absoluto é dada por

$$MARE = \frac{\sum_{i=1}^{n} ARE_i}{n},$$

onde n é o número de amostras, y_i é a saída estimada pela máquina e d_i é a saída esperada.

Definição 4.4. *Porcentual de estimativas dentro de 25% de erro relativo absoluto.*

Seja a função $h : \mathbb{N}^* \to \{0, 1\}$ definida por

$$h(i) = \begin{cases} 1 & se \quad ARE_i \leq 0{,}25 \\ 0 & \quad caso\ contrário \end{cases}.$$

O porcentual de estimativas dentro de 25% de erro relativo absoluto é dado por

$$PRED(0{,}25) = \frac{\sum_{i=1}^{n} h(i)}{n}.$$

A métrica MARE permite observar o desempenho geral do modelo preditor. Porém, como toda

média, ela é sensível à existência de *outliers*[3], o que pode fazer com que um excelente desempenho da máquina em um teste compense a má atuação nos testes restantes, ou que um fracasso total em um teste oculte bons resultados atingidos nos outros testes [22].

Uma vez que o objetivo prioritário é construir um modelo preditor com bons resultados na maior parte das vezes, torna-se útil a métrica PRED(0,25), porque ela permite atestar se um modelo que obteve um valor MARE baixo teve de fato a maioria de suas respostas dentro da margem de 25% de erro relativo absoluto, configurando um comportamento estável dentro de um intervalo de erro que é aceitável na literatura disponível.

Por exemplo, considere os dados da Tabela 4.3: enquanto a primeira série de dados possui um valor médio menor, a segunda série teve mais amostras que ficaram abaixo do valor de 25%. Dado que mais importante que um erro médio baixo é que o modelo preditor comporte-se assim para o número máximo de cenários, faz-se necessário considerar as duas medidas de desempenho na comparação.

Tab. 4.3: Exemplo de valores para MARE e PRED(0,25).

Valores									**Média**	**PRED(0,25)**
0,04	0,05	0,3	0,35	0,27	0,32	0,28	0,4	0,1	**0,23**	**0,33**
0,23	0,34	0,24	0,19	0,21	0,25	0,32	0,6	0,5	**0,32**	**0,56**

Com o intuito de observar também o comportamento dos modelos quanto à subestimação e à superestimação, principalmente o modelo de rede MLP ponderada, é definida uma terceira métrica de nome PRED(-0,25), que quantifica o número de respostas que superestimaram em até 25% o resultado. Note-se que o sinal negativo deve-se ao cálculo do erro relativo ser feito subtraindo o valor estimado do desejado.

Definição 4.5. *Porcentual de estimativas dentro de -25%.*

Seja

$$RE_i = \frac{d_i - y_i}{d_i},$$

e a função $g : \mathbb{N}^* \to \{0, 1\}$ *definida por*

$$g(i) = \begin{cases} 1 & \textit{se} \quad RE_i \in [-0{,}25, 0] \\ 0 & \textit{caso contrário} \end{cases}.$$

O porcentual de estimativas dentro de -25% é dado por

$$PRED(-0{,}25) = \frac{\sum_{i=1}^{n} g(i)}{n}.$$

[3]Em estatística, um *outlier* é uma observação que está numericamente distante do restante dos dados [4].

Tem-se ao fim deste experimento, para cada base de dados, oito grupos de dados compostos pelas combinações dos quatro algoritmos com as duas possibilidades de subconjuntos de variáveis (seleção por envoltório ou por filtro CFS). Cada grupo contém 1.000 medições das métricas MARE, PRED(0,25) e PRED(-0,25); analisando esses resultados pretende-se encontrar a configuração com o melhor desempenho.

4.5.4 Avaliação de modelos em ambiente real

O experimento anterior permite comparar de forma geral como se comporta cada configuração de máquina de aprendizado. Agora, o objetivo é avaliar o modelo escolhido supondo que ele é a ferramenta de estimativa da organização de testes, por meio de um experimento de validação cruzada. Para isso, os seguintes passos devem ser seguidos, para cada base de dados:

1. Dividir os dados em 70% para o conjunto de treinamento e 30% para o conjunto de testes, na ordem cronológica em que os ciclos ocorreram.

2. Treinar o modelo preditor com o conjunto de treinamento. Caso seja necessário um conjunto de validação (caso das redes MLP comum e ponderada), o conjunto de treinamento é novamente dividido em 75% para o treinamento propriamente dito, e 25% para o conjunto de validação utilizado pelo critério de parada.

3. Estimar o esforço por meio do modelo treinado, dadas as entradas de cada amostra do conjunto de teste.

4. Comparar as estimativas com os valores reais de esforço.

Definiu-se a divisão dos dados em 70% e 30% por duas razões: (1) é uma das divisões comumente utilizadas para validação cruzada, (2) esta divisão resulta em um número de ciclos (21 ciclos) da Base 1 disponíveis para treinamento que é aceitável, do ponto de vista de requisito mínimo de amostras para uma organização treinar um modelo de aprendizado de máquina. A segunda divisão em 75% e 25%, para quando se necessita de um conjunto de validação, também foi escolhida para tentar manter um número aceitável de ciclos da Base 1 para treinamento.

Devido à já discutida variabilidade dos resultados de treinamento de uma MLP de acordo com o valor inicial dos pesos, o procedimento de treinamento (passo 2) é repetido dez vezes tanto para o caso da rede MLP comum quanto para a MLP ponderada, e escolhe-se como modelo treinado aquele que teve melhor desempenho em termos de erro para o conjunto de validação.

Os resultados são também comparados com a proposta feita durante os estudos preliminares do problema de esforço [77]. A avaliação dos modelos visa a determinar qual modelo preditor é o mais adequado, se o modelo possui um desempenho bom para o problema, e como ele deve ser utilizado.

4.6 Síntese

Neste capítulo, faz-se uma análise do esforço de execução de testes de *software*, iniciando por meio do estudo de trabalhos anteriores sobre estimação de esforço tanto do processo completo de desenvolvimento como do processo de teste. Nota-se que há poucos trabalhos que utilizam aprendizado de máquina especificamente para estimar esforço em testes.

Desenvolveu-se uma análise sobre o processo de execução de testes funcionais, chegando à proposta de quatro dimensões de influência sobre o esforço. Foi examinado também o impacto que as estimativas, sejam acima ou abaixo do real, possuem sobre o processo de teste e consequentemente sobre a equipe, chegando-se à conclusão de que superestimativas são preferíveis a subestimativas; por isso, propõe-se o uso de uma função custo assimétrica para treinar uma MLP que estime o esforço de execução de testes.

Por fim apresentou-se o planejamento dos experimentos, que são divididos em quatro etapas: (1) coleta de dados, (2) seleção de variáveis, (3) comparação de modelos, e (4) avaliação de modelos em ambiente real.

O capítulo seguinte descreve os resultados obtidos nos experimentos com as duas bases de dados reais disponíveis. São apresentados também os procedimentos de ajuste dos parâmetros dos modelos e se discutem as implementações dos algoritmos.

Capítulo 5

Resultados experimentais

Este capítulo apresenta os resultados dos experimentos conforme o planejamento exposto no Capítulo 4. São discutidas as implementações dos algoritmos de aprendizado de máquina e de seleção de variáveis, e são apresentados os resultados experimentais com a Base 1 e com a Base 2. São também discutidos os ajustes de parâmetros dos modelos e cada resultado obtido.

5.1 Implementações dos algoritmos

Todos os algoritmos desta dissertação, exceto o filtro CFS, foram implementados na linguagem de programação do ambiente MATLAB©, Versão 7.5. A escolha deste ambiente de desenvolvimento se deu pela facilidade de visualização dos resultados bem como de uso de operações matemáticas. Os algoritmos foram executados em um microcomputador com processador Core 2 Duo© de frequência 1,73GHz, com 2GB de memória RAM e sistema operacional Windows Vista©. Nas próximas subseções as implementações feitas são descritas.

5.1.1 Algoritmos de aprendizado de máquina

Regressão Linear

O procedimento para realizar a regressão linear múltipla por mínimos quadrados é simples e calculado de forma analítica pela solução de um sistema linear via pseudo-inversão de matrizes. Há rotinas prontas no MATLAB© que foram utilizadas para o modelo.

Regressão por Vetor de Suporte

Como indicado na Seção 2.3.6 do Capítulo 2, existem diversos algoritmos de treinamento para máquinas de vetor de suporte. Para este trabalho utiliza-se o modelo de regressão do tipo ϵ-SV pertencente ao *toolbox* LIBSVM [15] para MATLAB©, cujo algoritmo de treinamento é o SMO para regressão.

Rede MLP comum

O treinamento de uma rede MLP, equivalente a solucionar um problema de otimização não-linear irrestrito, pode ser abordado por métodos de primeira ordem, ou seja, que utilizam o gradiente da função custo, ou por métodos de segunda ordem, que além do gradiente consideram a matriz hessiana para cálculo do passo otimizante [33].

Foi implementada a proposta do método de otimização LEVENBERG-MARQUARDT, de segunda ordem, incorporado ao algoritmo de retropropagação [30], porque os métodos de segunda ordem tem uma velocidade de convergência maior do que os métodos de primeira ordem. Embora o MATLAB© possua uma *toolbox* própria com este algoritmo de treinamento, além de diversos outros, escolheu-se realizar uma implementação própria para ter maior controle das medições durante o treinamento e para uso de métricas de desempenho não disponíveis por padrão na *toolbox*.

O critério de parada para o treinamento da rede é executar o algoritmo até atingir um número limite de épocas, e então retornar ao conjunto de pesos pertencentes à época de treinamento com o menor erro no conjunto de validação. Dado que não é a prioridade deste trabalho avaliar o tempo de treinamento dos modelos nos experimentos, este critério não prejudica as avaliações e enseja maior probabilidade do algoritmo convergir para um mínimo global.

Rede MLP ponderada

Embora um método de segunda ordem seja preferível, devido à sua maior velocidade de convergência, foi implementado o algoritmo de retropropagação generalizado, proposto por CRONE [19], para treinar a rede MLP com a função custo apresentada no Capítulo 4. Como já enfatizado, uma vez que os modelos são comparados em termos de desempenho do erro e não de velocidade de treinamento, tal decisão não afeta os resultados.

De toda forma, para melhorar a velocidade de convergência do algoritmo, incluiu-se o procedimento de busca unidimensional simples para garantir um passo minimizante a cada época de treinamento. A autoria da rotina é do professor Leandro Nunes de Castro, da Universidade Presbiteriana Mackenzie [13]. O critério de parada é idêntico ao utilizado para a rede MLP comum.

Não foram encontradas referências na literatura que quantificassem o impacto de uma subestimativa *versus* uma superestimativa, de forma que tal dado pudesse ser utilizado na determinação das constantes α e β da função custo ponderada (Definição 4.1). Assim, é proposta a utilização dos valores $\alpha = 1{,}5$ e $\beta = 1$ de forma que o valor subestimado tenha um peso 50% maior que o valor superestimado.

5.1.2 Algoritmos de seleção de variáveis

Para aplicar o filtro CFS aos dados, é utilizada a implementação disponível através da ferramenta WEKA [96]. No caso do método por envoltório, rotinas em MATLAB© foram implementadas para cada uma das quatro configurações de máquinas de aprendizado em estudo. Todas se baseiam na proposta de KOHAVI & JOHN [44], e diferem somente por qual modelo preditor chamam para avaliar cada subconjunto candidato de variáveis.

5.2 Resultados com a Base 1

5.2.1 Dados coletados

A Base 1 compreende os dados de 25 ciclos de testes executados no projeto HARPIA entre agosto de 2007 e novembro de 2008, de seis sistemas distintos de *software*, mais os dados de 5 ciclos de testes de um sétimo sistema de *software*, fornecidos por uma empresa especializada em testes fundada por ex-membros do projeto HARPIA. São dados de 30 ciclos de testes de 7 sistemas de *software*, abrangendo 3.017 execuções de casos de teste.

A consideração dos dados de ambas as organizações em uma só base é feita porque as equipes de testes seguem o mesmo processo, os engenheiros de testes da empresa haviam participado anteriormente do projeto HARPIA, e mesmas métricas são coletadas.

A Tabela 5.1 apresenta informações sobre os sistemas que fazem parte da base de dados, onde "UCs" indica o número de casos de uso, e "KLOCs" indica milhares de linhas de código. Os sistemas de número 1 a 5 são de plataforma *Web*, enquanto o sistema de número 6 tem plataforma *desktop*. Estes seis sistemas foram desenvolvidos com a linguagem Java e foram criados com o propósito de automatização e controle de diversos processos de negócios da Receita Federal do Brasil. O sistema de número 7 também é de plataforma *Web*, desenvolvido em Java, e provê automatização de processos de gestão para prefeituras.

Três critérios foram adotados para escolher as variáveis a serem coletadas:

1. Usar como referência uma das quatro dimensões de influência sobre o esforço apresentadas na Seção 4.5.1 do Capítulo 4: (1) a complexidade dos CTs, (2) a complexidade do sistema de

Tab. 5.1: Dados dos sistemas de *software*.

	SW #1	**SW #2**	**SW #3**	**SW #4**	**SW #5**	**SW #6**	**SW #7**
UCs	50	24	17	34	60	19	91
KLOCs	44	30	43	84	63	144	291

software que está sendo testado, (3) a capacidade da equipe de testes, e (4) a probabilidade de se provocar falhas.

2. Não escolher variáveis subjetivas[1].

3. Tentar aproveitar os dados coletados de acordo com o processo adotado pelas equipes.

O primeiro critério é considerado para que as variáveis possuam relação com a análise do esforço no processo de execução de testes, feito no Capítulo 4. Os outros dois critérios visam a facilitar o processo de coleta dos dados por parte das equipes.

A Tabela 5.2 mostra as variáveis definidas e que estão disponíveis na base junto com o valor de esforço, em pessoas-hora, realizado na execução do ciclo. Ela apresenta também o índice adotado para referência, e a dimensão de influência sobre o esforço que se deseja considerar para cada variável.

Tab. 5.2: Variáveis da Base 1 e mapeamento com as dimensões de influência.

Variável	**Dimensão mapeada**
1. Número de CTs total	Complexidade dos CTs
2. Número de passos de CTs total	
3. Número de CTs a executar	
4. Número de passos de CTs a executar	
5. Número de CTs a re-executar	
6. Número de passos de CTs a re-executar	
7. Número de testadores envolvidos	Capacidade da Equipe de Testes
8. Experiência, em número de dias no projeto, dos testadores envolvidos	
9. Número de Casos de Uso do sistema	Complexidade do Sistema
10. Número de linhas de código do sistema	
11. Número de classes do sistema	

Enquanto a variável 1 contempla a totalidade do número de CTs, as variáveis 3 e 5 separam os valores em CTs que são executados pela primeira vez e CTs que já foram executados em ciclos de testes passados, respectivamente. Analogamente, as variáveis 4 e 6 são assim estabelecidas em relação à variável 2; a razão para tal separação é a hipótese de que, tendo por base a experiência das equipes, um caso de teste realizado pela primeira vez toma mais tempo do que aquele já executado

[1]Uma variável subjetiva depende de avaliações e notas dadas por especialistas.

em situações passadas, ainda mais caso ele esteja sob a responsabilidade do mesmo testador. Se tal hipótese for de fato plausível, provavelmente as variáveis discriminadas serão relevantes para a determinação do esforço, e, assim, devem aparecer como futuro resultado da etapa de seleção de variáveis.

As variáveis 7 e 8 têm o propósito de contemplar informações a respeito do preparo e experiência da equipe na execução dos testes. Em especial, no caso da variável 8, diversas formas de medir a experiência dos testadores poderiam ser levantadas, mas resolveu-se usar o número de dias de cada testador no projeto porque nenhum testador possuía outra experiência profissional anterior com testes: assim, o tempo de trabalho desde o ingresso no projeto HARPIA pode ser um indicador bom e objetivo do grau de experiência. Para o caso de outras organizações ainda se pode considerar a mesma métrica, desde que o cálculo dos dias considere toda a experiência profissional.

As variáveis 9, 10 e 11 são informações do tamanho do sistema de *software* e assim podem estar relacionadas com a complexidade para testá-lo. Existem diversas outras métricas de complexidade de *software*, mas a adoção de qualquer uma aumentaria o esforço na coleta dos dados e prejudicaria a manutenção da simplicidade neste processo, o que não atende aos critérios estabelecidos.

Não foi possível adotar uma variável que pudesse indicar a probabilidade de falhas a ocorrer no sistema sem que os critérios fossem quebrados ou que fosse necessário se realizasse um estudo muito mais aprofundado; por isso tal dimensão foi desconsiderada.

Construída a base de dados, as variáveis de entrada foram transformadas para terem média 0 e desvio padrão igual a 1, com o objetivo de melhorar o desempenho dos algoritmos de treinamento dos modelos, principalmente das redes neurais [33]. É possível também realizar uma análise estatística básica na variável que é objeto de estimativa: o esforço realizado na execução do ciclo de teste. A Figura 5.1 apresenta o diagrama *box-plot*, em que se detecta a ausência de *outliers*, embora as medidas estejam bem distribuídas no quartil superior, o que reflete na mediana de 33,12 pessoas-horas, média de 41,85 pessoas-horas e o desvio padrão de 27,07 pessoas-horas.

5.2.2 Seleção das variáveis

Tendo em mãos uma base de apenas 30 amostras e espaço de entrada de dimensão 11, é vital garantir que as variáveis desse conjunto não sejam redundantes ou irrelevantes e assim se possa melhorar o desempenho das máquinas de aprendizado. A seguir é detalhada a aplicação dos métodos de envoltório e filtro para seleção das variáveis, incluindo o ajuste de parâmetros para cada modelo de aprendizado de máquina.

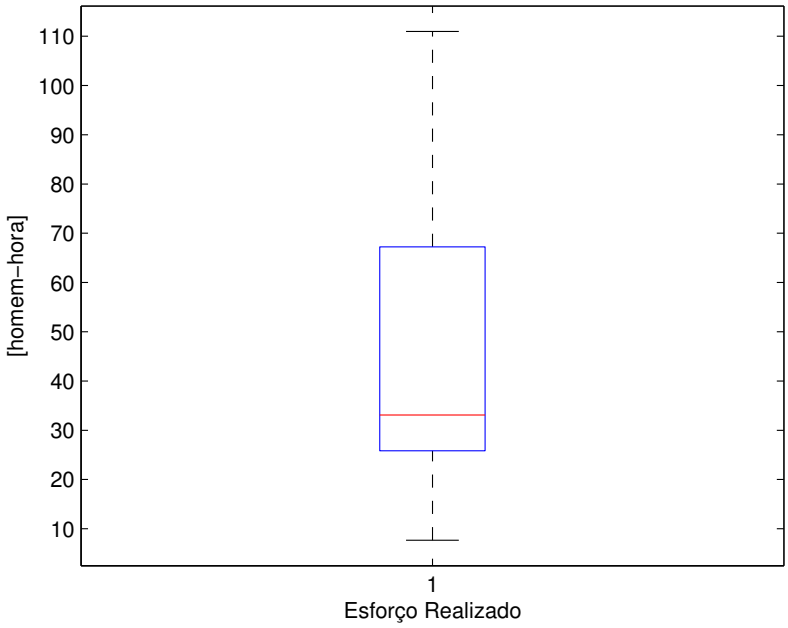

Fig. 5.1: Diagrama *box-plot* da variável esforço, para a Base 1.

Filtro CFS

O método de filtro CFS é utilizado diretamente por meio da ferramenta WEKA; foi escolhida a estratégia exaustiva de busca, pois o reduzido número de variáveis e amostras, aliado à eficiência inerente ao algoritmo, permitem que o processo se encerre em um tempo pequeno, da ordem de segundos.

A Tabela 5.3 apresenta o subconjunto de variáveis escolhido pelo algoritmo, após a avaliação de 2.048 subconjuntos candidatos. São 7 variáveis escolhidas entre as 11 iniciais, representando uma redução de aproximadamente 37% na dimensão do problema original.

Tab. 5.3: Variáveis selecionadas pelo método de filtro.

Índice	**Nome**
1	Número de CTs total
2	Número de Passos de CTs total
3	Número de CTs a executar
5	Número de CTs a re-executar
7	Número de Testadores
8	Experiência dos testadores [dias]
11	Número de classes

O subconjunto resultante contém variáveis das 3 dimensões de influência consideradas quando os

dados foram coletados, o que dá respaldo ao método utilizado para determinar as variáveis da base. Da dimensão de complexidade dos CTs foram excluídas as variáveis "Número de passos de CTs a executar" e "Número de passos de CTs a re-executar"; da dimensão de capacidade da equipe de testes, nenhuma variável foi excluída; e, finalmente, as variáveis "Número de Casos de Uso do sistema" e "Número de linhas de código do sistema", da dimensão de complexidade do sistema, foram excluídas.

Envoltório com SVR

O modelo SVR do tipo ϵ-SV e com núcleo RBF possui os seguintes parâmetros que devem ser definidos antes do seu treinamento: a constante de regularização C; a constante ϵ da função $|\xi|_\epsilon$, que determina o grau de tolerância a erros de aproximação; e a dispersão σ da função de base radial.

Para escolher o valor mais apropriado de cada parâmetro, é seguido o seguinte procedimento empírico, que é replicável para outros modelos:

1. Fixar todos os parâmetros, exceto aquele que se deseja definir.
2. Realizar execuções do algoritmo de envoltório, variando a cada vez o parâmetro em observação.
3. Comparar os valores de erro médio obtidos, bem como os seus desvios padrão.
4. Escolher o valor de parâmetro cuja execução teve menor erro médio (melhor desempenho) como primeiro critério, e menor desvio padrão como segundo critério. Caso haja dúvida devido à proximidade de valores, escolher pelo menor dos valores do produto entre o erro médio e o respectivo desvio padrão.

O procedimento é relativamente simples. Alternativamente poderia ser feito um procedimento de busca pelas combinações possíveis dos parâmetros, abordagem que seria muito mais custosa e questionável do ponto de vista prático, considerando que se deseja uma atividade de seleção de variáveis que seja viável para uso por equipes de testes.

O desvio padrão é considerado porque indica o grau de variação na medição do erro entre as partições observadas como conjunto de testes. O objetivo é garantir que os parâmetros escolhidos sejam os que dão o melhor desempenho, mas que tenham também um comportamento regular. Vale registrar que o número de iterações da validação cruzada ficou determinado para 1, porque, como já explicado, o treinamento de uma máquina de vetor de suporte é um problema de otimização que converge sempre para o único mínimo global. A repetição do treinamento da MVS com os mesmos dados resultaria nos mesmos resultados sempre, sendo, portanto, desnecessária.

A primeira série de execuções do algoritmo de envoltório foi feita com os valores $\epsilon = 3$, $\sigma = 0{,}1$, para escolher um valor adequado da constante C. A Tabela 5.4 apresenta os resultados obtidos, com os valores de média do MSE, desvio padrão e também as variáveis selecionadas.

Tab. 5.4: Variação do parâmetro C na execução do envoltório-SVR.

C	**1**	**10**	**100**	**1000**
Subconjunto escolhido	{2,4,1}	{2,1,7}	{2,7,10,9,11}	{1,3,8,10}
Média do MSE	681,11	303,73	**167,82**	309,28
Desvio padrão do MSE	488,06	189,52	**51,65**	173,04
Desvio Padrão / Média	72%	62%	**31%**	56%

A melhor configuração pelos critérios estabelecidos é obtida com o valor $C = 100$. Considerando este como o valor final de C e ainda utilizando $\sigma = 0{,}1$, outra rodada de execuções do algoritmo é feita, agora variando o valor da constante ϵ. A Tabela 5.5 apresenta os resultados alcançados, onde há um valor muito próximo na média de MSE para as configurações com $\epsilon = 2$, $\epsilon = 3$ e $\epsilon = 4$, assim a decisão final é pelo valor que tenha um menor desvio padrão relativo, ou seja, $\epsilon = 4$.

Tab. 5.5: Variação do parâmetro ϵ na execução do envoltório-SVR.

ϵ	**1**	**2**	**3**	**4**	**8**
Subconjunto escolhido	{2,7}	{2,7,10,9,11}	{2,7,10,9,11}	{2,7,9,11,10}	{2,7,9,11,10}
Média do MSE	169,21	167,44	167,82	**167,72**	181,04
Desvio padrão do MSE	88,12	61,62	51,65	**50,09**	81,83
Desvio Padrão / Média	52%	37%	31%	**30%**	45%

Acertados os valores de C e ϵ, resta estabelecer um valor adequado para σ. Após outra rodada de execuções variando os valores deste parâmetro, chega-se aos resultados apresentados na Tabela 5.6. A configuração de melhor desempenho é para $\sigma = 0{,}1$.

Tab. 5.6: Variação do parâmetro σ na execução do envoltório-SVR.

σ	**0,01**	**0,1**	**1**	**10**
Subconjunto escolhido	{2,7,1,9,11}	{2,7,9,11,10}	{1,4,5}	{2,10}
Média do MSE	173,93	**167,72**	371,89	285,51
Desvio padrão do MSE	72,81	**50,09**	310,44	310,79
Desvio Padrão / Média	42%	**30%**	83%	109%

Com os parâmetros definidos ($C = 100$, $\epsilon = 4$, $\sigma = 0{,}1$) foi realizada a execução definitiva do algoritmo de envoltório para determinar o subconjunto ideal de variáveis. A operação durou aproximadamente 0,4s e o subconjunto selecionado pode ser observado na Tabela 5.7. O valor médio do MSE para a solução encontrada é de 167,72, enquanto seu desvio padrão é 50,09, significando um valor relativo de aproximadamente 30%.

A solução encontrada apresenta uma redução de dimensão dos dados de entrada mais agressiva do que a promovida pelo filtro CFS: são 5 variáveis escolhidas dentre as 11 iniciais, reduzindo em 55% o tamanho inicial. As três dimensões ainda permanecem com variáveis representantes, porém

houve uma clara preferência por considerar todas as variáveis de complexidade do sistema (9, 10 e 11), enquanto apenas a variável 2 (Número de Passos de CTs total) caracteriza os casos de testes, e a única informação dos testadores é dada pelo número deles no ciclo (variável 7).

Tab. 5.7: Variáveis selecionadas pelo método de envoltório-SVR.

Índice	Nome
2	Número de Passos de CTs total
7	Número de Testadores
9	Número de Casos de Uso do sistema
10	Número de linhas de código do sistema
11	Número de classes

A Figura 5.2 apresenta o gráfico do comportamento dos melhores candidatos à solução (similar ao do exemplo na Figura 3.6 do Capítulo 3), à medida que a busca por seleção progressiva ocorria dentro do algoritmo. O ponto de mínimo da curva ocorre justamente no subconjunto de 5 variáveis escolhido pelo método.

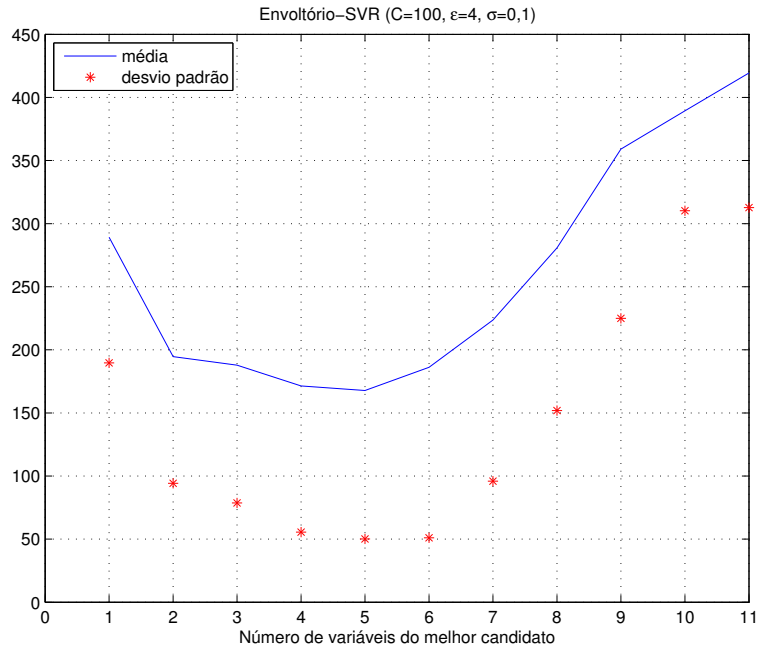

Fig. 5.2: Comportamento dos candidatos a solução para o envoltório com SVR, na Base 1.

Envoltório com Regressão Linear

O procedimento de regressão linear múltipla não requer ajuste de parâmetros; portanto, é possível partir diretamente para a execução do método por envoltório associado a este modelo preditor. O

número de iterações da validação cruzada em 5 partições também permanece como 1. A Tabela 5.8 mostra a solução do algoritmo, cujo tempo de execução foi de 2,1s e teve para o subconjunto de variáveis selecionado média do MSE igual a 171,59 e desvio padrão de 67,14 (39% relativo).

Tab. 5.8: Variáveis selecionadas pelo método por envoltório - Regressão Linear.

Índice	**Nome**
1	Número de CTs total
2	Número de Passos de CTs total
7	Número de Testadores

Novamente há uma redução grande na dimensão da entrada, com somente 3 variáveis das 11 iniciais, em uma redução de aproximadamente 73%. Nota-se que o algoritmo não escolheu variáveis pertencentes à dimensão da complexidade do sistema, e desconsiderou totalmente as variáveis que separam os casos de testes entre aqueles a executar pela primeira vez e aqueles a re-executar. A exclusão da maioria das variáveis pode indicar que elas guardam pouca relação com a variável de esforço, por um mapeamento linear. A solução assim se distancia da apresentada tanto pelo filtro como pelo envoltório-SVR, o que não surpreende se for considerada a diferença de concepção matemática das máquinas; no caso da regressão linear e da regressão por vetor de suporte, o primeiro é linear e o é segundo não-linear.

A Figura 5.3 apresenta o comportamento dos melhores candidatos à solução à medida que a seleção progressiva ocorria; desta vez o ponto de mínimo da curva está no subconjunto das 3 variáveis selecionadas.

Envoltório com rede MLP comum

Uma rede MLP, treinada com o algoritmo de segunda ordem LEVENBERG-MARQUARDT, possui um número maior de parâmetros a serem definidos, em comparação com uma máquina de vetor de suporte. É preciso definir as seguintes configurações:

1. Arquitetura da rede: número de camadas ocultas e número de neurônios em cada uma.
2. Número máximo de épocas de treinamento.
3. Coeficiente μ, que mantém a matriz de ajuste do algoritmo definida-positiva [30].
4. Coeficiente β de multiplicação/divisão da constante μ.

Pode-se aplicar o mesmo procedimento adotado para o envoltório-SVR para determinar todos estes parâmetros; entretanto, a complexidade seria muito maior, dado o maior número de combinações.

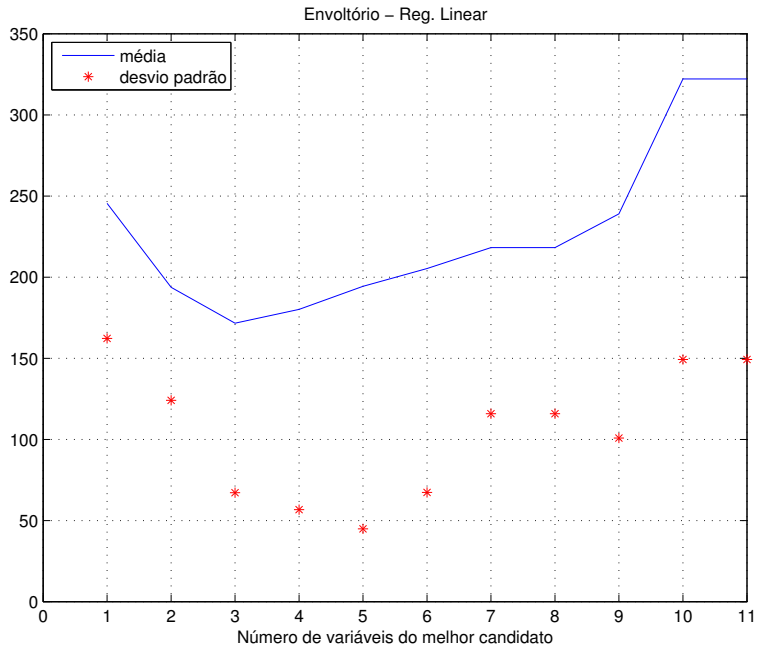

Fig. 5.3: Comportamento dos candidatos a solução para o envoltório com Regressão Linear, na Base 1.

Por isso, baseado em execuções de teste do algoritmo de treinamento, é estabelecido de antemão os valores $\mu = 0{,}001$, $\beta = 0{,}1$ e 200 para o limite de épocas de treinamento, os quais permanecem fixos para todas as execuções do treinamento da rede MLP comum. Resta assim aplicar o procedimento somente para determinar o número de camadas ocultas e o número de neurônios em cada camada.

Embora o teorema de aproximação universal, mostrado na Seção 2.2.4 do Capítulo 2, diga que é suficiente uma única camada oculta para aproximar uma função qualquer, na prática não é possível conhecer a função que se deseja aproximar e nem ter uma camada oculta com número ilimitado de neurônios, premissas do teorema. Em MLPs de camada oculta única, os neurônios nesta camada tendem a interagir entre si globalmente e, em situações complexas, o ajuste da aproximação em um ponto pode piorar outro ponto devido a este tipo de interação. Por isso uma rede MLP com duas camadas ocultas torna o processo de aproximação mais gerenciável, permitindo que as características locais seja extraídas na primeira camada oculta, enquanto as características globais ficam para extração na segunda camada [33].

Com base nestas considerações, a configuração de rede MLP com uma camada oculta e um neurônio linear na camada de saída é comparada com a configuração de duas camadas ocultas e um neurônio linear na camada de saída, na qual a segunda camada oculta possui sempre 4 neurônios. Procedimento análogo ao usado para definir os parâmetros do envoltório com o modelo SVR é adotado para

se determinar o número de neurônios na primeira camada oculta.

O número de iterações nesta atividade preliminar é definido em 20 uma vez que, diferentemente da SVR e da Regressão Linear, o treinamento de uma MLP converge para soluções distintas de acordo com sua iniciação. Desta forma, agora tem-se como valor de comparação a média de 20 repetições da medida de erro e o correspondente desvio padrão. Os dados foram divididos somente em conjuntos de treinamento e de validação, sendo que a medida de erro[2] é feita sobre o último conjunto; não se considerou incluir um conjunto de testes devido ao tamanho reduzido das bases.

Vale reforçar que se continua procurando o ajuste de melhor desempenho e mais regular, mas como o desvio padrão agora é em relação às repetições da validação cruzada, o que é diferente do que ocorreu com o envoltório-SVR, o ajuste que possui menor sensibilidade a variações no conjunto de pesos sinápticos iniciais terá menor desvio padrão.

A Tabela 5.9 apresenta o resultado da rodada preliminar de execuções do algoritmo de envoltório. Percebe-se uma diversidade de valores, tanto de média como de desvio padrão, e de subconjuntos selecionados. Ao se considerar os critérios pré-estabelecidos, a melhor configuração é a de uma camada oculta, com 20 neurônios.

Tab. 5.9: Ajuste de neurônios na primeira camada oculta, na execução do envoltório-MLP.

	Duas camadas ocultas				Uma camada oculta			
# Neurônios	**20**	**16**	**12**	**8**	**20**	**16**	**12**	**8**
Tempo gasto[min]	207	137	96	69	87	76	70	58
Subconjunto escolhido	{2,1,3}	{1,2,4,10}	{2}	{2,1}	{2,10}	{2,7}	{2,7}	{2,7}
Média do erro	273,54	257,88	291,03	265,95	**169,83**	193,21	184,21	185,42
Desvio padrão do erro	63,65	59,57	135,89	89,87	**45,35**	41,29	47,46	44,91
Desvio Padrão / Média	23%	23%	47%	34%	**27%**	21%	26%	24%

Ainda observando a Tabela 5.9, o tempo para execução de cada configuração candidata é muito superior ao gasto para os algoritmos de envoltório que utilizaram SVR e regressão linear, bem como o filtro CFS. O tempo variou de 58 minutos a até 207 minutos, para o caso de uma rede com duas camadas ocultas, com 20 neurônios na primeira camada oculta. As razões para esta diferença são adiante discutidas, ao apresentar o resultado final do algoritmo.

Estabelecidos os parâmetros e a arquitetura da rede MLP para sua execução, passa-se à execução definitiva do algoritmo de envoltório. Para prevenir mais variabilidade no resultado aumenta-se para 40 o número de iterações da validação cruzada. A Tabela 5.10 apresenta as variáveis selecionadas pelo método, cujo tempo de execução foi de 181 minutos, o valor médio de erro para o subconjunto escolhido foi de 182,99 e o desvio padrão de 40,60 (22% relativo).

[2] A medida de erro continua sendo a média dos valores de MSE obtidos com cada partição usada como conjunto de validação, na validação cruzada em 5 partições.

Tab. 5.10: Variáveis selecionadas pelo método por envoltório-MLP.

Índice	Nome
2	Número de Passos de CTs total
10	Número de linhas de código do sistema

Apenas duas variáveis compõem a seleção do algoritmo: a primeira, "Número de Passsos de CTs total", contempla a dimensão de complexidade dos CTs; a segunda, "Número de linhas de código do sistema", contempla a dimensão de complexidade do sistema. A dimensão referente à capacidade da equipe de testes foi excluída.

O tempo de execução do algoritmo é muito maior do que os tempos de execução dos outros métodos de seleção utilizados até então. Enquanto o método de filtro CFS e os métodos por envoltório com SVR e regressão linear gastaram segundos para terminar, o método por envoltório com a MLP gastou aproximadamente 3 horas para a mesma tarefa. Esta grande disparidade não é surpreendente porque o número de treinamentos da rede MLP para avaliar um único subconjunto é bem maior que o número equivalente nos outros dois algoritmos de envoltório e no filtro, que realiza uma simples avaliação por subconjunto.

Como é necessário repetir 40 vezes a validação cruzada na rede MLP, e cada validação cruzada compreende 5 treinamentos da rede com partições distintas sendo usadas como conjunto de teste, isto significa ter no total 200 treinamentos da rede MLP para atribuir uma avaliação a um subconjunto candidato. Para o caso do envoltório-SVR e envoltório com Regressão Linear, não há necessidade de repetição, por isso o número cai para cinco.

A Figura 5.4 apresenta o desempenho dos subconjuntos candidatos durante a execução da busca. A partir do candidato de 2 variáveis, escolhido como solução, nota-se que a curva de erro permanece em crescimento até o final da busca.

Envoltório com rede MLP ponderada

O treinamento da rede MLP ponderada com o algoritmo de retropropagação generalizado requer a definição prévia dos parâmetros de taxa de aprendizagem inicial (α), número máximo de épocas e número de neurônios nas camadas ocultas. De maneira similar à abordagem adotada para a rede MLP comum, são estabelecidos os valores $\alpha = 0{,}01$, limite de 200 épocas de treinamento e passa-se à determinação do uso de duas ou uma camada oculta, bem como o número de neurônios na primeira camada oculta. A segunda camada oculta, caso exista, permanece sempre com quatro neurônios e a camada de saída consiste de um neurônio linear.

Também é utilizado o mesmo número de repetições da validação cruzada (20) e a mesma divisão em conjunto de treinamento e conjunto de validação. Uma mudança importante em relação ao proce-

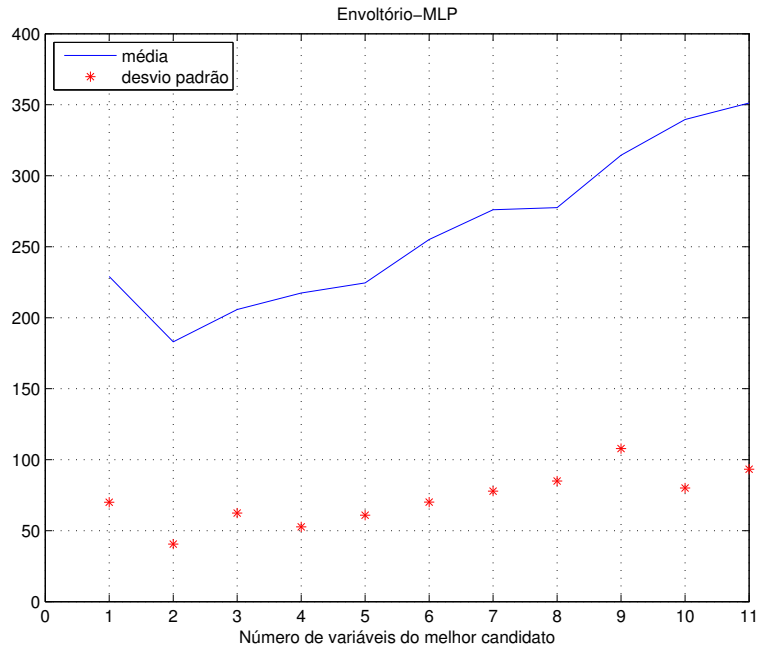

Fig. 5.4: Comportamento dos candidatos a solução para o envoltório com MLP, na Base 1.

dimento feito na rede MLP comum é que o valor calculado para cada avaliação de um subconjunto de variáveis é a média das 20 repetições da média do erro quadrático médio *ponderado* (Definição 4.2 do Capítulo 4) nas cinco partições de teste.

A Tabela 5.11 apresenta os resultados das execuções preliminares do algoritmo de envoltório. Coincidentemente, a arquitetura vencedora para a rede MLP comum foi a mesma escolhida para a rede MLPP: uma camada oculta com 20 neurônios. Nota-se também que a variação nos tempos gastos em cada execução foi menor, 42 a 63 minutos.

Tab. 5.11: Ajuste de neurônios na primeira camada oculta, na execução do envoltório-MLPP.

	Duas camadas ocultas				Uma camada oculta			
# Neurônios	**20**	**16**	**12**	**8**	**20**	**16**	**12**	**8**
Tempo gasto[min]	63	56	52	50	50	46	44	42
Subconjunto escolhido	{2,7,4,1}	{2,7,4}	{2,7,4}	{2,7,4}	{2,7,9}	{2,7,9}	{1,2,7,10}	{2,7,9}
Média do erro	297,61	292,47	273,90	254,95	**147,87**	155,35	173,78	187,98
Desvio padrão do erro	33,74	40,05	27,14	16,76	**18,81**	25,72	20,67	21,02
Desvio Padrão / Média	11%	14%	10%	7%	**13%**	17%	12%	11%

Com a arquitetura estabelecida e aumentado o número de repetições da validação cruzada para 40, realiza-se a execução final do algoritmo de envoltório-MLPP, tendo como resultado o subconjunto de variáveis apresentado na Tabela 5.12. O tempo de execução foi de 99 minutos, a média do erro foi

138,37 e o desvio padrão 19,37 (14% relativo).

O subconjunto escolhido possui apenas três variáveis, uma referente a cada dimensão mapeada: a variável 2 da dimensão de complexidade dos CTs, a variável 7 da dimensão de capacidade da equipe de testes, e a variável 9 da dimensão de complexidade do sistema. A redução do tamanho do problema, em relação ao conjunto inicial, é de aproximadamente 73%.

Tab. 5.12: Variáveis selecionadas pelo método por envoltório-MLPP.

Índice	Nome
2	Número de Passos de CTs total
7	Número de testadores envolvidos
9	Número de Casos de Uso do sistema

Embora o algoritmo de treinamento seja de primeira ordem, o tempo de execução do envoltório-MLPP foi menor do que o do envoltório-MLP, cujo algoritmo de treinamento é de segunda ordem. Não se pretende aqui realizar uma comparação formal de complexidade de cada algoritmo, mas pode-se levantar a hipótese de que isto ocorreu porque, embora algoritmos de segunda ordem necessitem de um menor número de épocas do que algoritmos de primeira ordem para convergir para um ponto de ótimo (global ou local), o número de operações realizadas em cada época pode ser maior na implementação do método de LEVENBERG-MARQUARDT do que no método de retropropagação.

O que reforça esta possibilidade é que na implementação do método de LEVENBERG - MARQUARDT há mais operações que utilizam estruturas condicionais e de laços, as quais são custosas e pouco otimizadas na linguagem do MATLAB©. Como exemplo há a verificação de se a matriz de ajuste pode ser invertida e se ela é definida-positiva, incrementando o valor μ iterativamente até que haja um ajuste minimizante.

Portanto, dado que o critério de parada escolhido obriga os dois algoritmos a executarem sempre o mesmo número de épocas, a vantagem que o algoritmo de segunda ordem possui acaba desaparecendo, frente ao método de primeira ordem.

A Figura 5.5 apresenta a curva de desempenho para os candidatos escolhidos a cada progressão da busca. Percebe-se também neste gráfico comportamento semelhante ao ocorrido no envoltório-MLP, com a curva monotonicamente crescente a partir do subconjunto vencedor, de 3 variáveis.

Visão geral dos resultados

Após executar os algoritmos para as cinco abordagens de seleção de variáveis planejadas, a Tabela 5.13 apresenta a consolidação dos resultados. Apesar da diversidade de resultados, que variam desde apenas duas variáveis selecionadas até sete, nota-se que o experimento atingiu o objetivo de promover redução no tamanho do problema, independentemente da abordagem.

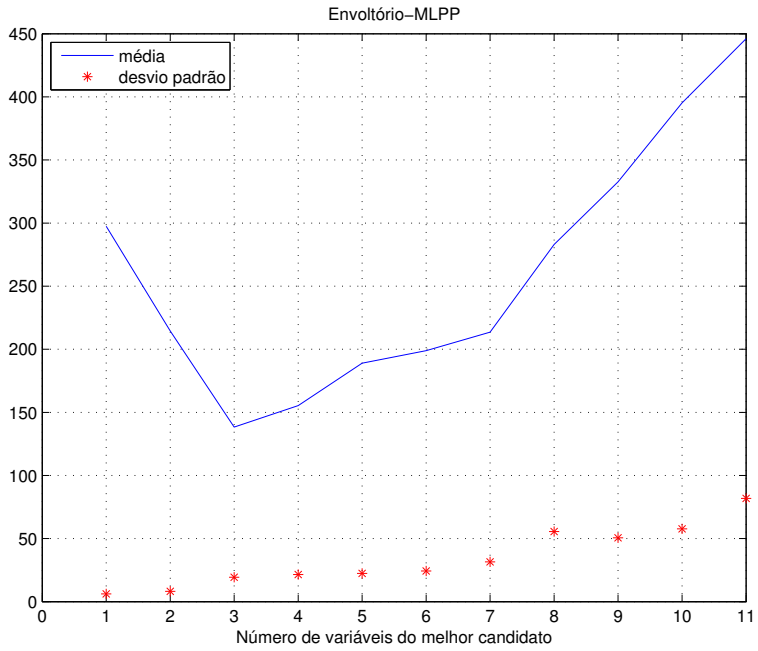

Fig. 5.5: Comportamento dos candidatos a solução para o envoltório com MLPP, na Base 1.

Em todos os cinco resultados selecionou-se a variável 2, "Número de passos de CTs total", o que indica uma forte relação de dependência desta com a medida de esforço. A segunda variável selecionada mais vezes é a de número 7, "Número de testadores envolvidos", que ocorre em todos os resultados, excetuando-se o do envoltório-MLP.

Tab. 5.13: Resultado final da etapa de seleção de variáveis, para a Base 1.

Método	**Variáveis selecionadas**
Filtro CFS	{1,2,3,5,7,8,11}
Envoltório-SVR	{2,7,9,10,11}
Envoltório-Reg. Linear	{1,2,7}
Envoltório-MLP	{2,10}
Envoltório-MLPP	{2,7,9}

Já as variáveis 4 — "Número de passos de CTs a executar" — e 6 — "Número de passos de CTs a re-executar" — não foram selecionadas por nenhum método, indicando que as informações obtidas por estas métricas não são, no contexto deste experimento, importantes para a determinação do esforço nesta base de dados.

Analisando a relação dos resultados com as variáveis propostas e coletadas para compor a base, percebe-se que a análise do processo de esforço de testes com o objetivo de identificar os fatores de influência no esforço não garante por si só que as melhores variáveis de entrada serão aplicadas

para treinar os modelos de aprendizado de máquina. Se esta hipótese fosse verdadeira, a maioria das variáveis disponíveis deveria aparecer nos resultados de cada abordagem de seleção de variáveis. Entretanto, não foi o que aconteceu: considerando os resultados dos métodos por envoltório para os dois tipos de redes neurais artificiais e para a regressão linear, nota-se que foi desconsiderada a maioria das variáveis disponíveis.

Por isso, reforça-se a recomendação de que, mesmo após uma análise por especialistas para encontrar as prováveis variáveis que determinam o esforço no processo de testes, aplicar um método de seleção de variáveis é útil e possivelmente trará tanto ganho de tempo no treinamento do modelo como melhoria da acurácia nas estimativas.

5.2.3 Comparação dos modelos

A pequena quantidade de dados na base impede que o experimento de comparação entre modelos seja passível de uma análise estatística. Seria necessário que as medidas do experimento obedecessem a critérios como normalidade nas distribuições, o que não é possível determinar, ou que elas fossem ao menos independentes [22].

Mesmo assim, pode-se realizar o experimento de comparação de modelos, conforme planejado na Seção 4.5.3 do Capítulo 4 e tentar identificar padrões nos resultados que possam trazer indicações úteis sobre o comportamento de cada modelo preditor. Os parâmetros e configurações de cada modelo são idênticos aos utilizados na etapa de seleção de variáveis.

A Tabela 5.14 apresenta os valores de média (μ) e desvio padrão (σ) para as 1.000 medidas das métricas MARE, PRED(0,25) e PRED(-0,25) do experimento. Os termos “Filtro” e “Envoltório”, na primeira coluna, distinguem, respectivamente, se as variáveis de entrada são aquelas selecionadas pelo método de filtro CFS ou se são aquelas selecionadas pela aplicação do método por envoltório usando o modelo preditor da coluna do dado em questão.

Os valores em negrito indicam que a rede MLP ponderada, utilizando as variáveis selecionadas pela aplicação do método por envoltório, apresentou o maior valor médio da métrica PRED(0,25) e também teve o menor valor médio da métrica MARE. Já para a métrica PRED(-0,25), o melhor modelo é a regressão linear utilizando as variáveis selecionadas por envoltório.

Se é considerada somente a aplicação dos modelos preditores usando as variáveis de entrada selecionadas pelo método de filtro, a rede MLPP é a melhor nas métricas PRED(0,25) e PRED(-0,25), enquanto o método de regressão linear apresenta o menor valor médio da métrica MARE. Percebe-se também que em nenhum modelo o desempenho global (medido pelas três métricas) das máquinas utilizando as variáveis escolhidas pelo filtro é melhor que o desempenho delas utilizando as variáveis escolhidas pelo método por envoltório. Isto pode ser considerado mais um indício de que os métodos por envoltório tendem a apresentar soluções de melhor qualidade do que os métodos de

Tab. 5.14: Resultado do experimento de comparação dos modelos para a Base 1.

		Reg. Linear		**MLP**		**SVR**		**MLPP**	
		μ	σ	μ	σ	μ	σ	μ	σ
Filtro	*PRED(0,25)*	0,473	0,187	0,408	0,208	0,440	0,198	0,575	0,210
	MARE	0,301	0,104	0,453	0,209	0,403	0,171	0,304	0,134
	PRED(-0,25)	0,219	0,170	0,180	0,156	0,187	0,151	0,235	0,170
Envoltório	*PRED(0,25)*	0,575	0,189	0,620	0,228	0,669	0,188	**0,702**	**0,181**
	MARE	0,267	0,101	0,274	0,154	0,271	0,129	**0,236**	**0,103**
	PRED(-0,25)	**0,337**	**0,181**	0,270	0,188	0,335	0,182	0,301	0,191

filtro, embora com maior custo computacional.

Tomando agora a escolha de variáveis por envoltório, destacam-se os desempenhos do modelo SVR e do modelo de Regressão Linear. A máquina de vetor de suporte apresentou o segundo maior valor médio da métrica PRED(0,25) – 0,669 – enquanto o modelo de regressão linear teve o melhor valor da métrica PRED(-0,25), já citado anteriormente, e o segundo melhor valor da métrica MARE.

Para o modelo MLPP é possível considerar que a modificação proposta na função custo aparentemente não trouxe prejuízos no funcionamento da máquina de aprendizado. Pelo contrário, comparando com os valores médios alcançados pela rede MLP, o novo modelo obteve ganho de desempenho em todas as métricas e ainda foi o modelo de melhor desempenho em duas delas. Tal fato reforça a proposta concebida neste trabalho.

Por fim, a avaliação dos resultados médios da métrica PRED(-0,25) não permite concluir que a proposta da rede MLPP aumentou a tendência do modelo em superestimar esforço. Embora o valor médio da métrica, com as variáveis de entrada escolhidas por envoltório, seja maior em comparação com a rede MLP comum (0,301 contra 0,270), tanto o modelo SVR como o de regressão linear apresentaram valores superiores, ainda que utilizem funções custo simétricas.

Espera-se que os melhores modelos deste experimento apresentem resultados similares na próxima etapa, que faz a simulação de uso dos modelos.

5.2.4 Simulação de uso

O segundo e último experimento para comparar os modelos preditores foi realizado conforme o plano dado na Seção 4.5.4 do Capítulo 4.

Está incluída na comparação a proposta de estimativa de esforço de execução de testes pela métrica de eficiência acumulada [77]. Este trabalho originou-se de dados prévios coletados no projeto HARPIA, compreendendo 20 ciclos de testes. Utilizando esta base propôs-se uma forma simples de estimativa de esforço de execução baseada somente no número acumulado de passos de CTs executados e no tempo gasto até então. Esta métrica denomina-se Eficiência Acumulada em Execução e foi

avaliada em um estudo de caso com resultados satisfatórios, dada a simplicidade de uso do modelo, mas que mostraram limitações quanto à consideração de outras variáveis que afetam o esforço da execução dos testes, como por exemplo a experiência dos testadores, as características dos sistemas em teste, etc..

A execução do experimento final para a Base 1 tem os resultados mostrados na Tabela 5.15, onde a primeira coluna indica o modelo preditor utilizado, com MEA sendo sigla para o modelo de eficiência acumulada e RL sendo a sigla para o modelo de Regressão Linear; a segunda coluna indica se o modelo foi testado com as variáveis de entrada determinadas pelo método de seleção por filtro ou se pelo método por envoltório ("Env."); as colunas centrais, cujos rótulos "SxCy" significam "Ciclo #y do *Software* #x", contêm os valores de erro relativo, definido por:

$$E_{rel} = \frac{E_{est} - E_{real}}{E_{real}} \tag{5.1}$$

e onde E_{est} é o esforço estimado e E_{real} é o esforço realizado de fato; a penúltima coluna contém a média do erro relativo absoluto e, por último, a coluna com os valores da métrica PRED(0,25), abreviada pela sigla P(0,25).

Tab. 5.15: Simulação de uso dos modelos preditores na Base 1 (valores percentuais).

		Ciclo de teste										
Modelo	**Seleção**	S4_C8	S5_C1	S5_C2	S5_C3	S5_C4	S7_C4	S6_C2	S5_C5	S7_C5	**MARE**	**P(0,25)**
MEA	N/A	177,1	139,6	81,6	172,4	135,1	-56,5	152,6	97,9	116,5	125,5	0,0
SVR	Filtro	52,4	17,2	-1,4	62,5	-50,2	48,6	62,5	-22,2	18,9	37,3	44,4
	Env.	87,4	21,5	23,1	112,0	-13,9	17,3	-10,4	19,0	19,5	36,0	**77,8**
MLP	Filtro	95,0	-22,3	-16,4	95,1	-35,7	0,8	13,5	-11,4	35,8	36,2	55,6
	Env.	92,6	-2,1	-5,1	90,9	-31,8	16,7	95,7	-3,6	59,0	44,2	44,4
RL	Filtro	66,4	43,0	-11,4	65,3	-2,7	18,6	56,0	-29,1	47,4	37,8	33,3
	Env.	52,8	45,1	2,1	66,8	7,8	-62,9	11,2	-14,6	17,0	**31,1**	55,6
MLPP	Filtro	120,9	7,1	8,5	120,6	-30,6	-18,1	53,0	-17,4	19,2	43,9	55,6
	Env.	142,0	18,4	16,4	139,6	-16,8	-4,7	53,8	16,5	18,1	47,4	66,7

A primeira observação que pode ser feita dos resultados mostrados na Tabela 5.15 diz respeito ao desempenho ruim do modelo baseado na eficiência acumulada: a média de erro relativo absoluto foi consideravelmente alta, de 125,5%, e nenhuma medida de erro absoluto está abaixo do valor de 25%. Embora no trabalho publicado por SILVA *et al.* [77] o desempenho inicial tenha sido satisfatório, o experimento de agora demonstra que de fato o modelo possui limitações que o levam ao pior desempenho entre todos os modelos analisados.

Para os modelos de regressão por vetor de suporte e regressão linear as duas métricas observadas obtiveram um melhor comportamento com as variáveis selecionadas por envoltório do que com

aquelas selecionadas por filtro. Por outro lado, a rede MLP comum teve melhor desempenho com as variáveis do filtro, enquanto a rede MLPP teve um valor de PRED(0,25) melhor para a simulação com as variáveis por envoltório e um valor de MARE melhor para a simulação com as variáveis por filtro. Considerando a proximidade dos valores nas duas métricas, pode-se dizer que houve um empate para a rede MLPP. Fazendo um balanço geral sobre todos os modelos, tem-se que o método por envoltório alcançou melhores resultados neste experimento, da mesma forma como ocorreu na etapa anterior.

O modelo de regressão linear com as variáveis selecionadas por envoltório apresentou o menor valor médio de erro relativo absoluto, de 31,1%, o que se assemelha com o bom desempenho apresentado na mesma métrica, no experimento anterior. Também usando as variáveis selecionadas por envoltório, a regressão por vetor de suporte obteve o melhor desempenho com a métrica PRED(0,25), de 77,8%, significando 7 ciclos estimados com erro relativo absoluto abaixo ou igual a 25%. Não obstante, o modelo ainda teve o segundo melhor valor da métrica MARE, de 36%. Já o modelo de rede MLPP, embora não tenha sido o melhor como no experimento anterior, obteve o segundo melhor resultado para a métrica PRED(0,25), também usando como entrada as variáveis selecionadas por envoltório.

A métrica PRED(0,25) pode fornecer uma melhor constatação sobre a regularidade das estimativas do modelo preditor. Outra forma de avaliar tal propriedade, agora no caso dos resultados deste experimento, é por gráficos de barras dos valores de erro relativo. A Figura 5.6 apresenta o gráfico dos erros obtidos utilizando cada configuração de modelo preditor de melhor desempenho no experimento, entre a opção com variáveis escolhidas por filtro e aquelas escolhidas por envoltório.

Nota-se no gráfico que nenhum modelo conseguiu um erro relativo menor do que 50% para as estimativas de número 1 e 4, que correspondem ao oitavo ciclo testes do *software* 4 e ao terceiro ciclo de testes do *software* 5. Tal fato pode indicar que o pequeno número de amostras disponíveis para a síntese dos modelos foi insuficiente para ajustá-los de forma a estimar todas as amostras do conjunto de testes de forma regular, e que estes dois ciclos podem possuir características razoavelmente distintas dos outros ciclos observados.

Também pela visualização do gráfico torna-se mais claro que o modelo de regressão por vetor de suporte com envoltório, indicado pelo segundo tom de cinza mais escuro, teve o comportamento mais regular entre todos os observados. Todas as estimativas, exceto os dois casos de ciclos citados anteriormente (1 e 4), pertencem ao intervalo $[-25\%, 25\%]$, sendo a maioria (5 em 7) delas positivas, o que significa que o modelo superestimou mais do que subestimou.

Em seguida, no quesito de regularidade, está indicado pela cor preta o modelo MLPP com envoltório, com 6 medidas dentro do valor absoluto de 25%. Entretanto, apresentou os maiores erros entre todos os modelos para as medidas 1, 4 e 7.

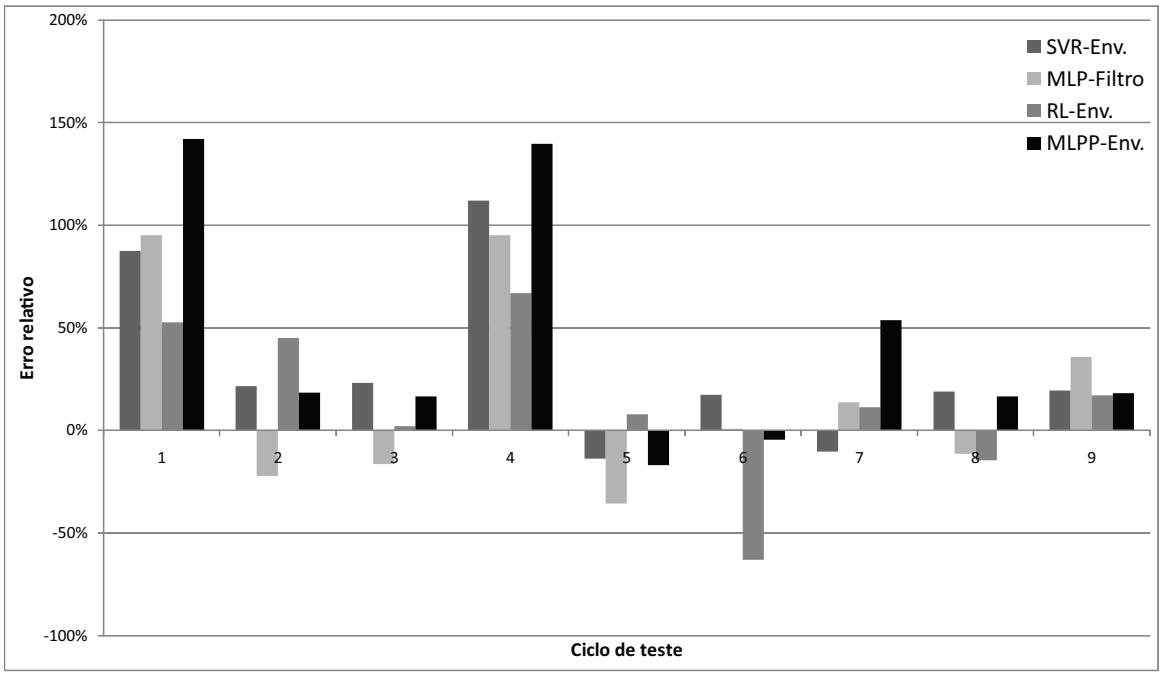

Fig. 5.6: Erro relativo a cada ciclo para os melhores modelos da simulação de uso na Base 1.

5.3 Resultados com a Base 2

5.3.1 Dados coletados

A Base 2 é composta por 70 ciclos de testes realizados sobre 18 diferentes versões de um único sistema de *software*, o qual foi desenvolvido pelo departamento de TI de uma organização chinesa de serviços financeiros. Desta forma, houve a realização de vários ciclos de teste para cada versão produzida. São no total 41.139 execuções de casos de testes funcionais.

Devido à dependência exclusiva da disponibilidade do pesquisador colaborador para coletar os dados, não foi possível adotar com ênfase os mesmos critérios usados para definir as variáveis da Base 1. Por isso, a Tabela 5.16 apresenta a relação de variáveis de entrada coletadas, que são apenas três: as duas primeiras estão relacionadas à dimensão de complexidade dos CTs, enquanto a terceira está relacionada à dimensão de complexidade do sistema em teste.

Enquanto as duas primeiras possuem a mesma definição usada na Base 1, a terceira variável, em especial, não foi coletada na primeira base e consiste no número de linhas de código atualizados em comparação à versão anterior do sistema.

Construída a Base 2, é aplicada nas variáveis de entrada a transformação para terem média 0

Tab. 5.16: Variáveis da Base 2 e mapeamento com as dimensões de influência.

Variável	**Dimensão mapeada**
1. Número de CTs total	
2. Número de passos de CTs total	Complexidade dos CTs
3. Número de linhas de código modificadas	Complexidade do Sistema

e desvio padrão igual a 1. Realiza-se uma análise estatística básica na variável que é objeto de estimativa: o esforço realizado na execução do ciclo de teste. A Figura 5.7 apresenta o diagrama *box-plot*, no qual se detecta a presença de 4 *outliers* superiores, com o restante dos dados distribuídos até o valor de aproximadamente 300 homens-horas. A mediana é de 88 pessoas-horas, média de 116,09 pessoas-horas e desvio-padrão de 101,87 pessoas-horas.

Para não complicar a preparação dos dados e, consequentemente, o uso prático do método, decidiu-se não excluir os *outliers* da base.

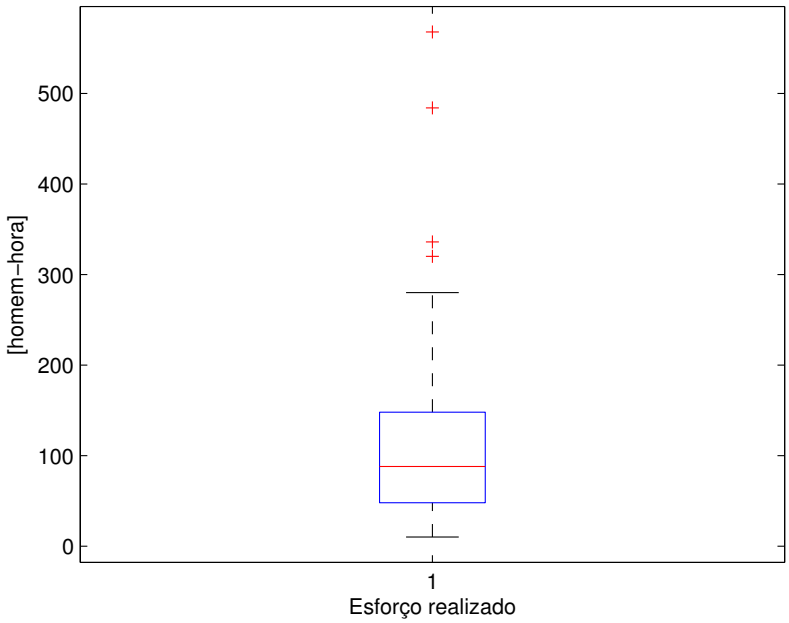

Fig. 5.7: Diagrama *box-plot* da variável esforço, para a Base 2.

A Base 2 é consideravelmente diferente da Base 1, como mostram as características apresentadas para as duas. Enquanto na primeira base foi possível realizar um processo mais apurado de determinação de variáveis e coleta dos dados em sintonia com o estudo desenvolvido sobre os fatores que influenciam o esforço, na segunda base apenas três variáveis estão disponíveis para uso. Por outro lado, o número de amostras da segunda base é mais do que o dobro da primeira (70 contra 30), provendo mais informação em termos de variabilidade nos dados.

O processo de testes seguido pela equipe da Base 2 é desconhecido, e também se sabe que a média de passos por caso de teste é muito menor se comparada com o valor da Base 1: 2,7 passos para a Base 2, contra 37,2 passos para a Base 1. Apenas um sistema é testado nos dados da Base 2, para a Base 1 são sete sistemas testados.

Todas estas divergências podem ser consideradas sob uma ótica positiva, porque é conhecido que a realidade das organizações de *software* é heterogênea [80]. Para poder estudar e elaborar uma metodologia, é valioso que os experimentos possam ser feitos sobre dados que reflitam essa diversidade. É claro que o número de amostras e bases disponíveis neste trabalho ainda não é adequado para se obter conclusões gerais, porém é um ponto de partida para se estabelecer direções na solução do problema de estimação de esforço de execução de testes.

5.3.2 Seleção das variáveis

Embora existam apenas três variáveis disponíveis na base de dados, é importante repetir o experimento de seleção de variáveis para avaliar a qualidade de cada uma delas, bem como estabelecer uma comparação nos procedimentos realizados nas duas bases. As próximas subseções descrevem os resultados obtidos.

Filtro CFS

Repete-se o procedimento usando a ferramenta WEKA, com a estratégia de busca exaustiva, e se obteve como resultado a seleção da primeira variável da base: **"1. Número de CTs total"**. Ficaram de fora da seleção as variáveis "Número de passos de CTs total" e "Número de linhas de código modificadas", implicando numa redução de aproximadamente 67% no tamanho da entrada do problema. O tempo de execução do algoritmo, também como no caso da Base 1, foi rápido, da ordem de segundos.

Envoltório com SVR

Usando exatamente os mesmos procedimentos e critérios descritos na subseção 5.2.2 para definição dos valores de parâmetros, realiza-se agora a atividade preliminar de estabelecer a configuração adequada para o modelo SVR a ser utilizado na seleção por envoltório.

Com os valores arbitrários de $\epsilon = 10$ e $\sigma = 0{,}1$, inicia-se o processo testando variações sobre a constante C, conforme mostram os resultados reunidos na Tabela 5.17. A variação de melhor desempenho foi para $C = 1000$, a qual é escolhida e passa-se para o próximo parâmetro.

Agora com $C = 1000$ e $\sigma = 0{,}1$, repetições do envoltório são feitas com variação no parâmetro ϵ, e a Tabela 5.18 apresenta os valores que indicam que $\epsilon = 45$ foi o melhor.

Tab. 5.17: Variação do parâmetro C na execução do envoltório-SVR, para a Base 2.

C	**10**	**100**	**1.000**	**10.000**
Subconjunto escolhido	{1}	{1}	{1,2}	{1,2}
Média do MSE	6856,39	4734,30	**2875,78**	6318,86
Desvio padrão do MSE	5877,70	3987,47	**2321,50**	7555,40
Desvio Padrão / Média	86%	84%	**81%**	120%

Tab. 5.18: Resultado final da etapa de seleção de variáveis, para a Base 2.

ϵ	**40**	**45**	**50**	**55**	**60**
Subconjunto escolhido	{1,2}	{1,2}	{1,2}	{1,2}	{1,2}
Média do MSE	2777,52	**2762,87**	2922,42	3109,29	3407,30
Desvio padrão do MSE	1857,44	**1845,06**	1825,34	1714,65	1780,76
Desvio Padrão / Média	67%	**67%**	62%	55%	52%

x

Encerra-se o processo de ajuste pela definição do valor de σ, com quatro repetições do algoritmo de envoltório, cujos valores são descritos na Tabela 5.19. A configuração final do algoritmo SVR, a ser utilizada tanto neste experimento de seleção de variáveis por envoltório como nos experimentos seguintes, consiste nos valores das constantes $C = 1.000$, $\epsilon = 45$ e $\sigma = 0{,}1$.

Tab. 5.19: Variação do parâmetro σ na execução do envoltório-SVR, para a Base 2.

σ	**0,05**	**0,1**	**0,2**	**0,4**
Subconjunto escolhido	{1,2}	{1,2}	{1}	{1,2}
Média do MSE	3070,39	**2762,87**	2888,61	2953,17
Desvio padrão do MSE	2357,48	**1845,06**	2045,30	2322,10
Desvio Padrão / Média	77%	**67%**	71%	79%

É executado o algoritmo de envoltório, agora devidamente configurado, para determinar as variáveis selecionadas. O tempo de execução do algoritmo foi rápido, de apenas 0,2s, e duas variáveis foram selecionadas: **"1. Número de CTs total"** e **"2. Número de Passos de CTs total"**. O valor médio do MSE para a solução encontrada é de 2.762,87, enquanto seu desvio padrão é 1.845,06, significando um valor relativo de aproximadamente 67%.

A única variável descartada no procedimento é a de número 3, "Número de linhas de código modificadas", e assim a redução de dimensão da entrada do problema fica na faixa de 33%.

A Figura 5.8 apresenta a curva de desempenho do algoritmo, a qual indica claramente que embora o número de variáveis seja pequeno (apenas três), isto não significa que um procedimento de seleção de variáveis torna-se inútil, pois neste caso ele atesta que o desempenho da predição pode melhorar caso se utilize apenas as duas variáveis selecionadas. Ainda que o melhor subconjunto escolhido fosse o próprio conjunto inicial, o método permaneceria útil porque estaria validando empiricamente

o conjunto inicial de variáveis.

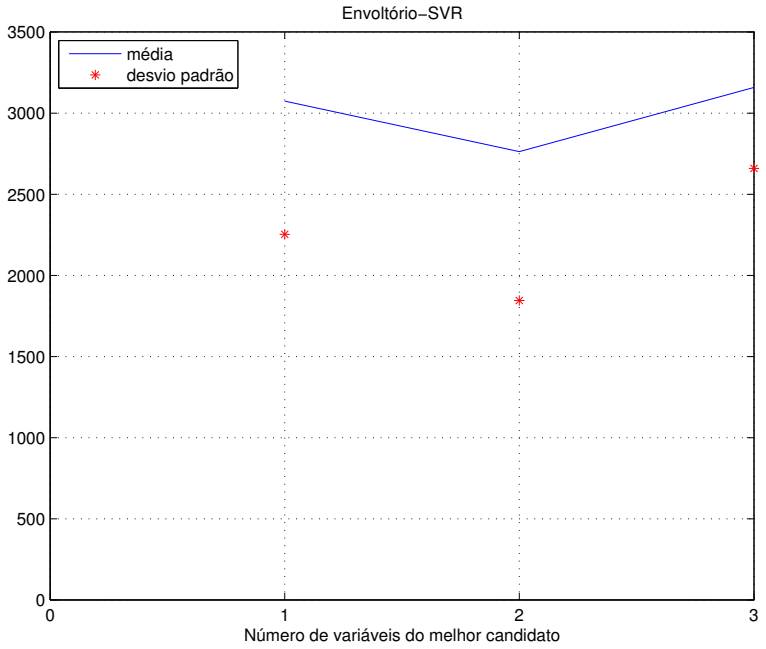

Fig. 5.8: Comportamento dos candidatos a solução para o envoltório com SVR, na Base 2.

Envoltório com Regressão Linear

Repete-se nos mesmos termos o algoritmo de envoltório com Regressão Linear, como antes realizado para a Base 1, agora para a segunda base. Foi escolhido como solução o subconjunto que compreende as variáveis **"1. Número de CTs total"** e **"2. Número de Passos de CTs total"**, com o tempo de execução de somente 0,2s, uma média do MSE de 2.161,49 e desvio padrão de 1.353,40 (63% relativo).

Mais uma vez houve uma redução de aproximadamente 33% na dimensão da entrada, sendo escolhidas as mesmas duas variáveis da solução pelo método por envoltório-SVR. A Figura 5.9 apresenta o comportamento dos melhores candidatos à solução à medida que a seleção progressiva ocorria, e novamente o ponto de mínimo da curva está no subconjunto de 2 variáveis selecionadas.

Envoltório com rede MLP comum

Mantendo iguais os valores $\mu = 0{,}001$, $\beta = 0{,}1$ e 200 para o limite de épocas de treinamento, aplica-se o procedimento empírico somente para determinar o número de camadas ocultas, e o número de neurônios em cada camada. Permanece também igual o número de repetições da validação

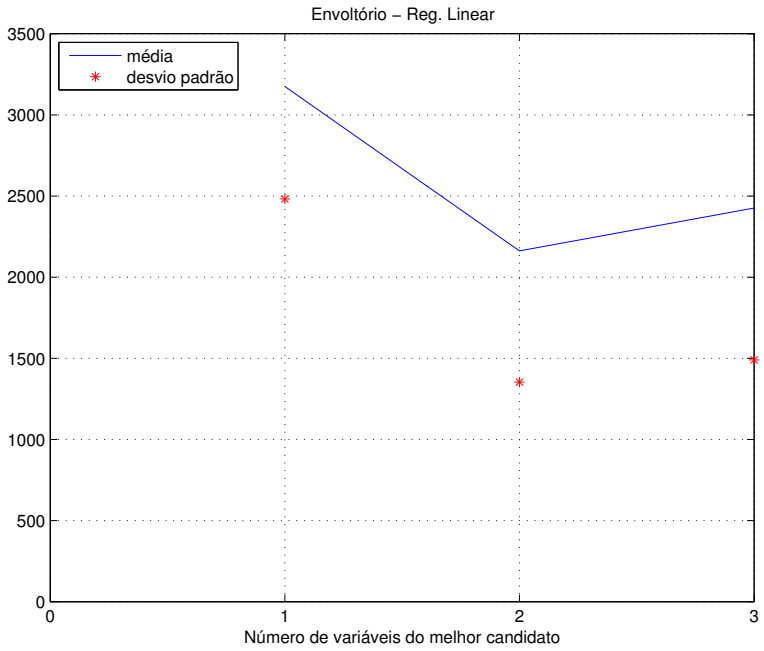

Fig. 5.9: Comportamento dos candidatos a solução para o envoltório com Regressão Linear, na Base 2.

cruzada, 20, bem como a divisão dos dados. A Tabela 5.20 apresenta o resultado da rodada preliminar de execuções do algoritmo de envoltório.

Nota-se que os valores de média de erro, obtidos com apenas uma camada oculta, vão decrescendo à medida que se diminui o número de neurônios. Por outro lado, os valores de desvio padrão são maiores, e segundo os critérios para decisão da melhor configuração, é melhor portanto a arquitetura de uma camada oculta com 20 neurônios, destacada em negrito na Tabela 5.20. Seu valor médio de erro não foi o menor, porém teve o menor desvio padrão relativo e, conforme o critério empírico em uso, teve o menor valor do produto entre a média do erro e o desvio padrão.

Tab. 5.20: Ajuste de neurônios na primeira camada oculta, na execução do envoltório-MLP para a Base 2.

	Duas camadas ocultas				Uma camada oculta			
# Neurônios	**20**	**16**	**12**	**8**	**20**	**16**	**12**	**8**
Tempo gasto[min]	12	11	9	8	13	11	10	8
Subconjunto escolhido	{1,2}	{1,2}	{1,2}	{1,2}	{1,2}	{1,2}	{1,2}	{1,2}
Média do erro	5936,44	5573,33	5580,29	5058,47	**2472,34**	2386,84	2239,26	2155,33
Desvio padrão do erro	1169,21	1256,77	1098,91	999,66	**320,82**	463,00	389,43	420,86
Desvio Padrão / Média	20%	23%	20%	20%	**13%**	19%	17%	20%

Estabelecida a configuração, passa-se à execução definitiva do algoritmo de envoltório-MLP.

Como feito na Base 1, novamente o número de repetições da validação cruzada é aumentado, passando para 40. A solução apresentada contém as mesmas variáveis selecionadas pelos outros algoritmos de envoltório: **"1. Número de CTs total"** e **"2. Número de Passos de CTs total"**. O tempo de execução foi de 26 minutos, o valor médio de erro para o subconjunto escolhido foi de 2.412,52 e o desvio padrão de 499,34 (21% relativo).

Como esperado e devido às mesmas razões do experimento com a Base 1, o tempo de execução do algoritmo foi outra vez maior do que os tempos de execução dos outros métodos de seleção já utilizados, para a Base 2. Entretanto, o tempo gasto de 26 minutos é aproximadamente um sexto do tempo gasto para o mesmo método na Base 1; essa diferença se explica pelo menor número de variáveis no espaço de busca. Retomando a discussão feita na Seção 3.4.1 do Capítulo 3, a primeira base possui 11 variáveis, o que implica em $11(11+1)/2 = 66$ avaliações de subconjuntos; já a segunda base possui 3 variáveis, resultando em somente $3(3+1)/2 = 6$ avaliações.

A Figura 5.10 apresenta o desempenho dos subconjuntos candidatos durante a execução da busca. Comparado com os métodos aplicados anteriormente e observando o gráfico, detecta-se que o valor de desvio padrão é relativamente o menor entre todos até o momento.

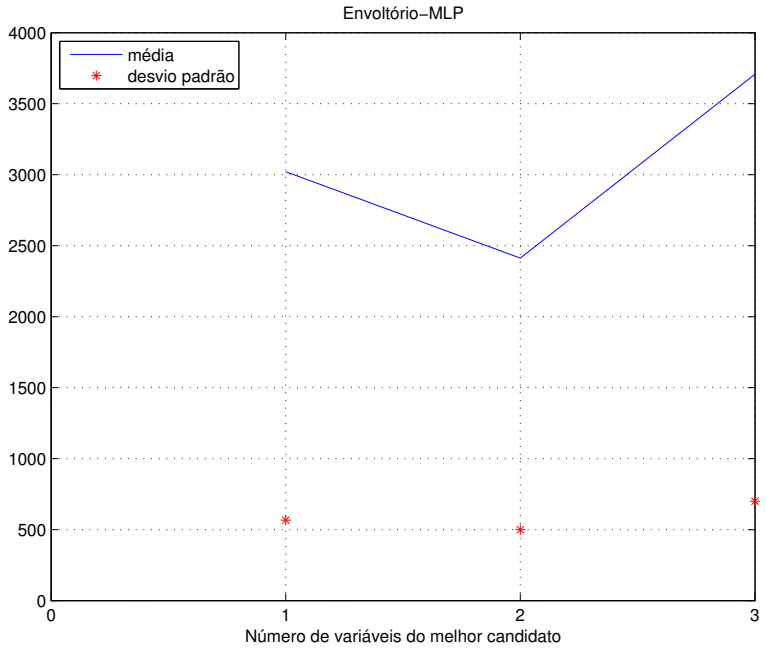

Fig. 5.10: Comportamento dos candidatos a solução para o envoltório com MLP, na Base 2.

Envoltório com rede MLP ponderada

Análogo ao procedimento feito para o envoltório-MLPP na Base 1, mantêm-se os valores $\alpha = 0{,}01$, limite de 200 épocas de treinamento e passa-se à determinação do uso de duas ou apenas uma camada oculta, bem como o número de neurônios na primeira camada oculta. Mantém-se o mesmo número de iterações da validação cruzada (20), a mesma divisão em conjunto de treinamento e conjunto de validação, e o uso da média das 20 repetições da média do erro quadrático médio *ponderado* (Definição 4.2 do Capítulo 4) nas cinco partições de teste.

A Tabela 5.21 apresenta os resultados das execuções preliminares do algoritmo de envoltório-MLPP. Outra vez a melhor arquitetura para a rede MLP comum também foi a escolhida para a rede MLPP: somente uma camada oculta, com 20 neurônios. Nota-se também que a variação nos tempos gastos em cada execução foi menor, ficando entre 9,67 e 3,5 minutos e que os valores relativos de desvio padrão, das configurações com uma camada oculta, são os menores já registrados até então neste trabalho.

Tab. 5.21: Ajuste de neurônios na primeira camada oculta, na execução do envoltório-MLPP para a Base 2.

	Duas camadas ocultas				Uma camada oculta			
# Neurônios	**20**	**16**	**12**	**8**	**20**	**16**	**12**	**8**
Tempo gasto[min]	5,00	4,00	3,75	3,50	9,67	8,57	8,40	8,23
Subconjunto escolhido	{1,3,2}	{1,3,2}	{1,2,3}	{2,1,3}	{1,2}	{1,2}	{1,2}	{1,2}
Média do erro	22419,76	23791,21	24407,57	24724,54	**4643,06**	4862,73	5537,99	7310,40
Desvio padrão do erro	3026,19	2544,12	1732,71	1237,63	**86,87**	121,41	240,37	360,49
Desvio Padrão / Média	13%	11%	7%	5%	**2%**	2%	4%	5%

Com o número de repetições da validação cruzada alterado para 40, realiza-se a execução final do algoritmo e a solução se repete: **"1. Número de CTs total"** e **"2. Número de Passos de CTs total"**. O tempo de execução foi de 19 minutos, a média do erro 4.605,00 e seu desvio padrão 85,32, o que significa um valor relativo de apenas 2%, o mais baixo anotado em todos os experimentos.

Vale registrar que há o mesmo padrão de comportamento, em termos de tempo de execução, ao comparar-se o envoltório-MLPP com o envoltório-MLP, para esta segunda base de dados. Dado que as hipóteses levantadas no cenário da primeira base independem dos dados, elas são válidas para explicar novamente esta situação, por isso não surpreende o fato que o tempo para executar o atual algoritmo foi menor que o tempo necessário para o envoltório-MLP. A curva de desempenho para os candidatos escolhidos a cada progressão da busca pode ser analisada na Figura 5.11, que exibe comportamento semelhante ao dos métodos por envoltório anteriores.

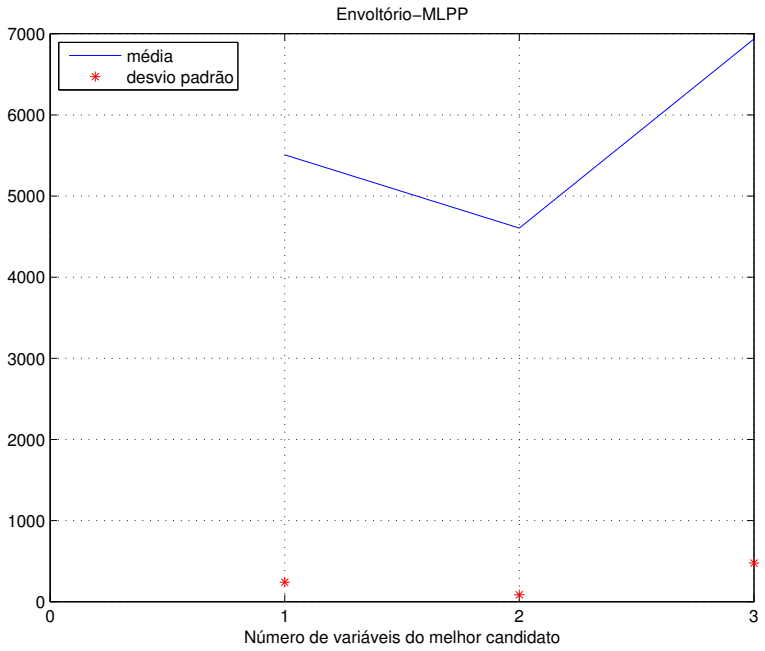

Fig. 5.11: Comportamento dos candidatos a solução para o envoltório com MLPP, na Base 2.

Visão geral dos resultados

A Tabela 5.13 exibe a consolidação dos resultados da execução dos algoritmos de seleção de variáveis. Houve redução da dimensão do problema em todas as abordagens. Observa-se que todos os resultados utilizando o algoritmo de envoltório foram iguais, tendo havido a seleção das variáveis 1 e 2, enquanto o método de filtro selecionou apenas a variável 1, "Número de CTs total". A variável 3, "Número de linhas de código modificadas", não foi escolhida em nenhuma abordagem, indicando uma falta de relação com o esforço no âmbito dos modelos empregados.

Tab. 5.22: Resultado final da etapa de seleção de variáveis, para a Base 2.

Método	**Variáveis selecionadas**
Filtro CFS	{1}
Envoltório-SVR	{1,2}
Envoltório-Reg. Linear	{1,2}
Envoltório-MLP	{1,2}
Envoltório-MLPP	{1,2}

Do ponto de vista prático, pode-se questionar ao final deste experimento a validade de se realizar seleção de variáveis sobre uma base com um número tão reduzido de variáveis. Ainda não é possível avaliar a qualidade dos dados disponíveis para se realizar estimativa do esforço, por meio dos modelos

preditores, porém é válido considerar que o fato de haver uma redução dimensional no problema trará um ganho na qualidade dessas estimativas. Talvez o ganho não seja significativo, mas isto só pode ser averiguado nos experimentos em sequência.

5.3.3 Comparação dos modelos

De forma idêntica, o experimento da Subseção 5.2.3 é repetido, agora para os dados da segunda base. A Tabela 5.23 apresenta os valores de média (μ) e desvio padrão (σ) para as 1000 medidas das métricas MARE, PRED(0,25) e PRED(-0,25).

A regressão linear utilizando as variáveis selecionadas pela aplicação do método por envoltório é destacadamente a técnica vencedora do experimento: os valores em negrito indicam que ela apresentou o maior valor médio da métrica PRED(0,25) e também teve o menor valor médio da métrica MARE. Não obstante, ela também teve o maior valor médio para a métrica PRED(-0,25).

Tab. 5.23: Resultado do experimento de comparação dos modelos para a Base 2.

		Reg. Linear		**MLP**		**SVR**		**MLPP**	
		μ	σ	μ	σ	μ	σ	μ	σ
Filtro	*PRED(0,25)*	0,353	0,128	0,390	0,131	0,348	0,118	0,353	0,131
	MARE	0,472	0,144	0,486	0,155	0,584	0,205	0,518	0,146
	PRED(-0,25)	0,163	0,101	0,189	0,105	0,215	0,101	0,153	0,097
Envoltório	*PRED(0,25)*	**0,577**	**0,123**	0,500	0,140	0,362	0,122	0,437	0,132
	MARE	**0,342**	**0,097**	0,399	0,144	0,561	0,190	0,468	0,149
	PRED(-0,25)	**0,313**	**0,122**	0,247	0,121	0,225	0,107	0,225	0,111

Ao se analisar a aplicação dos modelos somente usando as variáveis de entrada selecionadas pelo método de filtro, a rede MLP é vencedora na métrica PRED(0,25), o método de regressão linear apresenta o menor valor médio da métrica MARE, e o modelo SVR venceu pelo valor da métrica PRED(-0,25). Da mesma maneira que ocorreu com a primeira base de dados, nenhum modelo teve um desempenho global (medido pelas três métricas), utilizando as variáveis escolhidas por filtro, melhor do que o desempenho utilizando as variáveis escolhidas por método por envoltório.

A rede MLP com variáveis determinadas por envoltório foi o modelo que conseguiu o segundo melhor desempenho geral: teve o segundo melhor valor da métrica PRED(0,25), de 50%, e o segundo melhor valor da métrica MARE, de 39,9%.

Ao avaliar os valores obtidos em todos os modelos, fica claro que o resultado deste experimento foi aquém do esperado para um bom modelo preditor. Embora a caracterização do problema de esforço forneça indícios de que sua natureza é complexa, e portanto com maiores chances de um modelamento não-linear ser bem sucedido, os resultados mostraram que a regressão linear teve destacadamente a

melhor atuação. Ainda, os valores obtidos pelo melhor método são inferiores ao desempenho de outras propostas encontradas na literatura.

É importante considerar que a disponibilidade de variáveis nesta base é distante da situação ocorrida para a primeira base. Possivelmente, situações onde há limitação de informação sobre o problema dificultam a criação de mapeamentos adequados pelos modelos não-lineares e, por isso, as abordagens simplificadas podem se destacar.

Por fim, tanto o modelo de rede MLPP como o modelo SVR não apresentaram o mesmo padrão de desempenho visto no experimento com a primeira base de dados.

5.3.4 Simulação de uso

Conforme os mesmos procedimentos usados na primeira base, é realizada a simulação de uso agora envolvendo a segunda base de dados, para avaliar 21 estimativas dos quatro modelos.

Devido à ausência das datas de execução de cada caso de teste, as quais não foram disponibilizadas nesta base, não é possível aplicar o modelo de eficiência acumulada, e portanto ele não está incluso no experimento. A execução do experimento final para a Base 2 tem os resultados mostrados na Tabela 5.24, similar à Tabela 5.15, mas ocultando os valores individuais de erros. O cálculo de erro relativo continua sendo feito de acordo com a Equação 5.1.

Tab. 5.24: Simulação de uso dos modelos preditores na Base 2.

Modelo	Seleção	MARE	PRED(0,25)
SVR	Filtro	99,50%	38,10%
	Env.	91,81%	38,10%
MLP	Filtro	104,68%	23,81%
	Env.	128,44%	33,33%
Reg. Lin.	Filtro	70,73%	28,57%
	Env.	**38,36%**	**57,14%**
MLPP	Filtro	83,09%	23,81%
	Env.	82,02%	28,57%

À exceção da métrica MARE para o modelo MLP, em todos os outros valores de métricas de todos os modelos, a abordagem de seleção por envoltório teve melhor desempenho do que a abordagem por filtro. Em termos de desempenho geral dos modelos, mais uma vez, como tinha ocorrido no experimento anterior, os modelos tiveram um comportamento ruim. O único modelo que apresenta um desempenho razoável, e foi o melhor entre todos, é o de regressão linear, cujos valores de erro médio e PRED(0,25) estão destacados em negrito na Tabela 5.24.

O modelo de rede MLPP com seleção por envoltório teve o segundo melhor valor de MARE, o que entretanto foi extremamente alto: 82,02%. Já o modelo SVR, seja a seleção por envoltório ou

por filtro, obtiveram o segundo melhor valor de PRED(0,25): 38,10%, também ruim. A rede MLP comum apresentou os piores resultados entre os modelos.

Para avaliar a regularidade dos modelos nas estimativas, dispõe-se da Figura 5.12, que apresenta o gráfico dos erros relativos de cada modelo usando a abordagem de seleção de variável que trouxe melhor desempenho na métrica PRED(0,25): seleção por envoltório.

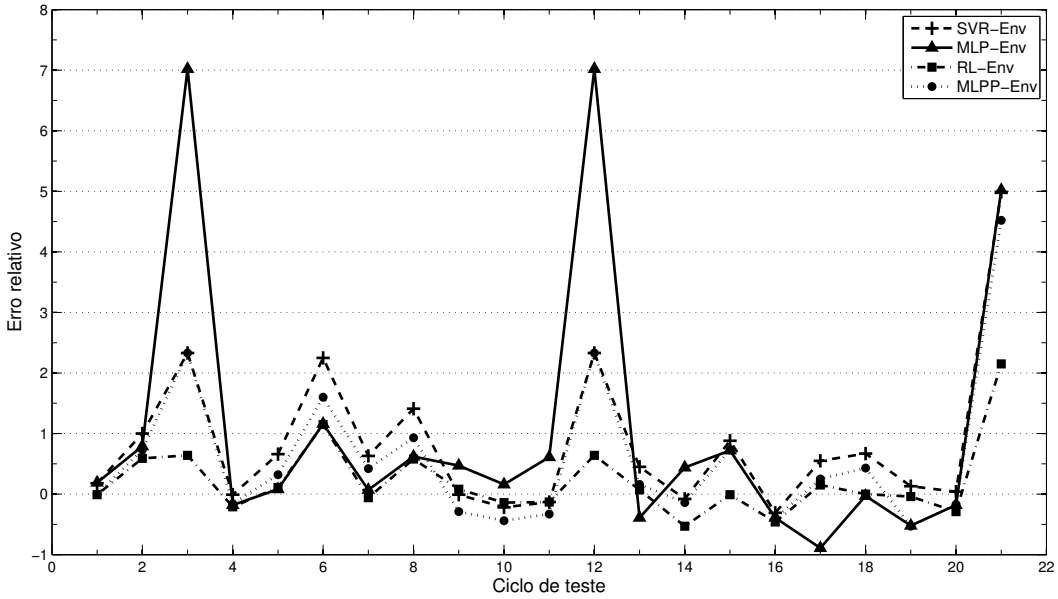

Fig. 5.12: Erro relativo a cada ciclo de testes, para os melhores modelos da simulação de uso na Base 2.

O gráfico consegue ilustrar a superioridade do modelo de regressão linear, representado pela linha com marcadores em formato de quadrado. Duas estimativas em 21 ficaram acima de 100% de erro relativo. Por outro lado, o modelo de rede MLP comum, representado pela linha com marcadores triangulares, tem uma elevada dispersão das suas predições, com três estimativas acima dos 500% de erro relativo. Os modelos de regressão por vetor de suporte e MLP ponderada possuem curvas semelhantes, com uma ligeira vantagem para o segundo.

A Base 2 proporcionou um maior número de amostras para o treinamento dos modelos, em comparação com a primeira. Porém, menos informação estava disponível, reflexo da existência de apenas três variáveis. O resultado geral indicou insucesso para os modelos não-lineares, com um desempenho razoável apenas para o modelo de regressão linear múltipla por quadrados mínimos.

5.4 Síntese

Neste capítulo descrevem-se os resultados dos experimentos executados em duas bases de dados reais. De maneira introdutória são apresentadas as implementações dos algoritmos; todos, com exceção do algoritmo do filtro CFS, são implementados na linguagem do ambiente MATLAB©.

Os resultados das quatro etapas de experimentos sobre a Base 1 indicam a predominância do método por envoltório para selecionar as variáveis; os melhores desempenhos de predição são os dos métodos de regressão por vetor de suporte e MLP ponderada. O método de regressão linear também apresenta um desempenho competitivo.

Para os resultados das quatro etapas sobre a Base 2, o método por envoltório permaneceu como o mais adequado para selecionar variáveis; entretanto, os modelos não-lineares não repetiram o mesmo nível de desempenho obtido com a Base 1, somente o modelo de regressão linear apresenta estimativas dentro de um intervalo aceitável.

O capítulo seguinte traz as conclusões e discussões gerais sobre este trabalho, além de propor melhorias e trabalhos futuros.

Capítulo 6

Conclusão e trabalhos futuros

6.1 Síntese do trabalho

Este trabalho de mestrado consistiu no estudo de técnicas de seleção de variáveis e aprendizado de máquina para o problema de estimar esforço de execução de testes funcionais de *software*. Os métodos de envoltório e filtro CFS foram as técnicas de seleção de variáveis consideradas, enquanto as propostas de redes neurais MLP, máquinas de vetor de suporte e regressão linear foram os modelos de aprendizado de máquina escolhidos para análise.

Realizou-se uma análise do processo de testes funcionais e dos fatores de influência no esforço, além de uma revisão de trabalhos anteriores sobre o tema. Como parte experimental do trabalho, executou-se um estudo de caso composto por uma série de experimentos em duas bases de dados reais, envolvendo todos os algoritmos escolhidos.

Com o objetivo de manter baixo o custo de coleta de dados, foram consideradas somente variáveis que não dependem de avaliação subjetiva para compor estas duas bases de dados. Equipes que possuam processos de testes bem definidos não terão dificuldade em implantar o mesmo esquema de coleta.

O custo computacional para aplicar cada modelo foi baixo, graças ao pequeno número de dados e variáveis das duas bases. Dado que a prioridade do trabalho era comparar a eficácia de predição de cada modelo, as implementações foram realizadas em ferramentas de natureza acadêmica para facilitar a análise dos resultados. Se trabalhos futuros confirmarem o uso de modelos de aprendizado de máquina para estimar esforço de testes de *software*, será necessário fazer uma nova implementação do modelo escolhido, desta vez como uma ferramenta de interface amigável com o usuário e otimizada também para um bom desempenho.

As conclusões são apresentadas de forma separada a seguir, de acordo com os objetivos propostos na Seção 1.3 e buscando responder às questões estabelecidas na Seção 1.4:

- Qual é a melhor abordagem para selecionar variáveis, nos dois estudos de caso?
- Qual é o melhor modelo para predizer esforço, nos dois estudos de caso?
- O melhor modelo nos dois estudos de caso apresenta um desempenho bom para os padrões da literatura?

6.1.1 Seleção de variáveis

Para cada base de dados, aplicou-se o método por filtro CFS e o método por envoltório, sendo este último realizado com cada um dos modelos de aprendizado de máquina em estudo. Cinco resultados de subconjuntos de variáveis foram obtidos para cada base, apresentados no Capítulo 5.

Ao avaliar esses resultados, nota-se que tanto o método por filtro CFS como as variações por envoltório com os modelos preditores apresentam soluções que garantem uma diminuição no tamanho do problema em termos de dimensão dos dados de entrada, seja para a primeira base de dados como para a segunda. Considerando o impacto desses resultados sobre o objetivo de estimar o esforço a partir dos modelos preditores, conclui-se que a inclusão da etapa de seleção de variáveis proporcionou à etapa seguinte (predição) uma execução do treinamento mais rápida e com estimativas de melhor qualidade, para ambas as bases de dados, em comparação a um cenário no qual fosse feito o treinamento e uso dos modelos com todas as variáveis disponíveis. Se a seleção não trouxesse ganhos de qualidade, o próprio procedimento apontaria como solução o conjunto completo de variáveis, fato este que não ocorreu com as duas bases de dados.

Em especial para a Base 1, na qual foi possível realizar uma coleta de métricas alinhada com o estudo anterior sobre o processo de testes, a etapa de seleção de variáveis foi ainda mais útil porque complementa esta etapa anterior de definição das variáveis. Enquanto o processo manual foi usado para escolher as métricas que se consideravam mais prováveis de influenciar o esforço, o processo automático realizou um refinamento pela seleção das variáveis realmente pertinentes ao modelo preditor.

Ainda sobre a Base 1, a variável "Número de passos de CTs total" pode ser considerada a de maior influência no estudo do esforço, pois ela esteve presente nos cinco resultados de subconjuntos de variáveis. Exceto pelo subconjunto escolhido pelo método por envoltório com o modelo MLP, a variável "Número de testadores envolvidos" esteve presente em todos os resultados e por isso pode ser considerada a de segunda maior influência no estudo. Para a Base 2, a variável "Número de CTs total" pode ser considerada a de maior influência por estar presente em todos os resultados, com a variável "Número de passos de CTs total" tendo a segunda maior influência, presente em quatro dos cinco resultados.

O fato de a variável que mede o número total de passos de CTs ter tido influência destacada nos experimentos das duas bases de dados reforça o conceito de que a complexidade do conjunto de casos de testes, ainda que medido por meio de informações indiretas, tem grande influência na determinação do esforço de execução de testes.

Na comparação da qualidade de resultados entre os dois métodos, as simulações dos modelos que utilizaram variáveis selecionadas por envoltório tiveram, na sua maioria, desempenho melhor do que quando utilizaram variáveis selecionadas por filtro. Essa constatação está alinhada com as características e avaliações anteriores desses métodos, apontadas no Capítulo 3, e responde à questão sobre qual é o melhor método para selecionar variáveis.

A velocidade na execução do método por filtro CFS e dos métodos por envoltório com regressão linear e envoltório com SVR foi consideravelmente superior à dos métodos por envoltório com MLP e MLPP. Já era esperado que o filtro CFS apresentasse um tempo de operação menor do que os outros; no caso dos métodos por envoltório com redes neurais a principal razão de obterem os maiores tempos de execução consiste no maior número de repetições da validação cruzada, feitas com o objetivo de contornar a variabilidade no treinamento.

É preciso esclarecer que esses resultados de tempo de execução não podem ser generalizados, pois as implementações feitas diferem consideravelmente. Enquanto o filtro está implementado na ferramenta WEKA e o algoritmo de SVR está implementado pela ferramenta LIBSVM, os algoritmos de redes neurais foram implementados pelo próprio autor na linguagem do ambiente MATLAB©. WEKA e LIBSVM já possuem anos de desenvolvimento e são implementadas em linguagens de programação otimizadas para uma execução mais rápida, como Java e C++. Já a linguagem oferecida pelo MATLAB© é otimizada para facilitar as operações matemáticas, mas tem um desempenho pior do que Java e C++ em instruções como comandos condicionais, laços, etc..

6.1.2 Técnicas de aprendizado de máquina

Realizaram-se dois experimentos comparativos dos modelos de aprendizado de máquina — MLP, MLPP, SVR e Regressão Linear — para as duas bases de dados disponíveis e considerando as variáveis de entrada selecionadas por filtro ou as selecionadas por envoltório. Os resultados são apresentados no Capítulo 5 e, no entanto, devido à divergência dos desempenhos, não é possível concluir qual é a melhor proposta de solução para o problema (segunda questão retomada no início do capítulo).

Há duas limitações nos experimentos: (1) a existência de apenas duas bases de dados para o experimento e os seus tamanhos reduzidos impediram de realizar experimentos que satisfaçam o pré-requisito mínimo para uma posterior avaliação estatística, que é a independência das medidas experimentais; e (2) a Base 2 tinha apenas três variáveis para uso pelos modelos de seleção e predição, distante assim da disponibilidade de onze variáveis da Base 1.

Para responder à terceira questão retomada no começo do capítulo, em comparação com outros métodos da literatura, os resultados com a Base 1 apontaram bons desempenhos para a regressão por vetor de suporte e para a rede MLP ponderada, considerando as variáveis selecionadas por envoltório.

O modelo SVR sobressaiu possivelmente graças à sua capacidade reconhecida de realizar mais facilmente mapeamentos não-lineares do que as redes neurais artificiais. O grau de contorção da função de aproximação sintetizada em uma RNA é determinado manualmente pelo número de neurônios e camadas ocultas que a rede possui [7], o que muitas vezes implica uma tarefa complexa e com o risco de causar sobreajuste no modelo, comprometendo a capacidade de generalização. Por outro lado, uma máquina de vetor de suporte tem o ajuste do número de funções núcleo feito de forma automática e com a restrição de regularização embutida no treinamento; as funções núcleo são componentes análogos ao neurônio artificial na tarefa de criar mapeamentos não-lineares — assim o ajuste de contorção do mapeamento na regressão por vetor de suporte ocorre independentemente do usuário e por isso com maiores chances de ser ajustado para garantir uma boa generalização.

O modelo de rede neural MLPP apresentou um bom resultado na primeira base apesar das dificuldades de ajuste citadas anteriormente. A proposta de uso de uma função custo assimétrica durante o treinamento pode ter sido responsável por essa melhoria, mas são necessários mais experimentos e estudos.

Observando os resultados dos experimentos com a Base 2, os modelos não-lineares obtiveram um desempenho muito aquém do desejado; somente o modelo de regressão linear obteve medidas razoáveis. Ressalte-se também que o modelo de regressão linear se mostrou competitivo ainda nos experimentos com a primeira base, principalmente na comparação utilizando a métrica MARE. Portanto, não é possível descartar a hipótese de que o problema pode ser solucionado por esta abordagem, desde que se disponha de uma base de dados apropriada.

É possível que os resultados para a segunda base fossem diferentes caso houvesse a disponibilidade das mesmas variáveis que foram coletadas para a primeira base. Afinal, a síntese de um modelo preditor requer que os dados possuam qualidade, tanto em número de amostras como em informações disponíveis para entrada [58]. Por isso é proposta a metodologia de escolha das métricas e coleta dos dados realizada para a Base 1, seguida pela seleção de variáveis.

Os resultados, em suma, confirmam a alta complexidade de solucionar o problema de estimar esforço, seja no processo de *software* como um todo, seja na execução de testes. Uma vez que, do ponto de vista prático, os usuários necessitam métodos de estimativas eficientes e fáceis de utilizar, os experimentos realizados não apontam nenhum modelo que atenda estes requisitos.

6.2 Discussões dos resultados obtidos

Apesar dos resultados do trabalho não terem levado a uma solução geral para o problema apresentado, pode-se conjeturar o seguinte:

- A aplicação do algoritmo de envoltório para selecionar variáveis pode melhorar a qualidade das estimativas; ao menos ela valida o conjunto de variáveis escolhido para fazer parte da base de dados;

- O desempenho do modelo de regressão linear, que foi competitivo na Base 1 e o melhor na Base 2, sendo que nesta última havia uma considerável restrição de variáveis, indica que ele pode ser bem sucedido como uma ferramenta de estimativa de esforço de testes em ambientes com pouca informação disponível e cuja prioridade é a simplicidade de uso, em detrimento da eficácia nas estimativas;

- Em ambientes com bases de dados mais robustas, contendo dez ou mais métricas do processo de testes e pelo menos 30 registros de dados, os modelos de regressão por vetor de suporte e rede neural MLP ponderada são recomendados para fornecer resultados mais precisos do que o modelo de regressão linear.

A principal contribuição deste trabalho para a área de Engenharia de *Software* está no estudo das técnicas de Redes Neurais Artificiais e Máquinas de Vetor de Suporte, associadas ao uso de seleção de variáveis, para solucionar o problema de estimar esforço de execução de testes funcionais. Como contribuições secundárias, este estudo propõe um processo para estimação de esforço e o uso de técnicas para remoção de qualquer caráter subjetivo nos dados que compõem a base de treinamento do processo.

Também é contemplado neste processo: uma proposta para escolha de dados e seleção de variáveis como parte do preparo prévio ao treinamento de um modelo preditor; e a adoção de uma função custo assimétrica no treinamento de redes MLP. O processo pode ser sintetizado pela realização dos seguintes passos, detalhados ao longo desta dissertação:

1. Estabelecer uma base de dados com variáveis objetivas e que estejam vinculadas às quatro dimensões de influência na determinação do esforço: a complexidade dos CTs; a complexidade do sistema de *software* em teste; a capacidade da equipe de testes; e a probabilidade de se provocar falhas.

2. Coletar o maior número possível de dados para a base, como sugestão pelo menos 30, considerando um número razoável de variáveis, pelo menos 10.

3. Selecionar as variáveis relevantes para predição pela técnica por envoltório, descartando as demais da base e do processo futuro de coleta.

4. Considerando as características levantadas neste trabalho e os requisitos de uso, adotar um dos três modelos para predição do esforço: regressão linear, rede neural artificial do tipo MLP ponderada, ou regressão por vetor de suporte.

5. Treinar o modelo preditor com a base de dados.

6. Utilizar o modelo treinado para efetuar as estimativas, realimentando a base com novos dados e repetindo o treinamento do modelo periodicamente.

6.3 Trabalhos futuros

São levantadas as seguintes perspectivas futuras para este trabalho:

- Para aprofundar qualquer estudo metódico que compare distintos algoritmos de aprendizado de máquina, várias bases de dados são necessárias para se ter resultados independentes do experimento de comparação. Por isso pretende-se repetir a metodologia experimental para mais bases de dados de diferentes organizações, com um número maior de amostras, de forma a ter condições de estabelecer comparações suportadas por testes estatísticos para chegar a conclusões mais amplas.

- A proposta de uso de uma função custo assimétrica no treinamento da rede MLP pode ser considerada um passo inicial na direção de se tentar descobrir qual é a função que modela o impacto em uma equipe de se fazer uma estimativa de esforço abaixo ou acima do real. As funções quadráticas têm sido adotadas tradicionalmente, porém é possível que funções lineares ou de outra natureza representem melhor o impacto de erros de estimativa em outros cenários de equipes. Portanto tem-se como trabalho futuro aprofundar o estudo sobre o uso de funções custo assimétricas na predição de esforço, por meio de experimentos específicos de calibração dos parâmetros de simetria e formato para distintas realidades de equipes de teste.

- A técnica de filtro CFS adotada no estudo usa conceitos de correlação linear para selecionar os subconjuntos adequados de variáveis; os subconjuntos selecionados obtiveram um desempenho na qualidade de estimativas pior do que os subconjuntos selecionados pelo método por envoltório. Entretanto, futuramente, pode-se estudar outros métodos de seleção de variáveis por filtro, como filtros que utilizam conceitos de teoria da informação [46] ou abordagens distintas de correlação [97] e que talvez sejam melhores do que a técnica por envoltório.

- Dado que se possa futuramente sintetizar e ajustar modelos de aprendizado de máquina capazes de produzir estimativas de esforço de execução de testes de boa qualidade, será necessário realizar o estudo comparativo deles sob a ótica do tempo de execução, de forma a viabilizar o uso prático de uma eventual ferramenta baseada nestes estudos.

- É possível que o problema de estimar esforço de execução de testes seja melhor abordado se for possível combinar, de alguma maneira, as soluções apresentadas por diferentes algoritmos de predição. Na situação ideal, a solução combinada dos algoritmos seria superior a qualquer solução individual apresentada. Para avaliar esta hipótese, tem-se como perspectiva futura o estudo de técnicas de Comitês de Máquinas [33, 90] como média de *Ensemble* ou Mistura de Especialistas, na solução do problema de estimar esforço.

Referências Bibliográficas

[1] Albrecht, A. J.: *Measuring application development productivity*. Procedures of IBM Applic. Dev. Joint SHARE/GUIDE Symposium, 1:83–92, 1979.

[2] Almeida, E.R.C. de, B.T. de Abreu e R. Moraes: *An Alternative Approach to Test Effort Estimation Based on Use Cases*. Proceedings of the International Conference on Software Testing Verification and Validation, 2009. ICST '09., 1:279–288, Abril 2009.

[3] Aranha, E. e P. Borba: *An Estimation Model for Test Execution Effort*. Proceedings of First International Symposium on Empirical Software Engineering and Measurement., 1, 2007.

[4] Barnett, V. e T. Lewis: *Outliers in statistical data*. Wiley New York, 1994.

[5] Becker, S.: *Unsupervised learning procedures for neural networks.* International Journal of Neural Systems, 2(1):17–33, 1991.

[6] Binder, R. V.: *Testing Object-Oriented Systems: Models, Patterns, and Tools.* Addison-Wesley Professional, 2000.

[7] Bishop, C. M.: *Neural networks for pattern recognition*. Oxford University Press, 1995.

[8] Boehm, B. W.: *Software Engineering Economics.* Prentice-Hall, 1981.

[9] Breiman, L., J. H. Friedman, R. A. Olshen e C. J. Stone: *Classification and Regression Trees*. Wadsworth and Brooks, 1984.

[10] Burges, C. J. C. e B. Schölkopf: *Improving the Accuracy and Speed of Support Vector Machines*. Advances in Neural Information Processing Systems 9, 1:375–381, 1997.

[11] Burges, C.J.C.: *A tutorial on support vector machines for pattern recognition.* Data mining and knowledge discovery, 2(2):121–167, 1998.

[12] Calzolari, F., P. Tonella e G. Antoniol: *Maintenance and testing effort modeled by linear and nonlinear dynamic systems*. Information and software technology, 43(8):477–486, 2001.

[13] Castro, L. N. de: *Rotina de Busca Unidimensional Simples em MATLAB*, Janeiro 1998.

[14] Castro, L. N. de: *Fundamentals of Natural Computing: Basic Concepts, Algorithms, And Applications*. Chapman & Hall/CRC, 2006.

[15] Chang, C.C. e C.J. Lin: *LIBSVM: a library for support vector machines*, 2001.

[16] Cheatham, T.J., J.P. Yoo e N.J. Wahl: *Software testing: a machine learning experiment*. Proceedings of the 1995 ACM 23rd annual conference on Computer science, 1:135–141, 1995.

[17] Chiricota, Y., F. Jourdan e G. Melançon: *Software components capture using graph clustering*. 11th IEEE International Workshop on Program Comprehension, 2003.

[18] Clarke, J., J.J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, B. Mitchell, K. Rees e M. Roper: *Reformulating Software Engineering as a Search Problem*. IEE Proceedings Software, 150, 2003.

[19] Crone, S.F.: *Training artificial neural networks for time series prediction using asymmetric cost functions*. Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP'02., 5, 2002.

[20] Cybenko, G.: *Approximation by superpositions of a sigmoidal function*. Mathematics of Control, Signals, and Systems (MCSS), 2(4):303–314, 1989.

[21] Decoste, D. e B. Schölkopf: *Training Invariant Support Vector Machines*. Machine Learning, 46:161–190, 2002.

[22] Demšar, J.: *Statistical comparisons of classifiers over multiple data sets*. The Journal of Machine Learning Research, 7:1–30, 2006.

[23] Dolado, J. J. e L. Fernandez: *Genetic Programming, Neural Networks and Linear Regression in Software Project Estimation*. Proceedings of the International Conference on Software Process Improvement, Research, Education and Training, páginas 157–171, 1998.

[24] Fan, R., P. Chen e C. Lin: *Working Set Selection Using Second Order Information for Training Support Vector Machines*. J. Mach. Learn. Res., 6:1889–1918, 2005.

[25] Fenton, N. E. e M. Neil: *Software metrics: roadmap*. ICSE '00: Proceedings of the Conference on The Future of Software Engineering, 1:357–370, 2000.

[26] Finnie, G. R., G. E. Wittig e J. M. Desharnais: *A comparison of software effort estimation techniques: using function points with neural networks, case-based reasoning and regression models*. The Journal of Systems & Software, 39(3):281–289, 1997.

[27] Fletcher, R.: *Practical methods of optimization*. Wiley-Interscience, 2ª edição, 1987.

[28] Gomes, D. T.: *Modelos de redes neurais recorrentes para previsão de séries temporais de memórias curta e longa*. Tese de Mestrado, Unicamp, Instituto de Matemática, Estatística e Computação Científica, Novembro 2005.

[29] Guyon, I. e A. Elisseeff: *An introduction to variable and feature selection*. Journal of Machine Learning Research, 3:1157–1182, 2003.

[30] Hagan, M.T. e M.B. Menhaj: *Training feedforward networks with the Marquardt algorithm*. IEEE Transactions on Neural Networks, 5(6):989–993, 1994.

[31] Hall, M. A.: *Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning*. ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning, 1:359–366, 2000.

[32] Hall, M.A.: *Correlation-based feature selection for machine learning*. Tese de Doutoramento, The University of Waikato, 1999.

[33] Haykin, S.: *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1998.

[34] Hearst, M.A., S.T. Dumais, E. Osman, J. Platt e B. Schölkopf: *Support vector machines*. IEEE Intelligent systems, 13(4):18–28, 1998.

[35] Holland, J.H.: *Adaptation in natural and artificial systems*. MIT Press Cambridge, MA, USA, 1975.

[36] Hopfield, J. J.: *Neural networks and physical systems with emergent collective computational abilities*. Proceedings of the national academy of sciences, 79(8):2554–2558, 1982.

[37] IEEE: *IEEE Std 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology*. IEEE, 1990.

[38] John, G.H., R. Kohavi e K. Pfleger: *Irrelevant features and the subset selection problem*. Proceedings of the Eleventh International Conference on Machine Learning, 129, 1994.

[39] Jørgensen, M. e M. Shepperd: *A systematic review of software development cost estimation studies*. IEEE Transactions on Software Engineering, páginas 33–53, 2007.

[40] Jørgensen, M. e D.I.K. Sjøberg: *Impact of effort estimates on software project work*. Information and Software Technology, 43(15):939–948, 2001.

[41] Karner, G.: *Resource Estimation for Objectory Projects*. Objective Systems SF AB, 1993.

[42] Karush, W.: *Minima of functions of several variables with inequalities as side constraints*. Tese de Mestrado, Dept. of Mathematics, Univ. of Chicago, 1939.

[43] Kirsopp, C. e M. Shepperd: *Making inferences with small numbers of training sets*. IEE Proceedings-Software, 149(5):123–130, Outubro 2002.

[44] Kohavi, R. e G.H. John: *Wrappers for feature subset selection*. Artificial intelligence, 97(1-2):273–324, 1997.

[45] Kohonen, T.: *Self-organization and associative memory*. Springer Information Sciences Series, página 312, 1989.

[46] Koller, D. e M. Sahami: *Toward optimal feature selection*. In *Proceedings of the Thirteenth International Conference on Machine Learning*, páginas 284–292. Citeseer, 1996.

[47] Kruchten, P.: *The rational unified process: an introduction*. Addison-Wesley Longman Publishing Co., 2000.

[48] Kuhn, H. W. e A. W. Tucker: *Nonlinear Programming*. Proceedings of 2nd Berkeley Symposium on Mathematical Statistics and Probabilistics, 1:481–492, 1951.

[49] Kushwaha, D.S. e A. K. Misra: *Software test effort estimation*. ACM SIGSOFT Software Engineering Notes, 33(3):1–5, Maio 2008.

[50] Lorena, A. C. e A. C. P. L. F. de Carvalho: *Introdução às máquinas de vetores suporte (Support Vector Machines)*. Relatório Técnico 192, Laboratório de Inteligência Computacional, ICMC/USP, São Carlos, Abril 2003.

[51] Lorena, A.C., G. Batista, A.C. de Carvalho e M.C. Monard: *Splice junction recognition using machine learning techniques*. Proc. of the Brazilian Workshop on Bioinformatics (WOB), 1:32–39, 2002.

[52] Mattera, D. e S. Haykin: *Support vector machines for dynamic reconstruction of a chaotic system*, páginas 211–241. MIT Press, 1999.

[53] McCulloch, W.S. e W. Pitts: *A logical calculus of the ideas immanent in nervous activity*. Bulletin of Mathematical Biology, 5(4):115–133, 1943.

[54] McGregor, J.D. e S. Srinivas: *A measure of testing effort*. Proceedings of the 2nd conference on USENIX Conference on Object-Oriented Technologies (COOTS), 2:10, 1996.

[55] McGregor, J.D. e D.A. Sykes: *A Practical Guide to Testing Object-oriented Software*. Addison-Wesley Professional, 2001.

[56] Mercer, J.: *Functions of positive and negative type, and their connection with the theory of integral equations*. Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character, páginas 415–446, 1909.

[57] Minsky, M. L. e S. A. Papert: *Perceptrons*. MIT Press, 1969.

[58] Mitchell, T. M.: *Machine Learning*. McGraw-Hill, 1997.

[59] Muller, K.R., A.J. Smola, G. Ratsch, B. Schölkopf, J. Kohlmorgen e V. N. Vapnik: *Predicting time series with support vector machines*. Lecture notes in computer science, 1:999–1004, 1997.

[60] Murtagh, B.A. e M.A. Saunders: *MINOS 5.1 User's Guide*. Relatório Técnico SOL 83-20R, Department of Operations Research, Stanford University, 1987.

[61] Myers, G.J., T. Badgett, T.M. Thomas e C. Sandler: *The art of software testing*. Wiley, 2004.

[62] Nageswaran, S.: *Test Effort Estimation Using Use Case Points*. 14th International Internet & Software Quality Week 2001, 2001.

[63] Ng, S. P., T. Murnane, K. Reed, D. Grant e T. Y. Chen: *A preliminary survey on software testing practices in Australia*. Proceedings of the 2004 Australian Software Engineering Conference, 1:116–125, 2004.

[64] Oliveira, A. L. I.: *Estimation of software project effort with support vector regression*. Neurocomputing, 69(13-15):1749 – 1753, 2006.

[65] Pargas, R.P., M.J. Harrold e R.R. Peck: *Test-data generation using genetic algorithms*. Software Testing Verification and Reliability, 9(4):263–282, 1999.

[66] Parkinson, C.N.: *Parkinson's law, and other studies in administration*. Houghton Mifflin, 1962.

[67] Platt, J. C.: *Fast training of support vector machines using sequential minimal optimization*, páginas 185–208. MIT Press, 1999.

[68] Pressman, R. S.: *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 6ª edição, 2005.

[69] Rakotomamonjy, A.: *Variable selection using SVM based criteria*. The Journal of Machine Learning Research, 3:1357–1370, 2003.

[70] Rosenblatt, F.: *The perceptron: A probabilistic model for information storage and organization in the brain*. Psychological review, 65(6):386–408, 1958.

[71] Rumelhart, D.E. e J.L. McClelland: *Parallel distributed processing: Explorations in the microstructure of cognition, Vol. 1: Foundations*. MIT Press, 1986.

[72] Russell, S.J., P. Norvig, J.F. Canny, J. Malik e D.D. Edwards: *Artificial intelligence: a modern approach*. Prentice Hall Englewood Cliffs, NJ, 1995.

[73] Schölkopf, B., C. Burges e V. Vapnik: *Incorporating invariances in support vector learning machines*. Proceedings of the International Conference on Artificial Neural Networks, 1, 1996.

[74] Schwitter, R.: *English as a formal specification language*. In *Proceedings of the 13th International Workshop on Database and Expert Systems Applications. DEXA02.*, 2002.

[75] Shan, Y., R. I. Mckay, C. J. Lokan e D. L. Essam: *Software project effort estimation using genetic programming*. Proceedings of International Conference on Communications Circuits and Systems, 1:1108–1112, 2002.

[76] Shukla, K. K.: *Neuro-genetic prediction of software development effort*. Information & Software Technology, 42(10):701–713, 2000.

[77] Silva, D.G., B.T. Abreu e M. Jino: *A Simple Approach for Estimation of Execution Effort of Functional Test Cases*. Proceedings of the International Conference on Software Testing Verification and Validation, 2009. ICST '09., 1:289–298, Abril 2009.

[78] Smola, A. J. e B. Schölkopf: *A tutorial on support vector regression*. Statistics and Computing, 14(3):199–222, Agosto 2004.

[79] Smola, A. J., B. Schölkopf e K. R. Müller: *The connection between regularization operators and support vector kernels*. Neural Networks, 11(4):637–649, 1998.

[80] Sommerville, I.: *Engenharia de Software*. São Paulo: Addison-Wesley, 6ª edição, 2003.

[81] Stikkel, G.: *Dynamic model for the system testing process*. Information and Software Technology, 48(7):578–585, 2006.

[82] Stone, M.: *Cross-validatory choice and assessment of statistical predictions*. Journal of the Royal Statistical Society. Series B (Methodological), páginas 111–147, 1974.

[83] Stone, M.: *Cross-validation: A review*. Statistics, 9(1):127–139, 1978.

[84] Üstün, B., W. J. Melssen e L. M. C. Buydens: *Facilitating the application of Support Vector Regression by using a universal Pearson VII function based kernel*. Chemometrics and Intelligent Laboratory Systems, 81(1):29–40, 2006.

[85] Vanderbei, R.J.: *LOQO: An interior point code for quadratic programming*. Optimization methods and Software, 11(1):451–484, 1999.

[86] Vapnik, V. e A. Chervonenkis: *A note on one class of perceptrons*. Automation and Remote Control, 25:821–837, 1964.

[87] Vapnik, V. e A. Lerner: *Pattern recognition using generalized portrait method*. Automation and Remote Control, 24(6):774–780, 1963.

[88] Vapnik, V. N.: *Estimation of Dependences Based on Empirical Data*. Springer, 1982.

[89] Vapnik, V.N.: *The nature of statistical learning theory*. Springer, 2000.

[90] Villanueva, W. J. P.: *Comitê de Máquinas em Predição de Séries Temporais*. Tese de Mestrado, Unicamp, Faculdade de Engenharia Elétrica e de Computação, 2006.

[91] Von Zuben, F. J.: *Modelos Paramétricos e Não-Paramétricos de Redes Neurais Artificiais e Aplicações*. Tese de Doutoramento, Unicamp, Faculdade de Engenharia Elétrica e de Computação, 1996.

[92] Weisberg, S.: *Applied linear regression*. Wiley-Interscience, 2005.

[93] Werbos, P.J.: *Beyond regression: New tools for prediction and analysis in the behavioral sciences*. Harvard University, 1974.

[94] Widrow, B. e M. E. Hoff: *Adaptive switching circuits. 1960 IRE WESCON Convention Record*. New York: IRE, 4:96–104, 1960.

[95] Wilson, D. G. e B. D. Rudin: *Introduction to the IBM optimization subroutine library*. IBM Systems Journal, 31(1):4–10, 1992.

[96] Witten, I. H. e E. Frank: *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2ª edição, 2005.

[97] Yu, L. e H. Liu: *Efficient Feature Selection via Analysis of Relevance and Redundancy*. Journal of Machine Learning Research, 5:1205–1224, 2004.

[98] Zhu, X., B. Zhou, L. Hou, J. Chen e L. Chen: *An Experience-Based Approach for Test Execution Effort Estimation*. The 9th International Conference for Young Computer Scientists - ICYCS, 1:1193–1198, Novembro 2008.

[99] Zien, A., G. Ratsch, S. Mika, B. Schölkopf, T. Lengauer e K. R. Muller: *Engineering support vector machine kernels that recognize translation initiation sites*. Bioinformatics, 16(9):799–807, 2000.