PUCRS

FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

LUCIANO ALVES

RECOMENDAÇÃO DE ALGORITMOS DE APRENDIZADO DE MÁQUINA PARA PREDIÇÃO DE FALHAS DE SOFTWARE POR MEIO DE META-APRENDIZADO

Porto Alegre

2016

PÓS-GRADUAÇÃO - *STRICTO SENSU*

Pontifícia Universidade Católica
do Rio Grande do Sul

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RECOMENDAÇÃO DE ALGORITMOS DE APRENDIZADO DE MÁQUINA PARA PREDIÇÃO DE FALHAS DE SOFTWARE POR MEIO DE META-APRENDIZADO

LUCIANO ALVES

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Rodrigo Coelho Barros

Porto Alegre
2016

Ficha Catalográfica

A474r Alves, Luciano

Recomendação de Algoritmos de Aprendizado de Máquina para Predição de Falhas de Software por Meio de Meta-Aprendizado / Luciano Alves . – 2016.

91 f.

Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Rodrigo Coelho Barros.

1. Predição de Falhas de Software. 2. Aprendizado de Máquina. 3. Meta-Aprendizado. I. Barros, Rodrigo Coelho. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS
com os dados fornecidos pelo(a) autor(a).
Bibliotecária responsável: Salete Maria Sartori CRB-10/1363

Luciano Alves

Recomendação de Algoritmos de Aprendizado de Máquina para Predição de Falhas de Software por Meio de Meta-Aprendizado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre em Ciência da Computação do Programa de Pós-Graduação em Ciencia da Computação, Escola Politécnica da Pontifícia Universidade Católica do Rio Grande do Sul.

Aprovado em 23 de Setembro de 2016.

BANCA EXAMINADORA:

Profa. Dra. Sabrina dos Santos Marczak (PPGCC/PUCRS)

Profa. Dra. Ana Trindade Winck (UFSM)

Prof. Dr. Rodrigo Coelho Barros (PPGCC/PUCRS - Orientador)

DEDICATÓRIA

Aos meus filhos Luiza e joaquim, a minha esposa Raquel e minha mãe Ivone.

AGRADECIMENTOS

À minha família, pelo apoio em todos os momentos de minha vida, em especial para minha esposa que apesar dos momentos difíceis, ela soube me aturar (olha que não foram poucos) durante minha trajetória acadêmica. Minha filha Luiza, ao qual passei algumas noites em claro cuidando dela quando estava doente e, ao mesmo tempo, estudando para as provas, fazendo trabalhos e escrevendo a dissertação.

Aos meus afiliados e sobrinho, às minhas irmãs, cunhados, sogra, em especial a minha mãe que nunca deixou de acreditar em mim. E olha, que ela rezou todos os dias, fazendo com que eu continuasse a acreditar em Deus, enfim cheguei até aqui né.

Ao meu orientador Prof. Rodrigo Coelho Barros que me ensinou muito durante toda essa trajetória. Ele sempre demonstrou muita capacidade técnica, sempre tentou extrair o melhor para este trabalho. Teve muita paciência para eu me manter no foco e pelas críticas que me fizeram aprender ainda mais com este trabalho. Além disso, agradeço pelo esforço em conseguir recursos que financiassem as taxas para minha pesquisa.

À colega Silvia N. das Dôres, por termos escrito um artigo juntos e por todas as sugestões, críticas e revisões ao trabalho. Também agradeço ao GPIN, pela companhia em alguns finais de semana e horários noturnos além dos dias em que eu podia estar lá. As sugestões nas prévias foram muito importantes. Aos colegas da PUCRS por dividirem experiências durante estes dois anos de aprendizado.

Ao Prof. Duncan que me apresentou ao Prof Rodrigo, trazendo uma oportunidade de realizar o meu sonho de fazer me tornar um mestre em ciência da computação.

À HP Inc., por deixar eu realizar horários alternativos nos dias em que eu estava em aulas pela parte da manhã. Agradeço muito a todos os meus colegas, em especial o colega Márcio Bortolini que me passou dicas de traduções para as escritas em inglês e também auxiliou nas revisões de português da escrita do trabalho.

À todos os que torceram por mim, muito obrigado.

RECOMENDAÇÃO DE ALGORITMOS DE APRENDIZADO DE MÁQUINA PARA PREDIÇÃO DE FALHAS DE SOFTWARE POR MEIO DE META-APRENDIZADO

RESUMO

A predição de falhas de software é uma parte significativa da garantia de qualidade do software e é normalmente utilizada para detectar módulos propensos a falhar baseados em dados coletados após o processo de desenvolvimento do projeto. Diversas técnicas de aprendizado de máquina têm sido propostas para geração de modelos preditivos a partir da coleta dos dados, porém nenhuma se tornou a solução padrão devido as especificidades de cada projeto. Por isso, a hipótese levantada por este trabalho é que recomendar algoritmos de aprendizado de máquina para cada projeto é mais importante e útil do que o desenvolvimento de um único algoritmo de aprendizado de máquina a ser utilizado em qualquer projeto. Para alcançar este objetivo, propõe-se nesta dissertação um *framework* para recomendar algoritmos de aprendizado de máquina capaz de identificar automaticamente o algoritmo mais adequado para aquele projeto específico. A solução, chamada FMA-PFS, faz uso da técnica de meta-aprendizado, a fim de aprender o melhor algoritmo para um projeto em particular. Os resultados mostram que o *framework* FMA-PFS recomenda tanto o melhor algoritmo, quanto o melhor *ranking* de algoritmos no contexto de predição de falhas de software.

Palavras-Chave: Predição de Falhas de Software, Aprendizado de Máquina, Meta-Aprendizado.

RECOMMENDATIONS OF MACHINE LEARNING ALGORITHMS FOR DEFECT PREDICTION BY METALEARNING

ABSTRACT

Software fault prediction is a significant part of software quality assurance and it is commonly used to detect faulty software modules based on software measurement data. Several machine learning based approaches have been proposed for generating predictive models from collected data, although none has become standard given the specificities of each software project. Hence, we believe that recommending the best algorithm for each project is much more important and useful than developing a single algorithm for being used in any project. For achieving that goal, we propose in this dissertation a novel framework for recommending machine learning algorithms that is capable of automatically identifying the most suitable algorithm according to the software project that is being considered. Our solution, namely FMA-PFS, makes use of the metalearning paradigm in order to learn the best learner for a particular project. Results show that the FMA-PFS framework provides both the best single algorithm recommendation and also the best ranking recommendation for the software fault prediction problem.

Keywords: Software Fault Prediction, Machine Learning, Metalearning.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AB. – *Ada Boost*

AM. – Aprendizado de Máquina

MLP. – *Multilayer Perceptron*

MS. – Métricas de Software

NB. – *Naïve Bayes*

PFS. – Predição de Falhas de Software

RF. – *Random Forest*

SVM. – *Support Vector Machine*

XGB. – *Extreme Gradient Boost*

SUMÁRIO

1. INTRODUÇÃO

Uma falha de software é um problema estrutural de um sistema que pode acarretar em uma falha total de um produto de software. Dentro do processo de desenvolvimento de software, a predição de uma falha é uma atividade muito importante para garantir a qualidade do produto, reduzir o esforço de manutenção e priorizar a entrega do software [Mal15, SSZ12].

Identificar e eleger os módulos de software mais propensos a falha pode ajudar os gerentes de projetos a tomar decisões estratégicas sobre esses módulos, poupando tempo de testes e revisão de código e, reduzindo custos e tempo de entrega, aumentando a confiabilidade e a qualidade de um sistema de software. Por essas razões a Predição de Falhas de Software (PFS) vem sendo um relevante tópico na área de Engenharia de Software há mais de 30 anos [SJS+11].

Para que se consiga prever se um software é propenso a falhas, é preciso que se colete dados após o processo de desenvolvimento do software [Pon08]. Tais dados, são as métricas de software que contém informações relacionadas ao código desenvolvido. No início dos anos 90, gerentes de qualidade coletavam essas métricas para verificar e validar a qualidade de um software. Juntamente com as métricas de software, os testadores de cada um dos módulos desenvolvidos relacionavam as falhas encontradas por eles. Assim, cada módulo de software desenvolvido era rotulado como um módulo falho, caso contivesse falhas durante as fases de teste.

Já no final dos anos 90, identifica-se que as métricas de software juntamente com a identificação das falhas poderia ser explorada por pesquisadores da área de Engenharia de Software para encontrar futuros módulos propensos a falhas mesmo antes de serem testados. Uma das maneiras de se encontrar módulos propensos a falha é utilizando técnicas de Aprendizado de Máquina (AM). AM pode ser definida como o campo de pesquisa, fundamentado em Inteligência Artificial e Estatística [Mit97]. Com esse fundamento torna-se possível aprender com as métricas de software e os módulos classificados previamente e assim identificar novos módulos a estarem propensos ou não a falhas através da experiência passada.

A utilização de AM para descobrir módulos propensos a falhas pode prover resultados consideráveis a um baixo custo para as empresas de software. Com isso, diversas técnicas (algoritmos) de AM foram aplicadas, tais como: métodos baseados em árvore [MGF07, SJS+11, GMCS04], as abordagens baseadas em analogia [EBGR01, KS03], redes neurais [KAHA97, QT03], métodos bayesianos [BWH08, TB09], dentre outras.

Embora tenham sido aplicados diversos algoritmos de AM para o domínio de PFS, não se têm um consenso da melhor técnica a ser utilizada. O fato de que nenhum algoritmo é melhor do que todos os outros algoritmos para cada conjunto de dados de PFS

é facilmente explicado pelo teorema "no-free-lunch"(NFL) [WM95, Wol01], segundo o qual uma vantagem obtida por um algoritmo para um específico conjunto de dados não deve ser levada em consideração quando aplicada em outro conjunto de dados. Tal constatação demonstra a importância de recomendar algoritmos de acordo com o domínio de aplicação. No que diz respeito à PFS, os trabalhos de [MGF07, LBMP08, SJS+11] corroboram esta afirmação a partir da realização de diversas análises experimentais.

Menzies et al. [MGF07] empiricamente comprovaram que utilizando o algoritmo Naïve Bayes (NB) e um pré-processamento dos valores numéricos nas base de dados supera-se os métodos baseados em árvore e também outras abordagens citadas anteriormente. Além disso, esse estudo demonstrou que a escolha do melhor algoritmo é mais importante do que a escolha de um subconjunto dos dados disponíveis. No entanto, Lessman et al. [LBMP08] concluíram empiricamente, através da comparação de 22 classificadores de AM e mais 10 conjuntos de dados públicos da NASA, que a importância de se escolher o melhor algoritmo é irrelevante, uma vez que os 17 melhores algoritmos de AM não obtiveram diferença significativa em seus desempenhos. Já, Song et al. [SJS+11] propuseram um *framework* que combina técnicas de pré-processamento de dados com técnicas de aprendizado de máquina para definir qual dos esquemas deve ser utilizado para um determinado conjunto de dados a ser analisado.

Duas coisas ficam evidentes sobre PFS: em primeiro lugar, nenhuma técnica de predição de falhas de software desponta sobre as outras [LBMP08], e em segundo lugar, a PFS é dificultada pela utilização de poucos conjuntos de dados nos experimentos e de poucas técnicas de pré-processamento, esquemas de validação e estatísticas de desempenho [LBMP08, MSS05, RHC10, SJS+11]. Essas duas evidências fazem com que não exista uma relação de concordância sobre o tema, e assim uma dificuldade em compartilhar código e algoritmos [IHGC12]. Em outras palavras, é difícil para pesquisadores e profissionais da área de Engenharia de Software terem uma noção bem clara de qual técnica de predição empregar na busca da realização dos problemas ocasionados por falhas de software [SBH14].

Com base no contexto apresentado, verifica-se que um desafio científico atual no desenvolvimento de soluções para a predição de falhas de software utilizando técnicas de AM não seria a construção de um modelo que se adapte a todos os contextos de desenvolvimento possíveis, mas sim *"como atribuir um algoritmo que construa um modelo mais adequado para um determinado problema"*, dada a existência de diversos algoritmos que se mostram satisfatórios para variadas situações.

A fim de resolver a questão de qual algoritmo de AM pode ter mais eficácia para qualquer projeto de software, esta dissertação propôs criar um *framework* para recomendação de algoritmos de AM por meio da técnica de meta-aprendizado. Visando apoiar a resolução deste desafio, este trabalho propõe a utilização de meta-aprendizado para a seleção de algoritmos de AM para o domínio de PFS.

De maneira genérica, meta-aprendizado pode ser entendido como a utilização de técnicas de AM para a construção de modelos que expliquem o relacionamento entre estratégias de aprendizagem e problemas, segundo alguma perspectiva [VGcSB05]. Sendo assim, meta-aprendizado consegue explorar conhecimento acumulado sobre diversas tarefas e aplicá-lo à resolução de problemas semelhantes [GCVB04]. Com isso, meta-aprendizado pretende determinar sob quais condições cada algoritmo é mais apropriado, possivelmente ampliando o entendimento do mesmo e levando a sugestões de uso mais adequadas.

No contexto de PFS essa solução seria aplicada para selecionar o(s) algoritmo(s) de AM mais adequado(s) para auxiliar o gerente de projetos a identificar de maneira mais eficaz se um novo projeto de software contém módulos propensos a falhas. Espera-se, com isso, contribuir para a ampliação do conhecimento científico referente à meta-aprendizado no contexto de Engenharia de Software, e estimular seu uso no contexto real de desenvolvimento de software para apoiar a predição de falhas de software.

1.1 Motivação

Para garantir a qualidade de um software, pode-se realizar vários esforços, tais como: inspecionar e revisar o código desenvolvido, desenvolver testes unitários ou criar casos de teste, prever falhas de software dentre outros. A predição de falhas de software consegue auxiliar em uma correção mais efetiva dos módulos de software falhos, consegue contribuir com os revisores de código em uma inspeção e principalmente auxiliar o especialista de domínio nas razões pela qual uma falha ocorre frequentemente em um determinado módulo de software [MGF07].

A motivação para esta pesquisa traz esse conjunto de problemas que a predição de falhas de software consegue atacar para garantir a qualidade de software. A PFS pode antecipar os problemas ocasionados no desenvolvimento de software e fazer com que eles sejam encarados de forma mais simples e sem onerar as estimativas de custo realizadas previamente. Além de contribuir para a qualidade de software, esta pesquisa pode trazer resultados superiores aos resultados obtidos até o presente momento, pois foram analisados uma variedade de conjunto de dados de vários projetos de software, que até então não foram analisados em outras pesquisas. Com esta análise mais diversificada, engenheiros de software podem ter mais confiança em empregar as técnicas e o *framework* apresentado em detrimento de outras técnicas utilizadas até o presente momento.

1.2 Objetivos

O objetivo geral desta pesquisa é desenvolver um *framework* de meta-aprendizado chamado FMA-PFS para recomendar algoritmos de AM para a predição de falhas de software. Os objetivos específicos relacionados são:

1. Extrair e pré-processar conjuntos de dados do repositório de dados públicos tera-PROMISE [Men15]. Esta etapa consiste em trazer o maior número de dados referente a projetos desenvolvidos por diversas empresas e que tenham sido disponibilizados em algum repositório de dados.
2. Executar técnicas de meta-aprendizado para extrair informações dos conjuntos de dados citados acima, a fim de coletar dados que representam as características destes conjuntos.
3. Aplicar algoritmos de AM sobre o conjunto de dados coletado. Experimentar técnicas e conceitos relacionados as pesquisas realizadas para PFS.
4. Identificar as medidas de desempenho de algoritmos mais usadas para PFS.
5. Criar uma única base de dados no nível meta. Tal base de dados deve ter toda a informação das características dos conjuntos de dados públicos e também os resultados dos desempenhos de cada algoritmo de AM conforme as medidas de desempenho investigadas.
6. Identificar algoritmos que possam ser capazes de recomendar algoritmos de AM através da experiência da meta-base de dados desenvolvida.
7. Tentar obter resultados superiores aos métodos existentes segundo as métricas especializadas para o domínio do problema.

1.3 Organização do Volume

Este trabalho está dividido em 5 capítulos:

- Capítulo 2: Apresenta a fundamentação teórica da pesquisa.
- Capítulo 3: Apresenta a visão geral do *framework* proposto e detalha todas as etapas realizadas para o desenvolvimento do *framework* de recomendação de algoritmos de AM.
- Capítulo 4: Apresenta os experimentos para validar as hipóteses levantadas.

- Capítulo 5: Apresenta as considerações finais e trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é apresentado o referencial teórico que constitui a base desta pesquisa, dividido nos seguintes tópicos: i) Visão geral sobre qualidade de software e a predição em falhas de software; ii) Aprendizado de máquina e meta-aprendizado para recomendar algoritmos de AM; iii) Por fim, serão apresentados os trabalhos relacionados a esta dissertação e as considerações finais.

2.1 Uma Visão Geral sobre Qualidade de Software

A principal premissa da Engenharia de Software é produzir um software de qualidade. Qualidade de software é o certificado para que um determinado módulo ou produto de software atenda todos os seus requisitos especificados [iee90]. Em outras palavras, a qualidade de software consiste de um conjunto de atividades pré-estabelecidas para proporcionar uma confiança no software de que todas as especificações exigidas tenham sido desenvolvidas, ou que estejam de acordo com o que foi exigido pelo cliente [iee90]. Dentre algumas atividades de qualidade de software, pode-se mencionar: revisão de código, refatoramento do código, testes, medidas de impacto das mudanças de código, desenvolvimento contínuo (falhas, melhorias e novas *features*), gerenciamento de configuração, gerenciamento das versões, integração do produto, dentre outras. Além disso, uma outra forma de se garantir ou aumentar a qualidade de um software é a predição de falhas de software [SSZ12].

2.1.1 Predição de Falhas de Software

Prever falhas em um software é saber se um software está propenso a falhas antes de ser entregue ao usuário final. Muitos pesquisadores têm dedicado esforços para descobrir algo apropriado para prever falhas em software. Em outras palavras, é preciso que exista uma maneira de antecipar a localização das falhas de um software.

A antecipação na localização das falhas em um software aumenta as chances de que as atividades de testes têm de encontrá-los e realizarem as ações corretivas de maneira rápida, efetiva e preventiva. Já se sabe que as atividades de teste de um software tem uma parcela considerável nos custos totais de um projeto, mas segundo Tosun [TTB08], existe a redução de aproximadamente 29% dos esforços de teste quando as atividades são direcionadas através da predição de falhas. Além da redução dos esforços de teste, a utilização de predição de falhas traz os seguintes benefícios [Pon08]:

- Suporte ao planejamento e execução de testes;
- Identificação de pontos de melhoria no código desenvolvido através do diagnóstico de trechos de código complexos que podem ser simplificados;
- Redução da taxa de inserção de falhas na manutenção e evolução do software;
- Planejamento do esforço de desenvolvimento de iterações futuras do projeto.

Para que uma empresa de software possa garantir esses benefícios, torna-se necessário entender como funciona o processo de prevenção de falhas em um âmbito geral do desenvolvimento de software. A Figura 2.1 exemplifica o processo e as etapas em que a predição de falhas de software ocorre. Suponha que uma empresa A queira incorporar a fase de predição em falhas de software em seus projetos. A empresa geralmente coleta as métricas de software ao final do ciclo de desenvolvimento de seu produto. As métricas de software contêm informações relativas a cada módulo desenvolvido. O especialista de domínio escolhe um algoritmo de aprendizado de máquina X para prever falhas de software baseado nas informações coletadas. Com base nas informações obtidas da predição de falhas, o especialista de domínio define quais módulos de software devem ter uma atenção maior, concentrando esforços nos módulos que estão propensos a falhas. Para que todo este processo seja bem sucedido, a coleta das métricas de software torna-se uma das etapas importantes, e portanto é apresentada a seguir.

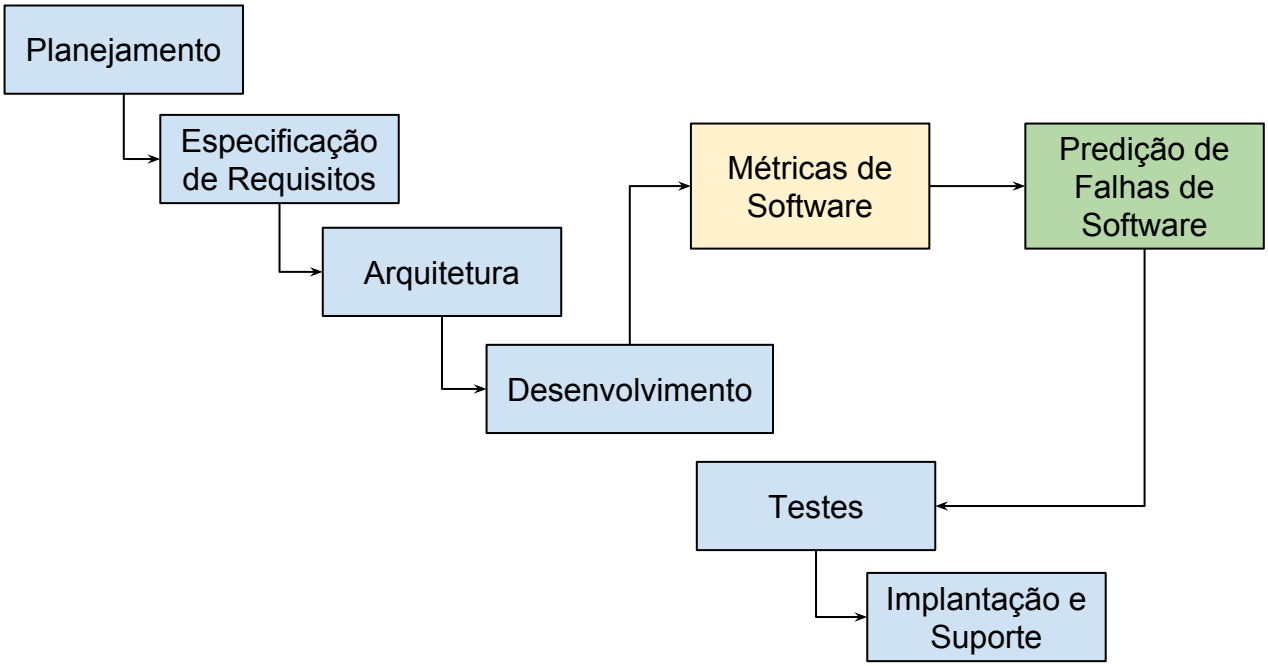

Figura 2.1: Processo de Desenvolvimento de Software com ênfase na etapa de PFS. Adaptado de [Som04].

2.1.2 Métricas de Software

Uma métrica de software é definida como uma medida quantitativa do grau em que um sistema, componente ou processo possui um determinado atributo [iee90]. Segundo Couto [Cou13], as métricas de software podem ser ilustradas como uma função que recebe como entrada os dados de software e retorna um valor numérico que representa o grau em que o software possui uma determinada qualidade. Normalmente as métricas fornecem uma visão que permitem que os engenheiros de software possam ajustar e controlar produtos de software, projetos e processos [Pre01]. As métricas geralmente são classificadas em três categorias: métricas de processo, métricas de projeto e métricas de produto [Kan02, Pre01], descritas a seguir:

- Métricas de Processo: permitem que a empresa possa avaliar o processo utilizado para desenvolver um sistema de software. Mais especificamente, um processo de software define técnicas, métodos de gestão, ferramentas, pessoas e tarefas relacionadas com o desenvolvimento de software. Com isso, as métricas de processo podem ser usadas para melhorar o desenvolvimento de software e práticas de manutenção. Exemplos de métricas de processo podem ser erros por pontos de função, números de arquivos envolvidos na correção de *bugs*, etc.

- Métricas de Projeto: permitem que um gerente e sua equipe possam adaptar o fluxo de trabalho do projeto e as atividades técnicas. Basicamente, métricas de projeto descrevem as características do projeto e sua execução. Número de desenvolvedores, custos, prazos e produtividade são exemplos.

- Métricas de produto: permitem que os engenheiros de software possam avaliar as propriedades internas de um produto de software. Os exemplos de métricas de produto podem ser tamanho, complexidade, acoplamento, coesão e herança do produto de software.

2.1.3 Métricas para Predição de Falhas de Software

As métricas de software para PFS, segundo Catal e Diri [**?**], se subdividem em seis categorias: i) método; ii) classe; iii) componente; iv) arquivo; v) processo; vi) quantitativas. Dentre essas seis categorias, duas delas método e classe são as categorias mais utilizadas para a PFS. Essas métricas são detalhadas a seguir, pois elas são as métricas dos conjuntos de dados utilizadas nos experimentos desta pesquisa. As outras categorias foram citadas, mas as suas medidas não constam em nenhum dos conjuntos de dados utilizados.

- Métricas em nível de método: Um dos exemplos mais conhecidos na literatura para as métricas baseadas em métodos são as métricas propostas por McCabe [McC76] e Halstead [Hal77] em meados dos anos 70. Estas métricas podem ser coletadas em programas desenvolvidos no âmbito de dois paradigmas (estruturados ou orientado a objetos), pois seus códigos consistem de estruturas baseadas em métodos. Quando as métricas de software em nível de métodos são usadas, elas conseguem predizer módulos propensos a falhas antes da execução dos testes de tal sistema.

- Métricas em nível de classe: Métricas de classe podem ser usadas somente em programas orientados a objetos, pois o conceito de classe é a característica de abstração de um objeto. Chidamber-Kemerer (CK) [CK94] são métricas em nível de classe que foram propostas em 1994 e são as métricas mais conhecidas para a predição de falhas de software. Além das métricas de CK, existem outras métricas como MOOD, QMOOD e L&K. As métricas de MOOD (*metrics for object-oriented design*) foram propostas em 1994 e validadas empiricamente em 1996. Lorenz e Kidd introduziram onze métricas de software que são conhecidas como métricas de L&K. QMOOD (*quality metrics for object-oriented design*) foram propostas por Banysia e Davis em 2002. No entanto, as métricas de CK continuam sendo as mais referenciadas em pesquisas nesta área.

A Tabela 2.1 sumariza as métricas de McCabe e Halstead e as métricas de CK, que são as métricas dos conjuntos de dados utilizados nesta dissertação.

2.2 Aprendizado de Máquina

O Aprendizado de Máquina (AM) é uma das mais importantes sub-áreas da Inteligência Artificial. Segundo Bishop [Bis06] o AM é um campo de pesquisa fundamentado na Inteligência Artificial e na Estatística. Como a complexidade dos problemas a serem tratados e o crescente volume de dados gerados por diferentes setores vem aumentando significativamente nos últimos anos, surgiu a necessidade de ferramentas computacionais mais sofisticadas e mais independentes sem a intervenção humana ou a consulta de um especialista [GFLDC11]. Com isso, a forma mais adequada de se resolver os problemas é fazer uma função ou hipótese em cima de experiências passadas. A esse processo de indução de uma hipótese (ou aproximação de função) a partir da experiência passada dá-se o nome Aprendizado de Máquina [GFLDC11].

Existem várias definições de AM na literatura. Uma delas, apresentada em Mitchel [Mit97], define AM como a capacidade de melhorar o desempenho na realização de alguma tarefa por meio da experiência.

Tabela 2.1: Métricas de Software para Predição de Falhas de Software

Categoria	Nome	Descrição
CK	WMC	A soma da complexidade de cada método de uma classe.
	DIT	Quantidade de classes que herdam de uma classe.
	NOC	Quantidade de subclasses de uma classe.
	RFC	Quantidade de elementos no retorno de uma classe.
	CBO	Quantidade de classes que usam uma determinada classe.
	LCOM	Diferença entre métodos que compartilham e não compartilham variáveis.
McCabe	v(g)	Complexidade ciclomática
	iv(G)	Complexidade Essencial
	ev(G)	Complexidade de Projeto
	LOC	Total de linhas de código (uma linha = um contador)
	LOC_B	Quantidade de linhas em branco
	LOC_CC	Quantidade de linhas de código e comentários
	LOC_C	Quantidade de linhas comentadas
	LOC_E	Quantidade de linhas executáveis
	NOL	Número de linhas de código entre as chaves {}
Halstead	N1	Número total de operadores
	N2	Número total de operando
	m1	Número de operadores únicos
	m2	Número de operando único
	N	tamanho: N = N1 + N2
	V	volume: V = N * log2m
	L	nível: L = V* / V onde V* = (2 + m2*)log2(2 + m2*)
	D	dificuldade: D = 1 / L
	I	conteúdo: I = ^L * V onde ^L = 2 / m1 * m2 / N2
	E	esforço: E = V / ^L
	B	Erro estimado
	T	tempo de programação: T = E / 18 (segundos)

2.2.1 Tarefas de Aprendizado

Segundo Faceli et al. [GFLDC11], os algoritmos de AM têm sido usados para diversas tarefas e podem ser organizados de acordo com diferentes critérios. Um deles é o paradigma de aprendizado que pode ser escolhido para tratar uma tarefa. Essas tarefas podem ser: descritivas e preditivas.

As tarefas descritivas tentam explorar um conjunto de dados e seguem o paradigma de aprendizado não supervisionado [GFLDC11]. Tais tarefas não são exploradas nesta pesquisa, onde o foco são às tarefas preditivas.

Em tarefas preditivas (paradigma de aprendizado supervisionado), o objetivo é obter uma função ou hipótese a partir de dados de treinamento para ser usada na predição de um rótulo (atributo alvo) com base em seus atributos de entrada [GFLDC11]. Para entender melhor como esse paradigma supervisionado acontece, a Figura 2.2 exemplifica como um algoritmo de AM utiliza as informações de entrada para induzir um modelo. Esse exemplo foi adaptado para apresentar um modelo induzido para a predição de falhas de software.

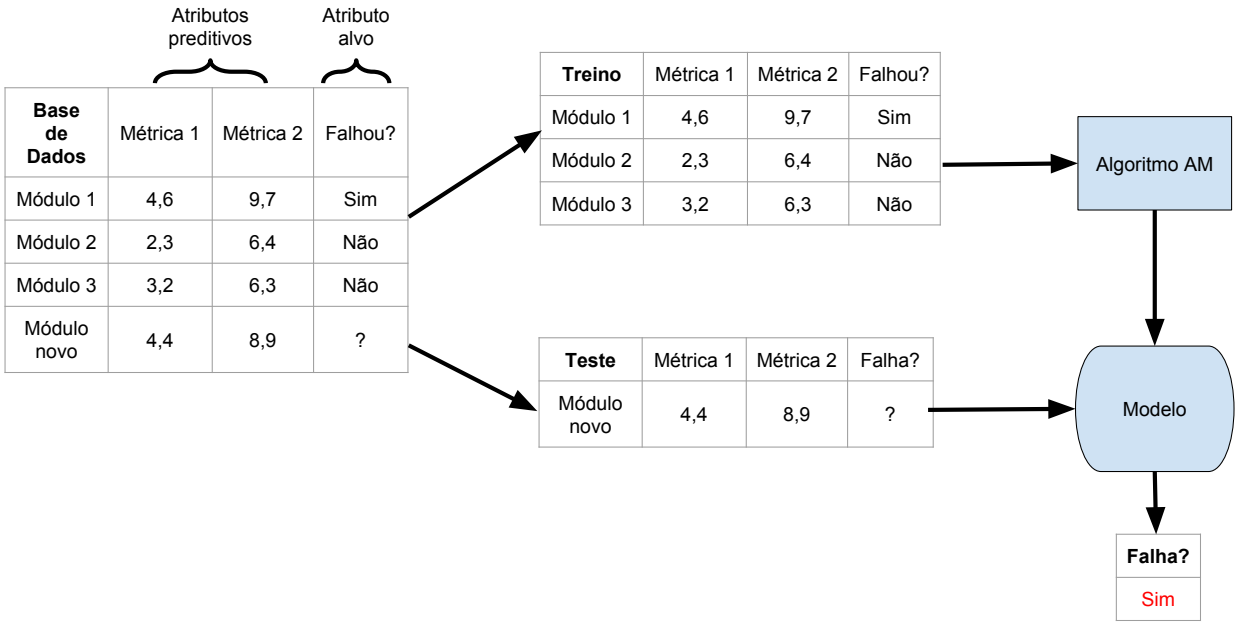

Figura 2.2: Modelo de classificação de um algoritmo de AM para PFS. Adaptado de [TSK05].

Basicamente, o algoritmo de AM recebe um conjunto de treino que contém os atributos preditivos (métricas de software) e o atributo alvo (indica a presença de falhas em um módulo de software). Com essas informações o algoritmo de AM adquire experiências e induz um modelo capaz de prever novos módulos (conjunto de teste) propensos a falhas de software. Com o modelo induzido, a partir de agora para cada novo módulo de dados, as métricas de software (atributos preditivos) são passadas para este modelo e a saída correspondente informa se o módulo está ou não propenso a falha.

2.2.2 Paradigmas Preditivos

Um algoritmo de AM preditivo é uma função que, dado um conjunto de exemplos rotulados, gera um estimador. De acordo com o atributo alvo, um problema supervisionado pode ser de classificação ou regressão. Para um problema de classificação os valores do atributo alvo são finitos e nominais. Já para um problema de regressão seus valores são numéricos, infinitos e ordenados. Um classificador ou regressor, por sua vez, também é uma função que, dado um exemplo ainda não rotulado, atribui este exemplo a uma das possíveis classes (ou a um valor real) [Die98].

Os principais métodos preditivos de AM são organizados da seguinte maneira: i) métodos baseados em distâncias; ii) métodos probabilísticos; iii) métodos baseados em procura; iv) métodos baseados em otimização[GFLDC11].

- **Métodos Baseados em Distâncias** - A estratégia nesse paradigma é a procura pelo vizinho mais próximo. Conhecido também como um algoritmo preguiçoso (*lazy*), pois não aprende um modelo compacto dos dados, apenas memoriza os objetos de treinamento. Uma forma de obter um vizinho mais próximo é através de uma distância, por exemplo, a distância Euclidiana. O *k*-NN é um exemplo deste paradigma.

- **Métodos Probabilísticos** - São baseados no teorema de *Bayes*, os métodos probabilísticos Bayesianos. A probabilidade de um evento A (classe), dado um evento B (conjunto de entrada) não depende apenas de uma relação entre A e B [Mit97], mas da probabilidade de ocorrência do evento B ser estimada pela observação da frequência que esse evento ocorre. *Naïve Bayes* (NB) é um exemplo deste paradigma.

- **Métodos Baseados em Procura** - São baseados em modelo que usam a estratégia de dividir para conquistar, e tentam assim resolver um problema de decisão. Um problema complexo é dividido em problemas mais simples e recursivos. As soluções dos subproblemas podem ser combinadas, gerando o modelo preditivo final. Um exemplo de algoritmo baseado em procura é o C4.5 [Qui93].

- **Métodos Baseados em Otimização** - Realizam a busca da hipótese através da otimização de alguma função. Seu objetivo consiste em minimizar (maximizar) uma função objetivo. Existem duas técnicas populares neste paradigma: as redes neurais artificiais (RNAs) e as máquinas de vetores de suporte (SVMs). As RNAs têm inspiração nas redes neurais biológicas presentes no cérebro, enquanto as SVMs têm suas origens na aplicação de conceitos da Teoria de Aprendizado Estatístico [Vap95].

2.2.3 Avaliação de Paradigmas Preditivos

Em uma aplicação de algoritmos de AM, um conjunto de exemplos é o único conhecimento que se tem de um domínio ou um problema real que se quer resolver. Os principais métodos preditivos brevemente explanados anteriormente são algumas das técnicas a serem utilizadas para a indução de um modelo a partir de um conjunto de exemplos rotulados. Porém, não é possível afirmar que um desses métodos preditivos se sairá melhor na resolução de qualquer tipo de experimentação. Por exemplo, em domínios que o conjunto de exemplos possui alta dimensionalidade, as SVMs são boas candidatas, enquanto que o algoritmo *k*-NN, usando a distância Euclidiana pode, a princípio, não parecer uma solução adequada. Isso evidencia que é preciso realizar experimentação. Essa avaliação experimental de um algoritmo de AM pode ser realizada através da acurácia do modelo gerado, tempo de aprendizado, compreensão do conhecimento, requisitos de armazenamento do modelo, entre outros.

Considerando apenas paradigmas preditivos, que é o foco desta pesquisa, será apresentado apenas as medidas relacionadas ao desempenho obtido nas predições realizadas. Mais especificamente, as medidas relacionadas a problemas de duas classes, pois os conjuntos de dados para a predição em falhas de software foram rotulados para serem propensos ou não a falhas de software.

Problemas de Duas Classes e o Espaço ROC

Considerando um problema de duas classes, uma classe pode ser denotada como sendo a classe positiva (+) e a outra como sendo da classe negativa (-). A Figura 2.3 ilustra uma matriz de confusão, em que:

- VP é o número de verdadeiros positivos (objetos que são da classe positiva e que foram classificados corretamente).
- VN é o numero de verdadeiros negativos (objetos que são da classe negativa e que foram classificados corretamente).
- FP é o número de falsos positivos (objetos que são da classe negativa, mas que foram classificados incorretamente como sendo da classe positiva).
- FN é o número de falsos negativos (objetos que são da classe positiva, mas que foram classificados incorretamente como sendo da classe negativa).

		PREDITO	
		+	-
REAL	+	VP	FN
	-	FP	VN

Figura 2.3: Matriz de Confusão.

Medidas de Desempenho

A partir da matriz de confusão, uma série de outras medidas de desempenho podem ser derivadas. Entre elas, têm-se:

- *Taxa de erro na classe positiva*: proporção de exemplos da classe positiva incorretamente classificados pelo preditor, também conhecida como taxa de falsos negativos (TFN).

$$\mathrm{T}FN = \frac{FN}{VP + FN} \tag{2.1}$$

- *Taxa de erro total*: dada pela soma dos valores da diagonal secundária da matriz, dividida pela soma de todos os elementos da matriz.

$$err = \frac{FP + FN}{n} \tag{2.2}$$

- *Taxa de acerto ou acurácia total*: calculada pela soma dos valores da diagonal principal da matriz, dividida pela soma dos valores de todos os elementos da matriz.

$$ac = \frac{VP + VN}{n} \tag{2.3}$$

- *Precisão*: proporção de exemplos positivos classificados corretamente entre todos aqueles preditos como positivos pelo preditor.

$$prec = \frac{VP}{VP + FP} \tag{2.4}$$

- *Especificidade*: corresponde à taxa de acerto na classe negativa.

$$esp = \frac{VN}{VN + FP} \tag{2.5}$$

- *Taxa de erro na classe negativa*: proporção de exemplos da classe negativa incorretamente classificadas pelo preditor, também conhecida como taxa de falsos positivos (TFP).

$$\mathrm{T}FP = \frac{FP}{FP + VN} \tag{2.6}$$

- *Sensibilidade ou revocação*: corresponde à taxa de acerto na classe positiva. Também é chamada de taxa de verdadeiros positivos (TVP)

$$\mathrm{T}VP = \frac{VP}{VP + FN} \tag{2.7}$$

- *Balance*: Medida proposta por Menzies et al. [MGF07] e utilizada especificamente para o contexto de predição de falhas de software. Conforme ilustrada pela Equação 2.8 a medida *Balance* é calculada por 1 menos a distância Euclidiana normalizada de TFP e TVP. A ideia básica é atingir o ponto correto para TFP = 0 e TVP = 1, ou seja, identificar corretamente os módulos que estão propensos a falhas de software e não indicar alarmes falsos que apresentaria módulos não propensos a falhas como sendo

falhos. Quanto maior ou mais próximo de 1, melhor será o desempenho de avaliação para esta medida.

$$balance = 1 - \frac{\sqrt{(0 - TFP)^2 + (1 - TVP)^2}.}{\sqrt{2}} \tag{2.8}$$

Análise ROC

Uma outra forma de se avaliar classificadores em problemas de duas classes é com o uso das curvas ROC (*Receive Operating Characteristics*), cuja utilização em algoritmos de aprendizado de máquina já se estende por várias décadas [GFLDC11]. O gráfico ROC é um gráfico bidimensional plotado em um espaço chamado ROC, tendo como eixos x e y a taxa de falsos positivos (TFP) Equação 2.6 e a taxa de verdadeiros positivos (TVP) Equação. 2.7, respectivamente. O desempenho de um classificador equivale a um ponto plotado nessa curva e no espaço bidimensional.

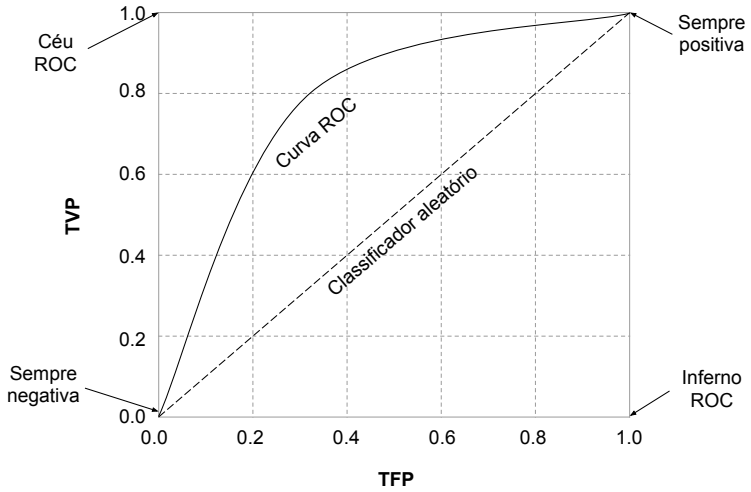

Figura 2.4: Espaço e exemplo de uma curva ROC. Adaptado de [GFLDC11].

Os principais aspectos do espaço ROC foram ilustrados na Figura 2.4. A linha diagonal tracejada representa classificadores com predições aleatórias e um classificador que apresentar um valor abaixo dessa linha pode ser considerado pior do que um aleatório. A figura inclui um exemplo de uma curva ROC. O ponto (0,1) representa as classificações perfeitas e foi denominado de *céu* ROC. O ponto (1,0), que é o oposto, representa o *inferno* ROC. Já o ponto (1,1) representa classificações sempre positivas e o ponto (0,0), classificações sempre negativas.

Com a curva ROC ilustrada na Figura 2.4, é comum comparar o desempenho dos algoritmos através de uma medida única: a área abaixo da curva ROC (AUC, do Inglês *Area Under ROC Curve*). A medida AUC, igualmente ao *Balance*, produz valores entre 0 e 1. Os valores mais próximos de 1 são considerados melhores. A AUC é uma das medidas mais usadas para o domínio de predição de falhas de software [Mal15].

2.3 Meta-Aprendizado

Segundo Faceli et al. [GFLDC11], um dos principais desafios enfrentados quando utiliza-se algoritmos de AM em novos conjuntos de dados é a escolha do melhor algoritmo. Conforme visto anteriormente, vários algoritmos de AM podem ser usados em problemas reais. Apesar de existir uma quantidade enorme de algoritmos, não existem regras que auxiliem na busca do algoritmo mais adequado. Tal escolha pode ocorrer por tentativa e erro ou acontece pela disponibilidade desse algoritmo na ferramenta computacional usada no projeto, ou pelo especialista em AM, ou pela experiência passada do usuário.

Conforme mencionado em [Wol96], não existe um único algoritmo de AM que seja melhor do que os outros, pois nem sempre um algoritmo se adapta a um determinado conjunto de dados. Pesquisas recentes em AM apontam que meta-aprendizado pode ser uma área capaz de determinar qual é o algoritmo mais recomendado para um novo conjunto de dados. Algumas análises experimentais indicam que meta-aprendizado pode não só recomendar algoritmos de AM, como também selecionar os melhores parâmetros de um algoritmo de AM para um conjunto de dados [SBK04, GPS+10, MPCS12]. Entretanto, na pesquisa que foi realizada, a técnica de meta-aprendizado foi especificamente utilizada para recomendar algoritmos de AM para o problema de PFS.

2.3.1 Recomendação de Algoritmos

A Figura 2.5 ilustra o processo de recomendação de algoritmos por meta-aprendizado. O processo inicia-se através da entrada do repositório de dados (Figura 2.5 - Item 1). Geralmente aqui pode-se definir um domínio específico ou repositórios de vários domínios. Em seguida, dois itens são definidos para o Item 1. Um deles é a caracterização de dados (Figura 2.5 - Item 2), que consiste em extrair características de cada uma das bases de dados do repositório de dados. A outra (Figura 2.5 - Item 3) é definida pela avaliação de um conjunto de algoritmos, novamente aplicado sobre o repositório de dados. Através da associação desses dois itens, forma-se uma meta-base, que contém os meta-atributos e o desempenho dos algoritmos (Figura 2.5 - Item 4). Para a meta-base que contém os meta-atributos e o desempenho dos algoritmos(chamado de meta-atributo alvo), escolhe-se um algoritmo de aprendizado de máquina, conhecido como meta-aprendiz (Figura 2.5 - Item 5), que fará a avaliação dessa meta-base para identificar se todo o processo de meta-aprendizado foi bem sucedido. E, por fim, gera-se o modelo de recomendação de algoritmos de aprendizado de máquina (Figura 2.5 - Item 6).

Em um processo de recomendação de algoritmos de aprendizado de máquina, Kalousis [Kal02] fez uma análise de acordo com os seguintes aspectos: i) propriedades

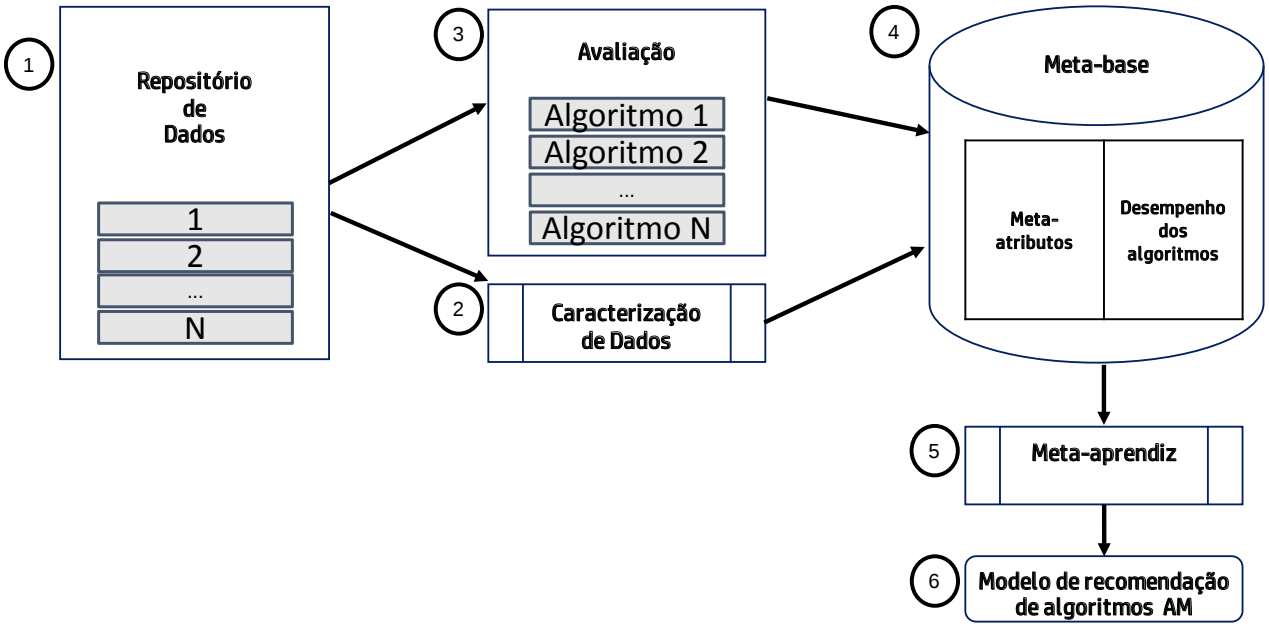

Figura 2.5: Processo de recomendação de algoritmos utilizando meta-aprendizado. Adaptado de [BCSV08].

usadas para caracterização do conjunto de dados; ii) medidas de avaliação dos algoritmos; iii) formas de apresentação de sugestões; iv) métodos para construção das sugestões. A seguir é apresentada uma breve explicação de cada um desses aspectos.

2.3.2 Caracterização do Conjunto de Dados

A caracterização do conjunto de dados tem como objetivo extrair determinadas características dos dados que, de alguma forma, possam influenciar o desempenho dos algoritmos de AM [GFLDC11]. Sendo assim, essas características são utilizadas em meta-aprendizado para recomendar os melhores algoritmos de AM. Uma das formas de se caracterizar um conjunto de dados, por exemplo, é extrair uma medida que demonstre a correlação dos atributos de um conjunto de dados. Suponha que existam dois atributos correlacionados, como peso e altura de um indivíduo. Sabe-se que uma pessoa mais alta tende a ser mais pesada. Se essa correlação entre os atributos ocorre, isso pode influenciar no desempenho do algoritmo. Um exemplo disso é o algoritmo de aprendizado de máquina *Naïve Bayes* (NB), que tem melhor desempenho quando essa correlação entre os atributos não acontece, ou seja, NB possui melhor desempenho com os atributos independentes.

Soares [SBK04] sugere que as medidas extraídas dos conjuntos de dados no nível base do meta-aprendizado precisa conter informação relevante que permita estimar o desempenho entre os algoritmos de AM. De acordo com [VGcSB05], os estudos realizados

em caracterização de dados são divididos nas três abordagens: i) caracterização direta; ii) caracterização por propriedades de modelos; iii) caracterização baseada em *landmarking*.

Caracterização Direta

A caracterização direta é a forma mais simples de retirar propriedades de um conjunto de dados. O projeto STATLOG [BH94] foi um dos primeiros trabalhos à utilizar essas características. Essas características (medidas) foram denominadas de meta-atributos e foram divididas em três classes:

- Medidas simples: são as descrições gerais do conjunto de dados como, por exemplo, número de classes, número de atributos, número de instâncias etc.
- Medidas baseadas em estatística: descrevem estatisticamente os dados, que podem ser valores de obliquidade, valores de curtose, correlação entre atributos por classe, desvio-padrão dos atributos.
- Medidas baseadas em teoria da informação: tenta quantificar a informação presente nos dados, usando medidas como a entropia e informação mútua.

Caracterização por Propriedades de Modelos

Na caracterização por propriedades de modelos, as propriedades de um conjunto de dados são os próprios valores de um modelo induzido. Quando isso ocorre, uma alteração no espaço de busca do processo de meta-aprendizado passa do espaço de objetos para o espaço de modelos [BGCK00]. Para citar um exemplo, Vilalta [VGcSB05] cita a indução de uma árvore de decisão para representar um conjunto de dados. Como propriedades do modelo para representar o conjunto de dados podem ser citadas: número de nós folha, formato da árvore, dentre outras. De acordo com Bensusan et al. [BGCK00], evidências empíricas mostram que propriedades dos conjuntos de dados podem estar ligadas a estruturas de árvores de decisão não podadas.

Caracterização Baseada em *Landmarking*

Na caracterização baseada em *landmarking*, as informações de um conjunto de dados são as informações de desempenho de algoritmos de classificação rápidos e simples. A estratégia é ter diferentes algoritmos de AM que sejam baseados nos métodos preditivos citados anteriormente. Um exemplo seria pegar um algoritmo de AM para cada categoria preditiva. Espera-se que o desempenho dos algoritmos de AM nos conjuntos de dados similares tenham valores aproximados.

2.3.3 Medidas de Avaliação dos Algoritmos

Para avaliar uma determinada base de dados, é preciso que uma medida de desempenho seja atribuída para selecionar um algoritmo de AM [GFLDC11]. Para problemas de classificação, podem ser utilizadas, por exemplo, medidas de acurácia preditiva, como taxa de classificação incorretas, revocação, precisão, medida F e área abaixo da curva ROC (AUC). Essas medidas de avaliação de desempenho dos algoritmos de aprendizado de máquina foram brevemente explanadas na Seção 2.2. Além das medidas de desempenho dos algoritmos de aprendizado de máquina, outras medidas podem ser utilizadas, como por exemplo: custo computacional e quantidade de memória.

2.3.4 Formas de Apresentação de Sugestões

Na forma de apresentação de sugestões, duas etapas distintas devem ser relacionadas. A primeira etapa consiste em definir a quantidade de algoritmos que serão selecionados. É importante ressaltar que cada algoritmo selecionado vai ser induzido para cada conjunto de dados relacionado no nível base. A segunda etapa é apresentar os resultados dos algoritmos ao usuário. De acordo com Kalousis [Kal02], as sugestões podem ser apresentadas de três formas:

- Sugestão do melhor algoritmo: é sugerido o melhor algoritmo de aprendizado de máquina para uma nova base de dados.
- Sugestão de um grupo com os melhores algoritmos: é recomendado um conjunto dos melhores algoritmos de aprendizado de máquina. Nesta forma, além de recomendar o melhor, é sugerido outros algoritmos que tiveram desempenhos estatísticos semelhantes ao melhor algoritmo.
- Sugestão de um *ranking* dos melhores algoritmos: Esta forma também apresenta um conjunto de algoritmos de aprendizado de máquina, porém a sua diferença para a segunda forma é que agora é recomendado um conjunto ordenado de algoritmos, na forma de um *ranking*.

2.3.5 Recomendações a partir da Caracterização

Uma das principais tarefas de meta-aprendizado é associar as características dos conjuntos de dados ao desempenho dos algoritmos de AM para poder recomendar os me-

lhores algoritmos para um novo conjunto de dados [GFLDC11]. Conforme visto recentemente, cada base de dados no nível base forma uma instância (exemplo) no novo conjunto de dados (nível meta). Esse nível contém os meta-atributos que são as características das bases de dados e o atributo alvo que é o resultado do desempenho dos algoritmos de AM. Esse único conjunto de dados é a meta-base a ser usada por um algoritmo de AM (meta-aprendiz) para a indução de um modelo formando assim um sistema de recomendação. O meta-aprendiz é um algoritmo de AM que avalia a meta-base, sendo assim, continua sendo uma técnica de aprendizado de máquina, porém avaliando e gerando um modelo induzido capaz de aprender sobre o aprendizado realizado pela avaliação e o desempenho dos algoritmos durante o processo realizado no nível base dos dados. O modelo induzido (sistema de recomendação) pode ser tanto um regressor (meta-regressor), que associa valores reais dos algoritmos de AM avaliados, quanto um classificador (meta-classificador), classificando o desempenho dos algoritmos de AM com o melhor algoritmo.

2.4 Trabalhos Relacionados

A área de Engenharia de Software com foco em predição de falhas no desenvolvimento de software tem avançado muito nos últimos anos. No entanto, até o presente momento, nenhuma delas mostra-se segura em afirmar que uma determinada técnica e ou algoritmo de Aprendizado de Máquina desempenha melhor sobre um outro modelo. Este trabalho tenta apresentar um novo conceito, que enfatiza ainda mais que não existe um único algoritmo (modelo) para desempenhar o melhor resultado para todas as bases de dados e que esse seja melhor do que todos os outros. Mas pode existir um conjunto de algoritmos pré-estabelecidos e essa pesquisa apresenta um *framework* para automatizar a escolha do(s) melhor(es) algoritmo(s) de AM para uma específica base de dados.

O *Framework* Meta-Aprendizado para Predição de Falhas de Software (FMA-PFS), usa as técnicas de meta-aprendizado para recomendar algoritmos de AM para o domínio de PFS. Algumas pesquisas que empregam a utilização de meta-aprendizado foram abordadas em um âmbito geral [SBK04, BCSV08, Sun14] e outras para dados de expressão gênica na área de biologia [Sou10]. Sendo assim, esta dissertação apresenta pesquisas levantadas em uma revisão sistemática feita por Malhotra [Mal15] e apresentará trabalhos relacionados na área de Engenharia de Software com foco na predição de falhas de software utilizando técnicas de aprendizado de máquina.

2.4.1 Menzies

Menzies et al. [MGF07] relataram a importância de se usar métricas em nível de método para construir modelos capazes de detectar falhas em módulos de software. Argumentaram que isso estava sendo muito debatido na literatura e que trabalhos anteriores a esse exploravam problemas entre as métricas de Mccabe, Halstead e linhas de código para a construção desses modelos preditivos. Além disso, referenciou trabalhos anteriores a sua pesquisa que concordavam com o uso das métricas em nível de método e aqueles que argumentavam que as métricas não poderiam ser usadas para os modelos preditivos.

Segundo Menzies et al. [MGF07], essas discordâncias aconteceram pela análise de diferentes dados. Então, propôs um experimento base sugerindo que, a partir de sua pesquisa, os dados utilizados deveriam ser armazenados em um repositório de dados de um domínio público para que então diferentes pesquisadores da área de Engenharia de Software pudessem usar e comparar suas técnicas. A sua pesquisa motivou esse experimento base, pois demonstrou através do algoritmo de AM *Naïve Bayes* com uma técnica de pré-processamento nas métricas de software, que conseguiu aumentar o desempenho desse modelo preditivo comparado a outros estudos preliminares que utilizavam algoritmos baseado em regras ou árvores de decisão. Com o desempenho do algoritmo de AM *Naïve Bayes* Menzies et al. [MGF07] concluíram que a escolha de um algoritmo de AM é mais importante do que escolher um conjunto de métricas de software como atributos preditivos para os algoritmos.

Além disso, Menzies et al. [MGF07] encontraram uma probabilidade de 71% de detecção de módulos propensos a falhas utilizando as métricas em nível de método. Shull et al. [SBB+02] concluíram que a técnica de revisão de código que é bastante empregada na indústria tem uma probabilidade em torno de 60% para a detecção de falhas de software. As métricas em nível de método podem ser automaticamente coletadas após o desenvolvimento de software, enquanto que as revisões de códigos despendem uma significativa parcela de tempo, uma vez que os métodos de revisão são realizados em 8 para 20 linhas de código por minuto e geralmente são realizados por um time de desenvolvimento entre 4 a 6 desenvolvedores [MRS+02].

Em sua pesquisa ainda, foram utilizados os conjuntos de dados do projeto da NASA. Em seu entendimento a NASA faz a contratação de empresas de software terceirizadas que empregam a certificação (ISO-9001), de forma a garantir as melhores práticas de desenvolvimento de software para a indústria. Basili et al. [BMPZ02] argumentaram que utilizar dados de projetos da NASA em geral podem ser relevantes, uma vez que foram desenvolvidos por empresas terceiras das indústrias de Engenharia de Software. Em contrapartida, a pesquisa utilizou somente 8 dos 10 conjuntos de dados da NASA, pois os autores argumentaram que dois desses continham dados muitos diferentes dos outros.

Isso, de certa forma, pode ser considerado uma limitação em sua pesquisa, pois não foram utilizados todos os dados disponíveis dos projetos da NASA.

Por fim, seus experimentos sugerem que os modelos devem ser construídos através de todas as métricas possíveis e somente depois encontrar o mais apropriado sub-conjunto de métricas para um particular conjunto de dados. Em resumo, os autores endorsam o uso de métricas em nível de método, ainda que trabalhos anteriores [FO00, SI94] argumentem contra isso, pois defendem que a alteração do sub-conjunto de métricas pode mudar de um conjunto de dados para outro conjunto de dados. De fato, os experimentos realizados pelos autores demonstraram bons resultados apenas para um algoritmo (*Naïve Bayes*) dos seis algoritmos de AM analisados. Sendo assim, enfatizaram ainda mais o emprego de mais de um algoritmo e um conjunto de dados, sugerindo que novos estudos devem ter extensiva experimentação.

2.4.2 Lessmann

Lessmann et al. [LBMP08] apresentaram um *framework* para avaliar uma comparação entre algoritmos de aprendizado de máquina para a predição em falhas de software. Os autores conduziram uma análise de 22 algoritmos de aprendizado de máquina para 10 conjuntos de dados do projeto da NASA. Foram os mesmos 8 conjuntos de dados com mais os 2 conjuntos de dados que Menzies et al. [MGF07] não acrescentaram em seus experimentos. Tal pesquisa foi conduzida sob argumento de falta de um consenso sobre as técnicas de predição em falhas de software.

De fato, Menzies et al. [MGF07] também fizeram tal constatação em sua pesquisa. De acordo com os autores, a quantidade de estudos, as formas que as técnicas foram mensuradas e avaliadas poderiam ter acarretado em uma descoberta inconsistente sobre a área de predição de falhas de software. Uma destas questões elucidou a forma de avaliação dos algoritmos de AM que apontavam para resultados contraditórios. Os autores propuseram, então, a utilização da medida de avaliação AUC para todos os algoritmos de aprendizado de máquina que foram avaliados e comparados. A utilização desta medida de avaliação foi debatida pelos autores que argumentaram que algumas pesquisas anteriores apresentaram valores diferentes de *thresholds* para determinar a classificação de um módulo de software propenso a falhas. Um *threshold* é um valor que determina se um módulo pode estar propenso ou não a falhas. Apenas relembrando que um algoritmo, após avaliar um novo dado ou um conjunto de métricas extraídas de um módulo de software, retorna um valor (que pode estar normalizado entre 0 e 1), que é a probabilidade de ele pertencer a uma classe ou não (propenso ou não a falhas). Um exemplo disso seria aplicar um valor de *threshold* igual a 0.5, por exemplo, determinando que valores maiores do que 0.5 indicariam que esse módulo estaria propenso a falhas. Caso contrário, para valores menores do que

0.5, o módulo estaria livre ou não propenso a falhas. Segundo os autores, esta aplicação de *thresholds* não é muito empregada para a predição de defeitos, pois o seu valor pode variar de acordo com o algoritmo de aprendizado de máquina e a sua variação pode confundir outros pesquisadores que quisessem comparar seus experimentos com outras pesquisas. Além disso, um valor de *threshold* aplicado a um determinado projeto e/ou conjunto de dados não necessariamente servirá para outro projeto de software, acarretando em mais desconfiança para o especialista de domínio que quisesse aplicar as técnicas utilizadas naquela pesquisa. Com isso, a medida de avaliação AUC consegue lidar melhor com todas as combinações de possíveis resultados de um módulo estar ou não propenso a falhas, pois um valor final é resultante das probabilidades de estar propenso ou não a falhas e isso pode ser então comparado entre um experimento e outro, ou até mesmo entre algoritmos de aprendizado de máquina.

Para comparar os algoritmos de aprendizado de máquina, Lessmann et al. [LBMP08] referenciaram em sua pesquisa a proposta de um artigo [Dem06] que revisou o problema que pode surgir com a comparação de algoritmos de aprendizado de máquina, e apresentou uma solução estatística para este problema. Essa solução recomenda que a comparação deve ser feita entre vários algoritmos e múltiplos conjuntos de dados. Porém, vale ressaltar aqui que os autores aplicaram uma significativa quantidade de algoritmos de aprendizado de máquina para apenas 10 conjuntos de dados. Além disso, os autores criticaram as formas de comparação de estudos preliminares que utilizaram inferência estatística como as análises de variância (ANOVA), pois um dos fatores negativos é justamente a limitação que seus experimentos têm com relação ao conjunto de dados ser inferior ao conjunto de dados mínimo pela ANOVA (30). Para suprir essas necessidades, foi proposta uma solução baseada em *ranking* médio [Dem06] que é obtido do desempenho que cada um dos algoritmos obteve contra todos os conjuntos de dados a serem avaliados. Aqueles algoritmos que tiverem as menores médias são reconhecidos como os melhores algoritmos a serem utilizados para a predição de falhas de software.

Outra constatação feita pelos autores é o fato de que muitos algoritmos de aprendizado de máquina obtiveram valores de AUC de 0.7 ou mais, isso traz uma probabilidade preditiva maior do que 70%. Essa conclusão demonstra o que Menzies et al. [MGF07] relataram em sua pesquisa, identificando que os algoritmos de aprendizado de máquina realmente são bons preditores em encontrar módulos de software propensos a falhas. Por fim, os autores argumentaram que poderia ser importante recomendar algoritmos de AM para diferentes conjuntos de dados, pois em uma simples análise entre 2 dos 22 algoritmos conseguiram comprovar tal fato.

2.4.3 Song

Song et al. [SJS+11] apresentaram um *framework* capaz de escolher o melhor esquema para um determinado projeto de software. O seu *framework* gera um modelo que se adapta as características específicas de um determinado projeto de software. Em sua entrada, o *framework* recebe os dados das métricas de software, faz um pré-processamento dos dados e indica qual o melhor modelo a ser seguido. O autor enfatizou em sua pesquisa, o fato de Menzies et al. [MGF07] terem violado uma das premissas da técnica de aprendizado de máquina, que recomenda que o pré-processamento de dados deve ser feito apenas para os dados de treino, o que não ocorreu em sua pesquisa. Menzies et al. [MGF07] fizeram o pré-processamento em todo o conjunto de dados e isso pode acarretar em modelos super ajustados aos treinamentos, porém com fraca capacidade de generalização. Song et al. [SJS+11] também argumentaram em sua pesquisa a importância de se recomendar um modelo diferente para determinados projetos de software que podem variar os valores de suas métricas de software. Nessa pesquisa, foram realizados 2 experimentos: i) comparar seus resultados com o *framework* desenvolvido por Menzies et al. [MGF07]; ii) validar como o seu *framework* funcionaria na prática. Em relação aos dados, os autores combinaram 17 conjuntos de dados, sendo 13 conjuntos de dados do projeto da NASA e mais 4 de outros projetos contidos no repositório de dados tera-PROMISE [Men15]. Já em relação aos algoritmos, foram avaliados 3 algoritmos de aprendizado de máquina: i) *Naïve Bayes*; ii) *OneR*; iii) C4.5. Juntamente com os algoritmos, foram realizados 2 técnicas de pré-processamento e 2 de seleção de atributos, totalizando 12 esquemas de aprendizados diferentes. Para avaliar os algoritmos de aprendizado de máquina, os autores utilizaram as medidas AUC e *Balance* proposta por Menzies et al. [MGF07].

E por fim, concluíram mais uma vez a importância de escolher um esquema diferente para novos projetos de software. Os autores destacaram que o seu *framework* pode trabalhar melhor com novos projetos, uma vez que o *framework* criado por Menzies et al. [MGF07] estavam super ajustando os modelos gerados.

2.5 Considerações Finais

Neste capítulo, apresentou-se a fundamentação teórica e os trabalhos relacionados desta dissertação. Uma visão geral sobre a qualidade de software demonstrou a importância em prever falhas no desenvolvimento de software. Além disso, a qualidade de software está ligada a PFS através das métricas de software que são empregadas para identificar módulos propensos a falha.

Este capítulo também apresentou a técnica de AM através de conceitos básicos de classificação, paradigmas preditivos, métodos de classificação e formas de avaliação. Além disso, foi ilustrado como AM pode ser utilizado para PFS.

Após a fundamentação de AM, foi apresentado genericamente a técnica de meta-aprendizado, assim como sua utilização, dando ênfase para a recomendação de algoritmos. Foram abordados os tópicos de caracterização dos conjuntos de dados, medidas de avaliação e recomendações a partir da caracterização (meta-aprendizes). Os conceitos de meta-aprendizado são utilizados para construção do *framework* FMA-PFS.

Em geral, os trabalhos relacionados mencionados previamente foram um grande avanço para a área de Engenharia de Software com foco em predição de falhas de software. Para os três trabalhos relacionados, os autores enfatizaram a questão de recomendar diferentes algoritmos de aprendizado de máquina para novos projetos de software. Além disso, conseguiram em seus resultados demonstrar a eficácia em utilizar a técnica de aprendizado de máquina para buscar módulos propenso a falhas em projetos de software, já que em suas pesquisas puderam constatar que em média tem-se mais do que 70% de predição em módulos propensos a falhas, ultrapassando os modelos estatísticos que ficam na média em 60%.

Entretanto, até o presente momento não existe um consenso na área sobre qual a melhor técnica para a recomendação desses algoritmos. Isso se torna evidente, pois em primeiro lugar, as pesquisas realizaram experimentos com um conjunto pequeno de dados, em média inferior a 20. Em segundo lugar, Menzies et al. [MGF07] violaram uma regra de aprendizado de máquina, quando utilizaram dados de treino e teste para o pré-processamento dos dados. Já Lessmann et al. [LBMP08] utilizaram técnicas de comparação de algoritmos que recomendam um maior conjunto de dados (os autores usaram apenas 10 e a técnica recomenda em média 30 [Zar99]). E, por último, os autores concordaram que era necessário mais experimentos com uma quantidade de dados de projetos de software ainda maior. Isso remete para a proposta desta pesquisa e reforça a necessidade de mais estudos para buscar novos conceitos e técnicas para a predição em falhas de software.

Em resumo, os trabalhos relacionados não apresentaram uma única técnica de aprendizado de máquina que possa prevalecer sobre todos os conjuntos de dados. De fato, isso reverte para o teorema NFL (*no-free-lunch*), que foi brevemente explanado na introdução desta pesquisa e que relata que nenhum algoritmo de aprendizado de máquina pode ser melhor para todos conjuntos de dados, mesmo que ele tenha sido recomendado para aquele específico conjunto de dados anteriormente avaliado. Ainda, ficou evidenciado nos trabalhos relacionados que os experimentos realizados foram feitos para um mínimo conjunto de dados, ou seja, as soluções de algoritmos de aprendizado de máquina podem funcionar para aquele domínio ou conjunto de dados que foi experimentado naquela pesquisa, mas definitivamente pode não ser a melhor solução para outro conjunto de dados.

Outra questão muito explorada por Song et al. [SJS+11] foi a questão do *framework* apresentado por Menzies et al. [MGF07] estar super estimando os resultados, uma vez que ele faz otimização de parâmetros dos algoritmos de aprendizado de máquina sobre os dados de treino e teste.

Sendo assim, esta pesquisa propõem um *framework* que possa resolver os problemas evidenciados nestas pesquisas relacionadas. Além de recomendar algoritmo(s) de aprendizado de máquina, os experimentos realizados podem utilizar um conjunto de dados distintos e com uma amostragem maior que os trabalhos relacionados. O próximo capítulo apresenta o *framework* FMA-PFS, trazendo todas as informações e detalhes de como ele será construído.

3. *FRAMEWORK* FMA-PFS PARA RECOMENDAÇÃO DE ALGORITMOS POR MEIO DE META-APRENDIZADO

Com base na fundamentação teórica do Capítulo 2, este capítulo apresenta o *Framework* de Meta-Aprendizado para Predição de Falhas de Software (FMA-PFS) para recomendação de algoritmo(s) de AM por meio de Meta-Aprendizado. Este *framework* é constituído por oito etapas que foram definidas e ilustradas conforme a Figura 3.1. Estas etapas consistem em:

1. Buscar conjuntos de dados públicos gerados para prevenir módulos propensos a falhas de software;
2. Coletar características estruturais dos conjuntos de dados públicos;
3. Utilizar algoritmos de AM para classificar os conjuntos de dados públicos;
4. Definir medidas de avaliação de desempenho de classificação;
5. Criar a meta-base;
6. Definir meta-aprendizes;
7. Coletar características estruturais de um novo conjunto de dados;
8. Testar o *framework* em conjuntos não vistos para recomendar o(s) algoritmo(s) mais apropriado(s) para estes conjuntos.

As seções a seguir detalham estas etapas.

3.1 Definição dos Conjuntos de Dados Públicos para Predição de Falhas

A primeira etapa do processo de meta-aprendizado é definir o conjunto de entrada de dados. Geralmente, tal conjunto de bases de dados é coletado de um determinado domínio ou de um conjunto de domínios. Particularmente, o domínio proposto foi a predição de falhas de software. Inúmeros conjuntos de dados para PFS têm sido usados ao longo dos anos por pesquisadores da área de Engenharia de Software [Mal15]. Um exemplo conhecido são os conjuntos de dados dos projetos desenvolvidos para a NASA.

A área de Engenharia de Software possui bases de dados disponibilizadas de forma pública sobre projetos de software. O tera-PROMISE [Men15] é um repositório de dados público que concentra uma grande quantidade dessas bases de dados. Este repositório de dados público contêm os conjuntos de dados da NASA, que é um dos conjuntos de

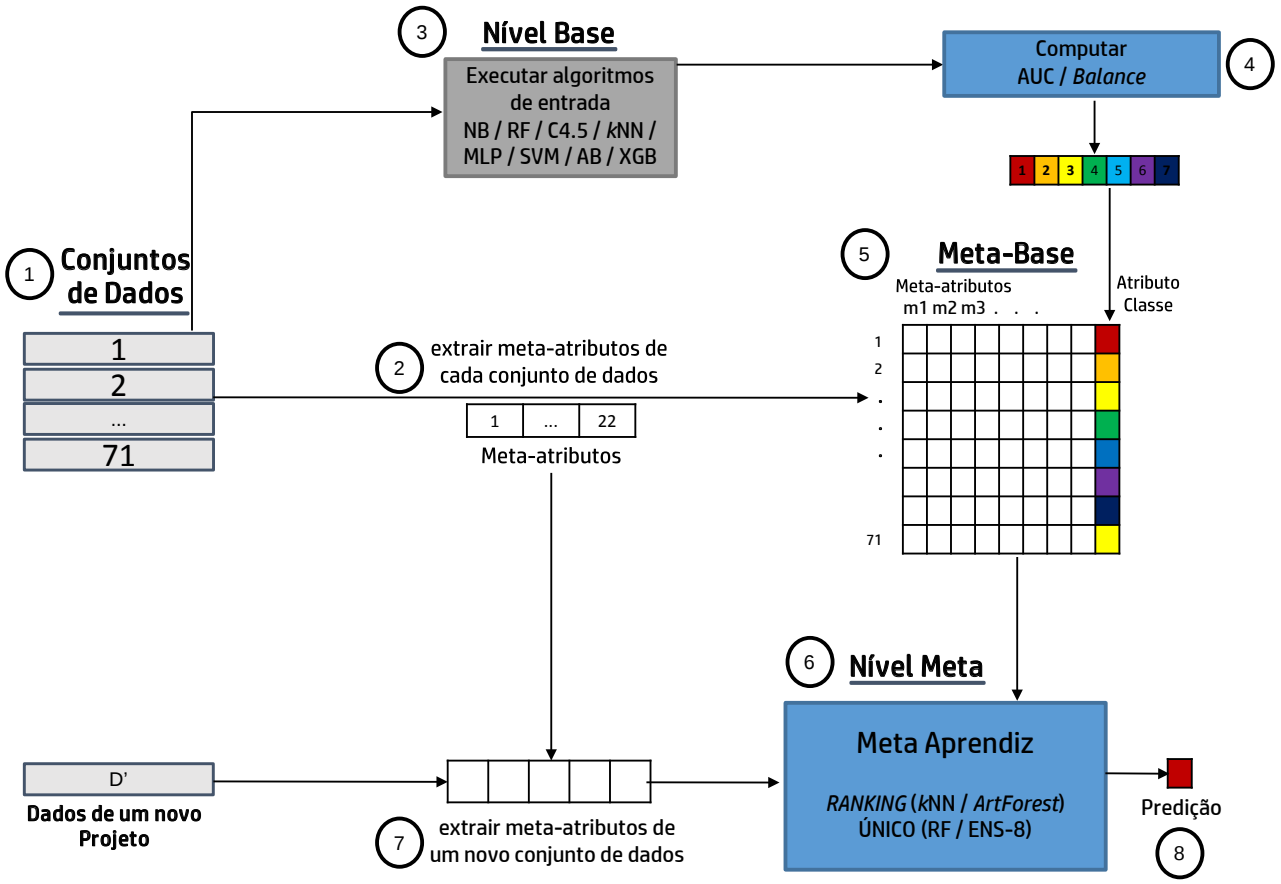

Figura 3.1: Visão geral do *framework* FMA-PFS para recomendação de algoritmos de AM por meio de meta-aprendizado.

dados mais empregados nos estudos em predição de falhas de software [MGF07, LBMP08, SJS+11]. Além disso, o tera-PROMISE foi criado para armazenar esses dados em um único lugar com o propósito de pesquisadores da área de Engenharia de Software usarem essas informações com o objetivo de produzirem trabalhos científicos replicáveis. O repositório de dados está dividido em várias categorias, como falhas de software, estimativa de esforço, análise de código, refatoramento de código e análise de requisitos. A categoria falhas de software concentra os conjuntos de dados utilizados nesta pesquisa. Em tal categoria, existem as subcategorias que são divididas pelos tipos de métricas de software: métricas de McCabe & Halstead e métricas *CK*.

Com base em tais informações é preciso destacar que os conjuntos de dados possuem uma heterogeneidade em função do conjunto de métricas de software ser diferente de acordo com as subcategorias citadas acima. Até o presente momento, e conforme relacionado anteriormente, os pesquisadores têm concentrado seus esforços em apenas um único ou poucos conjuntos de dados. Os projetos da NASA são um exemplo disto, pois possuem apenas 10 conjuntos de dados, o que é problemático, pois as suas soluções apresentam uma única técnica ou modelo a ser seguido e que muitas vezes serve apenas para aquele conjunto de dados específico. Além disso, existem alguns estudos [BSC03, Kal02, Sou10]

que utilizaram, em média, 50 conjuntos de dados de entrada para recomendação de algoritmos com a técnica de meta-aprendizado. Isto reforça ainda mais a ideia de que é preciso mais do que 10 conjuntos de dados para meta-aprendizado ser bem sucedido.

Sendo assim, o *framework* FMA-PFS utilizou 115 conjuntos de dados, ilustrados na Tabela 3.1, do repositório de dados público do tera-PROMISE. Com essa quantidade e com a heterogeneidade destes conjuntos de dados, o *framework* FMA-PFS será capaz de recomendar algoritmo(s) de aprendizado de máquina para um conjunto de dados ainda não conhecido (dados de outros projetos).

Tabela 3.1: Conjuntos de Dados do tera-PROMISE [Men15] para PFS utilizados pelo *framework* FMA-PFS.

Base de Dados				
1. ant-1.3	**2.** ant-1.4	**3.** ant-1.5	**4.** ant-1.6	**5.** ant-1.7
6. ar1	**7.** ar3	**8.** ar4	**9.** ar5	**10.** ar6
11. arc	**12.** berek	**13.** camel-1.0	**14.** camel-1.2	**15.** camel-1.4
16. camel-1.6	**17.** ckjm	**18.** CM1-v2	**19.** CM1	**20.** datatrieve
21. e-learning	**22.** eclipse34_debug	**23.** eclipse34_swt	**24.** forrest-0.7	**25.** forrest-0.8
26. intercafe	**27.** ivy-1.1	**28.** ivy-1.4	**29.** ivy-2.0	**30.** jedit-3.2
31. jedit-4.0	**32.** jedit-4.1	**33.** jedit-4.2	**34.** jedit-4.3	**35.** JM1-v2
36. JM1	**37.** kalkulator	**38.** KC1-v2	**39.** KC1	**40.** KC2-v2
41. KC2	**42.** KC3-v1	**43.** KC3-v2	**44.** KC3	**45.** KC4
46. log4j-1.0	**47.** log4j-1.1	**48.** log4j-1.2	**49.** lucene-2.0	**50.** lucene-2.2
51. lucene-2.4	**52.** MC1-v1	**53.** MC1-v2	**54.** MC1	**55.** MC2-v1
56. MC2-v2	**57.** MC2	**58.** mozilla4	**59.** MW1-v1	**60.** MW1-v2
61. MW1	**62.** nieruchomosci	**63.** pbeans1	**64.** pbeans2	**65.** PC1-v1
66. PC1-v2	**67.** PC1	**68.** PC2-v1	**69.** PC2-v2	**70.** PC2
71. PC3-v1	**72.** PC3-v2	**73.** PC3	**74.** PC4-v1	**75.** PC4-v2
76. PC4	**77.** PC5-v1	**78.** PC5-v2	**79.** PC5	**80.** pdftranslator
81. poi-1.5	**82.** poi-2.0	**83.** poi-2.5	**84.** poi-3.0	**85.** prop-1
86. prop-2	**87.** prop-3	**88.** prop-4	**89.** prop-5	**90.** prop-6
91. redaktor	**92.** serapion	**93.** skarbonka	**94.** sklebagd	**95.** synapse-1.0
96. synapse-1.1	**97.** synapse-1.2	**98.** systemdata	**99.** szybkafucha	**100.** termoproject
101. tomcat	**102.** velocity-1.4	**103.** velocity-1.5	**104.** velocity-1.6	**105.** workflow
106. wspomaganiepi	**107.** xalan-2.4	**108.** xalan-2.5	**109.** xalan-2.6	**110.** xalan-2.7
111. xerces-1.2	**112.** xerces-1.3	**113.** xerces-1.4	**114.** xerces-init	**115.** zuzel

3.2 Extração de Características dos Conjuntos de Dados

Uma das propriedades da técnica de meta-aprendizado é a caracterização dos dados. Depois de coletar um conjunto representativo de dados, torna-se necessário a extração de características estruturais de tais conjuntos. Esta etapa é definida como extração dos *meta-atributos*. Os meta-atributos são uma abstração dos conjuntos de dados. Para cada conjunto de dados, um conjunto de meta-atributos representa as informações daquele conjunto de dados. Com tal conjunto de meta-atributos, pode-se identificar o quão difícil foi resolver os problemas de classificação para aquele específico conjunto de dados.

Tabela 3.2: Meta-atributos utilizados no FMA-PFS.

Trabalho	Nome	Descrição
[BH94]	smpl	Número de exemplos (instâncias)
[BH94]	cls	Número de classes
[BH94]	NumAttr	Número de atributos
[BH94]	NumBin	Número de atributos binários
[BH94]	MSkew	Obliquidade média dos atributos
[BH94]	MKurt	Curtose média dos atributos
[BH94]	MMult	Múltipla correlação média dos atributos
[BH94]	SDRat	Taxa de desvio padrão
[HB02]	F1	Taxa máxima da discriminação de Fisher
[HB02]	F1v	Vetorização da taxa máxima da discriminação de Fisher
[HB02]	F2	Sobreposição dos limites definidos pelos exemplos de cada classe
[HB02]	F3	Maior eficiência de um atributo do conjunto de atributos
[HB02]	F4	Soma da eficiência de todos os atributos
[HB02]	L1	Soma da mínima distância de um erro para um classificador linear
[HB02]	L2	Quantidade de erros ao treinar um classificador linear
[HB02]	L3	Não linearidade de um classificador linear
[HB02]	N1	Fração de pontos situados na fronteira de classes
[HB02]	N2	Média entre as distâncias dos vizinhos próximos de uma classe
[HB02]	N3	Taxa de erro de um classificador 1-NN com estratégia *leave-one-out*
[HB02]	N4	Não linearidade de um classificador 1-NN
[HB02]	T1	Fração da esfera máxima de cobertura sobre os dados
[HB02]	T2	Dispersão dos dados (razão entre o número de exemplos e a quantidade de atributos)

Nesta dissertação, foi estabelecido que seriam utilizados dois conjuntos de meta-atributos que são bem conhecidos da área de meta-aprendizado:

- Meta-atributos do projeto STATLOG [BH94]: Consiste de 8 meta-atributos que foram coletados a partir do conjunto de dados de entrada definidos na etapa anterior. Estes atributos são as estatísticas simples de um conjunto de dados, conforme explicado no Capítulo 2. Dentre os exemplos de estatísticas simples, cita-se o número de classes, o número de atributos e a correlação entre os atributos.

- Meta-atributos baseados na complexidade dos dados [HB02, BH06]: São 14 meta-atributos que identificam problemas de classificação como a fronteira de decisão, a ambiguidade entre as classes e a geometria e topologia dos conjuntos de dados.

A Tabela 3.2 apresenta um sumário de todos os meta-atributos que foram coletados pelo *framework* FMA-PFS.

3.3 Definição dos Algoritmos de AM

Após a definição dos conjuntos de entrada para o *framework* FMA-PFS, a etapa representada pela (Figura 3.1 - Item 3) consiste em selecionar algoritmos de aprendizado de máquina para executar sobre a entrada do *framework* de meta-aprendizado. Para se recomendar algoritmo(s) de AM, faz-se necessário que eles sejam definidos como entrada em um processo de meta-aprendizado. De fato, os algoritmos escolhidos avaliam cada conjunto de dados de entrada e as medidas de avaliação de desempenho desses algoritmos são coletadas após suas execuções. Como exemplo, a acurácia de um algoritmo que informa se o algoritmo conseguiu predizer corretamente as saídas dos conjuntos de dados e o tempo que esse algoritmo de AM levou para executar.

Para definir os algoritmos, esta dissertação propôs um conjunto de algoritmos referenciados em uma recente revisão sistemática feita por Malhotra [Mal15]. Em sua revisão, a autora apresentou oito categorias de algoritmos de AM ilustradas na Tabela 3.3. Ainda na Tabela 3.3, apresenta-se os 8 algoritmos escolhidos para a entrada no *framework* FMA-PFS. Duas das oito categorias não tiveram algoritmos representados nesta pesquisa, por entender que a quantidade de estudos preliminares era inferior se comparado com as outras categorias. Por outro lado, a revisão sistemática apresentou o algoritmo *Random Forest* (RF) [Bre01] como o algoritmo de melhor desempenho para o domínio de predição de falhas de software. Desta forma, optou-se pelo uso de mais alguns algoritmos de AM da categoria *ensembles* para os experimentos. Os algoritmos escolhidos foram o *Ada Boost* (AB) [FS96] e *Extreme Gradient Boost* (XGB) [CG].

Após a definição dos algoritmos de AM, cada um dos 8 algoritmos foram executados sobre os 115 conjuntos de dados de entrada, o que resultou em uma lista de métricas através da avaliação de cada algoritmo de AM para cada um dos conjuntos de dados. Para a execução dos algoritmos, utilizou-se a ferramenta de aprendizado de máquina *WEKA* [HFH+09]. Cada um dos algoritmos de AM proposto possui um conjunto de parâmetros de configuração, e se optou por utilizar o conjunto de parâmetros sugeridos pela ferramenta *WEKA* (conjunto de parâmetros padrão), pois a proposta desta pesquisa é recomendar algoritmos de AM e não recomendar parâmetros para os algoritmos de AM. Após o *framework* FMA-PFS recomendar o(s) melhor(es) algoritmo(s), pode-se então definir os melhores parâmetros para aquele algoritmo.

3.4 Definição de Medidas de Avaliação de Desempenho dos Algoritmos

Na seção anterior foram definidos os 8 algoritmos de AM à serem executados sobre os 115 conjuntos de dados. Para cada um destes algoritmos de AM, é necessário

Tabela 3.3: Algoritmos de AM utilizados no FMA-PFS.

Categoria	Algoritmo(s)
Árvore de decisões (DT)	C4.5
Aprendizado Bayesiano (BL)	*Naïve Bayes* (NB)
Ensemble de algoritmos (EL)	*Random Forest* (RF) *Ada Boost* (AB) *Extreme Gradient Boost* (XGB)
Redes Neurais (NN)	Perceptron Multicamadas (MLP)
Máquina de Vetor de Suporte (SVM)	Máquina de Vetor de Suporte (SVM)
Aprendizagem baseada em regras (RBL)	—
Algoritmos evolucionários (EA)	—
Outras técnicas	*k*-NN

que uma avaliação de desempenho seja realizada. No Capítulo 2, algumas destas medidas de avaliação foram apresentadas. Dentre elas, a medida de avaliação AUC é uma medida conhecida na área de AM e, além disso, é uma das medidas de avaliação mais usadas no domínio de predição de falhas de software [Mal15]. A outra medida a ser utilizada nesta pesquisa foi a medida *Balance*, que também foi brevemente explanada no Capítulo 2, porém essa medida de avaliação é utilizada apenas no contexto de predição de falhas de software [MGF07, SJS+11].

A (Figura 3.1 - Item 4) destaca a relevância desta etapa no contexto de meta-aprendizado. Os valores das medidas de avaliação servem para formar os atributos alvos de uma meta-base que será detalhada na etapa seguinte.

3.5 Construção da Meta-Base

Após a definição das bases de dados, extração das características, classificação dos algoritmos de AM e escolha das medidas de avaliação, uma única base de dados chamada de meta-base foi formada. A quinta etapa, conforme ilustrado na (Figura 3.1 - Item 5) apresenta os dados da execução de todas as etapas anteriores realizadas no nível base dos conjuntos de dados, formando agora uma única base de dados. As informações relacionadas as características dos conjuntos de dados representam o conjunto de atributos preditivos (*features*) desse conjunto de dados. Os valores das medidas de avaliação dos algoritmos de AM são os rótulos ou atributo alvo que será exemplificado em seguida.

3.6 Seleção de um Meta-Aprendiz para Construção de Modelo de Recomendação

Assim que os dados de um novo projeto de software estiverem disponíveis, é preciso que os mesmos meta-atributos que foram extraídos dos conjuntos de dados de entrada sejam coletados deste novo conjunto de dados. Após, torna-se necessário escolher um meta-aprendiz (algoritmo de AM) que será aplicado sobre a meta-base para gerar um modelo preditivo capaz de recomendar algoritmo(s) para o problema de predição de falhas de software (Figura 3.1 - Item 6). Uma vez que o modelo é gerado, ele é aplicado sobre os meta-atributos gerados a partir dos dados de um novo projeto (Figura 3.1 - Item 7) e um conjunto do(s) melhor(es) algoritmo(s) de AM é apresentado como saída para ser empregado neste novo conjunto de dados (Figura 3.1 - Item 8).

O modelo obtido pelo meta-aprendiz pode ser utilizado para duas propostas diferentes [Kal02]: i) ranquear uma lista de algoritmos de AM para um novo conjunto de dados de um projeto de software; ii) recomendar o melhor algoritmo para o novo projeto. O meta-aprendiz que o *framework* FMA-PFS emprega vai depender da proposta definida pelo especialista de domínio. Para as tarefas de ranqueamento de algoritmos de AM, o *framework* FMA-PFS fez uso do algoritmo de aprendizado de máquina *k*-NN (*k* vizinhos mais próximos), pois este algoritmo permite a predição de uma lista de ranqueamento de forma natural [BSC03]. Além do algoritmo *k*-NN, foi utilizado também para a estratégia de *ranking*, um algoritmo *ArtForest* [Sun14] implementado recentemente, que faz uso de técnicas de *ensemble* de árvores de decisão. Já para as tarefas de recomendação do melhor algoritmo, o *framework* FMA-PFS permitiu a utilização do algoritmo *Random Forest* (RF) [Bre01] ou de um *ensemble* dos 8 algoritmos de entrada (ENS-8). A escolha do algoritmo RF como meta-aprendiz foi atribuída pelo fato de tal algoritmo ser um dos mais efetivos para o problema de predição de falhas de software [Mal15]. A segunda opção (ENS-8), refere-se a utilização de um esquema de votação (maior número de votos é eleito) feito pela predição dos 8 algoritmos escolhidos para a entrada do *framework* FMA-PFS.

3.7 Demonstração do *Framework* FMA-PFS

Para demonstrar o funcionamento do *framework* FMA-PFS para um usuário que queira comprovar a sua eficácia, torna-se necessário elucidar um exemplo didático. O *framework* recebeu em sua entrada os 115 conjuntos de dados para processamento. A seguir são coletadas as características estruturais dos 115 conjuntos de dados. Os dados devem ser coletados individualmente para cada conjunto de dados, ou seja, pegar o primeiro conjunto de dados, coletar as características e fazer isso para o segundo conjunto sucessivamente até que todos os conjuntos sejam concluídos. O próximo passo é executar os 8

algoritmos de AM para medir o desempenho de cada um dos algoritmos. Duas medidas de desempenho (AUC e *Balance*) são armazenadas para futuras análises. Com os dados das características estruturais e as medidas de desempenho, forma-se a meta-base que será utilizada no próximo instante.

Assim que a meta-base é constituída e armazenada o *framework* FMA-PFS está pronto para recomendar algoritmos de AM. Para sua utilização, basta passar um novo conjunto de dados ao *framework*. Da mesma forma que se extraiu as características estruturais para os 115 conjuntos de dados, o processo de caracterização é realizado novamente, mas somente para o novo conjunto de dados. Os meta-aprendizes geram seus modelos através da meta-base e baseados nos critérios de desempenhos. E, por fim, com os dados das características estruturais do novo conjunto de dados o *framework* FMA-PFS recomenda os algoritmos de AM ditos como ideais para o novo conjunto de dados. A saída final para o usuário é a combinação dos meta-aprendizes para *ranking* e algoritmo único, e mais a recomendação baseada no critério das medidas de desempenho. Sendo que foram desenvolvidos para o *framework*, 2 meta-aprendizes de *ranking*, 2 meta-aprendizes de recomendação única e 2 medidas de desempenho, totalizando-se 8 recomendações para o usuário.

Com a recomendação dos algoritmos de AM o gerente de projetos não precisa mais despender tempo e conhecimento para definir qual técnica de AM utilizar para o projeto de software desenvolvido recentemente na empresa. A partir de agora basta executar apenas o(s) algoritmo(s) que foram apresentados pelo *framework* FMA-PFS como sendo os algoritmos ideais para este projeto.

3.8 Considerações Finais

Este capítulo apresentou as etapas para a construção do *framework* FMA-PFS, que utiliza os conceitos de meta-aprendizado para recomendar algoritmo(s) de aprendizado de máquina para o domínio de predição de falhas de software. Com todas essas etapas definidas, o *framework* FMA-PFS foi desenvolvido na linguagem Java e está pronto para a execução de todos os experimentos. O Capítulo 4 irá relatar os experimentos e demonstrar a capacidade do *framework* em recomendar algoritmo(s) para a predição de falhas de software.

4. EXPERIMENTOS E RESULTADOS

4.1 Considerações Iniciais

Conforme apresentado no Capítulo 3, o *framework* FMA-PFS é constituído de 8 etapas baseadas na abordagem de meta-aprendizado para recomendação de algoritmos de AM. De maneira resumida, estas etapas operam da seguinte forma: entrada dos conjuntos de dados, coleta de meta-atributos e execução de algoritmos de AM sobre a entrada dos dados, sugestão das melhores métricas para avaliação dos algoritmos de AM, formação da meta-base, definição dos meta-aprendizes e a construção de um modelo de recomendação para predição de algoritmos de AM para novos problemas.

Neste capítulo, são reportados os resultados dos experimentos realizados para avaliar a utilização de meta-aprendizado em problemas de predição de falhas de software. Os experimentos e análises correspondentes foram divididos em dois grupos:

1. Recomendação de um *ranking* de algoritmos de aprendizado de máquina.

2. Recomendação de um único algoritmo de aprendizado de máquina.

Na primeira proposta, dois meta-aprendizes são providos ao usuário para a recomendação de algoritmos de AM na forma de um *ranking*. Um deles é o meta-aprendiz FMA-PFS-R*k*, que através do algoritmo *k*-NN, realiza a predição de *rankings*. O outro meta-aprendiz, FMA-PFS-RA, recomenda *rankings* utilizando o algoritmo ArtForest [Sun14], que utiliza a técnica de *ensemble* de algoritmos de árvores de decisão. Esses meta-aprendizes foram escolhidos para os experimentos desta pesquisa pois o meta-aprendiz *k*-NN ainda é considerado o estado da arte em predição de *rankings* e o meta-aprendiz *ArtForest* mostrou-se superior ao *k*-NN recentemente, para a recomendação de um *ranking* de algoritmos de AM.

O segundo grupo consiste basicamente em recomendar um único algoritmo de AM. Para esta estratégia, novamente foi proposto dois meta-aprendizes. O primeiro meta-aprendiz foi o algoritmo *Random Forest* [Bre01], FMA-PFS-RF. O segundo meta-aprendiz foi uma combinação de 8 algoritmos FMA-PFS-ENS-8 referenciados nesta pesquisa pela Figura (3.1 - Etapa 3), ilustrada no Capítulo 3. Para o meta-aprendiz FMA-PFS-ENS-8 recomendar um dos algoritmos de AM, foram executados cada um dos 8 algoritmos sobre a meta-base, cada um recomendando um dos algoritmos e no final aquele que obteve a maior votação [MRV+15], foi escolhido como o algoritmo de AM a ser recomendado.

Este capítulo está organizado como segue. Na Seção 4.2 é apresentada uma explanação sobre a técnica que os meta-aprendizes utilizaram para recomendar algoritmo(s)

de AM. Na Seção 4.3 é apresentada a primeira forma de recomendação através de *ranking*. Já na Seção 4.4, a segunda forma de recomendação é ilustrada, bem como seus resultados. E, por fim, na Seção 4.5 são tratadas as considerações finais sobre todos os resultados e experimentos analisados.

4.2 Métodos e Técnicas de Meta-Aprendizado Utilizadas nos Experimentos

Como forma de avaliar se o *framework* FMA-PFS consegue identificar qual o(s) algoritmo(s) obtém melhor desempenho para um determinado projeto de software, é necessário que se estabeleça uma técnica para verificar cada um dos 115 conjuntos de dados que foram pré-selecionados para os experimentos. A técnica de *leave one out* (LOO) foi empregada pela primeira vez na área de meta-aprendizado por Brazdil et al. [BSC03] e vem sendo utilizada em outras pesquisas na área de meta-aprendizado [Sou10, MRV+15]. A técnica LOO consegue analisar individualmente cada um dos 115 conjunto de dados. Ela funciona da seguinte maneira: um dos 115 conjuntos de dados é selecionado para ser o conjunto de dados a ser avaliado (conjunto de dados de teste) e os 114 conjuntos de dados restantes são tratados como conjunto de treino. Após a avaliação deste primeiro conjunto de dados, o segundo passa a ser o conjunto de teste e o restante o conjunto de treino. Isso ocorre sucessivamente até que todos os 115 conjuntos de dados sejam avaliados individualmente. Para todas as técnicas e medidas de avaliação, a técnica de LOO foi utilizada nesta pesquisa.

Além da técnica de LOO, os algoritmos utilizados como meta-aprendizes para o *framework* FMA-PFS necessitam apresentar uma configuração para os parâmetros de entrada. Nos experimentos realizados nesta pesquisa, os meta-aprendizes do *framework* utilizaram em suas configurações os valores *default* que são recomendados pelos pesquisadores que utilizaram as técnicas de meta-aprendizado [BSC03, Sun14]. A única exceção foi a configuração do número de vizinhos do algoritmo *k*-NN, onde foi empregado o valor de k = 3 [MRV+15], por entender que um valor inicial de k = 1 pudesse acarretar em um problema conhecido na área de aprendizado de máquina chamado de *outliers*.

Todos os experimentos realizados obtiveram resultados para os 115 conjuntos de dados. No entanto, como forma de apresentação, as seções a seguir ilustram apenas 10 dos 115 conjuntos de dados, e as médias (μ) e desvios-padrão (σ) finais. O Apêndice A contém o resultado completo de todos os experimentos de todos os 115 conjuntos de dados.

4.3 Recomendação por Meio de *Ranking*

Para demonstrar a eficácia de meta-aprendizado do *framework* FMA-PFS para FMA-PFS-R*k* e FMA-PFS-RA, é necessário que os métodos de construção de um *ranking* sejam capazes de produzir sugestões que limitem seu tempo com experimentação [Sou10]. Para resolver este problema, dois aspectos precisam ser avaliados. Um deles é a medida de avaliação para o *ranking* e a segunda é a estratégia para avaliar os métodos de meta-aprendizado.

A medida de avaliação para um *ranking* é dita como ideal quando sua representação mostra a exata ordem do *ranking* previsto pelos dados de teste [BSC03]. Porém, este cenário dificilmente acontece em uma situação real. No caso de dados serem pareados e dispostos como uma ordem de *ranking*, a Correlação de Spearman tem se demonstrado suficiente em diversas situações [NW88]. A Correlação de Spearman é definida por:

$$r_s = 1 - \frac{6 \sum_{i=1}^{m} (rc_i - rp_i)^2}{m^3 - m} \tag{4.1}$$

onde rc_i e rp_i são, respectivamente o *ranking* correto e o previsto por algum algoritmo de AM e m é o número de algoritmos de AM que se quer avaliar(no caso $m = 8$). Um valor para a Eq.(4.1) igual a 1 significa um *ranking* previsto corretamente. Um valor para a Eq. (4.1) igual a -1 significa o estado *ranking* inverso. E um valor para a Eq. (4.1) igual a 0 indica que o *ranking* correto não tem nenhuma relação com o *ranking* previsto.

Após a definição de avaliação do *ranking*, uma estratégia para avaliar se os meta-aprendizes conseguiram recomendar os *rankings* deve ser sugerida. Dois métodos foram propostos: *ranking* aleatório (RA) e *ranking* majoritário (MAJ-R). Estes dois métodos são utilizados como *baseline* para comparação com os métodos propostos.

Para o *ranking* aleatório (RA), foram apresentados um total de 30 *rankings* dispostos aleatoriamente entre os valores [1,8]. Para todos os *rankings* aleatórios formados, é calculado o Coeficiente de Correlação de Spearman e posteriormente a média dos 30 valores é apresentada para ser comparada com o cálculo do Coeficiente de Correlação de Spearman dos meta-aprendizes FMA-PFS-R*k* e FMA-PFS-RA.

Para o esquema de *ranking* majoritário, é proposto o *ranking* que teve maior expressão dentre todos os 115 conjunto de dados (aquele *ranking* que apareceu mais vezes entre os dados), sendo que esse *ranking* majoritário é informado então para cada um dos 115 conjunto de dados. O valor do Coeficiente de Correlação de Spearman é calculado para demonstrar a eficácia dos meta-aprendizes de FMA-PFS.

4.3.1 Resultados para Recomendação de Ranking

Os resultados dos experimentos analisados foram divididos por duas das medidas de desempenho que foram definidas no Capítulo 3. Uma das medidas de desempenho é a AUC, e a outra é a medida *Balance*.

As medidas de desempenho representam a ordem que um *ranking* é apresentado, ou seja, para cada um dos 8 algoritmos as medidas AUC e *Balance* informam a posição (ordem) que cada um dos algoritmos de AM ficou sobre aquele específico conjunto de dados. Suponha que para o conjunto de dados X, os algoritmos NB, RF e XGB tiveram desempenho de (0.650, 0.750 e 0.749) respectivamente. Neste caso a ordem dos algoritmos seria (3,1,2), onde RF ficou em primeiro lugar, XGB em segundo e NB em terceiro. Esta ordem não levou em consideração uma possível semelhança das avaliações de desempenho dos algoritmos (no caso RF e XGB que tiveram desempenhos semelhantes). Desta forma, a ordem dos algoritmos ficou indiscriminada independente de uma similaridade de desempenho, ou de valores aproximados entre o desempenho dos algoritmos.

A Tabela 4.1 apresenta os resultados dos experimentos analisados para FMA-PFS-R*k* e FMA-PFS-RA para *rankings* propostos através da medida de avaliação AUC. Nota-se que o valor médio do Coeficiente de Spearman para as duas estratégias adotadas FMA-PFS-R*k* (0.758) e FMA-PFS-RA (0.765) superou os valores de RA (-0.002) e MAJ-R (0.747), confirmando a eficácia em se prever *ranking* de algoritmos de AM.

Tabela 4.1: Amostra de 10 conjuntos de dados do projeto da NASA com a média (μ) e desvio padrão (σ) do Coeficiente de Correlação de Spearman dos 115 conjuntos de dados. Para FMA-PFS-R*k*, FMA-PFS-RA, MAJ-R e RA, considerando a ordenação dos valores pela medida de desempenho AUC.

BASE	FMA-PFS-R*k*	FMA-PFS-RA	MAJ-R	RA
CM1	0.934	**0.952**	0.905	-0.027
JM1	0.994	**1.000**	**1.000**	-0.051
KC1	**0.970**	0.929	0.929	0.042
KC2	0.874	**0.905**	0.738	-0.008
KC3	**0.970**	0.952	0.881	-0.102
MC1	**0.838**	0.762	0.643	0.062
MC2	**0.922**	0.881	0.714	0.060
MW1	0.934	**0.952**	0.905	0.066
PC1	0.898	**0.905**	0.881	-0.056
PC2	0.708	**0.738**	0.500	-0.021
μ	0.758	**0.765**	0.747	-0.002
σ	**0.234**	0.204	0.206	0.070

A Tabela 4.2 apresenta os resultados analisados para FMA-PFS-R*k* e FMA-PFS-RA ranqueados pela medida de desempenho *Balance*. Nesta análise, pode-se notar um valor mais expressivo de FS-FMA-R*k* (0.489) e FMA-PFS-RA (0.620) superando os valores de RR (-0.003) e MAJ-R (0.108).

Tabela 4.2: Amostra de 10 conjuntos de dados do projeto da NASA com a média (μ) e desvio padrão (σ) do Coeficiente de Correlação de Spearman dos 115 conjuntos de dados. Para FMA-PFS-R*k*, FMA-PFS-RA, MAJ-R e RA, considerando a ordenação dos valores pela medida de desempenho *Balance*.

BASE	FMA-PFS-R*k*	FMA-PFS-RA	MAJ-R	RA
CM1	0.714	**0.810**	0.048	-0.048
JM1	**0.952**	0.881	0.548	-0.006
KC1	0.524	**0.690**	0.238	0.118
KC2	0.333	0.286	**0.738**	0.018
KC3	**0.905**	0.762	-0.190	0.061
MC1	**0.952**	0.881	0.119	0.065
MC2	**0.833**	0.595	0.071	-0.054
MW1	0.333	**0.595**	-0.333	0.026
PC1	0.214	**0.643**	0.238	-0.102
PC2	0.238	**0.333**	-0.786	-0.044
μ	0.489	**0.620**	0.108	-0.003
σ	0.394	0.253	**0.472**	0.065

Após a análise de todos os experimentos realizados para FMA-PFS-R, pode-se concluir que o *framework* apresentou resultados que superaram as duas estratégias RA e MAJ-R para o propósito de comparação. Pode-se concluir também que o FMA-PFS-RA superou o método FMA-PFS-Rk. De fato isso foi comprovado empiricamente por Sun [Sun14] e mais uma vez nesta pesquisa que o meta-aprendiz *ArtForest* foi superior ao meta-aprendiz *k*-NN, parecendo ser a melhor opção também no domínio de PFS.

4.4 Recomendação Única de Algoritmos de AM

A segunda forma de recomendação do *framework* FMA-PFS foi empregada para recomendar um único algoritmo de aprendizado de máquina para um novo conjunto de dados. Os meta-aprendizes FMA-PFS-RF e o FMA-PFS-ENS-8, através da técnica de LOO, recomendaram um único algoritmo para cada um dos 115 conjuntos de dados analisados e definidos no Capítulo 3. Suponha que o meta-aprendiz Z tenha recomendado o algoritmo B para um conjunto de dados X. Ainda para o conjunto de dados X, a execução dos algoritmos (A,B,C) tenha seus resultados através da medida de desempenho Y com valores: A = 0.690(3), B = 0.780(1) e C = 0.730(2). Neste caso, o meta-aprendiz Z empata com o algoritmo B e seus valores são ajustados para 1.5. Consequentemente, um novo valor de ordem é atribuído aos algoritmos subsequentes, formando a nova ordem: meta-aprendiz Z = 1.5, A = 4, B = 1.5 e C = 3. Se o meta-aprendiz Z recomendar para os 115 conjuntos de dados o algoritmo que obteve o melhor desempenho, então o seu valor médio final será de 1.5. Com isso, o meta-aprendiz Z obteria o menor valor possível e seria considerado o melhor valor para as comparações entre os meta-aprendizes e os 8 algoritmos de AM definidos.

4.4.1 Resultados das Recomendações Únicas

A seguir são relatados os resultados dos experimentos para a estratégia de recomendação única para escolher o algoritmo de aprendizado de máquina que teve o melhor desempenho de acordo com um determinado critério. Para esta análise, são apresentados dois algoritmos (FMA-PFS-RF e FMA-PFS-ENS-8) com duas medidas de desempenho (AUC e *Balance*). Com isso, ficou um total de 4 experimentos a serem apresentados.

A Tabela 4.3 apresenta os experimentos realizados com a utilização do meta-aprendiz FMA-PFS-RF utilizando a medida de desempenho AUC. Nota-se que a estratégia FMA-PFS-RF, com o valor de (2.457), ficou um pouco acima, em comparação com o algoritmo RF (2,365). Estes dois valores representam um valor muito próximo ao valor ideal de 1.5. O segundo algoritmo, que mais se aproxima desses valores, é o algoritmo XGB (3.083). O terceiro algoritmo, com melhor desempenho é o AB (4.722). E o algoritmo que obteve a pior média foi o SVM, com valor médio de 8.409.

Tabela 4.3: Amostra de 10 conjuntos de dados do projeto da NASA com a média (μ) e desvio padrão (σ) dos *rankings* gerados segundo o critério AUC para os 115 conjuntos de dados. Valores médios de FMA-PFS-RF em comparação com os 8 algoritmos de AM.

BASE	FMA-PFS-RF	NB	RF	C4.5	*k*-NN	SVM	MLP	AB	XGB
CM1	2.500	5.000	2.500	8.000	6.000	9.000	7.000	4.000	**1.000**
JM1	**1.500**	6.000	**1.500**	8.000	7.000	9.000	5.000	4.000	3.000
KC1	**1.500**	4.000	**1.500**	8.000	7.000	9.000	6.000	5.000	3.000
KC2	2.500	**1.000**	2.500	8.000	7.000	9.000	5.000	6.000	4.000
KC3	2.500	4.000	**1.000**	8.000	6.000	9.000	7.000	5.000	2.500
MC1	**1.500**	6.000	5.000	8.000	3.000	9.000	7.000	4.000	**1.500**
MC2	**1.500**	4.000	3.000	7.000	6.000	9.000	5.000	8.000	**1.500**
MW1	**1.500**	5.000	3.000	8.000	6.000	9.000	7.000	4.000	**1.500**
PC1	2.500	6.000	2.500	8.000	5.000	9.000	7.000	4.000	**1.000**
PC2	2.500	**1.000**	6.000	9.000	5.000	8.000	7.000	4.000	2.500
μ	2.457	5.313	**2.365**	7.583	5.722	8.409	5.348	4.722	3.083
σ	1.471	2.199	1.239	1.532	1.613	**1.118**	1.621	1.730	1.718

A Tabela 4.4 apresenta os experimentos realizados com o meta-aprendiz FMA-PFS-ENS-8. Nota-se que a estratégia FMA-PFS-ENS-8 ficou em média com o valor de 2.578. O algoritmo RF com valor médio de 2.357 ficou em primeiro lugar. O segundo algoritmo que mais se aproxima destes valores é o algoritmo XGB (3.070) ficando na terceira posição. O terceiro algoritmo com melhor desempenho é o AB (4.717), e o algoritmo que obteve a pior média foi o SVM, com valor médio de 8.383.

Os resultados destas duas amostras para a primeira medida de desempenho AUC confirmam o que algumas pesquisas [LBMP08, Mal15] relatam sobre os algoritmos com estratégia de *ensemble* (RF, AB, XGB). Tais algoritmos obtém os melhores desempenhos para a predição de falhas de software. No entanto, as técnicas FMA-PFS-RF (2.457) e FMA-PFS-ENS-8 (2.578) obtiveram valores próximos ao algoritmo RF (2.365) e (2.357). Se fosse levado em consideração alguma estatística para critérios de avaliação, essa diferença mí-

Tabela 4.4: Amostra de 10 conjuntos de dados do projeto da NASA com a média (μ) e desvio padrão (σ) dos *rankings* gerados segundo o critério AUC para os 115 conjuntos de dados. Valores médios de FMA-PFS-ENS-8 em comparação com os 8 algoritmos de AM.

BASE	FMA-PFS-ENS-8	NB	RF	C4.5	*k*-NN	SVM	MLP	AB	XGB
CM1	**1.500**	5.000	3.000	8.000	6.000	9.000	7.000	4.000	**1.500**
JM1	**1.500**	6.000	**1.500**	8.000	7.000	9.000	5.000	4.000	3.000
KC1	**1.500**	4.000	**1.500**	8.000	7.000	9.000	6.000	5.000	3.000
KC2	2.500	**1.000**	2.500	8.000	7.000	9.000	5.000	6.000	4.000
KC3	2.500	4.000	**1.000**	8.000	6.000	9.000	7.000	5.000	2.500
MC1	4.500	6.000	4.500	8.000	2.000	9.000	7.000	3.000	**1.000**
MC2	**1.500**	4.000	3.000	7.000	6.000	9.000	5.000	8.000	**1.500**
MW1	**1.500**	5.000	3.000	8.000	6.000	9.000	7.000	4.000	**1.500**
PC1	2.500	6.000	2.500	8.000	5.000	9.000	7.000	4.000	**1.000**
PC2	2.500	**1.000**	6.000	9.000	5.000	8.000	7.000	4.000	2.500
μ	2.578	5.309	**2.357**	7.574	5.687	8.383	5.326	4.717	3.070
σ	1.684	2.182	1.237	1.555	1.638	**1.198**	1.621	1.738	1.669

nima poderia ser afirmada como um empate técnico, considerando que o *framework* FMA-PFS conseguiu obter bons resultados, se comparados com o algoritmo que é dito como um dos algoritmos mais recomendados para a predição de falhas de software.

Para os resultados a seguir, a medida *Balance* foi o critério para a escolha dos algoritmos de aprendizado de máquina. Se os resultados anteriores apresentaram um empate técnico, agora as técnicas FMA-PFS-RF e FMA-PFS-ENS-8 puderam apresentar resultados que demonstram a eficácia do *framework* FMA-PFS.

A Tabela 4.5 apresenta os experimentos realizados para a medida de desempenho *Balance*. Nota-se que a estratégia FMA-PFS-RF com o valor médio de 2.770 foi o valor mais próximo do ideal. O primeiro algoritmo que obteve a segunda melhor média foi o NB com o valor de 4.200. O segundo algoritmo, RF, aparece em terceiro lugar e ficou com o valor médio de 4.326. E, em último lugar, mais uma vez foi o algoritmo SVM com um valor médio de 7.583.

Tabela 4.5: Amostra de 10 conjuntos de dados do projeto da NASA com a média (μ) e desvio padrão (σ) dos *rankings* gerados segundo o critério *Balance* para os 115 conjuntos de dados. Valores médios de FMA-PFS-RF em comparação com os 8 algoritmos de AM.

BASE	FMA-PFS-RF	NB	RF	C4.5	*k*-NN	SVM	MLP	AB	XGB
CM1	**1.500**	**1.500**	7.000	3.000	6.000	9.000	5.000	8.000	4.000
JM1	**1.500**	5.000	3.000	4.000	**1.500**	9.000	7.000	8.000	6.000
KC1	4.500	**1.000**	2.000	3.000	4.500	8.000	7.000	9.000	6.000
KC2	4.500	8.000	2.000	6.000	**1.000**	9.000	7.000	4.500	3.000
KC3	**1.500**	**1.500**	8.000	3.000	7.000	9.000	4.000	6.000	5.000
MC1	**1.500**	**1.500**	3.000	6.000	4.000	9.000	5.000	8.000	7.000
MC2	**1.500**	4.000	7.000	3.000	5.000	9.000	**1.500**	8.000	6.000
MW1	**1.500**	**1.500**	7.000	6.000	8.000	9.000	4.000	3.000	5.000
PC1	2.500	2.500	4.000	5.000	**1.000**	8.000	7.000	9.000	6.000
PC2	**1.500**	**1.500**	6.000	8.000	7.000	4.000	3.000	5.000	9.000
μ	**2.770**	4.200	4.326	4.504	5.213	7.583	4.565	5.648	6.191
σ	1.834	2.988	2.149	2.004	2.156	1.969	**1.823**	2.502	2.325

A Tabela 4.6 novamente apresenta os resultados para a medida de desempenho *Balance*. Nota-se que a estratégia FMA-PFS-ENS-8 ficou com um valor médio de 2.883

(sendo este valor o mais próximo do ideal). O algoritmo que obteve a segunda melhor média foi o NB com o valor de 4.174. O algoritmo RF aparece em terceiro lugar e ficou com o valor médio de 4.322. E em último lugar, mais uma vez foi o algoritmo SVM com um valor médio de 7.591.

Tabela 4.6: Amostra de 10 conjuntos de dados do projeto da NASA com a média (μ) e desvio padrão (σ) dos *rankings* gerados segundo o critério *Balance* para os 115 conjuntos de dados. Valores médios de FMA-PFS-ENS-8 em comparação com os 8 algoritmos de AM.

BASE	FMA-PFS-ENS-8	NB	RF	C4.5	*k*-NN	SVM	MLP	AB	XGB
CM1	**1.500**	**1.500**	7.000	3.000	6.000	9.000	5.000	8.000	4.000
JM1	**1.500**	5.000	3.000	4.000	**1.500**	9.000	7.000	8.000	6.000
KC1	4.500	**1.000**	2.000	3.000	4.500	8.000	7.000	9.000	6.000
KC2	7.500	7.500	2.000	5.000	**1.000**	9.000	6.000	4.000	3.000
KC3	**1.500**	**1.500**	8.000	3.000	7.000	9.000	4.000	6.000	5.000
MC1	**1.500**	**1.500**	3.000	6.000	4.000	9.000	5.000	8.000	7.000
MC2	**1.500**	4.000	7.000	3.000	5.000	9.000	**1.500**	8.000	6.000
PC1	2.500	2.500	4.000	5.000	**1.000**	8.000	7.000	9.000	6.000
PC2	**1.500**	**1.500**	6.000	8.000	7.000	4.000	3.000	5.000	9.000
μ	**2.883**	4.174	4.322	4.491	5.183	7.591	4.543	5.639	6.174
σ	1.976	2.939	2.143	2.004	2.160	1.947	**1.857**	2.540	2.315

Os resultados da medida de desempenho *Balance* puderam comprovar a eficácia do *framework* FMA-PFS. Na média, os valores de FMA-PFS-RF e FMA-PFS-ENS-8 foram superiores ao algoritmo NB que ficou em segundo lugar. Ficou evidente nos experimentos realizados em relação ao *Balance* que o algoritmo NB pode ser considerado um algoritmo que gera um modelo preditivo eficaz para o domínio de predição de falhas de software. De fato, isto foi evidenciado por Menzies et al. [MGF07]. No entanto, se para a medida de desempenho AUC o *framework* FMA-PFS-RF e FMA-PFS-ENS-8 não conseguiram estabelecer um ganho e apenas empatou com o algoritmo RF, para a medida *Balance* um desempenho superior ficou evidente com os resultados apresentados.

4.5 Considerações Finais

No presente capítulo, foram analisadas experimentalmente as abordagens de recomendação de algoritmo(s) por meio de um *ranking* e a recomendação única que escolhe o algoritmo mais adequado para um novo projeto de software. Essas duas abordagens de recomendação representam bem as formas de sugestões de qualquer sistema de recomendação de algoritmos baseado em meta-aprendizado [Sou10].

Os resultados foram estratificados de acordo com as duas estratégias de recomendação. Para a recomendação de um *ranking*, foram empregadas duas abordagens, sendo uma baseada em vizinhos mais próximos (FMA-PFS-R*k*) e a outra sendo uma abordagem baseada em *ensemble* de algoritmos de árvore de decisão (FMA-PFS-RA). A avalia-

ção experimental conduzida comprovou que FMA-PFS-RA superou a estratégia FMA-PFS-R*k*, mas principalmente conseguiu superar o *ranking* aleatório (RA) e o *ranking* majoritário (MAJ-R). A vantagem mínima dos meta-aprendizes FMA-PFS-R*k* e FMA-PFS-RA para MAJ-R foi ocorrida segundo o critério de *rankings* pela medida de desempenho AUC. Com o *Balance*, os meta-aprendizes obtiveram clara vantagem, comprovando que usar a técnica de meta-aprendizado para recomendar *rankings* de algoritmos é eficaz para o domínio de PFS.

No segundo paradigma, a proposta era recomendar um único algoritmo que tenha gerado o melhor modelo dentre cada um dos 115 conjuntos de dados analisados. Para o *framework* de recomendação única, foram utilizados 2 meta-aprendizes: FMA-PFS-RF e FMA-PFS-ENS-8. Cada um dos meta-aprendizes foi comparado com os 8 algoritmos de AM. Utilizando o critério de AUC para identificar a ordem que os 8 algoritmos tiveram para os conjuntos de dados, o algoritmo RF apresentou uma vantagem mínima sobre os meta-aprendizes FMA-PFS-RF e FMA-PFS-ENS-8. Por outro lado, o *Balance* conseguiu superar a recomendação de um único algoritmo. Os meta-aprendizes FMA-PFS-RF e FMA-PFS-ENS-8 demonstraram clara vantagem sobre os 8 algoritmos de AM.

5. CONCLUSÃO

A proposta desenvolvida nesta dissertação foi de apresentar um *framework* para a recomendação de algoritmo(s) de aprendizado de máquina para o domínio de predição de falhas de software por meio da técnica de meta-aprendizado. A proposta visa contribuir em duas áreas. Da perspectiva da técnica utilizada, propôs-se afirmar a técnica de meta-aprendizado para problemas reais de domínio específico. A técnica de meta-aprendizado já vem sendo abordada para problemas oriundos de âmbitos diversos, como os representados pelos conjuntos de dados disponíveis em repositórios de propósito geral (por exemplo, o repositório de dados para AM da UCI [Asu07]). Meta-aprendizado também já foi experimentado para domínios específicos como por exemplo em dados de expressões gênicas [Sou10]. No entanto, tal técnica ainda não tinha sido empregada em domínios de predição de falhas de software.

Do ponto de vista da predição de falhas de software, a ideia foi combinar uma técnica que pudesse auxiliar o especialista de domínio que muitas vezes não é especialista em sistemas de classificação, tipicamente Engenheiros de Software que não tem conhecimento pleno das técnicas de aprendizado de máquina usadas para a predição de falhas de software. Isto facilitaria a análise e escolha do algoritmo de aprendizado de máquina para o projeto de software a ser utilizado.

5.1 Contribuições

A principal contribuição da pesquisa realizada nesta dissertação foi investigar empiricamente a utilização de meta-aprendizado para o contexto de predição de falhas de software. Neste contexto pesquisas [MGF07, LBMP08, SJS+11] despontavam para o problema em recomendar algoritmos de aprendizado de máquina para novos e diferentes projetos de software. No Capítulo 4, inúmeros experimentos e análises foram reportados. Os resultados obtidos forneceram evidências significativas que foram inclusive publicados em um artigo científico [dDARB16]. Além de cumprir o objetivo geral desta dissertação, que foi a proposta de construir um *framework* para a recomendação de algoritmo de aprendizado de máquina para PFS, o trabalho apresenta algumas contribuições pontuais relevantes durante o processo:

1. Recomendar algoritmos em vez de utilizar sempre o mesmo algoritmo para qualquer projeto de software;
2. Analisar e experimentar uma quantidade significativa de conjuntos de dados;
3. Corroborar com pesquisas publicadas que identificaram algoritmos de AM.

O *framewok* FMA-PFS analisa o desempenho de algoritmos de AM para uma coleção de conjuntos de dados e desenvolve uma meta-base para permitir a recomendação de um conjunto de algoritmos de AM para um novo conjunto de dados.

Os experimentos realizados utilizaram 115 conjuntos de dados público do repositório de dados tera-PROMISE [Men15]. Estes conjuntos de dados estão dispostos para a predição de falhas de software. Tal quantidade significativa ainda não tinha sido experimentada até este presente momento, pois algumas pesquisas [MGF07, LBMP08, SJS+11] apresentaram seus resultados para os conjuntos de dados oriundos de projetos da NASA, que representa em média 10 dos conjuntos de dados.

Além disso, os experimentos corroboraram para pesquisas anteriores que recomendaram o melhor algoritmo de aprendizado de máquina, Menzies et al. [MGF07] argumentaram a utilização do algoritmo NB para a geração de um modelo preditivo para predição de falhas de software. Através de experimentos realizados com base na medida de desempenho *Balance* sendo escolhido para definir os critérios de recomendação, os resultados mostraram que o algoritmo NB teve resultados inferiores apenas para o *framework* proposto.

Já a pesquisa feita por Lessmann et al. [LBMP08], através de seus experimentos concluiu que o algoritmo RF era um candidato para geração de um modelo preditivo para o domínio de predição de falhas de software. Desta vez o *framework* proposto FMA-PFS trouxe resultados através da ordenação dos algoritmos baseados pela medida de avaliação AUC, que o algoritmo RF teve uma pequena vantagem sobre o *framework* proposto nesta pesquisa.

5.2 Limitações

Os resultados reportados nesta dissertação cumpriram com o objetivo geral. No entanto, pode-se destacar algumas limitações nos experimentos conduzidos.

Uma das formas de recomendação de algoritmos propostas nesta dissertação foi a recomendação de *rankings* através de dois meta-aprendizes. Embora os resultados analisados tenham sido superiores as estratégias de *ranking* aleatório (RA) e *ranking* majoritário (MAJ-R), as médias do Coeficiente de Correlação de Spearman para FMA-PFS-R*k* (0.758) e FMA-PFS-RA (0.765) em comparação com a média do *ranking* majoritário MAJ-R (0.747) ficaram com uma diferença mínima para os experimentos conduzidos através da medida de desempenho AUC.

Outra forma de recomendação é a escolha do algoritmo de aprendizado de máquina mais adequado para um determinado conjunto de dados. Nesta situação e novamente

avaliando os resultados utilizados através da medida de desempenho AUC, que o algoritmo RF demonstrou uma eficácia superior a FMA-PFS-RF e FMA-PFS-ENS-8 respectivamente.

De fato, os problemas relatados acima podem ter sido acarretados pela não consideração da significância das diferenças entre os desempenhos dos algoritmos de aprendizado de máquina. Sendo assim, os algoritmos de AM podem ter ficado em posições diferentes na ordem de preferência, mas ter comportamentos estatisticamente semelhantes e, portanto, serem utilizados de maneira indiscriminada. Este problema é largamente difundido nos estudos de recomendação utilizando meta-aprendizado e uma solução ainda não é consenso [BCSV08, Sou10].

5.3 Trabalhos futuros

Esta pesquisa representa um primeiro esforço para a aplicação da técnica de meta-aprendizado para a recomendação de algoritmos de aprendizado de máquina para o domínio de predição de falhas de software. O principal objetivo neste trabalho foi recomendar algoritmos através de um modelo construído a partir de uma meta-base para um novo conjunto de dados de um projeto de software. A importância de recomendar um algoritmo baseado no novo conjunto de dados a ser analisado foi largamente debatida em [MGF07, LBMP08, SJS+11].

No entanto, este trabalho analisou o desempenho de 8 algoritmos de AM e a exploração de mais algoritmos de AM poderiam reforçar e ampliar o objetivo principal desta pesquisa. Lessman et al. [LBMP08] analisaram 22 algoritmos em sua pesquisa e isso poderia ser considerado a ser experimentado em novos estudos.

Além disso, este trabalho estabeleceu apenas a utilização de conjuntos de dados para as métricas de produto. Basicamente, estas métricas são coletadas a partir da conclusão do desenvolvimento do produto. No entanto, para garantir a qualidade de um software, existem uma série de outras métricas, como por exemplo, métricas de processo e de projeto. Algumas dessas métricas já foram experimentadas com técnicas de aprendizado de máquina [Mal15] e também poderiam ser adicionadas a trabalhos futuros.

Por fim, para apresentar definitivamente a técnica de meta-aprendizado para a área de Engenharia de Software, a técnica de meta-aprendizado no contexto de âmbito geral poderia ser relacionada com meta-aprendizado para predição de falhas de software. Uma maneira de comprovar essa teoria seria gerar um modelo de meta-aprendizado com o conjunto de dados de um repositório público [Asu07] para o propósito de âmbito geral. Posteriormente, aplicar o modelo para recomendar algoritmos de aprendizado de máquina para conjuntos de dados no contexto de predição de falhas de software.

REFERÊNCIAS BIBLIOGRÁFICAS

[Asu07] Asuncion, A., N. D. “UCI machine learning repository”, 2007.

[BCSV08] Brazdil, P.; Carrier, C. G.; Soares, C.; Vilalta, R. “Metalearning: Applications to Data Mining”. Springer, 2008.

[BGCK00] Bensusan, H.; Giraud-Carrier, C.; Kennedy, C. “A higher-order approach to meta-learning”, Relatório Técnico, Bristol, UK, 2000.

[BH94] Brazdil, P. B.; Henery, R. J. “Machine learning, neural and statistical classification”. , Michie, D.; Spiegelhalter, D. J.; Taylor, C. C.; Campbell, J. (Editores), Ellis Horwood, 1994, pp. 175–212.

[BH06] Basu, M.; Ho, T. K. “Data Complexity in Pattern Recognition”. Springer London, 2006.

[Bis06] Bishop, C. M. “Pattern Recognition and Machine Learning (Information Science and Statistics)”. Springer-Verlag New York, Inc., 2006.

[BMPZ02] Basili, V. R.; McGarry, F. E.; Pajerski, R.; Zelkowitz, M. V. “Lessons learned from 25 years of process improvement: The rise and fall of the nasa software engineering laboratory”. In: Proceedings of the 24th International Conference on Software Engineering, 2002, pp. 69–79.

[Bre01] Breiman, L. “Random forests”, *Machine Learning*, vol. 45, 2001, pp. 5–32.

[BSC03] Brazdil, P. B.; Soares, C.; Costa, J. P. d. “Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results”, *Machine Learning*, vol. 50, 2003, pp. 251–277.

[BWH08] Bouguila, N.; Wang, J. H.; Hamza, A. “A bayesian approach for software quality prediction”. In: Proceedings of the 4th International Conference in Intelligent Systems, 2008, pp. 11–49–11–54.

[CG] Chen, T.; Guestrin, C. “Xgboost: A scalable tree boosting system”, *Computer Research Repository*, vol. abs/1603.02754, pp. 1–14.

[CK94] Chidamber, S. R.; Kemerer, C. F. “A metrics suite for object oriented design”, *IEEE Transactions on Software Engineering*, vol. 20, 1994, pp. 476–493.

[Cou13] Couto, C. “Predicting software defects with causality tests”, Tese de Doutorado, UFMG, 2013.

[dDARB16] das Dôres, S. N.; Alves, L.; Ruiz, D. D.; Barros, R. C. “A meta-learning framework for algorithm recommendation in software fault prediction”. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing, 2016, pp. 1486–1491.

[Dem06] Demšar, J. “Statistical comparisons of classifiers over multiple data sets”, *Machine Learning Research*, vol. 7, 2006, pp. 1–30.

[Die98] Dietterich, T. G. “Approximate statistical tests for comparing supervised classification learning algorithms”, *Neural Computation*, vol. 10, 1998, pp. 1895–1923.

[EBGR01] Emam, K. E.; Benlarbi, S.; Goel, N.; Rai, S. N. “Comparing case-based reasoning classifiers for predicting high risk software components”, *Journal of Systems and Software*, vol. 55, 2001, pp. 301 – 320.

[FO00] Fenton, N. E.; Ohlsson, N. “Quantitative analysis of faults and failures in a complex software system”, *IEEE Transactions on Software Engineering*, vol. 26, 2000, pp. 797–814.

[FS96] Freund, Y.; Schapire, R. E. “Experiments with a new boosting algorithm”, 1996.

[GCVB04] Giraud-Carrier, C.; Vilalta, R.; Brazdil, P. “Introduction to the special issue on meta-learning”, *Machine Learning*, vol. 54, 2004, pp. 187–193.

[GFLDC11] Gama, J.; Faceli, K.; Lorena, A.; De Carvalho, A. “Inteligência Artificial: Uma Abordagem de Aprendizado de Máquina”. Grupo Gen - LTC, 2011.

[GMCS04] Guo, L.; Ma, Y.; Cukic, B.; Singh, H. “Robust prediction of fault-proneness by random forests”. In: Proceedings of the 15th International Symposium on Software Reliability Engineering, 2004, pp. 417–428.

[GPS+10] Gomes, T.; Prudencio, R.; Soares, C.; Rossi, A.; Carvalho, A. “Combining meta-learning and search techniques to SVM parameter selection”. In: Proceedings of the 11th Brazilian Symposium on Neural Networks, 2010, pp. 79–84.

[Hal77] Halstead, M. H. “Elements of Software Science (Operating and Programming Systems Series)”. Elsevier Science Inc., 1977.

[HB02] Ho, T. K.; Basu, M. “Complexity measures of supervised classification problems”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, 2002, pp. 356–370.

[HFH+09] Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; Witten, I. H. “The weka data mining software: An update”, *SIGKDD Explorations Newsletter*, vol. 11, 2009, pp. 10–18.

[iee90] "IEEE standard glossary of software engineering terminology", *IEEE Std 610.12-1990*, 1990, pp. 1–84.

[IHGC12] Ince, D. C.; Hatton, L.; Graham-Cumming, J. "The case for open computer programs", *Nature*, vol. 482, 2012, pp. 485–488.

[KAHA97] Khoshgoftaar, T.; Allen, E.; Hudepohl, J.; Aud, S. "Application of neural networks to software quality modeling of a very large telecommunications system", *IEEE Transactions on Neural Networks*, vol. 8, 1997, pp. 902–909.

[Kal02] Kalousis, A. "Algorithm selection via meta-learning", Tese de Doutorado, Geneva, 2002.

[Kan02] Kan, S. H. "Metrics and Models in Software Quality Engineering". Addison-Wesley Longman Publishing Co., Inc., 2002.

[KS03] Khoshgoftaar, T. M.; Seliya, N. "Analogy-based practical classification rules for software quality estimation", *Empirical Software Engineering*, vol. 8, 2003, pp. 325–350.

[LBMP08] Lessmann, S.; Baesens, B.; Mues, C.; Pietsch, S. "Benchmarking classification models for software defect prediction: A proposed framework and novel findings", *IEEE Transactions on Software Engineering*, vol. 34, 2008, pp. 485–496.

[Mal15] Malhotra, R. "A systematic review of machine learning techniques for software fault prediction", *Applied Soft Computing*, vol. 27, 2015, pp. 504–518.

[McC76] McCabe. "A complexity measure", *IEEE Transactions on Software Engineering*, vol. 2, 1976, pp. 308–320.

[Men15] Menzies, T., K. R. P. D. "The promise repository of empirical software engineering data", 2015.

[MGF07] Menzies, T.; Greenwald, J.; Frank, A. "Data mining static code attributes to learn defect predictors", *IEEE Transactions on Software Engineering*, vol. 33, 2007, pp. 2–13.

[Mit97] Mitchell, T. M. "Machine Learning". McGraw-Hill, Inc., 1997.

[MPCS12] Miranda, P. B. C. d.; Prudêncio, R. B. C.; Carvalho, A. C. P. L. F. d.; Soares, C. "An experimental study of the combination of meta-learning with particle swarm algorithms for SVM parameter selection". , Murgante, B.; Gervasi, O.; Misra, S.; Nedjah, N.; Rocha, A. M. A. C.; Taniar, D.; Apduhan, B. O. (Editores), Springer Berlin Heidelberg, 2012, pp. 562–575.

[MRS+02] Menzies, T.; Raffo, D.; Setamanit, S.-o.; Hu, Y.; Tootoonian, S. “Model-based tests of truisms”. In: Proceedings of the 17th International Conference on Automated Software Engineering, 2002, pp. 183–183.

[MRV+15] Mantovani, R. G.; Rossi, A. L. D.; Vanschoren, J.; Bischl, B.; Carvalho, A. C. P. L. F. “To tune or not to tune: Recommending when to adjust SVM hyper-parameters via meta-learning”. In: International Joint Conference on Neural Networks, 2015, pp. 1–8.

[MSS05] Myrtveit, I.; Stensrud, E.; Shepperd, M. “Reliability and validity in comparative studies of software prediction models”, *IEEE Transactions on Software Engineering*, vol. 31, 2005, pp. 380–391.

[NW88] Neave, H. R.; Worthington, P. L. “Distribution-free tests / H.R. Neave and P.L. Worthington”. Unwin Hyman London ; Boston, 1988.

[Pon08] Pontes, M. B. “Predição de defeitos em software”, *Engenharia de Software Magazine*, vol. 17, 2008, pp. 8–13.

[Pre01] Pressman, R. S. “Software Engineering: A Practitioner’s Approach”. McGraw-Hill Higher Education, 2001.

[QT03] Quah, T.-S.; Thwin, M. M. T. “Application of neural networks for software quality prediction using object-oriented metrics”. In: International Conference on Software Maintenance, 2003, pp. 116–125.

[Qui93] Quinlan, J. R. “C4.5: Programs for Machine Learning”. Morgan Kaufmann Publishers Inc., 1993.

[RHC10] Raeder, T.; Hoens, T.; Chawla, N. “Consequences of variability in classifier performance estimates”. In: Proceedings of the 10th International Conference on Data Mining (ICDM), 2010, pp. 421–430.

[SBB+02] Shull, F.; Basili, V.; Boehm, B.; Brown, A. W.; Costa, P.; Lindvall, M.; Port, D.; Rus, I.; Tesoriero, R.; Zelkowitz, M. “What we have learned about fighting defects”. In: Proceedings of the 8th International Symposium on Software Metrics, 2002, pp. 249.

[SBH14] Shepperd, M.; Bowes, D.; Hall, T. “Researcher bias: The use of machine learning in software defect prediction”, *IEEE Transactions on Software Engineering*, vol. 40, 2014, pp. 603–616.

[SBK04] Soares, C.; Brazdil, P. B.; Kuba, P. “A meta-learning method to select the kernel width in support vector regression”, *Machine Learning*, vol. 54, 2004, pp. 195–209.

[SI94] Shepperd, M.; Ince, D. “A critique of three metrics”, *Systems and Software*, vol. 26, 1994, pp. 197 – 210.

[SJS+11] Song, Q.; Jia, Z.; Shepperd, M.; Ying, S.; Liu, J. “A general software defect-proneness prediction framework”, *IEEE Transactions on Software Engineering*, vol. 37, 2011, pp. 356–370.

[Som04] Sommerville, I. “Software Engineering (7th Edition)”. Pearson Addison Wesley, 2004.

[Sou10] de Souza, B. F. “Meta-aprendizagem aplicada à classificação de dados de expressão gênica”, Tese de Doutorado, USP - São Carlos, 2010.

[SSZ12] Sun, Z.; Song, Q.; Zhu, X. “Using coding-based ensemble learning to improve software defect prediction”, *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 42, 2012, pp. 1806–1817.

[Sun14] Sun, Q. “Meta-learning and the full model selection problem”, Tese de Doutorado, The University of Waikato, 2014.

[TB09] Turhan, B.; Bener, A. “Analysis of naive bayes’ assumptions on software fault data: An empirical study”, *Data & Knowledge Engineering*, vol. 68, 2009, pp. 278–290.

[TSK05] Tan, P.-N.; Steinbach, M.; Kumar, V. “Introduction to Data Mining, (First Edition)”. Addison-Wesley Longman Publishing Co., Inc., 2005.

[TTB08] Tosun, A.; Turhan, B.; Bener, A. “Ensemble of software defect predictors: A case study”. In: Proceedings of the 2nd International Symposium on Empirical Software Engineering and Measurement, 2008, pp. 318–320.

[Vap95] Vapnik, V. N. “The Nature of Statistical Learning Theory”. Springer-Verlag New York, Inc., 1995.

[VGcSB05] Vilalta, R.; Giraud-carrier, C.; Sa, E. I.; Brazdil, P. “Chapter 1 meta-learning concepts and techniques”, 2005.

[WM95] Wolpert, D. H.; Macready, W. G. “No free lunch theorems for search”, 1995.

[Wol96] Wolpert, D. H. “The lack of a priori distinctions between learning algorithms”, *Neural Computation*, vol. 8, 1996, pp. 1341–1390.

[Wol01] Wolpert, D. H. “The supervised learning no-free-lunch theorems”. In: Proceedings of the 6th Online World Conference on Soft Computing in Industrial Applications, 2001, pp. 25–42.

[Zar99] Zar, J. “Biostatistical Analysis”. Prentice Hall, 1999.

APÊNDICE A – TABELAS

Tabela A.1: Coeficiente de Correlação de Spearman dos 115 conjuntos de dados com a Média (μ) e desvio padrão (σ). Para FMA-PFS-R*k*, FMA-PFS-RA, MAJ-R e RA, considerando a ordenação dos valores pela medida de desempenho AUC.

BASE	FMA-PFS-R*k*	FMA-PFS-RA	MAJ-R	RA
ant-1.3	0.747	**0.786**	0.690	0.014
ant-1.4	0.786	0.810	**0.905**	0.042
ant-1.5	**0.905**	**0.905**	0.833	-0.072
ant-1.6	0.929	**1.000**	0.929	0.068
ant-1.7	**0.898**	0.881	0.881	-0.132
ar1	0.455	0.548	**0.571**	0.001
ar3	**0.419**	0.286	0.214	-0.045
ar4	**0.826**	0.667	0.810	-0.043
ar5	**0.602**	0.238	0.167	-0.007
ar6	0.275	0.476	**0.548**	0.002
arc	**0.880**	0.738	0.810	-0.117
berek	0.439	0.476	**0.524**	-0.064
camel-1.0	0.695	**0.810**	0.714	0.033
camel-1.2	**0.857**	0.690	0.690	-0.146
camel-1.4	0.786	0.857	**0.881**	0.071
camel-1.6	0.881	**0.952**	0.952	-0.039
ckjm	0.108	0.190	**0.238**	0.001
CM1-v2	0.595	**0.970**	0.952	0.028
CM1	0.934	**0.952**	0.905	-0.027
datatrieve	0.647	0.738	**0.810**	0.154
e-learning	0.683	**0.786**	0.690	-0.009
eclipse34$_{d}$*ebug*	**0.228**	0.119	0.119	0.040
eclipse34$_{s}$*wt*	**0.952**	0.667	0.667	0.067
forrest-0.7	0.467	0.500	**0.643**	0.021
forrest-0.8	**0.575**	0.036	0.048	0.083
intercafe	**0.371**	0.119	0.286	-0.084
ivy-1.1	0.228	**0.476**	0.381	0.064
ivy-1.4	0.786	**0.857**	0.786	0.041
ivy-2.0	0.810	**0.976**	0.952	0.051
jedit-3.2	0.857	0.910	**0.929**	-0.014
jedit-4.0	0.850	0.810	**0.881**	-0.190
jedit-4.1	0.795	0.833	**0.857**	0.051
jedit-4.2	0.786	**0.810**	0.738	0.040
jedit-4.3	0.595	**0.738**	0.690	0.060
JM1-v2	**0.976**	**0.976**	0.976	0.098
JM1	0.994	**1.000**	**1.000**	-0.051
kalkulator	**0.551**	0.548	0.548	-0.084
KC1-v2	0.762	**0.905**	0.810	0.097
KC1	**0.970**	0.929	0.929	0.042
KC2-v2	**0.714**	0.619	0.595	-0.025
KC2	0.874	**0.905**	0.738	-0.008
KC3-v1	**0.905**	0.833	0.762	-0.032
KC3-v2	**0.905**	0.786	0.643	0.116
KC3	**0.970**	0.952	0.881	-0.102
KC4	0.659	0.833	**0.857**	0.156
log4j-1.0	**0.898**	0.690	0.690	0.075
log4j-1.1	0.714	**0.738**	0.595	0.073
log4j-1.2	**0.952**	0.929	0.952	0.048
lucene-2.0	0.548	0.500	**0.571**	-0.125

lucene-2.2	0.467	**0.595**	0.548	-0.063
lucene-2.4	**0.970**	0.833	0.786	-0.108
MC1-v1	**0.838**	0.833	0.643	0.074
MC1-v2	**0.976**	0.929	0.929	-0.002
MC1	**0.838**	0.762	0.643	0.062
MC2-v1	**0.802**	0.690	0.452	0.062
MC2-v2	**0.905**	0.786	0.643	0.032
MC2	**0.922**	0.881	0.714	0.060
mozilla4	**0.747**	0.690	0.667	-0.063
MW1-v1	**0.683**	0.548	0.476	0.073
MW1-v2	**0.590**	0.452	0.524	0.024
MW1	0.934	**0.952**	0.905	0.066
nieruchomosci	**0.524**	0.476	0.476	-0.056
pbeans1	0.204	**0.833**	0.810	0.010
pbeans2	0.619	**0.762**	0.667	0.058
PC1-v1	0.934	0.976	**1.000**	0.078
PC1-v2	0.934	0.976	**1.000**	-0.067
PC1	0.898	**0.905**	0.881	-0.056
PC2-v1	**0.892**	0.881	0.857	-0.007
PC2-v2	0.833	0.857	**0.881**	-0.035
PC2	0.708	**0.738**	0.500	-0.021
PC3-v1	**0.976**	0.929	0.976	-0.090
PC3-v2	**0.976**	0.929	0.976	-0.044
PC3	0.988	0.976	**1.000**	-0.052
PC4-v1	**1.000**	0.976	**1.000**	-0.095
PC4-v2	**1.000**	0.976	**1.000**	-0.095
PC4	**1.000**	0.976	**1.000**	-0.059
PC5-v1	0.838	0.929	**0.929**	-0.028
PC5-v2	**0.976**	**0.976**	0.976	0.138
PC5	0.874	**0.976**	0.905	0.005
pdftranslator	0.611	0.762	**0.762**	-0.077
poi-1.5	**0.934**	0.905	0.857	-0.182
poi-2.0	0.874	0.881	**0.881**	0.082
poi-2.5	**0.929**	0.786	0.786	0.017
poi-3.0	**0.892**	0.833	0.833	-0.048
prop-1	**1.000**	0.881	0.786	-0.043
prop-2	**0.970**	0.881	0.786	-0.084
prop-3	**1.000**	0.881	0.786	0.011
prop-4	**1.000**	0.881	0.786	-0.001
prop-5	**1.000**	0.786	0.786	-0.002
prop-6	0.790	**0.857**	0.857	-0.036
redaktor	0.524	0.762	**0.810**	0.019
serapion	0.857	0.857	**0.881**	0.021
skarbonka	**0.643**	0.595	0.500	-0.033
sklebagd	0.551	0.667	**0.667**	-0.127
synapse-1.0	0.488	**0.619**	0.571	0.023
synapse-1.1	0.778	**0.833**	0.786	0.024
synapse-1.2	0.857	0.714	**0.905**	0.009
systemdata	0.424	**0.476**	0.357	-0.083
szybkafucha	**0.881**	0.690	0.690	0.013
termoproject	0.333	0.595	**0.667**	-0.036
tomcat	**0.976**	0.929	0.929	0.077
velocity-1.4	0.886	0.857	**0.905**	-0.039
velocity-1.5	**0.881**	0.833	0.810	-0.056
velocity-1.6	0.922	0.946	**0.976**	0.048
workflow	-0.381	0.310	**0.333**	-0.003
wspomaganiepi	0.452	**0.738**	0.738	-0.034
xalan-2.4	**1.000**	0.976	0.976	-0.017
xalan-2.5	**0.854**	0.738	0.690	0.048
xalan-2.6	**0.881**	0.833	0.833	0.064

xalan-2.7	0.868	**0.881**	0.833	0.074
xerces-1.2	0.802	0.810	**0.810**	-0.006
xerces-1.3	**0.762**	0.738	0.738	0.035
xerces-1.4	0.738	**0.833**	0.833	-0.054
xerces-init	0.429	0.762	**0.857**	0.175
zuzel	0.898	**0.952**	0.905	-0.041
μ	0.758	**0.765**	0.747	-0.002
σ	**0.234**	0.204	0.206	0.070

Tabela A.2: Coeficiente de Correlação de Spearman dos 115 conjuntos de dados com a Média (μ) e desvio padrão (σ). Para FMA-PFS-R*k*, FMA-PFS-RA, MAJ-R e RA, considerando a ordenação dos valores pela medida de desempenho *Balance*.

BASE	FMA-PFS-R*k*	FMA-PFS-RA	MAJ-R	RA
ant-1.3	0.238	**0.810**	-0.357	-0.093
ant-1.4	-0.095	**0.333**	-0.048	0.051
ant-1.5	0.262	**0.452**	-0.262	0.117
ant-1.6	0.476	**0.595**	0.286	0.031
ant-1.7	-0.024	**0.048**	-0.071	0.029
ar1	0.595	**0.881**	-0.619	-0.053
ar3	0.524	**0.833**	-0.333	-0.025
ar4	**0.976**	0.714	-0.405	-0.016
ar5	-0.357	**0.333**	-0.214	0.004
ar6	**0.643**	0.310	-0.071	0.023
arc	0.405	**0.476**	-0.357	0.071
berek	0.310	**0.643**	-0.667	-0.017
camel-1.0	0.714	**0.762**	-0.619	0.052
camel-1.2	0.333	0.762	**0.881**	-0.002
camel-1.4	**0.881**	0.714	0.286	0.070
camel-1.6	**0.881**	0.810	0.071	0.021
ckjm	0.381	**0.595**	0.238	0.009
CM1-v2	0.429	**0.714**	0.048	-0.016
CM1	0.714	**0.810**	0.048	-0.048
datatrieve	**0.595**	0.476	-0.619	0.029
e-learning	-0.381	**0.810**	-0.095	0.018
eclipse34$_d$*ebug*	**0.310**	0.084	0.262	0.062
eclipse34$_s$*wt*	0.929	0.833	**1.000**	-0.020
forrest-0.7	0.071	**0.571**	-0.119	0.035
forrest-0.8	-0.381	-0.167	**-0.143**	-0.075
intercafe	-0.357	**0.190**	0.095	0.089
ivy-1.1	**0.190**	0.071	-0.167	0.009
ivy-1.4	0.238	**0.452**	-0.500	-0.110
ivy-2.0	0.405	**0.643**	-0.095	-0.063
jedit-3.2	0.595	**0.714**	0.595	-0.142
jedit-4.0	0.214	**0.571**	0.548	0.094
jedit-4.1	0.476	**0.738**	0.310	-0.078
jedit-4.2	0.262	**0.643**	-0.048	-0.056
jedit-4.3	**0.714**	0.595	-0.810	-0.114
JM1-v2	**0.952**	0.905	0.476	0.026
JM1	**0.952**	0.881	0.548	-0.006
kalkulator	**0.810**	0.595	-0.429	0.023
KC1-v2	0.167	0.262	**0.310**	0.003
KC1	0.524	**0.690**	0.238	0.118
KC2-v2	-0.024	0.238	**0.738**	-0.007
KC2	0.333	0.286	**0.738**	0.018
KC3-v1	**0.929**	0.762	-0.333	0.151
KC3-v2	**0.929**	0.667	-0.238	0.109

KC3	**0.905**	0.762	-0.190	0.061
KC4	**0.667**	**0.667**	0.429	0.067
log4j-1.0	0.048	**0.762**	-0.262	-0.017
log4j-1.1	**0.619**	0.238	-0.571	0.036
log4j-1.2	0.238	**0.286**	0.024	0.052
lucene-2.0	0.024	**0.381**	0.024	0.008
lucene-2.2	**0.929**	0.595	0.524	-0.139
lucene-2.4	**0.690**	0.548	0.476	-0.020
MC1-v1	**0.952**	0.762	0.119	-0.069
MC1-v2	0.595	**0.619**	0.214	0.001
MC1	**0.952**	0.881	0.119	0.065
MC2-v1	0.786	**0.833**	0.333	0.007
MC2-v2	**0.786**	0.595	0.571	-0.064
MC2	**0.833**	0.595	0.071	-0.054
mozilla4	-0.595	0.810	**0.976**	-0.039
MW1-v1	0.929	**0.952**	-0.310	-0.029
MW1-v2	**0.929**	0.833	-0.238	0.005
MW1	0.333	**0.595**	-0.333	0.026
nieruchomosci	**0.905**	0.850	-0.405	0.043
pbeans1	0.667	**0.714**	0.262	-0.002
pbeans2	0.048	**0.214**	0.190	-0.067
PC1-v1	**1.000**	0.714	-0.095	-0.050
PC1-v2	**1.000**	0.714	-0.095	-0.085
PC1	0.214	**0.643**	0.238	-0.102
PC2-v1	0.667	**0.810**	-0.714	0.014
PC2-v2	0.286	**0.643**	-0.667	-0.003
PC2	0.238	**0.333**	-0.786	-0.044
PC3-v1	**0.905**	0.881	0.167	-0.014
PC3-v2	0.905	**0.952**	0.119	0.069
PC3	0.429	**0.905**	0.095	-0.089
PC4-v1	**0.905**	0.667	0.476	-0.142
PC4-v2	**0.952**	0.333	0.667	-0.047
PC4	**0.905**	0.571	0.524	0.037
PC5-v1	0.905	**0.976**	0.143	0.006
PC5-v2	0.381	0.786	**0.786**	0.003
PC5	**0.905**	0.810	0.333	0.002
pdftranslator	-0.619	**0.762**	-0.048	-0.075
poi-1.5	0.690	**0.905**	0.857	-0.017
poi-2.0	0.071	0.048	**0.238**	0.016
poi-2.5	0.690	0.571	**0.929**	-0.086
poi-3.0	0.690	0.714	**0.833**	0.051
prop-1	**0.952**	0.833	0.690	0.036
prop-2	**0.952**	0.833	0.833	0.155
prop-3	**0.905**	0.833	0.833	0.017
prop-4	**0.905**	**0.905**	0.643	-0.170
prop-5	**0.952**	0.786	0.905	0.050
prop-6	0.071	**0.690**	0.024	-0.053
redaktor	0.238	**0.452**	-0.524	-0.133
serapion	0.071	**0.762**	-0.810	0.086
skarbonka	0.810	**0.929**	-0.452	0.044
sklebagd	**0.905**	0.548	-0.500	0.024
synapse-1.0	0.405	**0.786**	-0.286	0.067
synapse-1.1	0.048	**0.405**	0.119	0.017
synapse-1.2	**0.643**	**0.643**	0.429	-0.174
systemdata	0.095	**0.310**	-0.619	0.069
szybkafucha	**0.929**	0.571	-0.405	-0.040
termoproject	-0.405	-0.524	**0.071**	-0.075
tomcat	-0.119	**0.850**	-0.024	0.057
velocity-1.4	0.143	**0.429**	-0.048	0.071
velocity-1.5	0.619	**0.952**	0.667	-0.065

velocity-1.6	0.619	0.571	**0.833**	-0.017
workflow	0.357	**0.548**	0.333	0.003
wspomaganiepi	0.310	**0.762**	0.048	-0.009
xalan-2.4	0.476	**0.619**	-0.048	0.005
xalan-2.5	0.476	0.833	**1.000**	-0.079
xalan-2.6	0.024	0.738	**1.000**	0.006
xalan-2.7	0.286	**0.643**	-0.143	0.078
xerces-1.2	0.476	0.524	**0.667**	0.099
xerces-1.3	**0.476**	0.443	0.476	-0.009
xerces-1.4	0.524	**0.857**	0.500	0.008
xerces-init	0.143	**0.667**	0.429	-0.051
zuzel	**0.881**	0.786	-0.381	-0.025
μ	0.489	**0.620**	0.108	-0.003
σ	0.394	0.253	**0.472**	0.065

Tabela A.3: *Rankings* gerados segundo o critério AUC para os 115 conjuntos de dados com a média (μ) e desvio padrão (σ). Valores médios de FMA-PFS-RF em comparação com os 8 algoritmos de AM.

BASE	FMA-PFS-RF	NB	RF	C4.5	*k*-NN	SVM	MLP	AB	XGB
ant-1.3	4.500	**1.000**	3.000	8.000	7.000	9.000	6.000	2.000	4.500
ant-1.4	**1.500**	6.000	**1.500**	7.000	8.000	9.000	3.000	5.000	4.000
ant-1.5	**1.500**	4.000	**1.500**	8.000	6.000	9.000	7.000	3.000	5.000
ant-1.6	**1.500**	4.000	**1.500**	8.000	7.000	9.000	6.000	5.000	3.000
ant-1.7	**1.500**	4.000	**1.500**	8.000	6.000	9.000	7.000	5.000	3.000
ar1	3.500	6.000	2.000	8.000	**1.000**	9.000	5.000	7.000	3.500
ar3	5.500	2.000	3.000	7.000	**1.000**	9.000	5.500	4.000	8.000
ar4	4.500	4.500	3.000	9.000	6.000	8.000	7.000	**1.000**	2.000
ar5	4.500	**1.000**	2.000	8.000	3.000	7.000	4.500	6.000	9.000
ar6	2.500	6.000	5.000	9.000	4.000	8.000	**1.000**	7.000	2.500
arc	3.500	6.000	2.000	9.000	5.000	8.000	7.000	**1.000**	3.500
berek	**1.500**	3.000	4.000	8.000	6.000	9.000	**1.500**	5.000	7.000
camel-1.0	2.500	**1.000**	4.000	9.000	7.000	8.000	6.000	5.000	2.500
camel-1.2	**1.500**	7.000	**1.500**	6.000	5.000	9.000	4.000	8.000	3.000
camel-1.4	2.500	4.000	2.500	7.000	8.000	9.000	6.000	5.000	**1.000**
camel-1.6	**1.500**	7.000	**1.500**	8.000	6.000	9.000	4.000	5.000	3.000
ckjm	3.500	9.000	**1.000**	2.000	7.000	5.000	8.000	6.000	3.500
CM1-v2	**1.500**	5.000	3.000	8.000	7.000	9.000	6.000	4.000	**1.500**
CM1	2.500	5.000	2.500	8.000	6.000	9.000	7.000	4.000	**1.000**
datatrieve	4.500	6.000	4.500	9.000	7.000	8.000	2.000	3.000	**1.000**
e-learning	3.500	7.000	2.000	5.000	8.000	9.000	**1.000**	6.000	3.500
eclipse34$_d$*ebug*	**1.500**	9.000	**1.500**	3.000	8.000	4.000	7.000	5.000	6.000
eclipse34$_s$*wt*	**1.500**	9.000	**1.500**	6.000	4.000	8.000	7.000	5.000	3.000
forrest-0.7	**1.500**	8.000	4.000	5.000	7.000	9.000	**1.500**	6.000	3.000
forrest-0.8	2.500	9.000	5.000	**1.000**	6.000	7.000	2.500	8.000	4.000
intercafe	**1.500**	9.000	4.000	6.000	8.000	5.000	**1.500**	3.000	7.000
ivy-1.1	**1.500**	6.000	**1.500**	9.000	8.000	3.000	4.000	7.000	5.000
ivy-1.4	3.500	2.000	3.500	9.000	7.000	8.000	6.000	5.000	**1.000**
ivy-2.0	2.500	5.000	2.500	8.000	7.000	9.000	6.000	4.000	**1.000**
jedit-3.2	**1.500**	7.000	**1.500**	9.000	6.000	8.000	5.000	3.000	4.000
jedit-4.0	**1.500**	7.000	**1.500**	8.000	5.000	9.000	4.000	6.000	3.000
jedit-4.1	**1.500**	5.000	**1.500**	8.000	6.000	9.000	4.000	7.000	3.000
jedit-4.2	3.500	2.000	3.500	8.000	7.000	9.000	6.000	**1.000**	5.000
jedit-4.3	3.500	6.000	3.500	9.000	5.000	7.000	8.000	**1.000**	2.000
JM1-v2	**1.500**	6.000	**1.500**	7.000	8.000	9.000	5.000	4.000	3.000
JM1	**1.500**	6.000	**1.500**	8.000	7.000	9.000	5.000	4.000	3.000
kalkulator	**1.500**	3.000	5.000	9.000	7.000	8.000	**1.500**	4.000	6.000

KC1-v2	2.500	4.000	**1.000**	9.000	6.000	8.000	5.000	7.000	2.500
KC1	**1.500**	4.000	**1.500**	8.000	7.000	9.000	6.000	5.000	3.000
KC2-v2	4.500	**1.000**	4.500	8.000	7.000	9.000	2.000	6.000	3.000
KC2	2.500	**1.000**	2.500	8.000	7.000	9.000	5.000	6.000	4.000
KC3-v1	2.500	5.000	**1.000**	8.000	4.000	9.000	6.000	7.000	2.500
KC3-v2	2.500	5.000	2.500	7.000	4.000	9.000	6.000	8.000	**1.000**
KC3	2.500	4.000	**1.000**	8.000	6.000	9.000	7.000	5.000	2.500
KC4	**1.500**	7.000	**1.500**	5.000	8.000	9.000	6.000	4.000	3.000
log4j-1.0	3.500	**1.000**	2.000	9.000	8.000	7.000	6.000	5.000	3.500
log4j-1.1	4.500	**1.000**	4.500	9.000	6.000	8.000	7.000	2.000	3.000
log4j-1.2	3.500	6.000	2.000	7.000	8.000	9.000	5.000	3.500	**1.000**
lucene-2.0	2.500	**1.000**	2.500	9.000	5.000	8.000	7.000	4.000	6.000
lucene-2.2	**1.500**	4.000	**1.500**	9.000	7.000	5.000	6.000	8.000	3.000
lucene-2.4	**1.500**	6.000	**1.500**	8.000	4.000	9.000	5.000	7.000	3.000
MC1-v1	**1.500**	6.000	5.000	8.000	3.000	9.000	7.000	4.000	**1.500**
MC1-v2	**1.500**	5.000	**1.500**	8.000	6.000	9.000	7.000	4.000	3.000
MC1	**1.500**	6.000	5.000	8.000	3.000	9.000	7.000	4.000	**1.500**
MC2-v1	**1.500**	4.000	5.000	7.000	6.000	8.000	3.000	9.000	**1.500**
MC2-v2	**1.500**	4.000	5.000	9.000	6.000	8.000	3.000	7.000	**1.500**
MC2	**1.500**	4.000	3.000	7.000	6.000	9.000	5.000	8.000	**1.500**
mozilla4	**1.500**	9.000	**1.500**	4.000	7.000	8.000	6.000	5.000	3.000
MW1-v1	5.500	3.000	5.500	9.000	2.000	8.000	7.000	4.000	**1.000**
MW1-v2	6.500	3.000	**1.000**	9.000	2.000	8.000	5.000	4.000	6.500
MW1	**1.500**	5.000	3.000	8.000	6.000	9.000	7.000	4.000	**1.500**
nieruchomosci	5.500	2.000	**1.000**	9.000	4.000	7.000	5.500	3.000	8.000
pbeans1	6.500	5.000	2.000	4.000	8.000	9.000	6.500	3.000	**1.000**
pbeans2	**1.500**	4.000	3.000	9.000	5.000	7.000	8.000	6.000	**1.500**
PC1-v1	2.500	6.000	**1.000**	8.000	7.000	9.000	5.000	4.000	2.500
PC1-v2	**1.500**	6.000	**1.500**	8.000	7.000	9.000	5.000	4.000	3.000
PC1	2.500	6.000	2.500	8.000	5.000	9.000	7.000	4.000	**1.000**
PC2-v1	**1.500**	5.000	4.000	9.000	7.000	8.000	6.000	**1.500**	3.000
PC2-v2	**1.500**	6.000	4.000	9.000	7.000	8.000	5.000	**1.500**	3.000
PC2	2.500	**1.000**	6.000	9.000	5.000	8.000	7.000	4.000	2.500
PC3-v1	**1.500**	6.000	**1.500**	8.000	7.000	9.000	4.000	5.000	3.000
PC3-v2	**1.500**	6.000	**1.500**	8.000	7.000	9.000	4.000	5.000	3.000
PC3	**1.500**	6.000	**1.500**	8.000	7.000	9.000	5.000	4.000	3.000
PC4-v1	**1.500**	6.000	**1.500**	8.000	7.000	9.000	5.000	4.000	3.000
PC4-v2	**1.500**	6.000	**1.500**	8.000	7.000	9.000	5.000	4.000	3.000
PC4	**1.500**	6.000	**1.500**	8.000	7.000	9.000	5.000	4.000	3.000
PC5-v1	**1.500**	7.000	**1.500**	8.000	5.000	9.000	6.000	4.000	3.000
PC5-v2	**1.500**	7.000	**1.500**	8.000	6.000	9.000	5.000	4.000	3.000
PC5	**1.500**	6.000	**1.500**	8.000	5.000	9.000	7.000	4.000	3.000
pdftranslator	7.500	6.000	**1.000**	4.000	5.000	9.000	7.500	3.000	2.000
poi-1.5	2.500	7.000	2.500	8.000	5.000	9.000	4.000	6.000	**1.000**
poi-2.0	**1.500**	4.000	**1.500**	8.000	6.000	9.000	7.000	5.000	3.000
poi-2.5	2.500	8.000	2.500	7.000	4.000	9.000	6.000	5.000	**1.000**
poi-3.0	**1.500**	6.000	**1.500**	7.000	5.000	9.000	8.000	4.000	3.000
prop-1	**1.500**	8.000	**1.500**	7.000	4.000	9.000	5.000	6.000	3.000
prop-2	**1.500**	8.000	**1.500**	7.000	4.000	9.000	5.000	6.000	3.000
prop-3	**1.500**	8.000	**1.500**	7.000	4.000	9.000	5.000	6.000	3.000
prop-4	**1.500**	8.000	**1.500**	7.000	4.000	9.000	5.000	6.000	3.000
prop-5	**1.500**	8.000	**1.500**	7.000	4.000	9.000	5.000	6.000	3.000
prop-6	2.500	7.000	2.500	8.000	5.000	9.000	4.000	6.000	**1.000**
redaktor	2.500	7.000	**1.000**	9.000	4.000	8.000	5.000	6.000	2.500
serapion	2.500	5.000	2.500	9.000	8.000	7.000	6.000	4.000	**1.000**
skarbonka	3.500	**1.000**	3.500	8.000	5.000	9.000	6.000	2.000	7.000
sklebagd	2.500	8.000	2.500	9.000	6.000	5.000	7.000	4.000	**1.000**
synapse-1.0	4.500	**1.000**	4.500	8.000	7.000	9.000	3.000	2.000	6.000
synapse-1.1	**1.500**	6.000	**1.500**	8.000	4.000	9.000	5.000	7.000	3.000
synapse-1.2	**1.500**	7.000	**1.500**	8.000	5.000	9.000	6.000	3.000	4.000

systemdata	5.500	2.000	3.000	4.000	8.000	7.000	5.500	9.000	**1.000**
szybkafucha	3.500	2.000	3.500	8.000	5.000	9.000	7.000	6.000	**1.000**
termoproject	6.500	6.500	**1.000**	9.000	2.000	8.000	3.000	5.000	4.000
tomcat	**1.500**	4.000	**1.500**	8.000	7.000	9.000	6.000	5.000	3.000
velocity-1.4	**1.500**	7.000	**1.500**	9.000	5.000	8.000	6.000	4.000	3.000
velocity-1.5	2.500	7.000	2.500	8.000	4.000	9.000	5.000	6.000	**1.000**
velocity-1.6	**1.500**	7.000	**1.500**	8.000	6.000	9.000	5.000	4.000	3.000
workflow	8.500	3.000	**1.000**	4.000	5.000	7.000	6.000	2.000	8.500
wspomaganiepi	**1.500**	7.000	4.000	8.000	5.000	9.000	**1.500**	6.000	3.000
xalan-2.4	**1.500**	5.000	**1.500**	8.000	7.000	9.000	6.000	4.000	3.000
xalan-2.5	**1.500**	8.000	**1.500**	6.000	4.000	9.000	5.000	7.000	3.000
xalan-2.6	**1.500**	7.000	**1.500**	8.000	4.000	9.000	5.000	6.000	3.000
xalan-2.7	**1.500**	4.000	3.000	8.000	7.000	9.000	6.000	**1.500**	5.000
xerces-1.2	**1.500**	8.000	**1.500**	6.000	5.000	9.000	7.000	4.000	3.000
xerces-1.3	**1.500**	6.000	**1.500**	8.000	3.000	9.000	7.000	5.000	4.000
xerces-1.4	**1.500**	8.000	**1.500**	5.000	7.000	9.000	6.000	4.000	3.000
xerces-init	2.500	8.000	2.500	7.000	5.000	9.000	6.000	4.000	**1.000**
zuzel	**1.500**	5.000	3.000	8.000	6.000	9.000	7.000	4.000	**1.500**
μ	2.457	5.313	**2.365**	7.583	5.722	8.409	5.348	4.722	3.083
σ	1.471	2.199	1.239	1.532	1.613	**1.118**	1.621	1.730	1.718

Tabela A.4: *Rankings* gerados segundo o critério *Balance* para os 115 conjuntos de dados com a média (μ) e desvio padrão (σ). Valores médios de FMA-PFS-RF em comparação com os 8 algoritmos de AM.

BASE	FMA-PFS-RF	NB	RF	C4.5	*k*-NN	SVM	MLP	AB	XGB
ant-1.3	**1.500**	**1.500**	8.000	5.000	4.000	9.000	3.000	6.000	7.000
ant-1.4	2.500	2.500	7.000	4.000	5.000	9.000	**1.000**	8.000	6.000
ant-1.5	**1.500**	**1.500**	5.000	7.000	8.000	9.000	4.000	3.000	6.000
ant-1.6	**1.500**	5.000	4.000	6.000	8.000	9.000	7.000	**1.500**	3.000
ant-1.7	**1.500**	3.000	5.000	4.000	8.000	9.000	7.000	**1.500**	6.000
ar1	**1.500**	**1.500**	7.000	4.000	8.000	6.000	3.000	5.000	9.000
ar3	**1.500**	**1.500**	6.000	5.000	4.000	8.000	3.000	7.000	9.000
ar4	2.500	2.500	6.000	4.000	7.000	8.000	5.000	**1.000**	9.000
ar5	7.500	**1.000**	3.000	4.000	2.000	5.000	7.500	6.000	9.000
ar6	2.500	2.500	4.000	7.000	9.000	6.000	**1.000**	8.000	5.000
arc	**1.500**	**1.500**	7.000	5.000	3.000	9.000	6.000	4.000	8.000
berek	5.500	2.000	4.000	7.000	8.000	3.000	5.500	**1.000**	9.000
camel-1.0	**1.500**	**1.500**	6.000	8.000	5.000	7.000	3.000	4.000	9.000
camel-1.2	**1.500**	7.000	**1.500**	3.000	6.000	9.000	5.000	8.000	4.000
camel-1.4	3.500	3.500	7.000	**1.000**	6.000	8.000	5.000	9.000	2.000
camel-1.6	2.500	**1.000**	7.000	2.500	5.000	9.000	6.000	8.000	4.000
ckjm	7.500	7.500	**1.000**	3.000	5.000	4.000	6.000	2.000	9.000
CM1-v2	7.500	**1.000**	6.000	2.000	5.000	9.000	4.000	7.500	3.000
CM1	**1.500**	**1.500**	7.000	3.000	6.000	9.000	5.000	8.000	4.000
datatrieve	**1.500**	**1.500**	6.000	8.000	4.000	7.000	5.000	3.000	9.000
e-learning	3.500	3.500	5.000	2.000	7.000	8.000	**1.000**	6.000	9.000
eclipse34$_d$*ebug*	7.500	7.500	**1.000**	2.000	6.000	3.000	5.000	4.000	9.000
eclipse34$_s$*wt*	**1.500**	9.000	**1.500**	4.000	5.000	8.000	6.000	7.000	3.000
forrest-0.7	5.500	5.500	7.000	3.000	2.000	8.000	**1.000**	4.000	9.000
forrest-0.8	8.500	**1.000**	2.000	7.000	5.000	3.000	8.500	6.000	4.000
intercafe	8.500	8.500	5.000	**1.000**	6.000	4.000	2.000	3.000	7.000
ivy-1.1	2.500	7.000	2.500	6.000	8.000	**1.000**	4.000	5.000	9.000
ivy-1.4	**1.500**	**1.500**	8.000	9.000	4.000	7.000	3.000	6.000	5.000
ivy-2.0	**1.500**	**1.500**	6.000	7.000	4.000	8.000	3.000	9.000	5.000
jedit-3.2	2.500	9.000	**1.000**	7.000	6.000	8.000	4.000	2.500	5.000
jedit-4.0	3.500	8.000	5.000	6.000	7.000	9.000	2.000	3.500	**1.000**

jedit-4.1	**1.500**	8.000	4.000	7.000	5.000	9.000	3.000	**1.500**	6.000
jedit-4.2	**1.500**	**1.500**	6.000	4.000	3.000	9.000	5.000	8.000	7.000
jedit-4.3	**1.500**	**1.500**	7.000	6.000	8.000	4.000	3.000	5.000	9.000
JM1-v2	**1.500**	5.000	4.000	3.000	**1.500**	8.000	7.000	9.000	6.000
JM1	**1.500**	5.000	3.000	4.000	**1.500**	9.000	7.000	8.000	6.000
kalkulator	3.500	3.500	6.000	5.000	8.000	7.000	**1.000**	2.000	9.000
KC1-v2	3.500	9.000	**1.000**	6.000	5.000	2.000	3.500	7.000	8.000
KC1	4.500	**1.000**	2.000	3.000	4.500	8.000	7.000	9.000	6.000
KC2-v2	7.500	7.500	4.000	3.000	**1.000**	9.000	6.000	5.000	2.000
KC2	4.500	8.000	2.000	6.000	**1.000**	9.000	7.000	4.500	3.000
KC3-v1	**1.500**	3.000	8.000	4.000	7.000	9.000	5.000	**1.500**	6.000
KC3-v2	**1.500**	3.000	8.000	5.000	7.000	9.000	6.000	**1.500**	4.000
KC3	**1.500**	**1.500**	8.000	3.000	7.000	9.000	4.000	6.000	5.000
KC4	3.500	9.000	**1.000**	2.000	7.000	5.000	6.000	3.500	8.000
log4j-1.0	**1.500**	**1.500**	3.000	5.000	8.000	7.000	6.000	4.000	9.000
log4j-1.1	2.500	**1.000**	5.000	8.000	4.000	6.000	7.000	2.500	9.000
log4j-1.2	**1.500**	**1.500**	6.000	4.000	9.000	8.000	5.000	7.000	3.000
lucene-2.0	2.500	9.000	2.500	7.000	6.000	**1.000**	5.000	4.000	8.000
lucene-2.2	**1.500**	8.000	**1.500**	7.000	3.000	5.000	4.000	9.000	6.000
lucene-2.4	2.500	9.000	2.500	6.000	**1.000**	5.000	4.000	8.000	7.000
MC1-v1	**1.500**	**1.500**	3.000	7.000	4.000	8.000	5.000	9.000	6.000
MC1-v2	**1.500**	**1.500**	3.000	4.000	7.000	9.000	5.000	8.000	6.000
MC1	**1.500**	**1.500**	3.000	6.000	4.000	9.000	5.000	8.000	7.000
MC2-v1	**1.500**	6.000	5.000	4.000	3.000	9.000	**1.500**	8.000	7.000
MC2-v2	**1.500**	8.000	5.000	6.000	3.000	9.000	**1.500**	7.000	4.000
MC2	**1.500**	4.000	7.000	3.000	5.000	9.000	**1.500**	8.000	6.000
mozilla4	**1.500**	9.000	**1.500**	3.000	5.000	8.000	6.000	7.000	4.000
MW1-v1	**1.500**	**1.500**	6.000	5.000	8.000	9.000	3.000	4.000	7.000
MW1-v2	**1.500**	**1.500**	5.000	7.000	8.000	9.000	3.000	4.000	6.000
MW1	**1.500**	**1.500**	7.000	6.000	8.000	9.000	4.000	3.000	5.000
nieruchomosci	**1.500**	**1.500**	3.000	7.000	8.000	6.000	5.000	4.000	9.000
pbeans1	5.500	4.000	**1.000**	3.000	7.000	9.000	5.500	2.000	8.000
pbeans2	4.500	4.500	7.000	2.000	9.000	8.000	6.000	3.000	**1.000**
PC1-v1	**1.500**	**1.500**	6.000	5.000	3.000	9.000	4.000	8.000	7.000
PC1-v2	**1.500**	**1.500**	6.000	5.000	3.000	9.000	4.000	8.000	7.000
PC1	2.500	2.500	4.000	5.000	**1.000**	8.000	7.000	9.000	6.000
PC2-v1	**1.500**	**1.500**	7.000	5.000	8.000	6.000	4.000	3.000	9.000
PC2-v2	**1.500**	**1.500**	5.000	8.000	7.000	4.000	3.000	6.000	9.000
PC2	**1.500**	**1.500**	6.000	8.000	7.000	4.000	3.000	5.000	9.000
PC3-v1	2.500	**1.000**	6.000	2.500	5.000	9.000	7.000	8.000	4.000
PC3-v2	2.500	2.500	7.000	**1.000**	4.000	8.000	6.000	9.000	5.000
PC3	**1.500**	**1.500**	6.000	4.000	3.000	9.000	7.000	8.000	5.000
PC4-v1	3.500	6.000	7.000	**1.000**	5.000	9.000	3.500	8.000	2.000
PC4-v2	2.500	7.000	6.000	2.500	5.000	9.000	4.000	8.000	**1.000**
PC4	2.500	7.000	6.000	2.500	5.000	9.000	**1.000**	8.000	4.000
PC5-v1	3.500	**1.000**	5.000	2.000	3.500	9.000	7.000	8.000	6.000
PC5-v2	2.500	8.000	4.000	**1.000**	2.500	9.000	6.000	7.000	5.000
PC5	3.500	3.500	5.000	2.000	**1.000**	9.000	7.000	8.000	6.000
pdftranslator	5.500	4.000	2.000	3.000	8.000	7.000	5.500	**1.000**	9.000
poi-1.5	**1.500**	9.000	**1.500**	3.000	4.000	8.000	5.000	7.000	6.000
poi-2.0	5.500	5.500	2.000	4.000	3.000	9.000	7.000	**1.000**	8.000
poi-2.5	**1.500**	9.000	**1.500**	5.000	3.000	8.000	6.000	7.000	4.000
poi-3.0	**1.500**	9.000	**1.500**	5.000	7.000	8.000	6.000	4.000	3.000
prop-1	2.500	7.000	2.500	4.000	**1.000**	9.000	5.000	8.000	6.000
prop-2	2.500	7.000	**1.000**	4.000	2.500	9.000	6.000	8.000	5.000
prop-3	**1.500**	7.000	**1.500**	4.000	3.000	9.000	6.000	8.000	5.000
prop-4	**1.500**	7.000	4.000	3.000	**1.500**	9.000	5.000	8.000	6.000
prop-5	**1.500**	8.000	**1.500**	3.000	4.000	9.000	6.000	7.000	5.000
prop-6	**1.500**	**1.500**	4.000	7.000	5.000	9.000	3.000	8.000	6.000
redaktor	**1.500**	**1.500**	4.000	8.000	7.000	5.000	3.000	6.000	9.000

serapion	**1.500**	**1.500**	7.000	5.000	8.000	4.000	6.000	3.000	9.000
skarbonka	**1.500**	**1.500**	6.000	4.000	7.000	8.000	5.000	3.000	9.000
sklebagd	2.500	**1.000**	2.500	8.000	7.000	4.000	6.000	5.000	9.000
synapse-1.0	**1.500**	**1.500**	5.000	6.000	4.000	8.000	3.000	7.000	9.000
synapse-1.1	7.500	**1.000**	2.000	4.000	5.000	9.000	3.000	7.500	6.000
synapse-1.2	3.500	8.000	3.500	2.000	5.000	9.000	6.000	**1.000**	7.000
systemdata	**1.500**	**1.500**	8.000	5.000	9.000	4.000	3.000	7.000	6.000
szybkafucha	3.500	2.000	3.500	8.000	5.000	7.000	6.000	**1.000**	9.000
termoproject	3.500	3.500	6.000	8.000	5.000	7.000	**1.000**	9.000	2.000
tomcat	**1.500**	**1.500**	7.000	3.000	6.000	9.000	4.000	8.000	5.000
velocity-1.4	3.500	3.500	5.000	**1.000**	7.000	8.000	2.000	6.000	9.000
velocity-1.5	**1.500**	8.000	**1.500**	4.000	5.000	9.000	3.000	6.000	7.000
velocity-1.6	2.500	8.000	2.500	5.000	7.000	9.000	4.000	6.000	**1.000**
workflow	4.500	8.000	**1.000**	6.000	2.000	7.000	4.500	3.000	9.000
wspomaganiepi	3.500	3.500	2.000	6.000	7.000	9.000	**1.000**	5.000	8.000
xalan-2.4	**1.500**	**1.500**	7.000	5.000	6.000	9.000	3.000	8.000	4.000
xalan-2.5	**1.500**	9.000	**1.500**	4.000	5.000	8.000	6.000	7.000	3.000
xalan-2.6	**1.500**	9.000	**1.500**	4.000	5.000	8.000	6.000	7.000	3.000
xalan-2.7	**1.500**	**1.500**	4.000	3.000	6.000	8.000	7.000	5.000	9.000
xerces-1.2	4.500	4.500	2.000	**1.000**	7.000	9.000	6.000	8.000	3.000
xerces-1.3	6.500	6.500	4.000	3.000	2.000	9.000	**1.000**	8.000	5.000
xerces-1.4	3.500	9.000	2.000	**1.000**	6.000	7.000	5.000	3.500	8.000
xerces-init	3.500	9.000	3.500	2.000	6.000	8.000	5.000	**1.000**	7.000
zuzel	6.500	**1.000**	4.000	5.000	3.000	8.000	6.500	2.000	9.000
μ	**2.770**	4.200	4.326	4.504	5.213	7.583	4.565	5.648	6.191
σ	1.834	2.988	2.149	2.004	2.156	1.969	**1.823**	2.502	2.325

Tabela A.5: *Rankings* gerados segundo o critério AUC para os 115 conjuntos de dados com a média (μ) e desvio padrão (σ). Valores médios de FMA-PFS-ENS-8 em comparação com os 8 algoritmos de AM.

BASE	FMA-PFS-ENS-8	NB	RF	C4.5	*k*-NN	SVM	MLP	AB	XGB
ant-1.3	**1.500**	**1.500**	4.000	8.000	7.000	9.000	6.000	3.000	5.000
ant-1.4	**1.500**	6.000	**1.500**	7.000	8.000	9.000	3.000	5.000	4.000
ant-1.5	**1.500**	4.000	**1.500**	8.000	6.000	9.000	7.000	3.000	5.000
ant-1.6	**1.500**	4.000	**1.500**	8.000	7.000	9.000	6.000	5.000	3.000
ant-1.7	**1.500**	4.000	**1.500**	8.000	6.000	9.000	7.000	5.000	3.000
ar1	3.500	6.000	2.000	8.000	**1.000**	9.000	5.000	7.000	3.500
ar3	5.500	2.000	3.000	7.000	**1.000**	9.000	5.500	4.000	8.000
ar4	4.500	4.500	3.000	9.000	6.000	8.000	7.000	**1.000**	2.000
ar5	4.500	**1.000**	2.000	8.000	3.000	7.000	4.500	6.000	9.000
ar6	5.500	5.500	4.000	9.000	3.000	8.000	**1.000**	7.000	2.000
arc	2.500	6.000	2.500	9.000	5.000	8.000	7.000	**1.000**	4.000
berek	**1.500**	3.000	4.000	8.000	6.000	9.000	**1.500**	5.000	7.000
camel-1.0	2.500	**1.000**	4.000	9.000	7.000	8.000	6.000	5.000	2.500
camel-1.2	**1.500**	7.000	**1.500**	6.000	5.000	9.000	4.000	8.000	3.000
camel-1.4	2.500	4.000	2.500	7.000	8.000	9.000	6.000	5.000	**1.000**
camel-1.6	**1.500**	7.000	**1.500**	8.000	6.000	9.000	4.000	5.000	3.000
ckjm	3.500	9.000	**1.000**	2.000	7.000	5.000	8.000	6.000	3.500
CM1-v2	**1.500**	5.000	3.000	8.000	7.000	9.000	6.000	4.000	**1.500**
CM1	**1.500**	5.000	3.000	8.000	6.000	9.000	7.000	4.000	**1.500**
datatrieve	**1.500**	6.000	5.000	9.000	7.000	8.000	3.000	4.000	**1.500**
e-learning	3.500	7.000	2.000	5.000	8.000	9.000	**1.000**	6.000	3.500
eclipse34$_d$*ebug*	8.500	8.500	**1.000**	2.000	7.000	3.000	6.000	4.000	5.000
eclipse34$_s$*wt*	**1.500**	9.000	**1.500**	6.000	4.000	8.000	7.000	5.000	3.000
forrest-0.7	2.500	8.000	4.000	5.000	7.000	9.000	**1.000**	6.000	2.500
forrest-0.8	2.500	9.000	5.000	**1.000**	6.000	7.000	2.500	8.000	4.000

intercafe	6.500	9.000	3.000	5.000	8.000	4.000	**1.000**	2.000	6.500
ivy-1.1	**1.500**	6.000	**1.500**	9.000	8.000	3.000	4.000	7.000	5.000
ivy-1.4	3.500	2.000	3.500	9.000	7.000	8.000	6.000	5.000	**1.000**
ivy-2.0	2.500	5.000	2.500	8.000	7.000	9.000	6.000	4.000	**1.000**
jedit-3.2	**1.500**	7.000	**1.500**	9.000	6.000	8.000	5.000	3.000	4.000
jedit-4.0	**1.500**	7.000	**1.500**	8.000	5.000	9.000	4.000	6.000	3.000
jedit-4.1	**1.500**	5.000	**1.500**	8.000	6.000	9.000	4.000	7.000	3.000
jedit-4.2	3.500	2.000	3.500	8.000	7.000	9.000	6.000	**1.000**	5.000
jedit-4.3	2.500	6.000	4.000	9.000	5.000	7.000	8.000	**1.000**	2.500
JM1-v2	**1.500**	6.000	**1.500**	7.000	8.000	9.000	5.000	4.000	3.000
JM1	**1.500**	6.000	**1.500**	8.000	7.000	9.000	5.000	4.000	3.000
kalkulator	**1.500**	3.000	5.000	9.000	7.000	8.000	**1.500**	4.000	6.000
KC1-v2	2.500	4.000	**1.000**	9.000	6.000	8.000	5.000	7.000	2.500
KC1	**1.500**	4.000	**1.500**	8.000	7.000	9.000	6.000	5.000	3.000
KC2-v2	4.500	**1.000**	4.500	8.000	7.000	9.000	2.000	6.000	3.000
KC2	2.500	**1.000**	2.500	8.000	7.000	9.000	5.000	6.000	4.000
KC3-v1	2.500	5.000	**1.000**	8.000	4.000	9.000	6.000	7.000	2.500
KC3-v2	2.500	5.000	2.500	7.000	4.000	9.000	6.000	8.000	**1.000**
KC3	2.500	4.000	**1.000**	8.000	6.000	9.000	7.000	5.000	2.500
KC4	**1.500**	7.000	**1.500**	5.000	8.000	9.000	6.000	4.000	3.000
log4j-1.0	3.500	**1.000**	2.000	9.000	8.000	7.000	6.000	5.000	3.500
log4j-1.1	4.500	**1.000**	4.500	9.000	6.000	8.000	7.000	2.000	3.000
log4j-1.2	**1.500**	6.000	3.000	7.000	8.000	9.000	5.000	4.000	**1.500**
lucene-2.0	2.500	**1.000**	2.500	9.000	5.000	8.000	7.000	4.000	6.000
lucene-2.2	**1.500**	4.000	**1.500**	9.000	7.000	5.000	6.000	8.000	3.000
lucene-2.4	**1.500**	6.000	**1.500**	8.000	4.000	9.000	5.000	7.000	3.000
MC1-v1	**1.500**	6.000	5.000	8.000	3.000	9.000	7.000	4.000	**1.500**
MC1-v2	**1.500**	5.000	**1.500**	8.000	6.000	9.000	7.000	4.000	3.000
MC1	4.500	6.000	4.500	8.000	2.000	9.000	7.000	3.000	**1.000**
MC2-v1	**1.500**	4.000	5.000	7.000	6.000	8.000	3.000	9.000	**1.500**
MC2-v2	**1.500**	4.000	5.000	9.000	6.000	8.000	3.000	7.000	**1.500**
MC2	**1.500**	4.000	3.000	7.000	6.000	9.000	5.000	8.000	**1.500**
mozilla4	**1.500**	9.000	**1.500**	4.000	7.000	8.000	6.000	5.000	3.000
MW1-v1	5.500	3.000	5.500	9.000	2.000	8.000	7.000	4.000	**1.000**
MW1-v2	6.500	3.000	**1.000**	9.000	2.000	8.000	5.000	4.000	6.500
MW1	**1.500**	5.000	3.000	8.000	6.000	9.000	7.000	4.000	**1.500**
nieruchomosci	7.500	2.000	**1.000**	9.000	4.000	6.000	5.000	3.000	7.500
pbeans1	6.500	5.000	2.000	4.000	8.000	9.000	6.500	3.000	**1.000**
pbeans2	3.500	3.500	2.000	9.000	5.000	7.000	8.000	6.000	**1.000**
PC1-v1	2.500	6.000	**1.000**	8.000	7.000	9.000	5.000	4.000	2.500
PC1-v2	2.500	6.000	**1.000**	8.000	7.000	9.000	5.000	4.000	2.500
PC1	2.500	6.000	2.500	8.000	5.000	9.000	7.000	4.000	**1.000**
PC2-v1	**1.500**	5.000	4.000	9.000	7.000	8.000	6.000	**1.500**	3.000
PC2-v2	**1.500**	6.000	4.000	9.000	7.000	8.000	5.000	**1.500**	3.000
PC2	2.500	**1.000**	6.000	9.000	5.000	8.000	7.000	4.000	2.500
PC3-v1	**1.500**	6.000	**1.500**	8.000	7.000	9.000	4.000	5.000	3.000
PC3-v2	**1.500**	6.000	**1.500**	8.000	7.000	9.000	4.000	5.000	3.000
PC3	**1.500**	6.000	**1.500**	8.000	7.000	9.000	5.000	4.000	3.000
PC4-v1	**1.500**	6.000	**1.500**	8.000	7.000	9.000	5.000	4.000	3.000
PC4-v2	**1.500**	6.000	**1.500**	8.000	7.000	9.000	5.000	4.000	3.000
PC4	**1.500**	6.000	**1.500**	8.000	7.000	9.000	5.000	4.000	3.000
PC5-v1	**1.500**	7.000	**1.500**	8.000	5.000	9.000	6.000	4.000	3.000
PC5-v2	**1.500**	7.000	**1.500**	8.000	6.000	9.000	5.000	4.000	3.000
PC5	**1.500**	6.000	**1.500**	8.000	5.000	9.000	7.000	4.000	3.000
pdftranslator	7.500	6.000	**1.000**	4.000	5.000	9.000	7.500	3.000	2.000
poi-1.5	2.500	7.000	2.500	8.000	5.000	9.000	4.000	6.000	**1.000**
poi-2.0	**1.500**	4.000	**1.500**	8.000	6.000	9.000	7.000	5.000	3.000
poi-2.5	2.500	8.000	2.500	7.000	4.000	9.000	6.000	5.000	**1.000**
poi-3.0	**1.500**	6.000	**1.500**	7.000	5.000	9.000	8.000	4.000	3.000
prop-1	**1.500**	8.000	**1.500**	7.000	4.000	9.000	5.000	6.000	3.000

prop-2	**1.500**	8.000	**1.500**	7.000	4.000	9.000	5.000	6.000	3.000
prop-3	**1.500**	8.000	**1.500**	7.000	4.000	9.000	5.000	6.000	3.000
prop-4	**1.500**	8.000	**1.500**	7.000	4.000	9.000	5.000	6.000	3.000
prop-5	**1.500**	8.000	**1.500**	7.000	4.000	9.000	5.000	6.000	3.000
prop-6	2.500	7.000	2.500	8.000	5.000	9.000	4.000	6.000	**1.000**
redaktor	2.500	7.000	**1.000**	9.000	4.000	8.000	5.000	6.000	2.500
serapion	5.500	4.000	2.000	9.000	8.000	7.000	5.500	3.000	**1.000**
skarbonka	6.500	**1.000**	3.000	8.000	4.000	9.000	5.000	2.000	6.500
sklebagd	2.500	8.000	2.500	9.000	6.000	5.000	7.000	4.000	**1.000**
synapse-1.0	5.500	**1.000**	4.000	8.000	7.000	9.000	3.000	2.000	5.500
synapse-1.1	**1.500**	6.000	**1.500**	8.000	4.000	9.000	5.000	7.000	3.000
synapse-1.2	**1.500**	7.000	**1.500**	8.000	5.000	9.000	6.000	3.000	4.000
systemdata	**1.500**	3.000	4.000	5.000	8.000	7.000	6.000	9.000	**1.500**
szybkafucha	3.500	2.000	3.500	8.000	5.000	9.000	7.000	6.000	**1.000**
termoproject	4.500	7.000	**1.000**	9.000	2.000	8.000	3.000	6.000	4.500
tomcat	**1.500**	4.000	**1.500**	8.000	7.000	9.000	6.000	5.000	3.000
velocity-1.4	**1.500**	7.000	**1.500**	9.000	5.000	8.000	6.000	4.000	3.000
velocity-1.5	2.500	7.000	2.500	8.000	4.000	9.000	5.000	6.000	**1.000**
velocity-1.6	**1.500**	7.000	**1.500**	8.000	6.000	9.000	5.000	4.000	3.000
workflow	8.500	3.000	**1.000**	4.000	5.000	7.000	6.000	2.000	8.500
wspomaganiepi	**1.500**	7.000	4.000	8.000	5.000	9.000	**1.500**	6.000	3.000
xalan-2.4	**1.500**	5.000	**1.500**	8.000	7.000	9.000	6.000	4.000	3.000
xalan-2.5	**1.500**	8.000	**1.500**	6.000	4.000	9.000	5.000	7.000	3.000
xalan-2.6	**1.500**	7.000	**1.500**	8.000	4.000	9.000	5.000	6.000	3.000
xalan-2.7	**1.500**	4.000	3.000	8.000	7.000	9.000	6.000	**1.500**	5.000
xerces-1.2	**1.500**	8.000	**1.500**	6.000	5.000	9.000	7.000	4.000	3.000
xerces-1.3	**1.500**	6.000	**1.500**	8.000	3.000	9.000	7.000	5.000	4.000
xerces-1.4	**1.500**	8.000	**1.500**	5.000	7.000	9.000	6.000	4.000	3.000
xerces-init	2.500	8.000	2.500	7.000	5.000	9.000	6.000	4.000	**1.000**
zuzel	**1.500**	5.000	3.000	8.000	6.000	9.000	7.000	4.000	**1.500**
μ	2.578	5.309	**2.357**	7.574	5.687	8.383	5.326	4.717	3.070
σ	1.684	2.182	1.237	1.555	1.638	**1.198**	1.621	1.738	1.669

Tabela A.6: *Rankings* gerados segundo o critério *Balance* para os 115 conjuntos de dados com a média (μ) e desvio padrão (σ). Valores médios de FMA-PFS-ENS-8 em comparação com os 8 algoritmos de AM.

BASE	FMA-PFS-ENS-8	NB	RF	C4.5	*k*-NN	SVM	MLP	AB	XGB
ant-1.3	**1.500**	**1.500**	8.000	5.000	4.000	9.000	3.000	6.000	7.000
ant-1.4	2.500	2.500	7.000	4.000	5.000	9.000	**1.000**	8.000	6.000
ant-1.5	**1.500**	**1.500**	5.000	7.000	8.000	9.000	4.000	3.000	6.000
ant-1.6	4.500	4.500	3.000	6.000	8.000	9.000	7.000	**1.000**	2.000
ant-1.7	2.500	2.500	5.000	4.000	8.000	9.000	7.000	**1.000**	6.000
ar1	**1.500**	**1.500**	7.000	4.000	8.000	6.000	3.000	5.000	9.000
ar3	**1.500**	**1.500**	6.000	5.000	4.000	8.000	3.000	7.000	9.000
ar4	2.500	2.500	6.000	4.000	7.000	8.000	5.000	**1.000**	9.000
ar5	**1.500**	**1.500**	4.000	5.000	3.000	6.000	8.000	7.000	9.000
ar6	2.500	2.500	4.000	7.000	9.000	6.000	**1.000**	8.000	5.000
arc	**1.500**	**1.500**	7.000	5.000	3.000	9.000	6.000	4.000	8.000
berek	5.500	2.000	4.000	7.000	8.000	3.000	5.500	**1.000**	9.000
camel-1.0	**1.500**	**1.500**	6.000	8.000	5.000	7.000	3.000	4.000	9.000
camel-1.2	**1.500**	7.000	**1.500**	3.000	6.000	9.000	5.000	8.000	4.000
camel-1.4	3.500	3.500	7.000	**1.000**	6.000	8.000	5.000	9.000	2.000
camel-1.6	2.500	**1.000**	7.000	2.500	5.000	9.000	6.000	8.000	4.000
ckjm	7.500	7.500	**1.000**	3.000	5.000	4.000	6.000	2.000	9.000
CM1-v2	**1.500**	**1.500**	7.000	3.000	6.000	9.000	5.000	8.000	4.000
CM1	**1.500**	**1.500**	7.000	3.000	6.000	9.000	5.000	8.000	4.000

datatrieve	**1.500**	**1.500**	6.000	8.000	4.000	7.000	5.000	3.000	9.000
e-learning	3.500	3.500	5.000	2.000	7.000	8.000	**1.000**	6.000	9.000
eclipse34$_d$*ebug*	7.500	7.500	**1.000**	2.000	6.000	3.000	5.000	4.000	9.000
eclipse34$_s$*wt*	**1.500**	9.000	**1.500**	4.000	5.000	8.000	6.000	7.000	3.000
forrest-0.7	5.500	5.500	7.000	3.000	2.000	8.000	**1.000**	4.000	9.000
forrest-0.8	**1.500**	**1.500**	3.000	8.000	6.000	4.000	9.000	7.000	5.000
intercafe	8.500	8.500	5.000	**1.000**	6.000	4.000	2.000	3.000	7.000
ivy-1.1	2.500	7.000	2.500	6.000	8.000	**1.000**	4.000	5.000	9.000
ivy-1.4	**1.500**	**1.500**	8.000	9.000	4.000	7.000	3.000	6.000	5.000
ivy-2.0	**1.500**	**1.500**	6.000	7.000	4.000	8.000	3.000	9.000	5.000
jedit-3.2	**1.500**	9.000	**1.500**	7.000	6.000	8.000	4.000	3.000	5.000
jedit-4.0	7.500	7.500	4.000	5.000	6.000	9.000	2.000	3.000	**1.000**
jedit-4.1	7.500	7.500	3.000	6.000	4.000	9.000	2.000	**1.000**	5.000
jedit-4.2	**1.500**	**1.500**	6.000	4.000	3.000	9.000	5.000	8.000	7.000
jedit-4.3	**1.500**	**1.500**	7.000	6.000	8.000	4.000	3.000	5.000	9.000
JM1-v2	**1.500**	5.000	4.000	3.000	**1.500**	8.000	7.000	9.000	6.000
JM1	**1.500**	5.000	3.000	4.000	**1.500**	9.000	7.000	8.000	6.000
kalkulator	3.500	3.500	6.000	5.000	8.000	7.000	**1.000**	2.000	9.000
KC1-v2	8.500	8.500	**1.000**	5.000	4.000	2.000	3.000	6.000	7.000
KC1	4.500	**1.000**	2.000	3.000	4.500	8.000	7.000	9.000	6.000
KC2-v2	7.500	7.500	4.000	3.000	**1.000**	9.000	6.000	5.000	2.000
KC2	7.500	7.500	2.000	5.000	**1.000**	9.000	6.000	4.000	3.000
KC3-v1	2.500	2.500	8.000	4.000	7.000	9.000	5.000	**1.000**	6.000
KC3-v2	**1.500**	3.000	8.000	5.000	7.000	9.000	6.000	**1.500**	4.000
KC3	**1.500**	**1.500**	8.000	3.000	7.000	9.000	4.000	6.000	5.000
KC4	3.500	9.000	**1.000**	2.000	7.000	5.000	6.000	3.500	8.000
log4j-1.0	**1.500**	**1.500**	3.000	5.000	8.000	7.000	6.000	4.000	9.000
log4j-1.1	**1.500**	**1.500**	5.000	8.000	4.000	6.000	7.000	3.000	9.000
log4j-1.2	**1.500**	**1.500**	6.000	4.000	9.000	8.000	5.000	7.000	3.000
lucene-2.0	2.500	9.000	2.500	7.000	6.000	**1.000**	5.000	4.000	8.000
lucene-2.2	**1.500**	8.000	**1.500**	7.000	3.000	5.000	4.000	9.000	6.000
lucene-2.4	2.500	9.000	2.500	6.000	**1.000**	5.000	4.000	8.000	7.000
MC1-v1	**1.500**	**1.500**	3.000	7.000	4.000	8.000	5.000	9.000	6.000
MC1-v2	**1.500**	**1.500**	3.000	4.000	7.000	9.000	5.000	8.000	6.000
MC1	**1.500**	**1.500**	3.000	6.000	4.000	9.000	5.000	8.000	7.000
MC2-v1	**1.500**	6.000	5.000	4.000	3.000	9.000	**1.500**	8.000	7.000
MC2-v2	**1.500**	8.000	5.000	6.000	3.000	9.000	**1.500**	7.000	4.000
MC2	**1.500**	4.000	7.000	3.000	5.000	9.000	**1.500**	8.000	6.000
mozilla4	**1.500**	9.000	**1.500**	3.000	5.000	8.000	6.000	7.000	4.000
MW1-v1	**1.500**	**1.500**	6.000	5.000	8.000	9.000	3.000	4.000	7.000
MW1-v2	**1.500**	**1.500**	5.000	7.000	8.000	9.000	3.000	4.000	6.000
MW1	**1.500**	**1.500**	7.000	6.000	8.000	9.000	4.000	3.000	5.000
nieruchomosci	**1.500**	**1.500**	3.000	7.000	8.000	6.000	5.000	4.000	9.000
pbeans1	5.500	4.000	**1.000**	3.000	7.000	9.000	5.500	2.000	8.000
pbeans2	4.500	4.500	7.000	2.000	9.000	8.000	6.000	3.000	**1.000**
PC1-v1	**1.500**	**1.500**	6.000	5.000	3.000	9.000	4.000	8.000	7.000
PC1-v2	**1.500**	**1.500**	6.000	5.000	3.000	9.000	4.000	8.000	7.000
PC1	2.500	2.500	4.000	5.000	**1.000**	8.000	7.000	9.000	6.000
PC2-v1	**1.500**	**1.500**	7.000	5.000	8.000	6.000	4.000	3.000	9.000
PC2-v2	**1.500**	**1.500**	5.000	8.000	7.000	4.000	3.000	6.000	9.000
PC2	**1.500**	**1.500**	6.000	8.000	7.000	4.000	3.000	5.000	9.000
PC3-v1	2.500	**1.000**	6.000	2.500	5.000	9.000	7.000	8.000	4.000
PC3-v2	2.500	2.500	7.000	**1.000**	4.000	8.000	6.000	9.000	5.000
PC3	**1.500**	**1.500**	6.000	4.000	3.000	9.000	7.000	8.000	5.000
PC4-v1	5.500	5.500	7.000	**1.000**	4.000	9.000	3.000	8.000	2.000
PC4-v2	6.500	6.500	5.000	2.000	4.000	9.000	3.000	8.000	**1.000**
PC4	2.500	7.000	6.000	2.500	5.000	9.000	**1.000**	8.000	4.000
PC5-v1	3.500	**1.000**	5.000	2.000	3.500	9.000	7.000	8.000	6.000
PC5-v2	2.500	8.000	4.000	**1.000**	2.500	9.000	6.000	7.000	5.000
PC5	3.500	3.500	5.000	2.000	**1.000**	9.000	7.000	8.000	6.000

pdftranslator	5.500	4.000	2.000	3.000	8.000	7.000	5.500	**1.000**	9.000
poi-1.5	**1.500**	9.000	**1.500**	3.000	4.000	8.000	5.000	7.000	6.000
poi-2.0	5.500	5.500	2.000	4.000	3.000	9.000	7.000	**1.000**	8.000
poi-2.5	**1.500**	9.000	**1.500**	5.000	3.000	8.000	6.000	7.000	4.000
poi-3.0	**1.500**	9.000	**1.500**	5.000	7.000	8.000	6.000	4.000	3.000
prop-1	2.500	7.000	2.500	4.000	**1.000**	9.000	5.000	8.000	6.000
prop-2	2.500	7.000	**1.000**	4.000	2.500	9.000	6.000	8.000	5.000
prop-3	**1.500**	7.000	**1.500**	4.000	3.000	9.000	6.000	8.000	5.000
prop-4	3.500	7.000	3.500	2.000	**1.000**	9.000	5.000	8.000	6.000
prop-5	**1.500**	8.000	**1.500**	3.000	4.000	9.000	6.000	7.000	5.000
prop-6	**1.500**	**1.500**	4.000	7.000	5.000	9.000	3.000	8.000	6.000
redaktor	**1.500**	**1.500**	4.000	8.000	7.000	5.000	3.000	6.000	9.000
serapion	**1.500**	**1.500**	7.000	5.000	8.000	4.000	6.000	3.000	9.000
skarbonka	**1.500**	**1.500**	6.000	4.000	7.000	8.000	5.000	3.000	9.000
sklebagd	2.500	**1.000**	2.500	8.000	7.000	4.000	6.000	5.000	9.000
synapse-1.0	**1.500**	**1.500**	5.000	6.000	4.000	8.000	3.000	7.000	9.000
synapse-1.1	7.500	**1.000**	2.000	4.000	5.000	9.000	3.000	7.500	6.000
synapse-1.2	3.500	8.000	3.500	2.000	5.000	9.000	6.000	**1.000**	7.000
systemdata	**1.500**	**1.500**	8.000	5.000	9.000	4.000	3.000	7.000	6.000
szybkafucha	3.500	2.000	3.500	8.000	5.000	7.000	6.000	**1.000**	9.000
termoproject	3.500	3.500	6.000	8.000	5.000	7.000	**1.000**	9.000	2.000
tomcat	**1.500**	**1.500**	7.000	3.000	6.000	9.000	4.000	8.000	5.000
velocity-1.4	3.500	3.500	5.000	**1.000**	7.000	8.000	2.000	6.000	9.000
velocity-1.5	**1.500**	8.000	**1.500**	4.000	5.000	9.000	3.000	6.000	7.000
velocity-1.6	2.500	8.000	2.500	5.000	7.000	9.000	4.000	6.000	**1.000**
workflow	4.500	8.000	**1.000**	6.000	2.000	7.000	4.500	3.000	9.000
wspomaganiepi	3.500	3.500	2.000	6.000	7.000	9.000	**1.000**	5.000	8.000
xalan-2.4	**1.500**	**1.500**	7.000	5.000	6.000	9.000	3.000	8.000	4.000
xalan-2.5	**1.500**	9.000	**1.500**	4.000	5.000	8.000	6.000	7.000	3.000
xalan-2.6	**1.500**	9.000	**1.500**	4.000	5.000	8.000	6.000	7.000	3.000
xalan-2.7	**1.500**	**1.500**	4.000	3.000	6.000	8.000	7.000	5.000	9.000
xerces-1.2	4.500	4.500	2.000	**1.000**	7.000	9.000	6.000	8.000	3.000
xerces-1.3	6.500	6.500	4.000	3.000	2.000	9.000	**1.000**	8.000	5.000
xerces-1.4	8.500	8.500	2.000	**1.000**	5.000	6.000	4.000	3.000	7.000
xerces-init	3.500	9.000	3.500	2.000	6.000	8.000	5.000	**1.000**	7.000
zuzel	4.500	**1.000**	4.500	6.000	3.000	8.000	7.000	2.000	9.000
μ	**2.883**	4.174	4.322	4.491	5.183	7.591	4.543	5.639	6.174
σ	1.976	2.939	2.143	2.004	2.160	1.947	**1.857**	2.540	2.315

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Graduação
Av. Ipiranga, 6681 - Prédio 1 - 3º. andar
Porto Alegre - RS - Brasil
Fone: (51) 3320-3500 - Fax: (51) 3339-1564
E-mail: prograd@pucrs.br
Site: www.pucrs.br