UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MAIRIELI SANTOS WESSEL

OTIMIZAÇÃO DE RECOMENDAÇÕES DE MUDANÇAS DE SOFTWARE POR MEIO DE ALGORITMO GENÉTICO

MONOGRAFIA

CAMPO MOURÃO
2017

MAIRIELI SANTOS WESSEL

OTIMIZAÇÃO DE RECOMENDAÇÕES DE MUDANÇAS DE SOFTWARE POR MEIO DE ALGORITMO GENÉTICO

Trabalho de Conclusão de Curso de graduação apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso de Bacharelado em Ciência da Computação do Departamento Acadêmico de Computação da Universidade Tecnológica Federal do Paraná, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Igor Scaliante Wiese

CAMPO MOURÃO

2017

ATA DE DEFESA DO TRABALHO DE CONCLUSÃO DE CURSO

Às **14:00** do dia **28 de novembro de 2017** foi realizada na sala **E007** da UTFPR-CM a sessão pública da defesa do Trabalho de Conclusão do Curso de Bacharelado em Ciência da Computação do(a) acadêmico(a) **Mairieli Santos Wessel** com o título **Otimização de Recomendações de Mudanças de Software por meio de Algoritmo Genético**. Estavam presentes, além do(a) acadêmico(a), os membros da banca examinadora composta por: **Prof. Dr. Igor Scaliante Wiese** (orientador), **Prof. Dr. Diego Bertolini Gonçalves** e **Prof. Dr. Marco Aurélio Graciotto Silva**. Inicialmente, o(a) acadêmico(a) fez a apresentação do seu trabalho, sendo, em seguida, arguido(a) pela banca examinadora. Após as arguições, sem a presença do(a) acadêmico(a), a banca examinadora o(a) considerou _____ na disciplina de Trabalho de Conclusão de Curso **2** e atribuiu, em consenso, a nota _____ (_____). Este resultado foi comunicado ao(à) acadêmico(a) e aos presentes na sessão pública. A banca examinadora também comunicou ao acadêmico(a) que este resultado fica condicionado à entrega da versão final dentro dos padrões e da documentação exigida pela UTFPR ao professor Responsável do TCC no prazo de **onze dias**. Em seguida foi encerrada a sessão e, para constar, foi lavrada a presente Ata que segue assinada pelos membros da banca examinadora, após lida e considerada conforme.

Observações: _____

_____

_____

_____

_____

Campo Mourão, **28 de novembro de 2017**

Prof. Dr. Diego Bertolini Gonçalves
Membro 1

Prof. Dr. Marco Aurélio Graciotto Silva
Membro 2

Prof. Dr. Igor Scaliante Wiese
Orientador

A ata de defesa assinada encontra-se na coordenação do curso.

Agradecimentos

Gostaria de expressar meus sinceros agradecimentos a todos aqueles que, direta ou indiretamente, colaboraram no processo de elaboração deste trabalho. Amigos, professores, colegas de trabalho e familiares. Em especial gostaria de agradecer:

Ao meu orientador Igor Wiese, o qual com paciência me deu todo o suporte com suas correções e incentivos. Igor, sou muito grata a você por tudo o que me ensinou. Também sou grata aos pesquisadores Marco Aurélio Gerosa, Maurício Aniche e Gustavo Oliva, que apoiaram cada etapa da pesquisa e contribuíram com as revisões do conteúdo.

Aos membros da banca, os professores Diego Bertolini Gonçalves e Marco Aurélio Graciotto Silva, por aceitarem o convite e contribuírem para o enriquecimento científico da minha pesquisa.

Aos amigos que caminharam junto a mim durante a graduação. Em especial ao Bruno Mendes.

Agradeço à minha mãe, Carmeluci, que sempre foi minha maior fonte de inspiração e força. Sou grata ao meu pai, Wilson, por acreditar e apoiar meu sonho.

A Deus, por ter me dado esta oportunidade e me sustentado em todos os momentos.

Resumo

Wessel, Mairieli S. Otimização de Recomendações de Mudanças de Software por meio de Algoritmo Genético. 2017. 81. f. Monografia (Curso de Bacharelado em Ciência da Computação), Universidade Tecnológica Federal do Paraná. Campo Mourão, 2017.

Contexto: Pesquisadores têm tentado recomendar artefatos de software que provavelmente mudarão juntos para ajudar os desenvolvedores a fazer mudanças em um projeto de software. Os acoplamentos evolutivos constituem uma peça fundamental de técnicas de recomendação de mudanças e são geralmente descobertos por regras de associação. Um dos grandes problemas em utilizar essa técnica na prática é que o uso de regras de associação requer uma série de parâmetros de configuração. Determinar esses parâmetros significa realizar estudos empíricos para cada projeto, testando variações antes de escolher uma configuração que seja adequada para o projeto.
Objetivo: A fim de possibilitar a utilização de regras de associação para gerar recomendações de mudanças conjuntas que sejam utilizadas por desenvolvedores de software na prática, este trabalho investiga como determinar empiricamente os limiares de suporte e confiança, bem como o conjunto de treinamento que geram as recomendações de mudança com maior acurácia para um projeto de software.
Método: Este trabalho propõe a definição de um modelo de recomendação de mudança utilizando o algoritmo Apriori e Algoritmo Genético para selecionar o histórico de mudanças e os limiares de suporte e confiança, a fim de otimizar as recomendações de mudança geradas. Para a avaliação do modelo proposto, a acurácia das recomendações geradas foram comparadas com outros dois modelos definidos com base em uma função de regressão e no algoritmo TARMAQ, Regressão+Apriori e Regressão+TARMAQ. Experimentos foram executados em 16 projetos de código aberto, avaliando cenários com consultas geradas a partir de transações no conjunto de teste.
Resultados: Os resultados indicam que o modelo proposto, Genético+Apriori, possui acurácia maior do que o modelo Regressão+Apriori. Porém, para cenários com consulta fixa, o modelo proposto possui resultados semelhantes ao modelo Regressão+TARMAQ e, em cenários com consulta variável o Regressão+TARMAQ possui desempenho superior para, em média, 40% dos projetos. Os três modelos sofrem deterioração a medida que novas transações são inseridas.
Conclusões: Os resultados são encorajadores, pois repositórios como o GitHub hospedam muitos projetos com pouco histórico. Nossos resultados podem ser usados por pesquisadores ao

realizar estudos de previsão de mudanças conjuntas e por desenvolvedores para produzir suporte automatizado para ferramentas de recomendação de mudanças.

Palavras-chaves: Recomendação de Mudanças. Regras de Associação. Algoritmo Genético.

Abstract

Wessel, Mairieli S. Tweaking Association Rules to Optimize Software Change Recommendations. 2017. 81. f. Monograph (Undergradute Program in Computer Science), Federal University of Technology – Paraná. Campo Mourão, PR, Brazil, 2017.

Context: Past researchs have been trying to recommend artifacts that are likely to change together in a task to assist developers in making changes to a software system, often using techniques like association rules. Association rules learning is a data mining technique that has been frequently used to discover evolutionary couplings. These couplings constitute a fundamental piece of modern change prediction techniques. However, using association rules to detect evolutionary coupling requires a number of configuration parameters, such as measures of interest (e.g. support and confidence), their cut-off values, and the portion of the commit history from which co-change relationships will be extracted. To accomplish this set up, researchers have to carry out empirical studies for each project, testing a few variations of the parameters before choosing a configuration. This makes it difficult to use association rules in practice, since developers would need to perform experiments before applying the technique and would end up choosing non-optimal solutions that lead to wrong predictions.
Objective: In search of enabling the use of association rules to mine co-change recommendations that are actually used by software developers, this work research a way to empirically determine the threshold of support and confidence as well as a training set so that recommendations are more accurate for a given software project.
Method: To optimize co-change recommendations, this work proposes a model that make use of both Apriori and Genetic Algorithm to select the change history, support and confidence thresholds. The evaluation is done by comparing our proposed model accuracy with two other models, based on a regression function and TARMAQ algorithm proposed in the literature, Regression+Apriori and Regression+TARMAQ. Experiments were made in sixteen open source projects, comparing the accuracy of recommendations produced by a varying query scenarios.
Results: The results show that the Genetic+Apriori model has a better accuracy when compared to Regression+Apriori model. When testing against fixed query scenarios, the proposed model shows similar results when comparing with Regression+TARMAQ. In varying query scenarios however, Regression+TARMAQ has a better accuracy than the proposed model in 40% of the tested projects. We also find that, as more transactions are added on software change history, all

three models deteriorate in accuracy.
Conclusions: The results are particularly encouraging, because repositories such as GitHub host many young projects. Our results can be used by researchers when conducting co-change prediction studies and by tool developers to produce automated support to be used by practitioners.

Keywords: Change Recommendations. Association Rules. Genetic Algorithm

Lista de figuras

Lista de tabelas

Lista de acrônimos

AG Algoritmo Genético 23

AP *Average Precision* 23

MAP *Mean Average Precision* 23

TARMAQ *Targeted Association Rule Mining for All Queries* 14

API *Application Programming Interface* 44

HTTP *Hypertext Transfer Protocol* 44

SVD *Singular Value Decomposition* 27

CVS *Concurrent Version System* 26

Sumário

Capítulo 1

Introdução

De acordo com as leis de Lehman (LEHMAN, 1980), à medida que um sistema evolui, sua estrutura tende a tornar-se mais complexa. A complexidade afeta diretamente como o sistema é modificado. Particularmente, essa complexidade afeta em grande parte desenvolvedores novatos e contribuidores casuais de projetos de software livre, uma vez que a falta de conhecimento sobre a arquitetura do software cria dificuldades em encontrar quais artefatos devem ser modificados para resolver uma determinada tarefa (STEINMACHER et al., 2015; PINTO et al., 2016).

Para auxiliar os desenvolvedores, foram propostas técnicas que identificam artefatos de software que mudam frequentemente juntos. Diz-se que há um acoplamento de mudança entre esses artefatos (HASSAN; HOLT, 2004; CANFORA et al., 2010). Com base na identificação desses acoplamentos, recomendam-se artefatos com base na premissa de que artefatos que mudam juntos frequentemente no passado são propensos a mudar juntos no futuro (ZIMMERMANN et al., 2005).

Entretanto, a acurácia das recomendações de mudanças é afetada por um conjunto de fatores, tais como: tamanho do histórico de mudanças do projeto (também chamado de conjunto de treinamento), configuração de medidas das interesse (por exemplo, suporte e confiança) usadas para identificar os acoplamentos de mudança e pela inclusão de novas mudanças (*commits* que ocorrem durante a evolução do software). Embora os guias práticos atuais tenham investigado fatores que influenciam a acurácia das recomendações como, tamanho do histórico de mudanças e consultas invisíveis (MOONEN et al., 2016a, 2016b), eles não definiram valores recomendados para configuração das medidas de interesse.

O objetivo deste trabalho é investigar como determinar empiricamente limiares de suporte, confiança e o tamanho do conjunto de histórico de mudanças que geram as recomendações de mudança com maior acurácia. O modelo proposto utiliza um Algoritmo Genético que visa encontrar o conjunto de treinamento com base no histórico de mudanças (*commits*) e limiares de suporte e confiança que otimizam as recomendações de mudança para um projeto de software.

Para tal, definimos uma função de aptidão (do inglês, *fitness function*) que avalia a acurácia das recomendações de mudança geradas.

Para avaliação do modelo proposto, Genético+Apriori, comparamos a acurácia das recomendações produzidas com a acurácia das recomendações geradas por outros dois modelos, definidos a partir dos trabalhos de Moonen et al. (2016b) e Rolfsnes et al. (2016). Moonen et al. (2016b) propôs uma função de regressão que retorna o valor para o tamanho do conjunto de treinamento. O modelo Regressão+Apriori utiliza essa função de regressão em conjunto com o algoritmo Apriori para gerar as recomendações. O modelo Regressão+TARMAQ utiliza o algoritmo *Targeted Association Rule Mining for All Queries* (TARMAQ), proposto por Rolfsnes et al. (2016), para gerar as recomendações de mudança a partir do histórico definido pela função de regressão de Moonen et al. (2016b).

Foram utilizados o histórico de mudanças de 16 projetos de código aberto disponíveis no GitHub para responder duas questões de pesquisa: **($\mathbf{QP}_1$)** Como a acurácia do modelo de recomendação de mudanças baseado em Algoritmo Genético se compara àquelas dos modelos propostos por Moonen et al. (2016b) e Rolfsnes et al. (2016)? **($\mathbf{QP}_2$)** Como a acurácia dos modelos de recomendação de mudança se comporta quando o conjunto de teste aumenta?

Os resultados para a $\mathbf{QP}_1$ indicam que o modelo Genético+Apriori possui acurácia maior do que o modelo Regressão+Apriori. Porém, para cenários com consulta fixa no primeiro arquivo modificado por um desenvolvedor, o modelo proposto possui resultados semelhantes ao modelo Regressão+TARMAQ e em cenários com consulta de tamanho variável o modelo Regressão+TARMAQ possui desempenho superior para, em média, 40% dos projetos avaliados.

Para a $\mathbf{QP}_2$, os resultados mostram que acurácia dos modelos de recomendação de mudança se deteriora a medida que novas transações são inseridas. A inserção de novas mudanças implicou, em média, na deterioração de $37,5\%$ dos projetos tanto nos cenários com consultas fixas, quanto em cenários com consultas de tamanho variável para cada um dos modelos de recomendação.

A principal vantagem do modelo proposto está em se ajustar aos projetos com diferentes tamanhos de histórico de modificações, enquanto os modelos baseados na função de regressão não conseguiram identificar o tamanho do conjunto de treinamento capaz de ser usado na prática, uma vez que o tamanho do conjunto recomendado foi maior do que a quantidade de histórico disponível nos projetos. Esse resultado é particularmente interessante porque a maioria dos projetos hospedados em repositórios como o GitHub não têm histórico de mudanças grande (RAY et al., 2014) e, consequentemente, não conseguiriam usar a função de regressão na prática.

O restante deste trabalho está organizado da seguinte forma. O Capítulo 2 apresenta um referencial teórico, abordando os principais conceitos relacionados com este trabalho como, Recomendação de Mudanças Conjuntas, Regras de Associação e Algoritmo Genético e também apresenta os trabalhos relacionados. O Capítulo 3 apresenta em detalhes o modelo proposto de recomendação de mudanças conjuntas baseada em Algoritmo Genético. A metodologia aplicada

neste trabalho é apresentada Capítulo 4. Os resultados estão reportados no Capítulo 5. No Capítulo 6 estão descritas as ameaças à validade do estudo. Por fim, as conclusões são apresentadas no Capítulo 7.

Capítulo 2

Referencial Teórico

Neste capítulo são apresentados os fundamentos utilizados nesta monografia, assim como os trabalhos relacionados. Na Seção 2.1 expõe-se os conceitos primordiais sobre Recomendação de Mudanças Conjuntas, a definição de Regras de Associação e dois algoritmos capazes de gerar tais regras, Apriori e TARMAQ, e apresenta-se a definição e características dos Algoritmos Genéticos na Seção 2.1.4. A Seção 2.2 apresenta os trabalhos relacionados a esta monografia, bem como suas limitações. Por fim, a Seção 2.3 aborda as considerações finais referentes a este capítulo.

2.1. Fundamentação Teórica

Esta monografia aborda conceitos de Recomendação de Mudanças Conjunta de Software e Algoritmo Genético. Técnicas de recomendação de mudanças vem sendo propostas para auxiliar os desenvolvedores nas tarefas de manutenção e evolução do software (ZIMMERMANN et al., 2005; CANFORA et al., 2010; MOONEN et al., 2016b). Algoritmo Genético pode ser usado para otimização de tais recomendações, assim como tem sido usado para otimizar outros problemas em Engenharia de Software (FILHO; VERGILIO, 2015; COLARES et al., 2009).

2.1.1. Recomendação de Mudança Conjunta

Durante os períodos de manutenção e evolução do software, quando os desenvolvedores modificam um artefato de software, é muito importante ter conhecimento de quais outros artefatos podem ser afetados por essa mudança. Segundo Zimmermann et al. (2005), o conhecimento sobre o impacto de uma mudança tem consequências importantes na qualidade do software pois previne erros causados por mudanças incompletas e também guia decisões sobre as tarefas de manutenção e evolução do software.

Suponha que uma alteração no arquivo A pertencente ao modelo de dados afeta os arquivos B e C pertencentes à camada da lógica de negócios. Se uma determinada tarefa de um

desenvolvedor, por exemplo, envolver alterações apenas nos arquivos A e B, ele poderia ser avisado sobre prováveis mudanças necessárias no arquivo C para evitar que ocorram mudanças incompletas e possíveis inconsistências no sistema. O exemplo ilustra o que é conhecido na literatura como recomendação de mudança conjunta (do inglês, *co-change recommendation*). A recomendação de mudanças conjuntas tem sido utilizada para realizar recomendações de artefatos que são propensos a mudarem conjuntamente a aqueles que estão sendo alterados pelos desenvolvedores (ZIMMERMANN et al., 2005).

Tradicionalmente, as técnicas de análise de impacto de mudança são baseadas na análise estática e dinâmica do código fonte (ARNOLD, 1996). Recentemente, técnicas baseadas na análise de dependências de mudanças (ZIMMERMANN et al., 2005; CANFORA et al., 2010) têm sido utilizadas para analisar o impacto de mudanças e recomendar mudanças conjuntas de artefatos de software.

Essas abordagens, tanto a análise estática e dinâmica do código fonte, quanto a análise de dependências de mudanças, se baseiam na relação de dependência entre os artefatos para prever a ocorrência de uma mudança conjunta e gerar uma recomendação. Entretanto, a análise de dependência de mudança se difere por se basear em acoplamentos de mudança, ou seja, na forma como o software evoluiu ao longo do tempo.

O acoplamento de mudança (do inglês, *change coupling*) (BALL et al., 1997) é um relacionamento "oculto" entre artefatos que mudam conjuntamente durante a evolução de um software. Quanto mais mudanças conjuntas dois artefatos tiverem, mais acoplados por mudanças se tornam e maior é a probabilidade de estes dois artefatos continuarem mudando conjuntamente no futuro (maior a dependência de mudança entre eles). Assim, analisando o histórico de modificações de um software, um relacionamento de dependência de mudança é inferido entre artefatos que frequentemente mudam conjuntamente durante o desenvolvimento de software.

Baseado na premissa de que "artefatos que mudaram juntos no passado tendem a mudar juntos no futuro" (ZIMMERMANN et al., 2005), acoplamentos de mudança se tornaram um componente fundamental de diversas técnicas e ferramentas de recomendação (ou predição) de mudanças (DIT et al., 2014; MALIK; HASSAN, 2008).

2.1.2. Regras de Associação

Uma regra de associação (AGRAWAL et al., 1993) é um tipo especial de implicação expressa por $X \Rightarrow Y$ e que é lida da seguinte forma "quando X ocorre, Y tende a ocorrer" (o contrário não é necessariamente verdade). Mais especificamente, X e Y são conjuntos disjuntos de itens, sendo X chamado de *antecedente* e Y chamado de *consequente*.

Por exemplo, considere que x, y e z são livros a venda em uma livraria online. Uma regra $\{x, y\} \Rightarrow \{z\}$ encontrada ao minerar dados de vendas indicaria que se um cliente compra os livros x e y, então ele tem uma grande chance de comprar o livro z. Note que, para obter as regras, é

necessário que se tenha acesso às transações. No exemplo anterior, cada transação corresponde ao conjunto de itens comprados por um cliente.

No contexto de desenvolvimento de software, os sistemas de controle de versão armazenam o histórico de mudanças dos artefatos. Logo, um *commit* pode ser interpretado como uma transação que contêm um conjunto de arquivos modificados. Assim, minerando o sistema de controle de versão, é possível extrair regras de associação como $\{x\} \Rightarrow \{y\}$, indicando que "quando um desenvolvedor modifica o artefato x, ele tende a também modificar o artefato y". Portanto, regras de associação são uma forma natural de se identificar acoplamentos de mudança entre artefatos de software (BAVOTA et al., 2013; ZIMMERMANN et al., 2005; CANFORA et al., 2010). Analogamente às regras de associação, dizemos que um artefato de código y está evolucionariamente acoplado a um outro artefato x se y frequentemente muda junto com x.

Duas medidas de interesse são calculadas para filtrar as regras relevantes: suporte e confiança. O suporte é a medida que representa a frequência de uma regra no conjunto de transações. Um conjunto de regras que aparece em muitas transações é dito ser frequente. O suporte da regra $X \Rightarrow Y$, escrito como $Suporte(X \Rightarrow Y)$, indica o número de transações que contêm tanto X quanto Y. A força de uma regra, por sua vez, é medida pela confiança. A confiança, escrita como $Confiana(X \Rightarrow Y)$, determina o quão frequente Y aparece em transações que contêm X.

Seja a frequência de um conjunto de itens X em um conjunto de transações θ definida como $freq_\theta(X) = |\{T \in \theta, X \subseteq T\}|$, as medidas de interesse $Suporte$ e $Confiana$ são definidas formalmente como:

$$Suporte(X \Rightarrow Y) = \frac{freq_\theta(X \cup Y)}{|T|} \tag{2.1}$$

$$Confiana(X \Rightarrow Y) = \frac{Suporte(X \Rightarrow Y)}{Suporte(X)} = \frac{freq_\theta(X \cup Y)}{freq_\theta(X)} \tag{2.2}$$

Como originalmente definida (AGRAWAL et al., 1993), a mineração de regras de associação gera regras que expressam padrões em um conjunto de dados completo. Isso significa que, dado um conjunto de transações, serão geradas regras que expressam padrões encontrados em todo o conjunto. Um exemplo dessa técnica será apresentado adiante pelo algoritmo Apriori.

A Mineração de Regra de Associação Alvo (do inglês, *Targeted Association Rule Mining*) (SRIKANT et al., 1997) é um refinamento da técnica de mineração de regra de associação que restringe a geração das regras com base em uma restrição fornecida pelo usuário. Dado um conjunto de restrições são geradas apenas regras que o satisfaçam. As regras não relacionadas com o conjunto de restrições são filtradas previamente, permitindo uma redução drástica do tempo de execução do algoritmo e na quantidade de regras que são criadas (SRIKANT et al., 1997).

No contexto de recomendação de mudanças conjuntas de software, uma restrição, também chamada de consulta Q, é constituída pelos artefatos modificados por um desenvolvedor desde a

última sincronização com o controle de versão. Seja $A = \{a, b\}$ o conjunto de arquivos modificados por um desenvolvedor, se faz necessário encontrar os padrões no histórico de mudanças do projeto que envolvam os arquivos modificados para que seja possível realizar recomendações.

Para tal, as técnicas de mineração de regras de associação alvo irão gerar regras restringindo os antecedentes ao conjunto A. Por exemplo, considerando o histórico de modificações do projeto $H = [\{a, b, c\}, \{a, b, d\}]$ e a consulta $Q = A$, duas regras são geradas $\{a, b\} \Rightarrow \{c\}$ e $\{a, b\} \Rightarrow \{d\}$. Um exemplo dessa técnica será apresentado adiante pelo algoritmo TARMAQ.

O Algoritmo Apriori

O algoritmo Apriori (AGRAWAL; SRIKANT, 1994) é frequentemente usado para se obter regras de associação eficientemente. Para encontrar todos os conjuntos de itens frequentes (L_k), esse algoritmo requer um valor de suporte e confiança mínimo como entrada, de modo que itens não frequentes são removidos preventivamente.

Empregando busca em profundidade, o algoritmo é capaz de encontrar conjunto de itens frequentes de tamanho k a partir dos conjunto de itens frequentes de tamanho $k - 1$. A premissa deste algoritmo é que se um conjunto de itens frequentes L tem suporte mínimo, todos os seus subconjuntos também terão (AGRAWAL; SHAFER, 1996).

Desta forma, o algoritmo Apriori decompõe a obtenção das regras de associação em dois passos:

- **Encontrar conjuntos de itens frequentes**: Encontrar todos os conjuntos de itens com suporte maior que o suporte mínimo. Os conjuntos de itens que estão acima do suporte mínimo são denominados conjunto de itens frequentes.
- **Gerar as regras de associação**: A partir de cada um dos conjuntos de itens obtidos são criadas regras. Todas as regras criadas a partir de um único conjunto de itens têm o mesmo suporte, porém somente as regras que estão acima da confiança mínima são retornadas.

O primeiro passo do algoritmo Apriori é realizar a geração do conjunto de itens candidatos C_k de tamanho k a partir do conjunto de itens frequentes $L_k - 1$ de tamanho $k - 1$. O conjunto C_k é gerado juntando pares de itens de tamanho $k - 1$ que possuem $k - 2$ elementos em comum. Assim, será gerado um conjunto de itens de tamanho k onde ao menos dois de seus subconjuntos de tamanho $k - 1$ é frequente. Em seguida é realizada uma busca nas transações para encontrar o valor de suporte de cada item em C_k em que, itens com suporte menor que o suporte mínimo são descartados, restando apenas os conjuntos de itens frequentes.

Após obter todos os conjuntos de itens frequentes, são descobertas as regras de associação. A geração dessas regras, para qualquer conjunto de itens frequentes L, significa encontrar todos os subconjuntos não vazios de L. Assim, para toda regra candidata A, onde $A \subset L$, é produzida uma regra $A \Rightarrow (I - A)$ somente se a $Confiana(A \Rightarrow (I - A))$ for maior ou igual a confiança mínima estabelecida.

O Algoritmo TARMAQ

O TARMAQ, definido por Rolfsnes et al. (2016), é um algoritmo que implementa mineração de regra de associação alvo. TARMAQ foi criado para de resolver a limitação fundamental da aplicação de mineração direcionada de regra de associação na recomendação de mudanças a um desenvolvedor.

Essa limitação ocorre porque, dado um histórico de transações H e uma consulta Q, uma técnica de mineração direcionada de regras de associação é dita não aplicável a consulta Q se o histórico filtrado, após a remoção de todas as transações não relacionadas a Q, for vazio (ROLFSNES et al., 2016). Assim, TARMAQ busca transações que contém subconjuntos de Q, obtendo recomendações na ausência de transações envolvendo todos os itens da consulta.

Esse algoritmo recebe como entrada um histórico de transações H e uma consulta Q e gera uma lista ordenada com os arquivos melhor classificados que estão mais relacionados com a Q. O Algoritmo TARMAQ consiste em três etapas: filtragem das transações, criação das regras e a criação da lista ordenada.

Os autores do TARMAQ demonstram sua aplicabilidade com o seguinte exemplo. Consideraremos que o histórico de versão de um software H contém três transações:

$$H = [\{a, b, c\}, \{a, d\}, \{c, d\}]$$

Na primeira etapa do algoritmo é realizada a filtragem das transações. Desta forma, dada uma consulta $Q = \{arquivo_1, arquivo_2, \cdots, arquivo_n\}$, uma transação $T \in H$ é mantida se $\mid T \cup Q \mid = k$, onde k é o tamanho do maior subconjunto de Q visto no histórico. Considerando a consulta $q_1 = \{a, c, d\}$, a filtragem não remove nenhuma das transações do histórico, visto que cada uma das transações inclui dois dos aquivos da consulta q_1 e neste caso $k = 2$. Todavia, para a consulta $q_2 = \{a\}$ a transação $\{c, d\}$ é removida pela filtragem, porque $\{a\} \cap \{c, d\} =$, e neste caso $k = 1$.

Após filtrar as transações, TARMAQ gera as regras de associação na forma $Q^{'} \Rightarrow x$, onde $Q^{'}$ é um subconjunto da consulta Q, com $\mid Q^{'} \mid = k$, e x representa um único arquivo no consequente da regra. Essa regra é criada se e somente se existir uma transação $T \in H$ tal que $Q^{'} \cup \{Q\} \subseteq T$. TARMAQ gera três regras para a consulta q_2: $\{a\} \Rightarrow \{b\}$, $\{a\} \Rightarrow \{c\}$ e $\{a\} \Rightarrow \{d\}$.

A última etapa do algoritmo é responsável por produzir uma lista ordenada com os arquivos recomendados. Isso é realizado ordenando as regras pelo valor de suporte, e se houverem empates, o valor da confiança é considerado para a ordenação.

Exemplo dos Algoritmos Apriori e TARMAQ

Para demonstrar a aplicabilidade e limitações dos algoritmos Apriori e TARMAQ, serão apresentados exemplos de geração de regras dado um histórico de mudanças:

$$H = [\{a, b, c\}, \{a, b\}, \{b, c\}, \{a, b, c\}, \{b\}, \{a, c, d\}, \{e, f\}, \{f\}]$$

O conjunto H representa o histórico de mudanças dos arquivos de um projeto de software. Esse histórico contém oito transações, e cada uma delas representa um *commit*.

O algoritmo Apriori necessita dos limiares de suporte e confiança para restringir a geração das regras. Neste exemplos, o limiar de suporte utilizado será de $0,16$ e o limiar de confiança de $0,10$. Com este limiar de suporte serão geradas regras que aconteceram no mínimo uma vez no histórico, uma vez que o histórico contém oito transações e $\frac{1}{8} = 0,16$.

As regras de associação geradas pelo algoritmo Apriori nestes exemplos correspondem as regras que possuem o antecedente igual a consulta Q fornecida. A mesma abordagem de filtragem das regras foi estabelecida no trabalho de Zimmermann et al. (2005).

Considerando a consulta $q_1 = \{a\}$, constituída por apenas um arquivo, em que o mesmo está presente no histórico H, os algoritmos Apriori e TARMAQ geram as regras listadas nas Tabela 2.1 e Tabela 2.2, respectivamente.

Tabela 2.1. Regras geradas pelo Apriori para a consulta $\{a\}$.

Regra	Suporte	Confiança
$\{a\} \Rightarrow \{b\}$	$0,375$	$0,75$
$\{a\} \Rightarrow \{c\}$	$0,375$	$0,75$
$\{a\} \Rightarrow \{d\}$	$0,125$	$0,25$

Tabela 2.2. Regras geradas pelo TARMAQ para a consulta $\{a\}$.

Regra	Suporte	Confiança
$\{a\} \Rightarrow \{b\}$	$0,75$	$0,75$
$\{a\} \Rightarrow \{c\}$	$0,75$	$0,75$
$\{a\} \Rightarrow \{d\}$	$0,25$	$0,25$

As regras de associação geradas pelos dois algoritmos para a consulta q_1 são iguais. Foram geradas três regras de associação. A diferença entre as regras geradas pelo Apriori e as regras geradas pelo TARMAQ está nos valores de suporte de cada regra. Essa diferença é decorrente da forma como o algoritmo TARMAQ calcula o suporte de cada regra. No processo de filtragem das transações realizado pelo TARMAQ, algumas transações não relacionadas com a consulta ou com um subconjunto da consulta são removidas, restando um histórico com um número menor de transações. As regras e o suporte de cada uma das regras são calculados com base no histórico filtrado, e por esta razão o valor do suporte é maior do que o calculado pelo Apriori.

O segundo exemplo utiliza uma consulta composta por dois arquivos. A consulta é definida por $q_2 = \{a, c\}$. Assim como na consulta q_1, os arquivos da consulta q_2 estão presentes no histórico H. Há também em H, transações em que os dois arquivos ocorrem juntos. Para a consulta q_2 os algoritmos Apriori e TARMAQ geram as regras listadas nas Tabela 2.3 e Tabela 2.4, respectivamente.

Tabela 2.3. Regras geradas pelo Apriori para a consulta $\{a, c\}$.

Regra	Suporte	Confiança
$\{a, c\} \Rightarrow \{b\}$	$0,25$	$0,66$
$\{a, c\} \Rightarrow \{d\}$	$0,125$	$0,33$

Tabela 2.4. Regras geradas pelo TARMAQ para a consulta $\{a, c\}$.

Regra	Suporte	Confiança
$\{a, c\} \Rightarrow \{b\}$	$0,66$	$0,66$
$\{a, c\} \Rightarrow \{d\}$	$0,33$	$0,33$

A terceira consulta $q_3 = \{b, g\}$ é composta por dois arquivos e apresenta um cenário em que apenas um dos seus arquivos está presente no histórico. A consulta q_3 tem um arquivo novo. Para esse tipo de consulta o algoritmo Apriori não é capaz de gerar regras de associação. A Tabela 2.5 apresenta as regras geradas pelo algoritmo TARMAQ.

Tabela 2.5. Regras geradas pelo TARMAQ para a consulta $\{b, g\}$.

Regra	Suporte	Confiança
$\{b\} \Rightarrow \{a\}$	0,6	0,6
$\{b\} \Rightarrow \{c\}$	0,6	0,6

Observando as regras é possível notar que apenas regras com o antecedente b foram geradas. Como o os dois arquivos da consulta q_3 não foram encontrados juntos em nenhuma transação em H, o antecessor das regras passou a ser b, pois ele é o maior subconjunto de q_3 encontrado em H.

A quarta e ultima consulta $q_4 = \{a, f\}$ é composta por dois arquivos. Ambos os arquivos estão presentes no histórico H, porém nunca ocorreram juntos em uma mesma transação. Assim como para a consulta q_3, o algoritmo Apriori não é capaz de gerar regras de associação para a consulta q_4. A Tabela 2.6 apresenta as regras geradas pelo algoritmo TARMAQ.

Tabela 2.6. Regras geradas pelo TARMAQ para a consulta $\{a, f\}$.

Regra	Suporte	Confiança
$\{a\} \Rightarrow \{b\}$	0,5	0,75
$\{a\} \Rightarrow \{c\}$	0,5	0,75
$\{f\} \Rightarrow \{e\}$	0,16	0,5
$\{a\} \Rightarrow \{d\}$	0,16	0,25

As regras geradas pelo algoritmo TARMAQ para a q_4 são padrões encontrados no histórico que representam os maiores subconjuntos de q_4 encontrados nas transações. Como os arquivos a e f não foram encontrados em uma mesma transação, os subconjuntos de q_4 encontrados em H foram os próprios arquivos a e f.

2.1.3. Avaliação de algoritmos de predição de mudanças conjuntas

A capacidade das regras de associação de produzir recomendações precisas e completas pode ser medida por meio de *Precisão* e *Sensibilidade* (SU et al., 2015). A *Precisão* de uma recomendação é a razão entre o número de recomendações corretas e o total de itens recomendados. A *Sensibilidade*, por sua vez, é a razão entre o número de recomendações corretas e o total de itens esperados.

Formalmente, *Precisão* e *Sensibilidade* podem ser definidos como segue:

$$Precisão = \frac{|F \cap E|}{|F|} \tag{2.3}$$

$$Sensibilidade = \frac{|F \cap E|}{|E|} \tag{2.4}$$

De acordo com o trabalho de Rolfsnes et al. (2016), Precisão Média, do inglês *Average Precision* (AP), é a métrica mais indicada para avaliar listas de recomendações. Essa métrica leva em consideração a ordem em que os arquivos aparecem na lista de recomendações. Uma recomendação correta no início da lista é mais provável de ser considerada pelo desenvolvedor, e por este motivo deve receber uma avaliação melhor do que as recomendações corretas que estiverem ao final da lista.

Dado um resultado esperado E para uma consulta Q e lista de recomendação ordenada F, a AP é definida como:

$$AP = \sum_{k=1}^{|F|} P(k) * \Delta r(k) \tag{2.5}$$

Onde, $P(k)$ é a precisão calculada nos primeiros k arquivos da lista de recomendações e $\Delta r(k)$ é a diferença entre a *Sensibilidade* calculada nos $k-1$ primeiros arquivos e a *Sensibilidade* calculada nos k primeiros arquivos.

Como medida de desempenho global, a Média das Precisões Médias, do inglês *Mean Average Precision* (MAP), pode ser usada sobre o conjunto de todas as consultas executadas. O valor de MAP representa uma média dos valores da métrica AP obtido para cada consulta e é definido como:

$$MAP = \frac{1}{|Q|} \sum_{q \in Q} AP(q) \tag{2.6}$$

2.1.4. Algoritmo Genético

Algoritmo Genético (AG) (HOLLAND, 1975) é uma técnica de busca e otimização inspirada nos princípios da evolução e seleção natural. Os AGs fornecem um mecanismo de busca que pode ser usado tanto em problemas de classificação, quanto em problemas de otimização. O AG simula o processo de evolução natural, ou biológica, no qual os indivíduos mais aptos dominam sobre os mais fracos imitando mecanismos biológicos de evolução, tais como seleção natural, recombinação e mutação.

A estrutura básica de um AG que será descrita a seguir está representada na Figura 2.1.

A população inicial é o ponto de partida para um AG. Na forma mais simples de AG (HOLLAND, 1975) a população inicial é gerada aleatoriamente, na qual cada indivíduo representa uma solução candidata para o dado problema. Cada um desses indivíduos é constituído por genes, responsáveis pela herança genética de características através das gerações. O ciclo de evolução se inicia com a avaliação da aptidão de cada indivíduo para solucionar o problema.

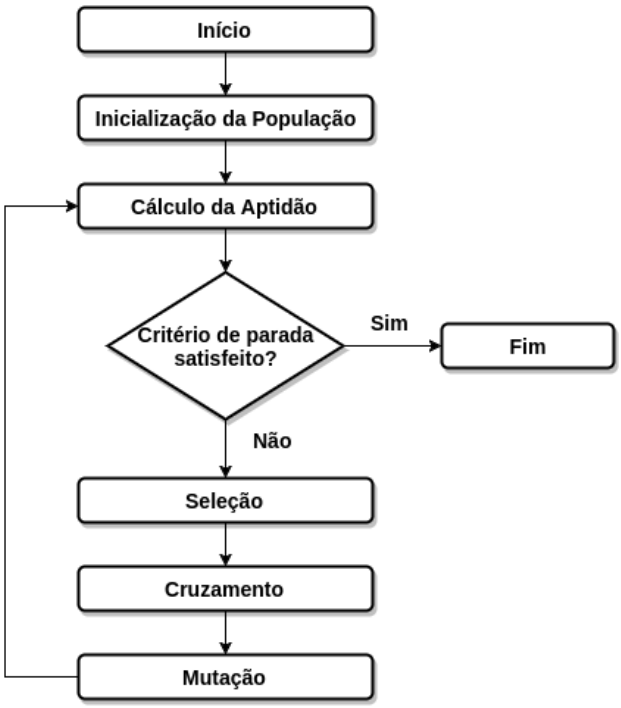

Figura 2.1. Estrutura de um Algoritmo Genético.
Fonte: Autoria própria. Adaptado de Miranda (2007).

Em um problema de busca e otimização, deve haver uma forma de determinar o quão boa é uma solução. Em um AG, isso é realizado pela função de aptidão (do inglês, *fitness function*), que confere uma nota para cada indivíduo de acordo com a sua aptidão para solucionar o problema (TANOMARU, 1995). Assim, o objetivo é maximizar o valor da função de aptidão de modo que a cada nova geração, os indivíduos estejam mais próximos do ótimo global, que é a melhor solução para o problema.

Após o cálculo da aptidão dos indivíduos, o próximo passo consiste em reproduzir esses indivíduos, formando uma nova população com o mesmo tamanho da população inicial. O operador de seleção tende a selecionar, com probabilidade proporcional ao respectivo valor de aptidão dos indivíduos, os indivíduos que serão os próximos progenitores.

Os novos novos indivíduos (isto é, descendentes) são então gerados após a aplicação de um operador de recombinação (do inglês, *crossover*). Na ausência do operador de recombinação, os descendentes são apenas clones dos indivíduos progenitores. Esse operador tem associado a si uma probabilidade de ocorrência. Assim, dado dois progenitores, na ocorrência de recombinação um ponto de recombinação é definido e com isso os progenitores são recombinados gerando dois novos indivíduos, como mostra a Figura 2.2.

Após a recombinação, o operador de mutação é aplicado aos descendentes com uma baixa probabilidade. Esse operador de mutação diversifica a população, adicionando uma informação inteiramente nova ao indivíduo. Basicamente, uma posição no cromossomo é selecionada e seu valor é alterado, como pode ser observado na Figura 2.3.

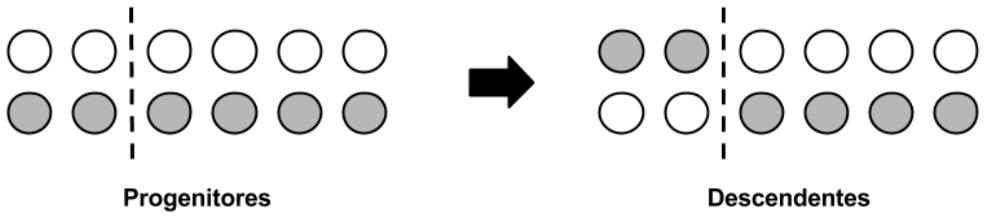

Figura 2.2. Comportamento do Operador de Recombinação
Fonte: Autoria própria.

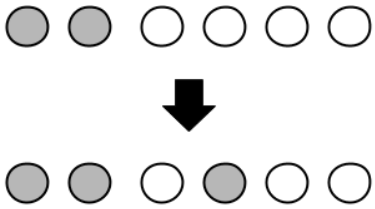

Figura 2.3. Comportamento do Operador de Mutação
Fonte: Autoria própria.

O processo de evolução é encerrado com base em critérios de convergência, geralmente um número máximo de gerações (HOLLAND, 1975). Alternativamente, o processo de evolução é interrompido quando um grande número de gerações não apresenta melhoria no melhor valor de aptidão, ou quando um valor predefinido é atingido.

O desempenho de um AG está fortemente relacionado com os parâmetros utilizados (TANOMARU, 1995). Em um AG (HOLLAND, 1975), o usuário deve definir o tamanho da população, além das probabilidades de recombinação e mutação.

2.2. Trabalhos Relacionados

Esta Seção apresenta os principais trabalhos relacionados a mineração de regras de associação, recomendação de mudança conjunta e trabalhos que utilizam AG para otimizar problemas na Engenharia de Software.

Desde o trabalho de Agrawal et al. (1993) introduzindo o conceito de Mineração de Regras de Associação, técnicas vem sendo propostas para melhorar sua execução e eficiência. O algoritmo Apriori (AGRAWAL; SRIKANT, 1994), apresentado na Seção 2.1.2, é o mais conhecido e utiliza um limiar de suporte e confiança para realizar um eficiente cálculo prévio dos candidatos à geração das regras de associação. Um refinamento desses algoritmos é trazido pelas técnicas de Mineração de Regras de Associação Alvo (SRIKANT et al., 1997). Tais técnicas focam a geração das regras em uma consulta fornecida pelo usuário (SRIKANT et al., 1997; HAFEZ et al., 1999; KUBAT et al., 2003), filtrando previamente as transações que não estão relacionadas permitindo uma redução drástica no tempo de execução do algoritmo (SRIKANT et al., 1997).

2.2.1. ROSE

Utilizando a técnica de mineração de regras de associação alvo e o algoritmo Apriori, Zimmermann et al. (2005) construíram ROSE, um componente para o ambiente de desenvolvimento Eclipse para alertar os desenvolvedores sobre possíveis mudanças conjuntas baseado em arquivos ou métodos que estão sendo alterados.

A Figura 2.4 ilustra o fluxo básico através da ferramenta ROSE. O Servidor ROSE lê o histórico de mudanças do projeto, agrupa as mudanças em transações e gera as regras de associação que descrevem implicações entre artefatos do software (arquivos, funções e variáveis).

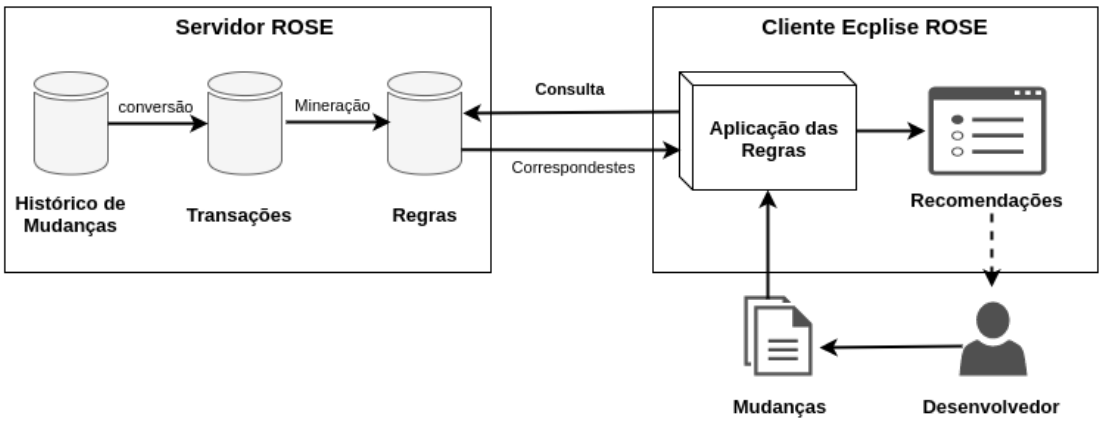

Figura 2.4. O fluxo de dados através do ROSE.
Fonte: Autoria própria. Adaptado de Zimmermann et al. (2005).

Quando um desenvolvedor altera algum artefato do projeto de software, o cliente ROSE realiza uma consulta ao servidor e as regras de associação correspondentes são geradas. Dada as regras correspondentes a consulta, o cliente ROSE apresenta uma lista de recomendações de novas mudanças ao desenvolvedor.

O servidor ROSE recupera as mudanças e as transações, conforme descrito acima, do sistema de versões concorrentes, do inglês *Concurrent Version System* (CVS). Esse sistema de controle de versões fornece apenas controle de versão de arquivos individualmente. Transações envolvendo um ou mais arquivos necessitam serem inferidas das modificações individuais realizadas nos arquivos, com base no tempo.

A fim de agrupar os arquivos modificados conjuntamente em transações, Zimmermann et al. (2005) seguiram a abordagem de janela deslizante: duas mudanças subsequentes feitas pelo mesmo autor são parte de uma transação se estiverem a uma distância máxima de 200 segundos.

Dada as transações, o objetivo do servidor ROSE é minerar regras dessas transações. Como parte do pré-processamento, foram filtradas transações com mais de 30 artefatos modificados. ROSE usa o algoritmo Apriori para computar as regras de associação. O uso clássico do algoritmo Apriori toma como entrada um limiar de suporte e um limiar de confiança e gera um conjunto

de regras sobre todas transações do histórico de mudanças. No entanto, foram realizadas duas otimizações no algoritmo Apriori.

A primeira otimização no algoritmo Apriori, realizada por Zimmermann et al. (2005), está relacionada com a decisão de restringir a geração das regras com base em uma consulta fornecida para o algoritmo utilizando a técnica de mineração de regras de associação alvo. A segunda otimização é a mineração de regras com único consequente. Segundo Zimmermann et al. (2005), essas otimizações foram adotadas para reduzir o tempo de execução do Apriori.

Para avaliar a abordagem proposta, oito projetos de software livre foram utilizados. Dentre os projetos utilizados estão Eclipse, GCC, Gimp, JBoss, JEdit, KOffice, Postgresql, Python. Foram usadas as últimas 1.000 transações para verificar se os arquivos poderiam ser previstos pela história anterior. As transações mais antigas foram destinadas a geração das regras e as mais recentes a um conjunto de avaliação.

Para avaliar a capacidade de prever arquivos usando a história anterior, foram usadas as transações do conjunto de avaliação. Para cada transação do conjunto de avaliação t foram criados casos de teste $q = (Q, E)$, consistindo em uma consulta $Q \subset t$ e um resultado esperado $E = t - Q$.

Os resultados de ROSE foram satisfatórios ao sugerir prováveis mudanças a partir de uma mudança inicial, usando um limiar de confiança de 10% e um limiar de frequência das regras de 1 ocorrência. Dada uma alteração inicial em um artefato $Q = \{e\}$, sendo $\{e\}$ o primeiro artefato modificado, ROSE pode prever 15% dos artefatos que mudaram em conjunto com $\{e\}$, e para 64% das transações do conjunto de avaliação ROSE conseguiu informar as três primeiras recomendações da lista corretamente.

Avaliando a capacidade de ROSE em ajudar na prevenção de erros devido a alterações incompletas, para cada uma das transações no conjunto de avaliação foram realizadas consultas $Q = t - \{e\}$ e $E = \{e\}$. Para a geração das regras foram usados um limiar de confiança de 10% e um limiar de frequência das regras de 1 ocorrência. Dada uma transação em que uma mudança está faltando, ROSE pôde prever 4% das entidades que precisavam ser alteradas.

2.2.2. Consultas Invisíveis e TARMAQ

O trabalho de Rolfsnes et al. (2016) apresenta TARMAQ, um algoritmo para mineração de regras de associação alvo, descrito em detalhes na Seção 2.1.2. Além disso, Rolfsnes et al. (2016) contribuíram com a classificação das limitações dos algoritmos existentes e uma comparação entre a acurácia das recomendações geradas pelo TARMAQ e a acurácia das recomendações geradas por ROSE (ZIMMERMANN et al., 2005) e também pela Decomposição em Valores Singulares, do inglês *Singular Value Decomposition* (SVD).

No contexto da análise do impacto de mudança de software, Rolfsnes et al. (2016) identificaram uma classe particular de consultas para as quais as abordagens de mineração de regras de associação alvo até então encontradas na literatura (ZIMMERMANN et al., 2005; SRIKANT

et al., 1997) não podem gerar recomendações. A essas consultas foi dado o nome de consultas invisíveis (do inglês, *unseen queries*).

Estas consultas são ditas invisíveis pois apresentam um padrão não observado no conjunto de dados. Por exemplo, consideraremos que o histórico de versão de um software H contém três transações $H = [\{a, b, c\}, \{a, d\}, \{c, d\}]$. A consulta $q_1 = \{e, d\}$ é invisível pois inclui um novo arquivo e que ainda não havia ocorrido em nenhuma transação de H. O algoritmo TARMAQ é capaz de gerar recomendações para todos os tipos de consultas fornecidas.

Para avaliar a abordagem proposta e realizar a comparação entre os três algoritmos de recomendação de mudança conjunta foram analisados seis projetos de software. Destes, dois são projetos da indústria e quatro são software livre e têm seu código disponível no GitHub. Dentre os projetos utilizados estão MySQL, Git, Apache HTTP Server, Linux Kernel e os dois projetos da indústria, Kongsberg Maritme e Cisco Norway.

A fim de simular um cenário prático em que um desenvolvedor necessita de uma recomendação de mudança, foram criadas consultas que simulam os arquivos alterados por um desenvolvedor. Desta maneira, cada consulta representa o conjunto de arquivos que um desenvolvedor modificou desde a última sincronização com o sistema de controle de versão.

Como parte do pré-processamento dos dados, foram removidos *commits* com mais de 30 arquivos, seguindo o que foi realizado no trabalho de Zimmermann et al. (2005). Dos *commits* restantes, para cada um dos tamanhos de *commit* (2 a 30) foram retiradas aleatoriamente 40 amostras. As 1.160 transações resultantes são usadas para gerar as consultas. Cada transação T foi particionada em uma consulta Q e um resultado esperado $E = T - Q$, sendo possível avaliar a habilidade do algoritmo prever E a partir de Q. Foram criadas para cada transação consultas de tamanho 1 até $|T| - 1$.

Para avaliar uma consulta gerada a partir de uma transação T, foram usados os 10.000 *commits* anteriores a T para gerar as recomendações de mudança. Cada uma das consultas é executada, e então a lista de recomendação gerada é comparada com o resultado esperado E. A métrica que avalia a acurácia das recomendações geradas por uma consulta é o AP. Para avaliar os três algoritmos é usado o MAP. Os resultados mostraram em 90% dos casos, pelo menos uma recomendação correta estava entre as dez primeiras recomendações das recomendações geradas por TARMAQ e ROSE.

2.2.3. Regressão Linear

O trabalho de Moonen et al. (2016a) apresenta um estudo sistemático dos efeitos do tamanho do histórico usado para gerar as recomendações e do número de transações que ocorreram desde que as regras foram mineradas. As duas contribuições principais deste trabalho são investigar o quanto o tamanho e o envelhecimento das historico afetam a qualidade das recomendações de mudança de um projeto de software, e estabelecer diretrizes práticas que possam auxiliar os desenvolvedores a determinar o tamanho do histórico.

A avaliação da abordagem proposta foi realizada em dois grandes projetos da indústria e dezessete projetos de software livre. Dentre os projetos de software livre estão CPython, Mozilla Gecko, Git, Apache Hadoop, HTTPD, IntelliJ IDEA, Liferay Portal, Linux Kernel, LLVM, MediaWiki, MySQL, PHP, Ruby on Rails, RavenDB, Subversion, WebKit, Wine. Os projetos Cisco Norway e Kongsberg Maritime foram cedidos pela indústria. De cada um dos projetos foram utilizadas e extraídas as 50.000 transações mais recentes. Devido a diversidade dos projetos, essas 50.000 transações representam 20 anos de desenvolvimento em alguns projetos e cerca de 10 meses em outros.

Como parte do pré-processamento dos dados, foram removidos *commits* com mais de 8 arquivos. Com os *commits* restantes, foram retiradas 660 amostras e para cada uma dessas transações foram aleatoriamente gerada uma consulta Q e um resultado esperado E.

Para avaliar a abordagem foram criados cenários que são uma combinação fixa de um tamanho de histórico e um envelhecimento. O tamanho do histórico é quantidade de transações do histórico usadas para minerar as regras de associação. O envelhecimento é número de transações que ocorreram desde que essas regras foram mineradas. Ao todo foram criados vinte e quatro cenários, os tamanhos de histórico variaram entre 5.000, 15.000, 25.000, 35.000 transações, e o envelhecimento entre 0, 1.000, 2000, 3.000, 4.000 e 5.000 transações.

O estudo com os vinte e quatro cenários mostrou que a qualidade das recomendações diminui com o envelhecimento do histórico ao mesmo tempo que aumenta com históricos maiores. A fim de investigar tais descobertas foram adicionados dois estudos mais detalhados: tamanhos de histórico menores para um envelhecimento fixo de 0 transações, e envelhecimentos menores para um tamanho de histórico fixo de 25.000 transações.

Dado que ocorre um nivelamento na aplicabilidade a medida que o tamanho do histórico cresce, a analise sugeriu que 25.000 *commits* é o ponto em que há histórico suficiente para aprender sobre o projeto em questão e produzir recomendações aceitáveis.

Moonen et al. (2016b) definem uma função de regressão baseada no número de arquivos e média do tamanho dos *commits* do projeto para prever o valor do tamanho do histórico que deve ser usado como conjunto de treinamento. A função de regressão é dada por:

$$\begin{aligned} \mathit{Tamanho\ do\ treinamento} = {} & 33974,1 \\ & + 208,4 \times \mathit{Número\ de\ arquivos} \\ & - 3958,9 \times \mathit{Média\ do\ tamanho\ dos\ commits} \end{aligned} \tag{2.7}$$

2.2.4. Limitações

A literatura de mineração de repositório de software frequentemente menciona que a escolha do tamanho do histórico usado para extrair as regras interfere na acurácia dos resultados (GRAVES et al., 2000; HASSAN, 2008; ZIMMERMANN et al., 2005). Isso ocorre tanto ao escolher um histórico

extremamente pequeno ou extremamente grande, seja porque não há informações suficientes para gerar conhecimento sobre o sistema, ou porque algumas informações já estão desatualizadas.

O trabalho de Moonen et al. (2016b), apresentado na Seção 2.2.2, analisa impacto do tamanho do histórico de mudanças utilizado para extrair as regras de associação. Porém, foram considerados apenas projetos com mais de $50k$ transações no histórico. Considerar projetos menores na análise é importante para que sejam avaliados projetos comuns no cenário prático. A maioria dos projetos hospedados em repositórios como o GitHub não têm histórico de mudanças grande (RAY et al., 2014).

Pesquisadores de mineração de regras de associação (ZHENG et al., 2001; MAIMON; ROKACH, 2010) expõem a necessidade de investigar a influência dos parâmetros dos algoritmos como a escolha dos limiares das medidas de interesse. No contexto de recomendação de mudanças de software, determinar os limiares ideais para que as regras sejam suficientemente relevantes é uma tarefa complicada pois depende das características do projeto em questão.

No trabalho de Zimmermann et al. (2005), por exemplo, apenas regras de associação com frequência maior ou igual a 2 e confiança maior que $0,5$ são consideradas relevantes. Por sua vez, Bavota et al. (2013) consideraram relevante regras com limiar de confiança $0,8$ e frequência mínima de 2 ou 3. No entanto, nenhum trabalho anterior relata como escolher automaticamente um limiar de suporte e confiança, e também um conjunto de treinamento para um projeto de software a fim de otimizar as recomendações geradas.

Neste sentido, pesquisadores de Engenharia de Software têm utilizado AGs em vários cenários que necessitam de otimização. Por exemplo, na área de testes, AGs foram utilizados para gerar e avaliar um conjunto de teste para linha de produto de software a partir de parâmetros como operadores de mutação (FILHO; VERGILIO, 2015). Colares et al. (2009) usaram um AG multiobjetivo para mostrar a aplicabilidade de sua abordagem para o problema de planejamento de *realease* de software.

2.3. Considerações Finais

Neste capítulo, foram apresentados os principais conceitos relacionados a esta monografia, como a definição de Recomendação de Mudanças Conjuntas, as técnicas e formas de avaliação de recomendação de mudança utilizadas na literatura e definição de AG.

Apesar do esforço realizado pelos pesquisadores, existem ainda questões em aberto em relação a aspectos práticos da aplicação dos acoplamentos de mudança para a recomendação de mudanças conjuntas (OLIVA; GEROSA, 2015). Em particular, o tamanho do histórico do projeto, assim como a escolha dos limiares para suporte e confiança, interferem na qualidade das regras geradas (ZIMMERMANN et al., 2005; MOONEN et al., 2016b). Uma abordagem baseada em AG pode ser usada para automatizar a obtenção desses parâmetros em conjunto com a otimização das recomendações de mudança.

Capítulo 3

Modelo de Recomendação de Mudança Genético+Apriori

Neste capítulo, o modelo de recomendação de mudanças baseado em AG é apresentado. Tal modelo visa automatizar as recomendações de tamanho do histórico de mudanças e limiares de suporte e confiança para um projeto de software. Na Seção 3.1 é apresentada a função de aptidão responsável por avaliar as possíveis soluções do problema. As configurações do AG estão descritas na Seção 3.2.

Para construir o modelo proposto, um AG foi utilizado para explorar efetivamente o espaço de busca de possíveis combinações de tamanho do histórico de mudanças, e limiares de suporte e confiança para selecionar as recomendações de mudanças com maior acurácia esperadas para um projeto de software.

3.1. Função de Aptidão

A função de aptidão para o AG, responsável por avaliar cada uma das potencias soluções (indivíduos), foi modelada considerando o tamanho do conjunto de treinamento (em percentual) e os limiares de suporte e confiança para otimizar as recomendações de mudanças conjuntas de um projeto de software.

Essa função de aptidão utiliza a porcentagem do conjunto treinamento para construir o modelo, gerando regras de associação, e utiliza também consultas para gerar as regras de associação e as avaliar, similarmente à avaliação feita por outros pesquisadores na área de recomendação de mudanças (ZIMMERMANN et al., 2005; MOONEN et al., 2016b, 2016a).

A otimização nas recomendação de mudança realizada pelo AG está baseada na métrica MAP, das consultas feitas sobre regras geradas. Portanto, o MAP é a métrica escolhida para avaliar o desempenho das recomendações de mudança conjunta.

Para o problema em questão, deve ser encontrado um limiar de suporte e confiança, bem como um tamanho de histórico de mudança que otimize as recomendações para um projeto de software, as potenciais soluções serão representadas por uma porcentagem do conjunto de treinamento, um limiar de suporte e um limiar de confiança chamados respectivamente de $porcentagem\_treinamento$, $suporte$ e $confianca$.

Além dos três parâmetros que representam uma potencial solução do AG, um conjunto de treinamento com transações do histórico de mudanças do projeto e um conjunto de teste com consultas e resultados esperados são passados como parâmetros adicionais para a função de aptidão e são mantidos inalterados durante todo o processo de evolução.

Algoritmo 1: FuncaoAptidao

Entrada: porcentagem_treinamento, suporte, confianca
Saída: O valor de MAP correspondente à aptidão das recomendações geradas

```
1  início
2      {Extração do treinamento}
3      treinamento_n ← extrair_transacoes(Treinamento, porcentagem_treinamento)
4      para cada ft ∈ Teste faça
5          Q ← ft.consulta
6          E ← ft.resultado_esperado
7          {Geração das regras}
8          R ← apriori(treinamento_n, suporte, confianca, Q)
9          R_p ← ordena(R, 10) {ordenação usando suporte e k=10}
10         F ← consequente(R_p)
11         {Avaliação da consulta}
12         ap ← calcula_ap(E, F)
13     fim
14     map ← media(ap)
15 fim
16 retorna map
```

Como pode ser observado no Algoritmo 1, a função de aptidão consiste em três passos principais: extração da porcentagem de treinamento, geração das regras e avaliação das consultas. Cada um desses passos são descritos a seguir.

Extração do treinamento. O primeiro passo da função de aptidão consiste em gerar um novo conjunto de treinamento contendo apenas a porcentagem de transações correspondente ao indivíduo que está sendo avaliado. Portanto, são extraídas do conjunto $Treinamento$ a porcentagem de transações mais recente correspondente à potencial solução $porcentagem\_treinamento$ determinada pelo AG e informada por parâmetro para a função. Um novo conjunto de treinamento $Treinamento_n$ é formado com as transações extraídas.

Geração das regras. A função de aptidão verifica para cada uma das transações do conjunto $Teste$ se seus itens podem ser previstos a partir das transações do conjunto $Treinamento_n$. Para desempenhar essa tarefa, a partir de $Treinamento_n$ regras de associação são geradas com base em uma consulta Q e nas medidas de interesse $suporte$ e $confiana$. Optou-se então por selecionar as mudanças conjuntas, que gerarão as recomendações de mudança, com o uso de regras de associação porque são comumente utilizadas na literatura (ZIMMERMANN et al., 2005; CANFORA et al., 2010; YING et al., 2004; BALL et al., 1997).

O algoritmo Apriori (AGRAWAL et al., 1993), descrito anteriormente na Seção 2.1.2, é utilizado para minerar as regras de associação. Seguindo o trabalho de Zimmermann et al. (2005), são computadas apenas regras de associação R com um único item em seu consequente $X \Rightarrow \{e\}$. Regras com um único consequente são suficientes para o problema em questão, porque nesse modelo é considerada uma lista de recomendações formada pela união dos consequentes da regra X. Desta forma, o conjunto de regras R referentes a Q é obtido, significando que R contém apenas regras nas quais a consulta é o antecessor: $Q \Rightarrow \{e\}$.

As regras R, no cenário prático, representam uma lista de sugestão que o desenvolvedor recebe após alterar os arquivos da consulta Q. Assumindo que listas muito grandes não são viáveis no cenário prático por apresentarem ao desenvolvedor muitas sugestões e o forçarem a navegar por muitos arquivos, neste trabalho são consideradas apenas as primeiras 10 regras. Para isso, são selecionadas a partir de R as 10 primeiras regras, ordenadas por valor de suporte e confiança. Após essa filtragem por valor de suporte e confiança, um novo conjunto de regras R_p é gerado, apresentando tamanho menor ou igual a 10.

Avaliação das regras. A última etapa da função de aptidão consiste em devolver o valor de aptidão do indivíduo com base na avaliação das consultas realizadas. Trabalhos anteriores (ZIMMERMANN et al., 2005; CANFORA et al., 2010) utilizaram *Precisão* e *Sensibilidade* como métricas de desempenho. Entretanto, a avaliação da execução de uma consulta é realizada neste trabalho por meio de AP.

3.2. Cenário de Avaliação

A função de aptidão, apresentada na Seção 3.1, foi implementada na linguagem R (versão 3.3.1 - "Bug in Your Hair"), utilizando o pacote GA (SCRUCCA, 2013) (versão 3.0.2) para implementação e execução do AG. O pacote GA possui um flexível conjunto de ferramentas para otimização usando AG. Por padrão, as buscas usando o GA são executadas sequencialmente, entretanto, podem ser executadas utilizando um mecanismo de paralelização ofertado pelo próprio pacote. A paralelização foi utilizada para acelerar a execução do AG.

Para gerar as regras de associação, o pacote *arules* (HAHSLER et al., 2011) (versão 1.5.2) foi utilizado. Esse pacote provê toda a infraestrutura para representar, manipular e analisar dados

de transações, assim como a mineração de padrões com regras de associação por meio de uma interface para o algoritmo Apriori implementado em C.

3.2.1. Configurações do Algoritmo Genético

Quanto às configurações do AG, usamos uma probabilidade de cruzamento de $0,8$ e uma probabilidade de mutação de $0,1$. Determinamos quatro tamanhos diferentes para a população: 25, 50, 100 e 200 indivíduos. O objetivo foi avaliar o impacto do tamanho da população na otimização das recomendações de mudanças, a fim de definir a quantidade de indivíduos necessária para oferecer total cobertura do espaço de busca.

Quanto maior o tamanho da população, maior a complexidade computacional gasta para avaliar os indivíduos em cada uma das iterações do algoritmo. No entanto, pequenas populações não possuem a diversidade necessária para convergir rapidamente para uma boa solução do problema (HAUPT; HAUPT, 1998). A população ideal para um determinado problema é quando há um equilíbrio entre um baixo número de gerações para a convergência e uma maior precisão à medida que a população aumenta (GOTSHALL; RYLANDER, 2002).

O pacote GA utiliza um conjunto de configurações padrão para operadores genéticos. Seguimos as recomendações do pacote e utilizamos os mesmos operadores. A seleção dos indivíduos foi realizada com base na escala linear, do inglês *linear scaling*. O operador de cruzamento usado foi o cruzamento aritmético local, do inglês *local arithmetic crossover*, e o operador de mutação foi a mutação uniforme aleatória, do inglês *uniform random mutation*.

Seguindo as configurações usadas no trabalho que originou este (WESSEL et al., 2017), no qual avaliamos a acurácia das recomendações de mudanças usando AG para cinco projetos de código aberto, escolhemos uma população de 200 indivíduos para utilizar nas demais execuções. Essa escolha deve-se ao fato de termos observado uma convergência mais rápida para o ótimo global ao utilizar uma população de 200 indivíduos no problema em questão. Como condição de parada para o AG, terminamos a evolução após atingir o número máximo de 100 gerações.

3.2.2. Restrição do espaço de busca

Para limitar o espaço de busca do AG, estabelecemos o limite máximo e mínimo para cada uma das variáveis de decisão. Esses limites são usados pelo algoritmo ao gerar novos indivíduos. Ao limitar o espaço de busca também estamos reduzindo o tempo gasto na execução do AG.

Para a variável de decisão que representa a porcentagem de treinamento, adotamos o limite mínimo como 10% e o limite máximo em 100% das transações do conjunto de treinamento. Desta forma, o AG utiliza no mínimo 10% das transações do conjunto de treinamento de projeto para gerar as regras e recomendações.

Para a variável de decisão $confiana$, que corresponde à confiança mínima das regras geradas, estabelecemos 0.1 como limite mínimo e 1 para o limite máximo. Esses valores são

empregados nos trabalhos que calculam acoplamento de mudança (ZIMMERMANN et al., 2005; CANFORA et al., 2010; ROLFSNES et al., 2016).

Os limites da variável de decisão *suporte*, que corresponde ao suporte mínimo das regras geradas, são determinados de acordo com a quantidade de transações disponíveis no conjunto de treinamento de cada projeto, como segue:

$$Suporte\ m\acute{i}nimo = \frac{1}{Tamanho\ do\ treinamento}$$

$$Suporte\ máximo = \frac{20}{Tamanho\ do\ treinamento}$$

Assim, serão sempre geradas regras que correspondem a arquivos que mudaram juntos no mínimo de uma a vinte vezes em todo o período de treinamento.

3.3. Considerações Finais

Neste capítulo apresentamos como utilizamos o AG para encontrar os parâmetros (tamanho do conjunto de treinamento e limiares de suporte e confiança) que otimizam as recomendações de mudança para um projeto de software. Foram mostradas ainda as configurações definidas para o AG, como probabilidade de mutação, probabilidade de cruzamento e tamanho da população. Por fim, explicamos as técnicas utilizadas para limitar o espaço de busca do AG. O próximo capítulo abordará a metodologia usada neste trabalho, como as questões de pesquisa e como iremos respondê-las, coleta dos dados, pré-processamento e formas de avaliação.

Capítulo 4

Metodologia

Este capítulo descreve o método utilizado neste trabalho. Na Seção 4.1 é apresentado o objetivo, bem como o problema de pesquisa. As questões de pesquisa são apresentadas na Seção 4.2. Os cenários de avaliação utilizados para comparar o modelo proposto com os trabalhos de Rolfsnes et al. (2016) e Moonen et al. (2016b) estão descritos na Seção 4.3. Por fim, a Seção 4.4 descreve o método de seleção dos projetos para o experimento.

4.1. Objetivo

O objetivo deste trabalho é investigar como determinar empiricamente os limiares de suporte e confiança, assim como o tamanho do histórico de mudanças que geram as melhores recomendações de mudança para um projeto de software.

Essa investigação se faz necessária pois utilizar um histórico com poucas transações pode resultar em regras que não expressam conhecimento suficiente sobre o sistema. No entanto, um histórico com muitas transações pode conter informações desatualizadas e inserir ruídos nas regras geradas. Segundo Zimmermann et al. (2005), há um custo-benefício entre a quantidade de recomendações e a qualidade dessas recomendações. Usar valores baixos de suporte e confiança possibilita uma maior quantidade de regras de associação, mas com menor precisão de recomendações.

Determinar limiares ótimos das medidas de interesse e o tamanho do histórico ainda é uma tarefa difícil. Deste modo, é necessário que novas técnicas determinem a configuração ideal para que os desenvolvedores possam se valer de acoplamentos evolucionários na prática.

4.2. Questões de Pesquisa

Este trabalho propõe o uso de AG para otimizar a seleção do tamanho do histórico de mudanças e dos limiares de suporte e confiança para um projeto de software, conforme problematizado anteriormente. Para tal, duas questões de pesquisa precisam ser avaliadas empiricamente. A seguir são descritas as questões de pesquisa e um resumo da abordagem utilizada para responder cada uma delas.

QP_1: Como a acurácia do modelo de recomendação de mudanças baseado em Algoritmo Genético se compara àquelas dos modelos propostos por Moonen et al. (2016b) e Rolfsnes et al. (2016)?

Para avaliar o desempenho do modelo proposto, Genético+Apriori, em encontrar o tamanho do conjunto de treinamento e os limiares de suporte e confiança capazes de otimizar recomendações de mudanças para um projeto de software, utilizamos o trabalho de Moonen et al. (2016b) e de Rolfsnes et al. (2016).

O trabalho de Moonen et al. (2016b) propõe uma função de regressão para definir o tamanho do histórico de mudanças baseado no número de arquivos e na média do tamanho dos *commits* de um projeto. Definimos então, a partir da função de regressão, o modelo Regressão+Apriori. Com interesse em avaliar o impacto do tamanho do histórico de mudanças definido pela regressão com o encontrado pelo AG, comparamos a acurácia das recomendações geradas pelo modelo Genético+Apriori com as geradas pelo modelo Regressão+Apriori.

O algoritmo TARMAQ, proposto por Rolfsnes et al. (2016), será usado em conjunto com a função de regressão para a construção do modelo Regressão+TARMAQ. O TARMAQ será responsável por gerar as recomendações de mudanças de acordo com as transações do histórico de mudanças definido pela regressão. Com interesse em avaliar o impacto do algoritmo usado para a geração das regras de associação, comparamos a acurácia das recomendações geradas pelo modelo Genético+Apriori com as geradas pelo modelo Regressão+TARMAQ.

O objetivo desta questão de pesquisa é verificar se o modelo de recomendação proposto possui maior acurácia do que o modelo Regressão+Apriori e o modelo Regressão+TARMAQ. Para responder a $\mathbf{QP}_1$ serão utilizados métodos estatísticos de comparação de amostras. A ideia chave por trás da avaliação proposta para responder a $\mathbf{QP}_1$ consiste em executar os três modelos, utilizando o histórico mais recente para teste, e selecionar o melhor resultado de cada um dos modelos para análise.

QP_2: Como a acurácia dos modelos de recomendação de mudança se comporta quando o conjunto de teste aumenta?

A mudança de um software é um processo inevitável segundo a primeira lei de Lehman (LEHMAN, 1980). O ambiente de desenvolvimento muda constantemente, surgem novos requisitos e o software deve ser modificado para não se tornar progressivamente menos satisfatório. Nesta questão de pesquisa será investigado o quanto novas mudanças no sistema deterioram a estabilidade

do modelo de recomendações de mudanças. Para isso, serão inseridas incrementalmente novas mudanças no período de teste a cada execução dos modelos e será analisado o impacto na acurácia das recomendações geradas.

Para esta análise serão usados quatro períodos de teste, sendo eles 5%, 10%, 20% e 30% das transações mais recentes do histórico do projeto. Após a execução dos três modelos, os resultados com maior acurácia de cada modelo, para cada um dos períodos de teste, serão selecionados para análise. Para responder a **QP2** serão utilizados métodos estatísticos de comparação de amostras nos resultados obtidos.

O objetivo desta questão de pesquisa é verificar se a variação do tamanho do conjunto de teste de 5% até 30% afeta a estabilidade dos modelos fazendo com que eles percam acurácia. Se isso ocorrer, pode ser concluído que usar o modelo proposto neste trabalho é melhor, uma vez que ele conseguiria otimizar as recomendações em diferentes cenários sem a necessidade de realizar um estudo empírico para definir qual deveria ser o limiar de suporte e confiança que seria usado combinado com o tamanho do histórico de modificações sugerido pela regressão proposta por Moonen et al. (2016b), ou utilizando o TARMAQ.

4.3. Método

Nesta seção, apresentamos o método utilizado para avaliar os modelos de recomendação de mudança conjunta e responder as questões de pesquisa é apresentado. A Figura 4.1 mostra uma visão geral do modelo.

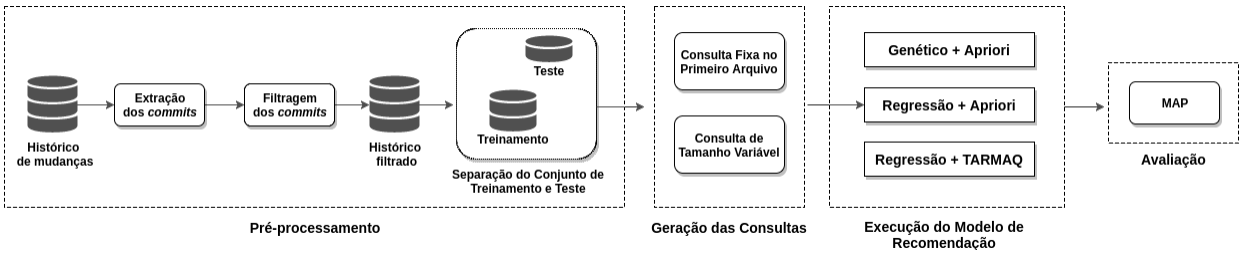

Figura 4.1. Visão geral do método de recomendação de mudanças conjuntas
Fonte: Autoria própria.

O método proposto foi dividido em quatro passos: pré-processamento, geração das consultas, execução do modelo de recomendação de mudanças e avaliação do modelo. Esses passos são detalhados a seguir.

4.3.1. Pré-processamento

O pré-processamento dos dados consiste nas etapas de extração dos dados, filtragem dos dados e separação do conjunto de treinamento e conjunto de teste. Cada uma das três etapas do pré-processamento dos dados será descrita em detalhes a seguir.

Extração dos *commits*. Na etapa de extração dos *commits*, o repositório de código fonte do projeto é clonado para que as informações de todo o histórico de mudanças (*commits*) sejam recuperadas. Foram consideradas apenas modificações presentes no ramo principal (*branch master*). As informações do histórico de mudanças do projeto recuperadas pela extração de dados são os arquivos que foram modificados, o tipo de modificação realizada em cada arquivo, o autor e data de cada *commit*. Essa etapa resulta em um arquivo CSV com os dados coletados do histórico de *commits* do projeto.

Seguindo os trabalhos de Moonen et al. (2016b) e Rolfsnes et al. (2016) que também coletaram dados de repositórios Git, cada *commit* do projeto é considerado uma transação. Isto significa que, teoricamente, os arquivos modificados para realizar uma determinada tarefa estão agrupados em um *commit* do projeto. Podem haver casos em que arquivos são erroneamente esquecidos de serem adicionados a um *commit* e por este motivo, são inseridos em um novo *commit*. Casos assim são considerados como dois *commits* separados e serão analisados em detalhes em um trabalho futuro.

Para desempenhar a tarefa de extração dos *commits* será utilizado o RepoDriller[1], um framework Java que auxilia na mineração de repositórios de software e possibilita a extração de informações de um repositório Git e a exportação para arquivos CSV. A necessidade de clonar os repositórios dos projetos que serão estudados está relacionada com a forma como o RepoDriller coleta os dados dos projetos.

Filtragem dos *commits*. Na filtragem dos *commits*, são removidos do histórico do projeto *commits* com mais de trinta arquivos, assim como no trabalho de Zimmermann et al. (2005), e também *commits* que possuem apenas imagens. Essas filtragens removem grandes transações que não apresentam relevância para se obter os acoplamentos de mudança dos arquivos (MOONEN et al., 2016a) e mudanças que não envolvem código-fonte.

Dentre todas as possíveis modificações em um arquivo (adicionar, deletar, copiar, renomear e modificar seu conteúdo), consideramos no *commit* apenas arquivos que foram adicionados, copiados, renomeados ou modificados, pois essas modificações são relevantes no cenário de recomendação de mudanças conjuntas. Para os arquivos renomeados em determinado *commit*, a renomeação foi aplicada também aos *commits* anteriores nos quais o arquivo estava presente. A filtragem dos *commits* resulta em um conjunto de transações $T = \{t_1, t_2, \cdots, t_n\}$, sendo n o número de *commits* que permaneceram no histórico após a filtragem.

Após a filtragem, a ordem dos arquivos em cada uma das transações em T é aleatorizada. Esta ação se deve ao fato de não sabermos qual a ordem em que os arquivos de um *commit* foram modificados pelo desenvolvedor. Assim, para simular um cenário prático de desenvolvimento será sempre considerado a ordem dos arquivos do *commit*, após a aleatorização, como sendo a sequência que o desenvolvedor modificou os arquivos.

[1] https://github.com/mauricioaniche/repodriller

Separação do conjunto de treinamento e teste. A etapa de separação do conjunto de treinamento e teste é responsável por dividir o conjunto de transações T, resultante da etapa de filtragem dos dados, em dois novos conjuntos. Uma porcentagem x das transações mais antigas de T são destinadas ao conjunto de treinamento e uma porcentagem y das subsequentes transações ao conjunto de teste.

Dado o conjunto de transações T, são gerados um conjunto de treinamento $Treinamento = \{t_1, t_2, \cdots, t_{s-1}\}$ e um conjunto de teste $Teste = \{t_s, t_{s+1}, \cdots, t_m\}$, em que s corresponde ao índice do primeiro elemento do conjunto de teste obtido a partir da porcentagem x escolhida para gerar o conjunto de treinamento e m corresponde ao último elemento do conjunto de teste obtido a partir da porcentagem y escolhida para gerar o conjunto de teste.

Para responder a $\mathbf{QP}_1$ foram definidas três configurações de separação do conjunto de treinamento e teste. Essas três configurações foram definidas com 5%, 10% e 20% das transações mais recentes para teste e, respectivamente, 95%, 90% e 80% das transações mais antigas para treinamento.

Para exemplificar a separação do histórico de mudanças para responder a $\mathbf{QP}_1$, consideraremos um projeto de software com vinte *commits* em seu histórico. Sendo a porcentagem x igual a 95% e a porcentagem y igual a 5%, os conjuntos de treinamento e teste resultantes da divisão podem ser vistos na Figura 4.2. Os 95% primeiros *commits* do histórico formam o conjunto de treinamento. O conjunto de teste é formado pelos 5% *commits* subsequentes ao treinamento.

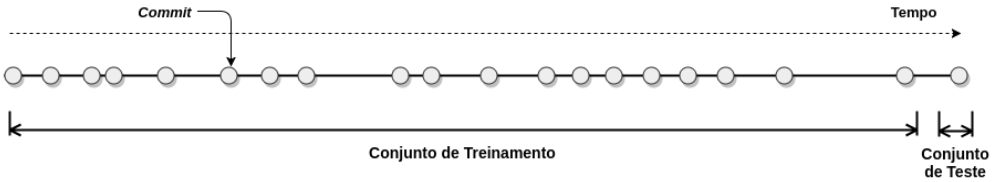

Figura 4.2. Divisão do histórico em 95% de treinamento e 5% de teste
Fonte: Autoria própria.

Para responder a $\mathbf{QP}_2$ foram definidas quatro configurações, semelhantemente ao realizado para responder a $\mathbf{QP}_1$. Todavia, estas configurações possuem uma porcentagem de treinamento fixa em 70% das transações mais antigas do histórico. Os conjuntos de teste são formados incrementalmente por 5%, 10%, 20% e 30% das transações subsequentes ao treinamento.

Para provar a robustez do modelo gerado e mostrar que o modelo é generalista e não especialista para um determinado conjunto, por se tratarem de dados que estão temporalmente relacionados, definimos os cenários de avaliação de $\mathbf{QP}_2$. Mostraremos se a inserção de novas mudanças no projeto faz com que o modelo perca acurácia.

A Figura 4.3 exemplifica a separação do histórico de mudanças, considerando o mesmo projeto de software com vinte *commits* do exemplo anterior.

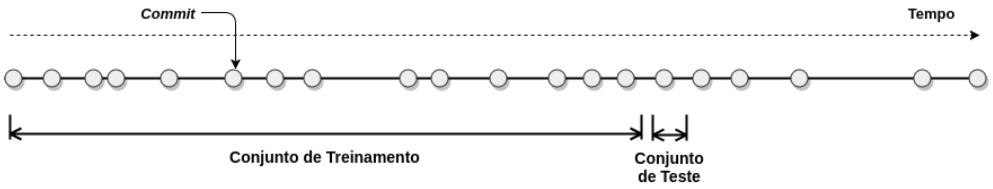

Figura 4.3. Divisão do histórico em 70% de treinamento e 5% de teste
Fonte: Autoria própria.

4.3.2. Geração das consultas

As transações do conjunto $Teste$ foram usadas para simular contextos reais de desenvolvimento de software. Nesses contextos, recomendações de mudança são necessárias para que seja possível completar uma determinada tarefa. Desta forma, usando as transações do conjunto $Teste$ é possível verificar se seus itens podem ser previstos a partir das transações do conjunto $Treinamento$.

Para desempenhar essa tarefa, uma consulta $q = (Q, E)$ é criada para cada uma das transações F_t. Sendo $F_t = \{f_1, \cdots, f_n\}$ uma transação pertencente ao conjunto $Teste$, ela é então particionada em uma consulta $Q = \{f_1, \cdots, f_m\}$ de tamanho $\mid Q \mid = m$, com $1 \leq m < n$, e um resultado esperado $E = F_t - Q$.

A quantidade de arquivos m presente em uma consulta Q está intimamente ligada ao que se deseja analisar. Neste trabalho foram utilizadas duas maneiras para definir, dada uma transação, a quantidade de arquivos usados para gerar a consulta. Foram criados um conjunto de consultas para analisar a recomendação de mudanças a partir da primeira modificação realizada e um conjunto de consultas para analisar a efetividade na prevenção de erros devido a alterações incompletas.

Consulta fixa no primeiro arquivo modificado. Para simular a recomendação de mudanças com base na primeira modificação realizada, assumimos o cenário em que um desenvolvedor está iniciando o processo de resolução de uma determinada tarefa por um arquivo arbitrário e a partir deste arquivo são geradas recomendações de mudança conjunta para o auxiliar a completar as demais alterações que a tarefa envolve.

Dada a descrição do primeiro cenário, as transações do conjunto $Teste$ são particionadas em uma consulta $Q = \{f_1\}$ e um resultado esperado $E = F_t - Q$. O arquivo f_1 representa o arquivo pelo qual o desenvolvedor iniciou a resolução da tarefa e os demais arquivos da transação, contidos em E são esperados como resultado da lista de recomendações.

Consulta de tamanho variável. Para verificar a efetividade das recomendações na prevenção de alterações incompletas, assumimos o cenário em que um desenvolvedor finalizou a resolução de uma tarefa, porém ainda não enviou suas modificações ao controlador de versões. Para evitar que ocorram erros ou inconsistências no projeto causadas por modificações incompletas, recomendações são realizadas com base em todos os arquivos que foram modificados.

Para o segundo cenário, as transações do conjunto $Teste$ são particionadas em uma consulta $Q = \{f_1 \cdots, f_m\}$ com um ou mais arquivos e um resultado esperado $E = F_t - Q$. O tamanho da consulta para cada transação $| Q | = m$ foi determinado aleatorizando um valor inteiro maior ou igual a 1 e menor que o tamanho da transação n. Para um transação com cinco arquivos, por exemplo, a consulta pode assumir um tamanho entre 1 e 4 arquivos, restando no mínimo o último arquivo para servir como resultado esperado.

4.3.3. Execução do modelo de recomendação de mudanças

Aplicar as consultas geradas a partir do conjunto $Teste$ no conjunto $Treinamento$, usando os modelos de recomendação de mudanças, resulta em uma lista de recomendação ordenada F, a qual corresponde aos consequentes das regras R geradas.

Para uma mesma consulta e um mesmo projeto de software, diferentes listas de recomendações podem ser geradas, dependendo do tamanho do histórico mudanças e do algoritmo e limiares utilizados para gerar as regras de associação. Os efeitos do tamanho do histórico de mudanças e do algoritmo de recomendação serão explorados com a execução e comparação dos modelos a seguir com o modelo proposto, Genético+Apriori.

Nos modelos Regressão+Apriori e Regressão+TARMAQ, a quantidade de transações do histórico de mudanças utilizadas para gerar as regras de associação é determinada pela função de regressão definida no trabalho por (MOONEN et al., 2016a). Os dois modelos foram implementado na linguagem R (versão 3.3.1 - “Bug in Your Hair”), tanto a extração das transações do histórico quanto a geração das regras de associação.

Modelo Regressão+Apriori. Neste modelo, o algoritmo Apriori é usado para gerar as regras que correspondem a uma consulta Q e aos limiares de suporte e confiança. Contudo, Moonen et al. (2016b) não apresentam quais limiares de suporte e confiança devem ser usados. Assim, o modelo Regressão+Apriori será gerado variando o limiar de frequência das regras de associação entre 2 a 20 ocorrências e os limiares de confiança em 0, 10, 0, 5 e 0, 9, de acordo com as recomendações de Zimmermann et al. (2005) e Bavota et al. (2013).

O modelo Regressão+Apriori foi implementado utilizando o pacote *arules* (HAHSLER et al., 2011) (versão 1.5.2) para gerar as regras de associação.

Modelo Regressão+TARMAQ. Neste modelo, o algoritmo TARMAQ é usado para gerar as regras que correspondem a uma consulta Q informada e ao histórico de mudanças definido pela função de regressão. Nenhum parâmetro adicional, como limiares de suporte e confiança, necessitam ser informados ao algoritmo.

4.3.4. Avaliação das consultas

Ao comparar uma lista de arquivos recomendados F, gerados a partir de uma consulta Q, com os resultados esperados E, são obtidos dois novos conjuntos de arquivos:

- **Recomendações corretas**: arquivos que estão em F e correspondem a um resultado esperado E.
- **Recomendações incorretas**: arquivos que estão em F, mas não correspondem a um resultado esperado E.

A avaliação de cada uma das consultas consiste em testar a capacidade dos modelos de recomendação de mudança em produzirem recomendações precisas e completas. Listas de recomendações mais precisas são aquelas que possuem maior quantidade de recomendações corretas. Já a completude de uma lista de recomendações está relacionada com conseguir recomendar os arquivos esperados.

Como descrito na Seção 2.1.3, a métrica AP considera a posição de uma recomendação correta na lista de recomendações. Devido ao contexto, a métrica AP foi escolhida para que a posição ocupada pelo arquivo recomendado seja levada em consideração na avaliação da recomendação. Como medida de desempenho global, foi escolhido para ser usado o MAP, sobre o conjunto de todas as consultas executadas.

4.3.5. Exemplo de Geração e Avaliação das Recomendações

A Figura 4.4 exemplifica a execução de uma consulta e geração das regras. Neste exemplo, $c = \{A, B, D, E\}$ representa um *commit* do conjunto de teste. No pré-processamento foi aleatorizada a ordem em que os arquivos aparecem nos *commits*, assumindo que a nova ordem em que os arquivos aparecem representa a ordem que foram modificados pelo desenvolvedor. Os arquivos são divididos em consulta $Q = A$ e resultado esperado $E = \{B, D, E\}$.

O conjunto de treinamento contém oito transações. Sobre essas transações a consulta $Q = A$ é aplicada. O algoritmo de recomendação de mudanças conjuntas se encarrega de gerar as regras de associação. Três regras são geradas e ordenadas por suporte e confiança. Os consequentes dessas regras, os arquivos $[B, C, D]$, formam a lista de recomendação.

A Figura 4.5 ilustra o cálculo da precisão $P(k)$ nos k primeiros arquivos da lista e da a diferença na sensibilidade $\Delta r(k)$ entre os $k-1$ e k primeiros arquivos, dada lista de recomendações $[B, C, D]$ e os arquivos relevantes $\{B, D, E\}$.

Feito o cálculo de $P(k)$ e $\Delta r(k)$ para todos os k arquivos recomendados, a métrica AP pode ser calculada da seguinte maneira:

$$AP = (1/1 * 1/3) + (1/2 * 0) + (2/3 * 1/3) \simeq 0.55$$

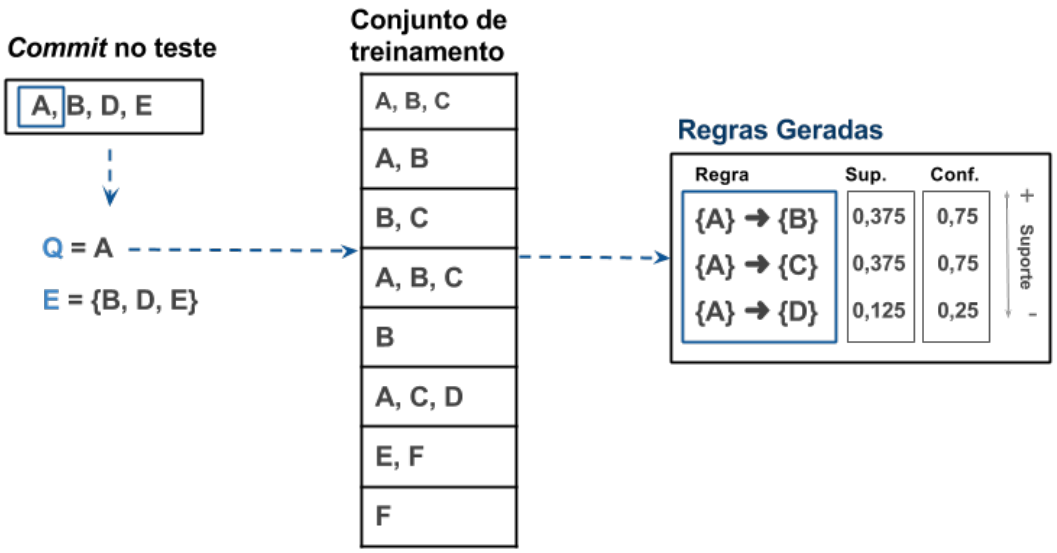

Figura 4.4. Exemplo de execução da consulta e geração das regras

Posição (k)	Arquivo	P(k)	Δr(k)
1	B	1/1	1/3
2	C	1/2	0
3	D	2/3	1/3

Figura 4.5. Exemplo de cálculo de $P(k)$ e $\Delta r(k)$

4.4. Projetos Estudados

Para avaliar os três modelos de recomendação de mudanças em várias condições, foram utilizados 16 projetos de código aberto. O objetivo da escolha dos projetos era selecionar projetos com características diferentes entre si: tamanhos de histórico de transações de tamanhos, frequência de *commits*, linguagem e domínio diferentes.

A escolha dos projetos se baseou nos projetos mais populares por linguagem no GitHub. O critério de seleção se refere ao número de estrelas que um repositório recebeu dos usuários do GitHub. Utilizando a *Application Programming Interface* (API) do GitHub v3, é possível realizar requisições *Hypertext Transfer Protocol* (HTTP) para obter os projetos mais populares por linguagem.

Foram selecionados quatro projetos por linguagem. Dentre as linguagens selecionadas estão: C, Java, JavaScript e Python. Após a coleta dos dados referentes aos projetos mais populares destas quatro linguagens, foram selecionados quatro projetos para cada linguagem com base na análise da distribuição de *commits* do histórico do projeto.

A partir da análise da distribuição dos *commits*, foram selecionados para cada linguagem projetos que correspondem ao primeiro quartil, mediana, terceiro quartil e média da distribuição. A Tabela 4.1 apresenta as características relevantes dos projetos selecionados para a avaliação e sua diversidade.

Tabela 4.1. Características dos projetos estudados

Projeto	URL do GitHub	Linguagem usada	Tamanho médio das transações	Arquivos únicos	Histórico de versão	Número de transações
RobotJS	https://github.com/octalmage/robotjs	C	1.49	48	31-08-2014 – 21-08-2017	472
s2n	https://github.com/awslabs/s2n	C	2.98	335	02-09-2014 – 21-08-2017	1487
netdata	https://github.com/firehol/netdata	C	2.30	595	17-06-2013 – 21-08-2017	5114
Emscripten	https://github.com/kripken/emscripten	C	1.98	2972	26-08-2010 – 21-08-2017	16656
SwitchButton	https://github.com/kyleduo/SwitchButton	Java	3.67	112	09-09-2014 – 21-08-2017	131
EventBus	https://github.com/greenrobot/EventBus	Java	2.38	143	15-07-2012 – 21-08-2017	415
MaterialDrawer	https://github.com/mikepenz/MaterialDrawer	Java	2.41	226	15-03-2014 – 21-08-2017	1720
Apache Zeppelin	https://github.com/apache/zeppelin	Java	3.39	1375	19-06-2013 – 21-08-2017	3347
rainyday.js	https://github.com/maroslaw/rainyday.js	JavaScript	1.46	22	01-08-2013 – 21-08-2017	255
lazysizes	https://github.com/aFarkas/lazysizes	JavaScript	5.88	109	11-10-2014 – 21-08-2017	604
Primus	https://github.com/primus/primus	JavaScript	1.66	148	25-06-2013 – 21-08-2017	1489
Styled Components	https://github.com/styled-components/styled-components	JavaScript	2.46	6703	16-08-2016 – 21-08-2017	1475
dev-setup	https://github.com/donnemartin/dev-setup	Python	1.13	44	08-07-2015 – 21-08-2017	342
Gmvault	https://github.com/gaubert/gmvault	Python	2.00	135	16-12-2011 – 21-08-2017	1085
Mantl	https://github.com/mantl/mantl	Python	2.28	803	04-11-2014 – 21-08-2017	3262
Keras	https://github.com/fchollet/keras	Python	2.01	289	27-03-2015 – 21-08-2017	3783

A Figura 4.6 mostra a distribuição da quantidade de arquivos por *commit* em cada um dos projetos estudados após a filtragem dos *commits*. Dentre os dezesseis projetos, treze possuem a mediana em 1 arquivo por *commit*. Dos outros três projetos, dois possuem a mediana em 2 arquivos por *commit*.

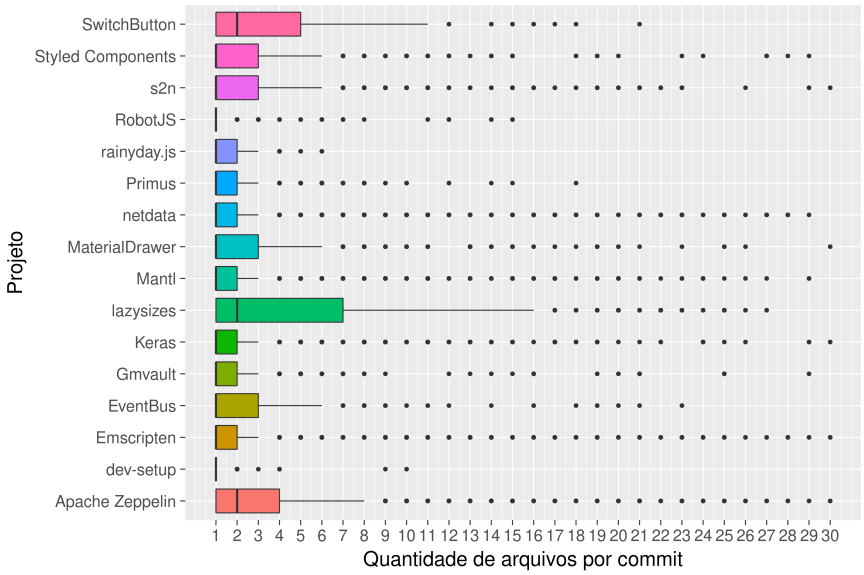

Figura 4.6. Distribuição da Quantidade de arquivos por *commit*

4.5. Considerações Finais

Neste capítulo apresentamos a metodologia utilizada neste trabalho. Explicamos as questões de pesquisa que serão investigadas e como as responderemos. Apresentamos os modelos de

recomendação de mudança Regressão+Apriori e Regressão+TARMAQ, definidos para serem usados em comparação ao modelo proposto no Capítulo 3, Genético+Apriori. Foi mostrado ainda todo o processo de coleta e pré-processamento dos dados, bem como os processos de geração e avaliação das consultas. A próxima seção irá mostrar os experimentos feitos e os resultados obtidos.

Capítulo 5

Resultados e Discussão

Neste capítulo, apresentamos os resultados dos experimentos realizados com o modelo Genético+Apriori para otimização de recomendações de mudanças conjuntas para 16 projetos de código aberto, apresentados anteriormente na Seção 4.4.

Os resultados obtidos com o modelo Genético+Apriori, em termos de acurácia do modelo, são comparados estatisticamente com os resultados provenientes dos outros dois modelos definidos, Regressão+Apriori e Regressão+TARMAQ. A implementação dos modelos está disponível na plataforma Github, no seguinte endereço: https://github.com/mairieli/Genetic-Apriori, sob a licença MIT.

Os resultados são separados em duas seções, de forma a responder cada uma das duas questões de pesquisa definidas neste trabalho. A Seção 5.1 aborda os resultados e discussões referentes a $\mathbf{QP}_1$, apresentando uma comparação entre a acurácia do modelo proposto, Genético+Apriori, com os modelos Regressão+Apriori e Regressão+TARMAQ. A Seção 5.2 aborda os resultados e discussões referentes a $\mathbf{QP}_2$, apresentando uma investigação da avaliação da deterioração dos modelos à medida que o conjunto de teste aumenta.

5.1. Avaliação da Acurácia dos Modelos de Recomendação de Mudanças Conjuntas

Para responder a $\mathbf{QP}_1$, como descrito anteriormente na Seção 4.3.1, foram definidos três configurações de separação do conjunto de treinamento e teste (95% e 5%, 90% e 10%, e por fim 80% e 20%), e também dois tipos de consultas (fixas no primeiro arquivo modificado e de tamanho variável). Cada uma das configurações de separação de treinamento e teste foram combinadas com os dois tipos de consultas, gerando então 6 cenários de avaliação diferentes. Tais cenários de avaliação foram usados para avaliar cada um dos três modelos de recomendação de mudanças, e executados para os 16 projetos estudados.

Aplicar no conjunto de treinamento uma consulta gerada a partir do conjunto de teste, usando um dos três modelos de recomendação de mudanças, resulta em uma lista de recomendações. A métrica precisão média foi usada para comparar essa lista com os resultados esperados pela consulta. Como as recomendações geradas dependem de fatores como o tamanho do histórico de mudanças, o algoritmo e os limiares das medidas de interesse usados para gerar as regras de associação, diferentes listas podem ser geradas dependendo do modelo utilizado para uma mesma consulta e um mesmo projeto de software.

Como visto na descrição dos projetos investigados, todos eles possuem a maior parte de seus históricos constituídos de transações contendo apenas um arquivo. Esse fato pode ser observado no gráfico de caixa da Figura 4.6. Transações contendo apenas um arquivo no conjunto de teste não servem como consulta, uma vez que não é possível fazer a separação entre consulta e resultado esperado. Assim, a Figura 5.1 mostra a porcentagem de transações removidas do teste após a separação dos conjuntos de treinamento e teste, no pré-processamento dos dados.

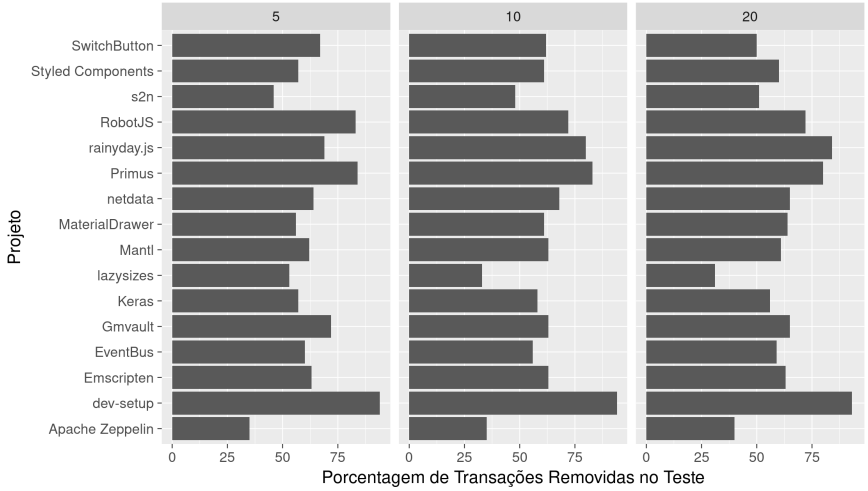

Figura 5.1. Porcentagem de transações removidas no conjunto de teste para a $\mathbf{QP}_1$

A fim de ilustrar o desempenho geral baseado na precisão média de cada consulta executada, analisamos como cada modelo de recomendação se comporta independente do projeto analisado e da configuração de treinamento e teste. A Figura 5.2 mostra a distribuição geral da precisão média para cada um dos modelos investigados, em cenários de avaliação com consultas fixas no primeiro arquivo modificado pelo desenvolvedor, por meio de um gráfico de caixa.

Os resultados de MAP mostrados para o modelo Genético+Apriori são os resultados encontrados pelo AG ao final da evolução de 100 gerações para uma população de 200 indivíduos. Para o modelo Regressão+Apriori, que foi executado variando a frequência das regras de associação entre 2 a 20 e os limiares de confiança em $0,10$, $0,5$ e $0,9$, os resultados considerados são os que obtiveram a maior acurácia para cada cenário de avaliação. Por fim, para o modelo Regressão+TARMAQ, que foi executado uma vez para cada cenário de avaliação, todos seus resultados são considerados e listados.

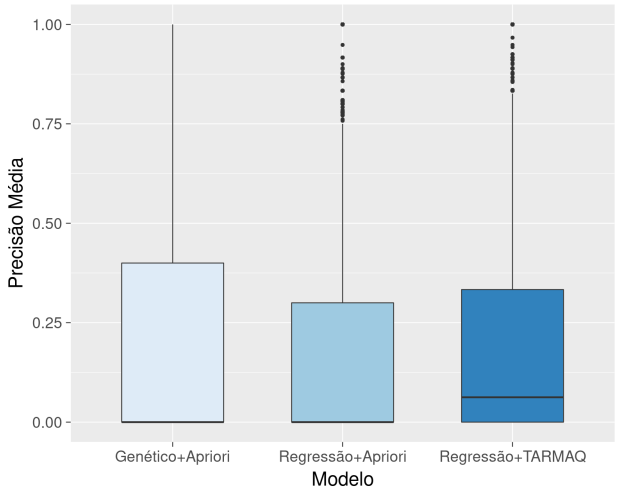

Figura 5.2. Distribuição Geral da Precisão Média em Cenários de Consulta Fixa

Os três modelos não puderam produzir resultados para um número significativo de consultas e, portanto, acabaram com uma alta porcentagem de AP com valor 0. O modelo Regressão+TARMAQ possui uma mediana maior que os demais modelos. Além disso, observando o terceiro quartil é possível notar que houveram consultas em que o modelo Genético+Apriori obteve recomendações com maior acurácia do que os outros dois modelos.

A distribuição geral da precisão média para cada um dos modelos, em cenários de avaliação com consultas variáveis, pode ser vista no diagrama de caixa da Figura 5.3. Assim como na distribuição dos modelos com consulta fixa, mostrada na Figura 5.2, os três modelos não puderam produzir resultados para um número significativo de consultas e, portanto, acabaram com uma alta porcentagem de AP com valor 0.

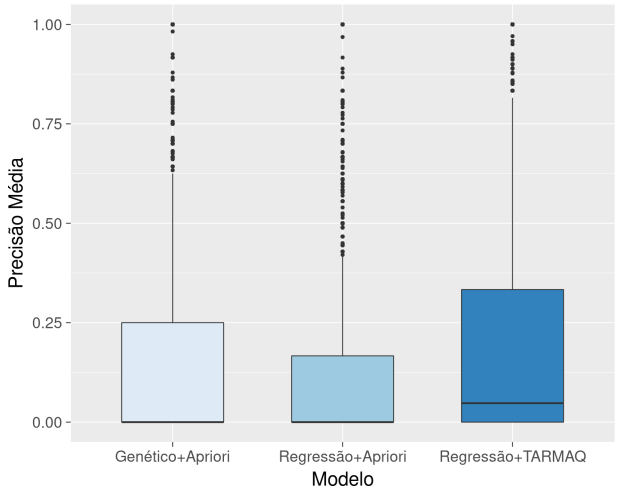

Figura 5.3. Distribuição Geral da Precisão Média em Cenários de Consulta Variável

O modelo Regressão+TARMAQ continuou com sua distribuição semelhante a obtida com consultas fixas no primeiro arquivo modificado. Porém, para os cenários com consultas de tamanho variável, o algoritmo TARMAQ conseguiu alcançar recomendações com maior acurácia

do que o modelo Genético+Apriori e o modelo Regressão+Apriori. Os resultados de precisão média das consultas dos demais modelos, no entanto, foram afetados pela consulta de tamanho variável.

A distribuição do tamanho das consultas variáveis, por projeto e tamanho do teste, encontra-se ilustrada no gráfico de caixa da Figura 5.4. É possível notar que, exceto para o projeto SwitchButton na configuração de 5% de teste, os projetos tiveram parte da distribuição do tamanho das consultas em 1 arquivo. Nos casos onde o tamanho da consulta variável foi igual a 1, essa consulta coincide com a consulta realizada no cenário de consulta fixa no primeiro arquivo.

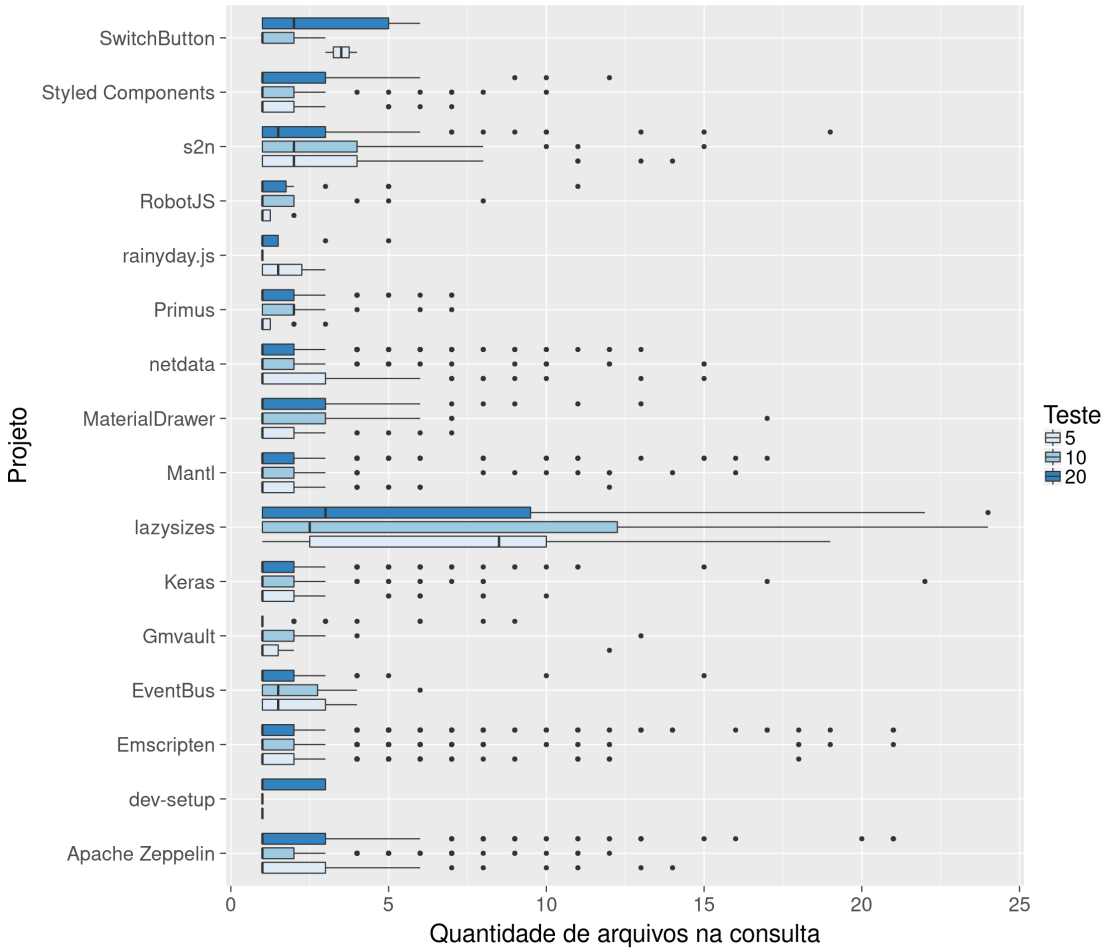

Figura 5.4. Distribuição da quantidade de arquivos nas consultas em Cenários de Consulta Variável para a $\mathbf{QP}_1$

Nas consultas de tamanho variável, por se tratarem de consultas que podem conter mais de um arquivo, os modelos Genético+Apriori e Regressão+Apriori tiveram uma diminuição nos valores de precisão média das consultas. Para as consultas de tamanho igual a 1, o valor de precisão média obtido foi igual ao obtido para a mesma consulta na configuração de consulta fixa, uma fez que a consulta é a mesma.

Nos casos em que a consulta avaliada era uma consulta invisível, o algoritmo Apriori usado pelos modelos Genético+Apriori e Regressão+Apriori não conseguiu gerar regras. Neste

caso, o algoritmo TARMAQ foi capaz de gerar regras referentes a um subconjunto da consulta, e por este motivo alcança resultados superiores para consultas com múltiplos arquivos.

O desempenho geral dos modelos de recomendação de mudança pode ser afetado por um conjunto de fatores, são eles: tamanho do histórico, algoritmo usado para minerar as regras de associação, limiares das medidas de interesse e fatores específicos de cada projeto de software. Por este motivo, os demais resultados serão apresentados projeto a projeto, discriminando as diferentes configurações de separação do conjunto de treinamento e teste, e também os dois tipos de consulta.

A Figura 5.5 ilustra a porcentagem de acertos do modelo Genético+Apriori em cada um dos cenários de avaliação. A porcentagem de acertos é referente a quantidade de arquivos esperados pelas consulta em cada cenário.

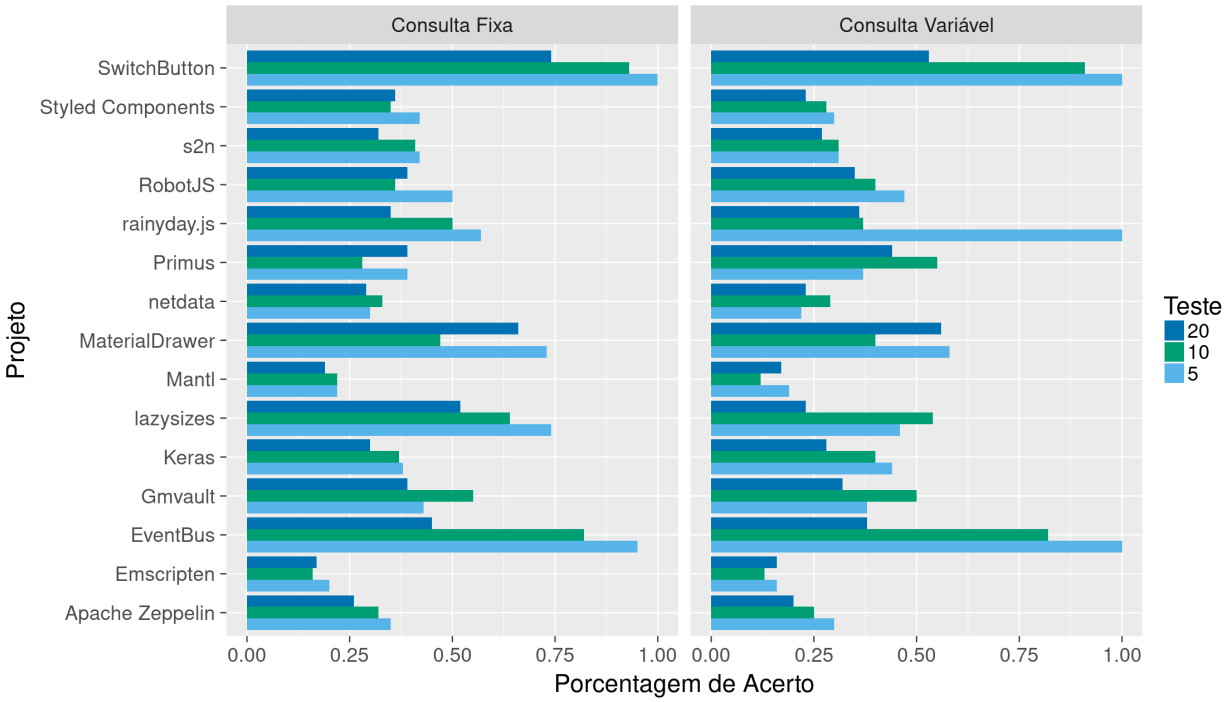

Figura 5.5. Porcentagem de acertos do modelo Genético+Apriori

Independente do tipo da consulta realizada, em 75% dos cenários com 5% de teste há uma maior quantidade de acertos por projeto. Para o projeto SwitchButton, em cenários de consulta fixa e variável com 5% de teste, todos os arquivos esperados pelas consultas foram recomendados pela lista de recomendações. O mesmo ocorre para os projetos EventBus e rainyday.js para os cenários de consulta variável com 5% de teste. Além disso, para o projeto rainyday.js com 5% de teste, ao mudar o tipo de consulta a porcentagem de acerto mudou de 57% para 100%.

A porcentagem de acerto do modelos Regressão+Apriori é ilustrada na Figura 5.6. Os resultados obtidos para este modelo são similares aos obtidos para o modelo Genético+Apriori.

Para o modelo Regressão+TARMAQ, é possível notar que há uma maior porcentagem de acerto em cenários de consulta variável. Isso se deve ao fato de que o algoritmo TARMAQ,

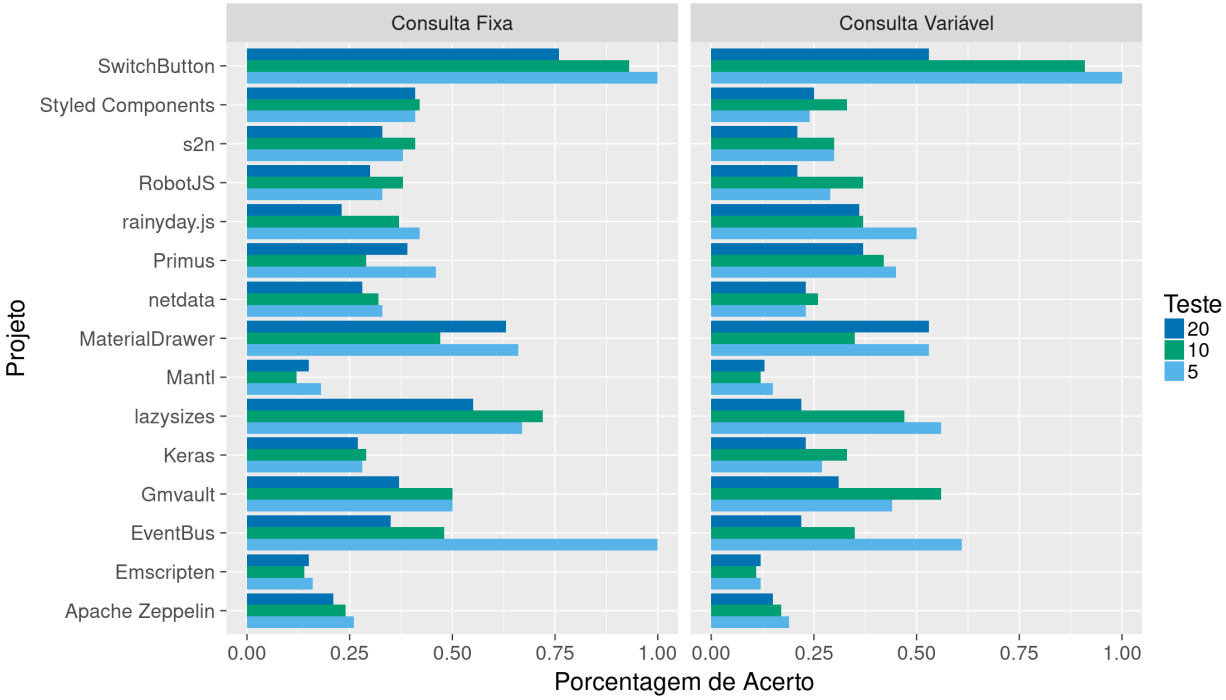

Figura 5.6. Porcentagem de acertos do modelo Regressão+Apriori

ao não encontrar regras que corresponde a todos os arquivos de uma consulta, gera regras que correspondem a sub-consultas, tendo assim a chance de uma maior porcentagem de acerto quando há mais de um arquivo na consulta. A porcentagem de acerto do modelo se encontra na Figura 5.7.

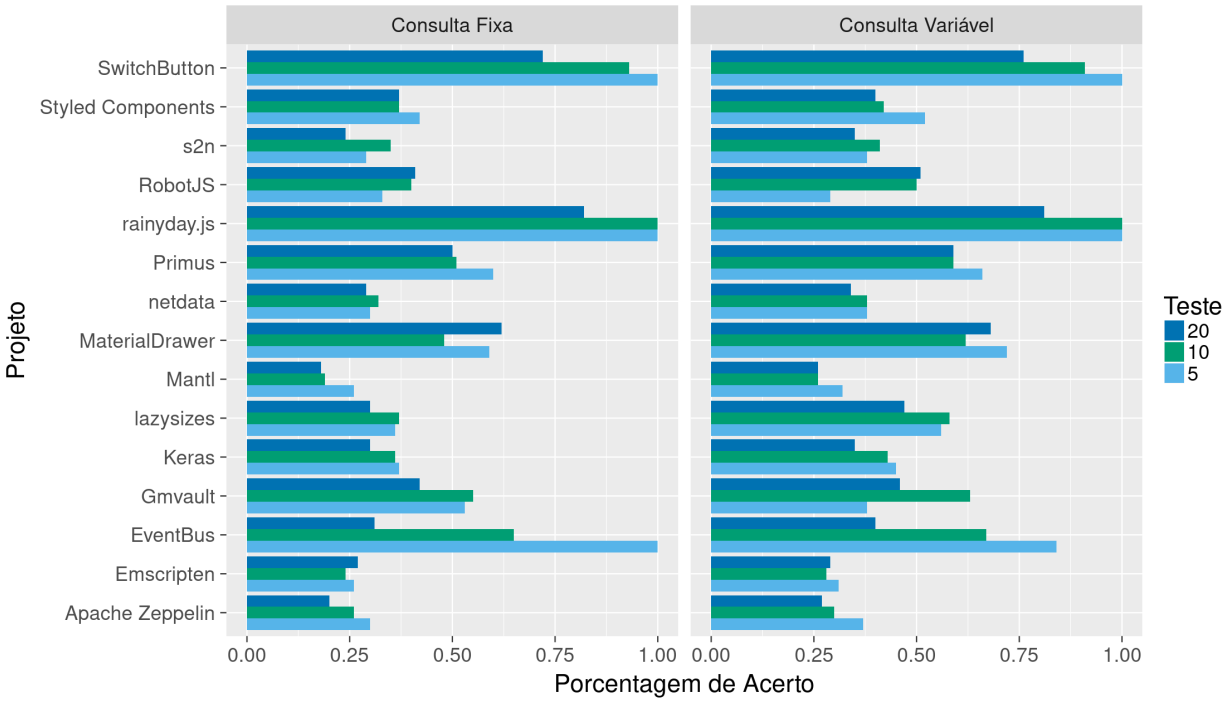

Figura 5.7. Porcentagem de acertos do modelo Regressão+TARMAQ

O desempenho geral dos modelos de recomendação é, então, refletido nos resultados mais específicos, resultantes da média das precisões médias das consultas executadas em cada cenário de avaliação dos 16 projetos analisados. A média das precisões médias, ao contrário da porcentagem de acerto mostrada anteriormente, leva em consideração a posição dos arquivos corretos na lista de recomendação.

Os valores de MAP para cada um dos cenários de avaliação da $\mathbf{QP}_1$ e para cada projeto estão descritos na Tabela 5.1.

Tabela 5.1. Valores de MAP obtidos para os cenários de avaliação da $\mathbf{QP}_1$

Projeto	Consulta Fixa									Consulta Variável								
	5%			10%			20%			5%			10%			20%		
	G+A	R+A	R+T	G+A	R+A	R+T	G+A	R+A	R+T	G+A	R+A	R+T	G+A	R+A	R+T	G+A	R+A	R+T
RobotJS	**0,875**	0, 800	0, 791	0, 498	0, 455	**0,500**	0, 569	0, 365	**0,583**	**0,875**	0, 800	0, 791	0, 458	0, 417	**0,589**	0, 516	0, 316	**0,569**
s2n	**0,348**	0, 216	0, 233	**0,304**	0, 228	0, 233	0, 169	0, 171	**0,186**	**0,264**	0, 161	0, 244	0, 275	0, 186	**0,289**	0, 141	0, 127	**0,196**
netdata	0, 258	0, 264	**0,275**	0, 264	0, 252	**0,285**	0, 231	0, 200	**0,239**	0, 213	0, 212	**0,282**	0, 224	0, 200	**0,275**	0, 200	0, 200	**0,254**
Emscripten	**0,180**	0, 138	0, 175	0, 127	0, 097	**0,133**	**0,147**	0, 100	0, 144	0, 159	0, 119	**0,189**	0, 100	0, 078	**0,140**	0, 130	0, 088	**0,145**
SwitchButton	**0,900**	0, 854	0, 854	**0,661**	0, 595	0, 575	**0,529**	0, 487	0, 491	0, 583	0, 583	**0,666**	**0,539**	0, 501	0, 500	0, 345	0, 322	**0,530**
EventBus	0, 897	**0,900**	0, 855	0, 590	0, 479	**0,600**	**0,374**	0, 326	0, 336	0, 775	0, 750	**0,792**	0, 461	0, 333	**0,495**	0, 312	0, 238	**0,323**
MaterialDrawer	**0,517**	0, 430	0, 444	**0,461**	0, 398	0, 432	**0,582**	0, 531	0, 551	**0,524**	0, 399	0, 475	0, 400	0, 347	**0,477**	0, 522	0, 484	**0,555**
Apache Zeppelin	**0,267**	0, 200	0, 228	**0,279**	0, 197	0, 256	**0,194**	0, 166	0, 169	**0,232**	0, 143	0, 230	0, 230	0, 144	**0,264**	0, 172	0, 139	**0,190**
rainyday.js	0, 250	0, 200	**0,308**	0, 200	0, 166	**0,286**	0, 197	0, 125	**0,347**	**0,375**	0, 166	0, 291	0, 166	0, 166	**0,286**	0, 125	0, 125	**0,200**
lazysizes	**0,686**	0, 630	0, 638	**0,545**	0, 531	0, 517	**0,375**	0, 345	0, 342	0, 447	0, 489	**0,685**	0, 320	0, 311	**0,539**	0, 152	0, 128	**0,333**
Primus	0, 348	0, 358	**0,425**	0, 279	0, 282	**0,379**	0, 283	0, 272	**0,319**	0, 285	0, 300	**0,400**	0, 363	0, 331	**0,432**	0, 257	0, 238	**0,324**
Styled Components	**0,311**	0, 199	0, 256	**0,386**	0, 343	0, 358	**0,300**	0, 285	**0,300**	**0,281**	0, 153	0, 200	0, 379	0, 333	**0,389**	0, 258	0, 232	**0,319**
dev-setup	0, 000	0, 000	**0,250**	0, 000	0, 000	**0,125**	0, 000	0, 000	**0,091**	0, 000	0, 000	**0,250**	0, 000	0, 000	**0,125**	0, 000	0, 000	**0,091**
Gmvault	**0,400**	0, 295	0, 300	**0,249**	0, 221	0, 230	0, 262	0, 247	**0,281**	**0,313**	0, 190	0, 190	**0,228**	0, 196	0, 213	0, 232	0, 220	**0,259**
Mantl	0, 161	0, 126	**0,176**	0, 174	0, 146	**0,185**	**0,191**	0, 170	0, 186	0, 133	0, 100	**0,187**	0, 156	0, 128	**0,193**	0, 177	0, 151	**0,196**
Keras	**0,412**	0, 351	0, 381	**0,385**	0, 348	0, 365	**0,296**	0, 270	0, 286	0, 365	0, 314	**0,392**	**0,372**	0, 337	0, 363	0, 262	0, 242	**0,285**

Na Tabela 5.1, os valores de MAP que se encontram em negrito correspondem ao maior valor obtido para determinado projeto em um cenário de avaliação. Por exemplo, para o projeto RobotJS no cenário de avaliação com 5% de teste e consulta fixa, o maior valor de MAP foi obtido com o modelo Genético+Apriori.

Analisando os cenários de avaliação com consulta fixa no primeiro arquivo, é possível observar que, para cenários com 5% de teste, 63% dos melhores resultados de MAP foram dados pelo modelo proposto neste trabalho, apenas um caso pelo modelo Regressão+Apriori. Para os cenários com 10% de teste, 50% dos melhores resultados foram obtidos com o modelo Genético+Apriori e os outros 50% pelo modelo Regressão+TARMAQ. Por fim, nos cenários com 20% de teste, 57% dos melhores resultados foram obtidos com o modelo Genético+Apriori e restante pelo modelo Regressão+TARMAQ.

Da mesma forma que no resultado do desempenho geral das precisões médias da Figura 5.3, o modelo Regressão+TARMAQ foi capaz de obter melhores resultados de MAP em relação aos outros modelos nos cenários mais específicos para cada projeto, com consultas de tamanho variável, exceto para cenários com 5% e 10% de teste em que, respectivamente, 44% e 17% dos melhores resultados foram obtidos com o modelo proposto.

Os valores de MAP listados na Tabela 5.1 em alguns casos possuem pouca diferença (em percentual) entre os modelos. É o caso do projeto RobotJS, no cenário de avaliação de 5% de teste e consulta fixa, entre o MAP obtido pelo modelo Genético+Apriori e o MAP obtido pelo

Regressão+Apriori há uma diferença de 0,075, cerca de 7%. Todavia, na prática essa diferença representa que além de serem recomendados todos os arquivos esperados, as recomendações estão sendo mostradas ao desenvolvedor de forma que os arquivos corretos (arquivos esperados pela consulta) aparecem primeiro na lista, facilitando a escolha do desenvolvedor ao realizar uma tarefa.

É possível notar que, para o projeto dev-setup os valores de MAP para os modelos Genético+Apriori e Regressão+Apriori são 0. Por se tratar de uma diferença de valores muito grande entre o resultado obtido nos dois modelos e no modelo Regressão+TARMAQ, foi realizada uma inspeção manual nas regras geradas em cada um dos três modelos. O algoritmo Apriori não foi capaz de gerar regras de associação para o projeto dev-setup. Isso se deve ao fato de que não terem sido encontradas regras relacionadas as consultas que tivessem uma confiança mínima de 0,10. Como o algoritmo TARMAQ não utiliza limiares de suporte e confiança para filtrar as regras geradas, ele foi capaz de gerar recomendações.

A seguir serão descritos em detalhe os fatores que influenciaram na obtenção dos valores de MAP apresentados. Na Seção 5.1.1 será apresentado o impacto do tamanho do histórico de mudanças usado para minerar as regras de associação. Na Seção 5.1.2 o impacto das medidas de interesse usadas, ou não, para gerar as recomendações serão apresentadas. Para finalizar as análises da $\mathbf{QP}_1$, a Seção 5.1.3 apresenta resultados dos métodos estatísticos de comparação de amostras.

5.1.1. Análise do Impacto do Tamanho do Histórico de Mudanças

Enquanto o AG utilizado pelo modelo Genético+Apriori conseguiu otimizar o tamanho do histórico de treinamento utilizado para gerar as regras de associação, a função de regressão usada pelos modelos Regressão+Apriori e Regressão+TARMAQ não foi capaz de prever o tamanho do histórico de treinamento para os projetos.

Como pode ser visto na Tabela 5.2, a quantidade de transações de treinamento recomendada pela função de regressão de Moonen et al. (2016b) é superior ao tamanho de histórico disponível em cada um dos projetos, para cada um dos cenários de tamanho de treinamento. A coluna Regressão apresenta o resultado da função de regressão para cada um dos projetos. Como os projetos não possuíam histórico suficiente para suprir a quantidade de transações recomendada pela regressão, as colunas subsequentes, 80%, 90% e 95%, apresentam a quantidade de transações usadas nos respectivos cenários de treinamento.

A Figura 5.8 mostra a porcentagem de transações usada pelo AG para otimizar as recomendações em cada um dos cenários. Dado que os modelos que são baseados na função de regressão sempre utilizaram todo o histórico de mudanças disponível como treinamento, pode-se observar que o AG possui a vantagem de se adaptar ao tamanho do projeto e ao mesmo tempo selecionar um conjunto de treinamento capaz de otimizar as recomendações de mudança geradas a partir dele.

Tabela 5.2. Resultado da Função de Regressão para os cenários de avaliação da $\mathbf{QP}_1$

Projeto	Regressão	80%	90%	95%
RobotJS	28085	378	425	449
s2n	22246	1190	1339	1413
netdata	24992	4092	4603	4859
Emscripten	26754	13325	14991	15824
SwitchButton	19468	105	118	125
EventBus	24581	332	374	395
MaterialDrawer	24480	1376	1548	1634
Apache Zeppelin	20839	2678	3013	3180
rainyday.js	28198	204	230	242
lazysizes	10718	484	544	574
Primus	27433	1192	1341	1415
Styled Components	24283	1180	1328	1401
dev-setup	29509	274	308	325
Gmvault	26084	868	977	1031
Mantl	25115	2610	2936	3099
Keras	26076	3027	3405	3594

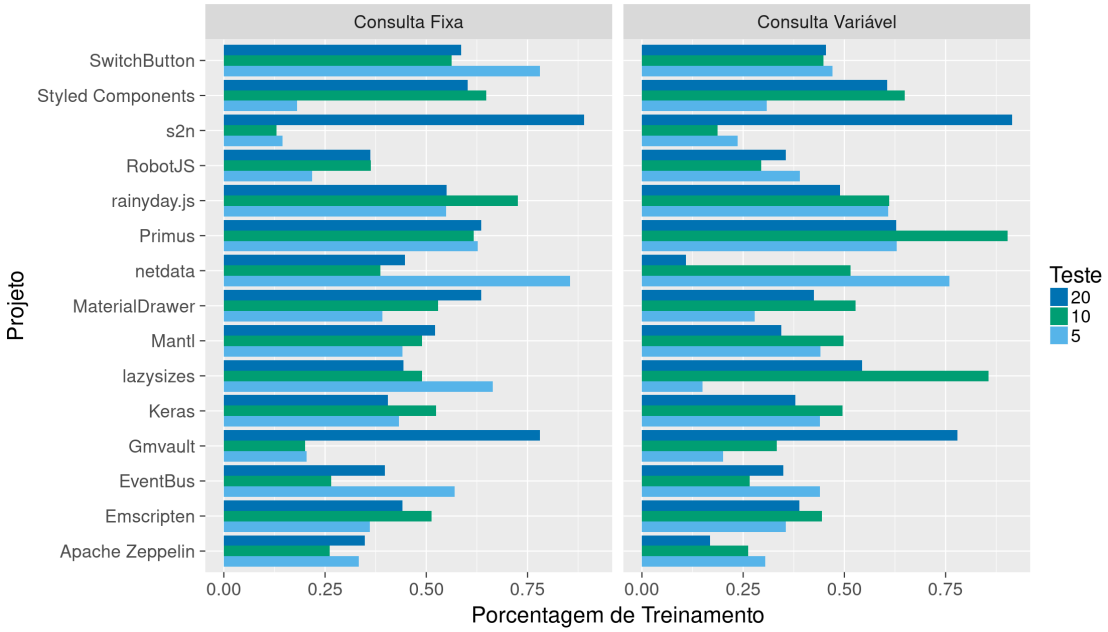

Figura 5.8. Porcentagem do histórico de mudanças selecionado pelo AG para os cenários de avaliação da $\mathbf{QP}_1$

5.1.2. Análise do Impacto dos Limiares das Medidas de Interesse

A distribuição dos valores de MAP obtidos com o modelo Regressão+Apriori, em cenários de consulta fixa no primeiro arquivo modificado pelo desenvolvedor, com diferentes configurações dos limiares de suporte e confiança pode ser vista na Figura 5.9. Essas distribuições representam o resultado obtido por um desenvolvedor ao tentar configurar manualmente limiares de suporte e confiança para o algoritmo Apriori.

Para projetos como SwitchButton, EventBus e RobotJS, no cenário de avaliação com 5% de teste e as demais transações no treinamento, a configuração manual pode levar a até 80% de diferença na acurácia das recomendações geradas pelo modelo. É possível perceber que, há uma variabilidade na acurácia das recomendações independente da quantidade de *commits* no projeto.

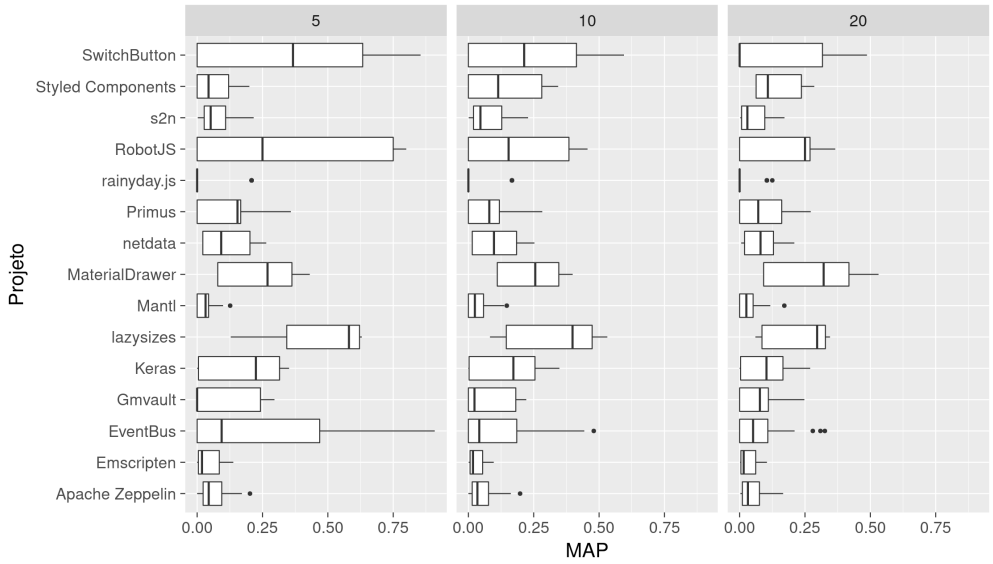

Figura 5.9. Distribuição do MAP obtido com o modelo Regressão+Apriori e consulta fixa no primeiro arquivo modificado

O mesmo ocorre na distribuição dos valores de MAP obtidos com o modelo Regressão+Apriori, em cenários de avaliação com consulta variável, com diferentes configurações dos limiares de suporte e confiança, como mostra o gráfico de caixa na Figura 5.10.

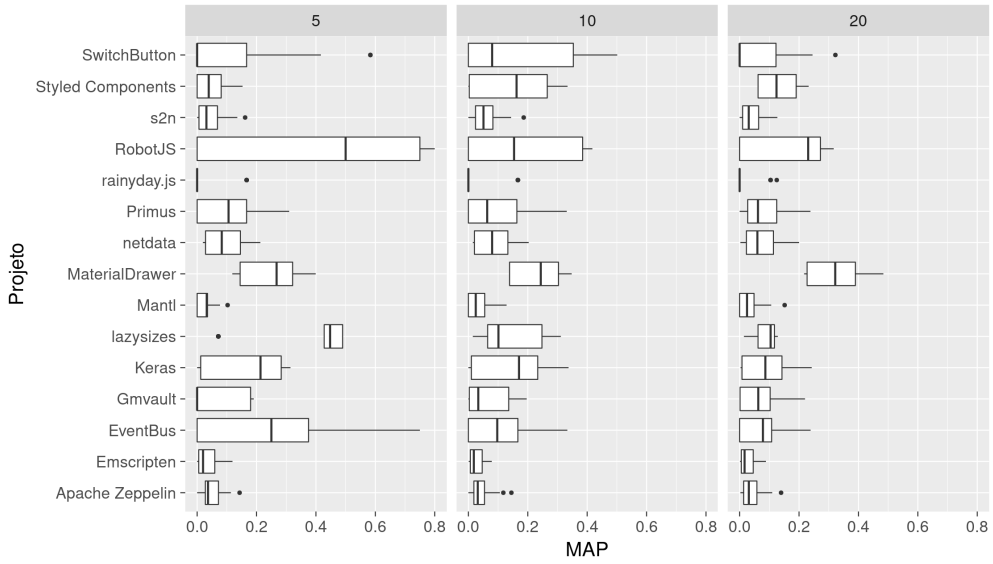

Figura 5.10. Distribuição do MAP obtido com o modelo Regressão+Apriori e consulta de tamanho variável

Essa grande diferença (até 80%) na acurácia dos resultados obtidos com os diferentes limiares de suporte e confiança para o modelo Regressão+Apriori, reforça a necessidade da experimentação de diferentes parâmetros antes da escolha do limiar a ser usado. Essa escolha é tratada de modo automático pelo modelo proposto. O AG se encarrega de selecionar os limiares das medidas de interesse que otimizam as recomendações.

Os limiares de suporte e confiança selecionados pelo AG, que otimizam as recomendações de mudança para cada um dos projetos em cenários de consulta fixa, estão descritos na Tabela 5.3.

Os limiares de frequência ou confiança em negrito representam os casos em que o AG selecionou um limiar menor do que o limiar usado no modelo Regressão+Apriori que obteve a maior acurácia das recomendações.

Tabela 5.3. Limiares de Suporte e Confiança selecionados pelo AG em cenários de consulta fixa para a $\mathbf{QP}_1$

Projeto	5%			10%			20%		
	Sup	Conf	Freq	Sup	Conf	Freq	Sup	Conf	Freq
RobotJS	0,0057	0,21	2	0,017	0,15	3	0,0072	0,13	2
s2n	0,0027	0,12	2	0,0029	0,13	2	0,0017	0,12	3
netdata	0,0002	0,13	2	0,0004	0,12	2	0,0011	0,12	**1**
Emscripten	0,0002	0,11	2	0,0001	0,12	2	0,0002	0,11	2
SwitchButton	0,0500	0,24	4	0,0501	0,27	4	0,0397	0,27	3
EventBus	0,0109	0,29	3	0,0085	0,14	2	0,0073	0,14	2
MaterialDrawer	0,0020	0,14	2	0,0012	0,15	2	0,0015	0,12	2
Apache Zeppelin	0,0009	0,12	2	0,0011	0,11	2	0,0014	0,12	2
rainyday.js	0,0179	0,11	4	0,0189	0,12	4	0,0185	0,11	3
lazysizes	0,0109	0,18	2	0,0042	0,24	3	0,0037	0,19	2
Primus	0,0011	0,13	2	0,0015	0,10	3	0,0013	0,12	2
Styled Components	0,0019	0,11	2	0,0033	0,14	4	0,0024	0,12	3
Gmvault	0,0074	0,13	3	0,0074	0,15	3	0,0013	0,11	2
Mantl	0,0007	0,12	2	0,0006	0,16	2	0,0010	0,12	2
Keras	0,0009	0,10	2	0,0006	0,12	2	0,0008	0,11	2

Independente do cenário de avaliação, a maior acurácia das recomendações com o modelo Regressão+Apriori foi obtida ao usar um limiar de frequência das regras de 2 ocorrências e um limiar de confiança de $0,10$. Em apenas um dos casos, para o projeto s2n no cenário 20% com consulta fixa, o AG selecionou um limiar de frequência de regra menor do que o utilizado pelo modelo Regressão+Apriori. Entretanto, o limiar de confiança selecionado pelo AG foi 2% maior. Para este caso a diferença no MAP entre os dois modelos foi de $0,002$, sendo a maior acurácia obtida pelo modelo Regressão+Apriori.

Para os demais projetos, no cenário de avaliação com consulta fixa no primeiro arquivo, as medidas de interesse encontradas pelo AG são iguais ou superiores as medidas definidas para o modelo Regressão+Apriori que obtiveram a melhor acurácia das recomendações geradas.

Os limiares de suporte e confiança selecionados pelo AG, que otimizam as recomendações de mudança para cada um dos projetos em cenários de consulta de tamanho variável, estão descritos na Tabela 5.4.

Assim como para os cenários de consulta fixa, em todos os casos, o limiar de confiança selecionado pelo AG foi igual ou superior ao usado pelo modelo Regressão+Apriori, e também em apenas um dos casos, para o projeto s2n no cenário 10%, o AG selecionou um limiar de frequência de regra menor do que o utilizado pelo modelo Regressão+Apriori, entretanto o limiar de confiança

Tabela 5.4. Limiares de Suporte e Confiança selecionados pelo AG em cenários de consulta variável para a $\mathbf{QP}_1$

Projeto	5%			10%			20%		
	Sup	Conf	Freq	Sup	Conf	Freq	Sup	Conf	Freq
RobotJS	0,0088	0,22	2	0,0179	0,15	4	0,0139	0,12	3
s2n	0,0031	0,15	2	0,0026	0,11	**1**	0,0009	0,12	2
netdata	0,0004	0,12	3	0,0005	0,13	2	0,0005	0,15	2
Emscripten	0,0002	0,10	2	0,0001	0,11	2	0,0002	0,10	2
SwitchButton	0,0145	0,47	2	0,0283	0,25	3	0,0297	0,28	3
EventBus	0,0041	0,31	2	0,0092	0,30	2	0,0067	0,13	2
MaterialDrawer	0,0014	0,11	2	0,0012	0,10	2	0,0011	0,17	2
Apache Zeppelin	0,0007	0,16	2	0,0009	0,11	2	0,0011	0,12	2
rainyday.js	0,0065	0,14	2	0,0200	0,19	4	0,0219	0,15	3
lazysizes	0,0148	0,74	7	0,0037	0,19	2	0,0037	0,19	2
Primus	0,0011	0,12	2	0,0012	0,11	2	0,0011	0,12	2
Styled Components	0,0026	0,11	2	0,0031	0,23	4	0,0025	0,12	3
Gmvault	0,0129	0,18	4	0,0045	0,15	2	0,0013	0,12	2
Mantl	0,0007	0,16	2	0,0007	0,21	2	0,0007	0,13	2
Keras	0,0006	0,14	2	0,0004	0,13	2	0,0006	0,11	2

selecionado pelo AG foi 1% maior. Apesar da diferença no limiar de frequência, o MAP obtido pelo modelo Genético+Apriori foi $0,089$ maior.

Algo importante de mencionar é que o limiar de suporte e confiança selecionados pelo modelo Genético+Apriori são calculados a partir do tamanho do treinamento, também selecionado pelo AG, e tanto as medidas de interesse quanto o tamanho do treinamento selecionado para gerar as regras de associação são os valores que otimizam as recomendações geradas.

5.1.3. Comparação Estatística dos Modelos

Com o objetivo de verificar se o modelo de recomendação proposto possui maior acurácia do que o modelo Regressão+Apriori e o modelo Regressão+TARMAQ, foram formuladas e testadas as seguintes hipóteses:

$\mathbf{H}_0$ A distribuição de precisão média gerada por cada modelo é a mesma.

$\mathbf{H}_1$ A distribuição de precisão média gerada por cada modelo é diferente.

Para testar as hipóteses, utilizamos o teste de Kruskal-Wallis, um método não-paramétrico para comparação de três ou mais distribuições independentes. A Tabela 5.5 reporta o resultado obtido por esse teste. Assim, 78% dos diferentes cenários produzem um $p - value < 0,05$, e, portanto, rejeitamos $\mathbf{H}_0$ nestes casos e concluímos que há uma diferença significativa entre os modelos.

Não foram testados 10 cenários de avaliação pois, estes cenários possuem menos de cinco arquivos no conjunto de teste e na literatura o recomendado é que utilize distribuições com mais de cinco amostras (COHEN, 1988). Os teste não realizados foram marcados na tabela.

Tabela 5.5. Resultado do teste de Kruskal-Wallis para a $\mathbf{QP}_1$

Projeto	Consulta Fixa			Consulta Variável		
	5%	10%	20%	5%	10%	20%
RobotJS	-	$p < 0,05$	$p < 0,05$	-	0,211	$p < 0,05$
s2n	0,094	0,055	$p < 0,05$	0,148	$p < 0,05$	$p < 0,05$
netdata	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$
Emscripten	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$
SwitchButton	-	-	0,170	-	-	0,537
EventBus	0,240	0,221	$p < 0,05$	$p < 0,05$	0,182	$p < 0,05$
MaterialDrawer	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$
Apache Zeppelin	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$
rainyday.js	-	-	0,320	-	-	0,220
lazysizes	0,072	$p < 0,05$	$p < 0,05$	0,457	0,149	$p < 0,05$
Primus	0,201	$p < 0,05$	$p < 0,05$	0,138	$p < 0,05$	$p < 0,05$
Styled Components	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$
Gmvault	0,270	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$
Mantl	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$
Keras	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$

Uma vez que aceitamos $\mathbf{H}_1$, podemos aplicar um teste *post-hoc* para realmente identificar quais modelos, e seus respectivos cenários de avaliação, possuem uma distribuição diferente. Usamos o teste de Wilcoxon, do inglês *Wilcoxon signed-ranks test*, um método não-paramétrico para comparação de duas amostras pareadas, a fim de testar as seguintes hipóteses:

$\mathbf{H}_0^{AGvRA}$ A distribuição da precisão média gerada pelo modelo Genético+Apriori é menor que a gerada por Regressão+Apriori.

$\mathbf{H}_1^{AGvRA}$ A distribuição da precisão média gerada pelo modelo Genético+Apriori é maior que a gerada por Regressão+Apriori.

$\mathbf{H}_0^{AGvRT}$ A distribuição da precisão média gerada pelo modelo Genético+Apriori é menor que a gerada por Regressão+TARMAQ.

$\mathbf{H}_1^{AGvRT}$ A distribuição da precisão média gerada pelo modelo Genético+Apriori é maior que a gerada por Regressão+TARMAQ.

Para analisar os resultados do teste de Wilcoxon, executado par a par entre o modelo Genético+Apriori e os demais modelos, além do respectivo valor de p, do inglês *p-value*, foram considerados o Tamanho do Efeito, do inglês *Effect Size*. O tamanho do efeito é definido como o grau em que o fenômeno está presente na população (COHEN, 1988), e é avaliado utilizando os limiares previstos em Romano e Kromrey (2006), como pode ser visto na Tabela 5.6. Os resultados negativos do tamanho do efeito indicam um desempenho inferior para a primeira distribuição avaliada.

O teste de Wilcoxon foi realizado em R, utilizando a função *wilcox.test* com: *alternative = 'greater', paired = TRUE*. O tamanho de efeito também foi calculado em R, usando a função *cliff.delta* do pacote *effsize*.

Tabela 5.6. Limiares do tamanho de efeito

Limiar	Tamanho do Efeito
$\|d\| < 0,147$	insignificante
$\|d\| < 0,33$	pequeno
$\|d\| < 0,474$	médio
$\|d\| \geq 0,474$	grande

O resultado do teste de Wilcoxon e do tamanho do efeito, para o par Genético+Apriori e Regressão+Apriori, para cada um dos cenários de avaliação de cada projeto está descrito na Tabela 5.7. Valores em negrito na tabela representam os testes em que o $p - value < 0,05$, e para estes rejeitamos $\mathbf{H}_0$ e aceitamos que a distribuição da precisão média gerada pelo modelo Genético+Apriori é maior.

Tabela 5.7. Resultado do teste Wilcoxon e Tamanho de efeito: Genético+Apriori e Regressão+Apriori para a $\mathbf{QP}_1$

Projeto	Consulta Fixa			Consulta Variável		
	5%	10%	20%	5%	10%	20%
RobotJS	-	insignificante (0, 041)	**pequeno (0, 254)**	-	insignificante (0, 071)	**pequeno (0, 227)**
s2n	**pequeno (0, 24)**	insignificante (0, 117)	insignificante (−0, 003)	**insignificante (0, 121)**	insignificante (0, 082)	insignificante (0, 031)
netdata	insignificante (0, 0004)	insignificante (0, 029)	insignificante (−0, 006)	insignificante (0, 002)	insignificante (0, 024)	insignificante (0, 010)
Emscripten	**insignificante (0, 07)**	**insignificante (0, 037)**	**insignificante (0, 053)**	**insignificante (0, 057)**	**insignificante (0, 037)**	**insignificante (0, 051)**
SwitchButton	-	-	insignificante (0, 029)	-	-	insignificante (0, 017)
EventBus	insignificante (−0, 015)	**pequeno (0, 179)**	insignificante (0, 081)	insignificante (0, 062)	**pequeno (0, 219)**	**insignificante (0, 137)**
MaterialDrawer	**insignificante (0, 086)**	**insignificante (0, 075)**	**insignificante (0, 056)**	**insignificante (0, 129)**	**insignificante (0, 062)**	**insignificante (0, 049)**
Apache Zeppelin	**insignificante (0, 104)**	**pequeno (0, 155)**	**insignificante (0, 055)**	**pequeno (0, 148)**	**pequeno (0, 150)**	**insignificante (0, 058)**
rainyday.js	-	-	pequeno (0, 25)	-	-	insignificante (0, 031)
lazysizes	insignificante (0, 061)	insignificante (0, 01)	**insignificante (0, 015)**	insignificante (−0, 045)	insignificante (0, 022)	insignificante (0, 017)
Primus	insignificante (−0, 013)	insignificante (−0, 009)	insignificante (0, 016)	insignificante (−0, 02)	insignificante (0, 086)	insignificante (0, 03)
Styled Components	**pequeno (0, 237)**	insignificante (0, 065)	insignificante (0, 032)	**pequeno (0, 203)**	insignificante (0, 058)	insignificante (0, 027)
Gmvault	insignificante (0, 066)	insignificante (0, 0006)	**insignificante (0, 025)**	insignificante (0, 04)	insignificante (−0, 013)	**insignificante (0, 01)**
Mantl	insignificante (0, 011)	**insignificante (0, 04)**	insignificante (0, 029)	insignificante (−0, 014)	insignificante (0, 01)	**insignificante (0, 028)**
Keras	**insignificante (0, 108)**	**insignificante (0, 071)**	**insignificante (0, 044)**	**insignificante (0, 080)**	**insignificante (0, 053)**	**insignificante (0, 04)**

Comparando estatisticamente a distribuição da precisão média gerada pelo modelo Genético+Apriori e pelo modelo Regressão+Apriori, é possível observar que em 74% dos casos em que rejeitamos $\mathbf{H}_0$, o tamanho do efeito é insignificante. Considerando cenários com consulta fixa no primeiro arquivo e também com consulta de tamanho variável, para o projeto RobotJS com 20% de teste, a distribuição dos valores de precisão média gerada pelo Genético+Apriori é maior do que para o modelo Regressão+Apriori, e o tamanho do efeito é pequeno. Neste caso usar o modelo Genético+Apriori para gerar as recomendações é melhor.

O mesmo pode ser visto para o projeto Styled Components com 5% de teste e para os projetos EventBus e Apache Zeppelin com 10% de teste. No caso do projeto Apache Zeppelin com cenário de consulta variável, há dois casos em que o modelo com melhor resultado é o modelo proposto: 5% e 10% de teste. Para o projeto s2n, apenas no cenário de consulta fixa com 5% de teste o modelo Genético+Apriori tem uma distribuição de AP maior e com tamanho de efeito pequeno em relação ao modelo Regressão+Apriori.

O resultado do teste de Wilcoxon e do tamanho do efeito, para o par Genético+Apriori e Regressão+TARMAQ, para cada um dos cenários de avaliação de cada projeto está descrito na Tabela 5.8. Valores em negrito na tabela representam os testes em que o $p - value < 0,05$, e

para estes rejeitamos $\mathbf{H}_0$ e aceitamos que a distribuição da precisão média gerada pelo modelo Genético+Apriori é maior.

Tabela 5.8. Resultado do teste Wilcoxon e Tamanho de efeito: Genético+Apriori e Regressão+TARMAQ para a $\mathbf{QP}_1$

Projeto	Consulta Fixa			Consulta Variável		
	5%	10%	20%	5%	10%	20%
RobotJS	-	insignificante (−0, 029)	insignificante (−0, 026)	-	pequeno (−0, 195)	insignificante (−0, 094)
s2n	**pequeno (0, 191)**	insignificante (0, 041)	insignificante (−0, 036)	insignificante (−0, 093)	pequeno (−0, 149)	pequeno (−0, 164)
netdata	insignificante (−0, 042)	insignificante (−0, 05)	insignificante (−0, 086)	pequeno (−0, 228)	pequeno (−0, 156)	insignificante (−0, 139)
Emscripten	insignificante (−0, 098)	insignificante (−0, 115)	insignificante (−0, 1)	pequeno (−0, 2)	pequeno (−0, 182)	pequeno (−0, 153)
SwitchButton	-	-	insignificante (0, 017)	-	-	pequeno (−0, 31)
EventBus	insignificante (0, 125)	insignificante (−0, 015)	**insignificante (0, 022)**	insignificante (−0, 031)	insignificante (−0, 08)	insignificante (−0, 04)
MaterialDrawer	insignificante (0, 053)	insignificante (0, 011)	insignificante (0, 021)	insignificante (0, 022)	pequeno (−0, 164)	insignificante (−0, 055)
Apache Zeppelin	insignificante (0, 038)	**insignificante (0, 035)**	insignificante (0, 019)	insignificante (−0, 057)	insignificante (−0, 107)	insignificante (−0, 113)
rainyday.js	-	-	pequeno (−0, 312)	-	-	pequeno (−0, 25)
lazysizes	insignificante (0, 045)	insignificante (0, 011)	**insignificante (0, 01)**	médio (−0, 336)	pequeno (−0, 273)	pequeno (−0, 313)
Primus	pequeno (−0, 159)	pequeno (−0, 201)	insignificante (−0, 109)	pequeno (−0, 243)	insignificante (−0, 137)	pequeno (−0, 196)
Styled Components	insignificante (0, 127)	insignificante (0, 041)	insignificante (−0, 012)	insignificante (−0, 093)	insignificante (−0, 075)	pequeno (−0, 169)
Gmvault	insignificante (0, 057)	insignificante (−0, 032)	insignificante (−0, 05)	insignificante (0, 044)	insignificante (−0, 08)	insignificante (−0, 091)
Mantl	insignificante (−0, 092)	insignificante (−0, 067)	insignificante (−0, 026)	pequeno (−0, 192)	pequeno (−0, 183)	insignificante (−0, 115)
Keras	insignificante (0, 03)	insignificante (0, 008)	insignificante (−0, 004)	insignificante (0, 067)	insignificante (−0, 029)	insignificante (−0, 077)

Dentre os oitenta testes estatísticos realizados, apenas 5% (quatro testes) possuem o $p - value < 0,05$ e rejeitamos $\mathbf{H}_0$. Destes quatro testes, apenas um possui um tamanho de efeito pequeno, os demais são insignificantes.

Como já havíamos analisado anteriormente, comparando os valores de MAP dos diferentes cenários de avaliação, o modelo Regressão+TARMAQ possui um desempenho superior ao se tratar de consultas de tamanho variável. O mesmo pode ser visto no resultado do tamanho de efeito. Exceto um teste estatístico realizado para os cenários de consulta variável não resultou em efeito negativo. O que significa que para consultas de tamanho variável, o modelo Genético+Apriori apresenta um desempenho inferior ao modelo Regressão+TARMAQ.

5.2. Avaliação da Estabilidade dos Modelos de Recomendação de Mudanças Conjuntas

A fim de responder a $\mathbf{QP}_2$, foram definidos quatro configurações de separação do conjunto de treinamento e teste, sendo que o treinamento é fixo em 70% do histórico mais antigo e o teste é composto por incrementalmente 5%, 10%, 20% e 30% do histórico mais recente a partir do treinamento, como descrito anteriormente na Seção 4.3.1. As duas configurações de consulta (fixas no primeiro arquivo modificado e de tamanho variável) também são usadas e combinadas com as quatro configurações de separação de treinamento e teste, gerando então 8 cenários de avaliação diferentes.

O objetivo é investigar o quanto novas mudanças em um projeto deterioram a estabilidade de cada um dos três modelos de recomendação de mudança. Fixar um período de treinamento e inserir novas mudanças no período de teste a cada nova execução do modelo permite analisar o impacto de novas mudanças e como cada um dos modelos se comporta mediante a deterioração.

Assim como na $\mathbf{QP}_1$, o método para responder a questão envolve uma etapa de pré-processamento dos dados. As transações contendo apenas um arquivo no conjunto de teste não podem ser usadas como consulta, uma vez que não é possível fazer a separação da transação em consulta e resultado esperado. Assim, a Figura 5.11 mostra a quantidade de transações removidas do teste após a separação dos conjuntos de treinamento e teste, no pré-processamento dos dados.

Figura 5.11. Porcentagem de transações removidas no conjunto de teste para a $\mathbf{QP}_2$

A distribuição do tamanho das consultas variáveis, por projeto e tamanho do teste, encontra-se ilustrada no gráfico de caixa da Figura 5.12. O projeto dev-setup, para o cenário de 5% de teste, possui a distribuição dos tamanhos das consultas em 0. Isso ocorre pois, após dividir os conjuntos de treinamento e teste e remover as transações com apenas um arquivo, não sobraram transações no conjunto de teste. Então, para esse projeto não foi possível analisar os resultados usando 5% de teste com treinamento fixo em 70% das transações mais antigas do projeto.

De forma semelhante ao que foi realizado na $\mathbf{QP}_1$, os resultados de MAP mostrados para o modelo Genético+Apriori são os resultados encontrados pelo AG ao final da evolução de 100 gerações para uma população de 200 indivíduos. Para o modelo Regressão+Apriori, que foi executado variando a frequência das regras de associação entre 2 a 20 e os limiares de confiança em 0, 10, 0, 5 e 0, 9, os resultados considerados são os que obtiveram a maior acurácia para cada cenário de avaliação. Por fim, para o modelo Regressão+TARMAQ, que foi executado uma vez para cada cenário de avaliação, todos seus resultados são considerados e listados.

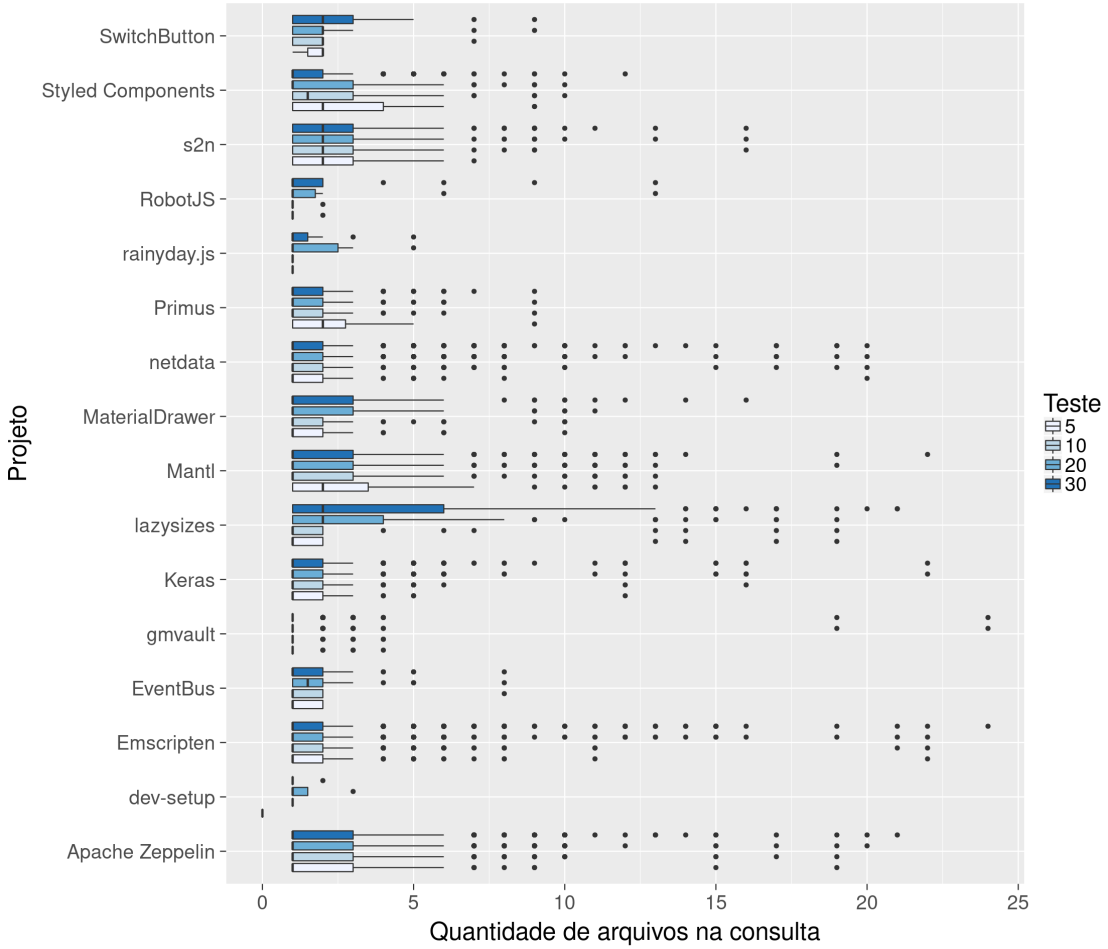

Figura 5.12. Distribuição da quantidade de arquivos nas consultas em Cenários de Consulta Variável para a $\mathbf{QP}_2$

5.2.1. Análise do Impacto da Inserção de novas Mudanças

A análise do impacto da inserção de novas mudanças em um projeto de software é realizada ao avaliar o comportamento dos modelos de recomendação a medida em que novas transações são inseridas no conjunto de teste.

A acurácia obtida em cada uma das quatro configurações de teste, para o modelo Genético+Apriori com consulta fixa no primeiro arquivo modificado é mostrado na Figura 5.13. Nela é possível observar que dos dezesseis projetos analisados, seis perderam acurácia a medida que o teste foi incrementado: Emscriptem, Keras, lazysizes, netdata, rainyday.js e s2n.

Os resultados de MAP obtidos com o modelo Genético+Apriori em cenários de consulta com tamanho variável são mostrados na Figura 5.14. Dentre os dezesseis projetos, novamente seis perderam acurácia a medida que o teste foi incrementado, são eles: Apache Zeppelin, Emscriptem, Keras, lazysizes, netdata e SwitchButton.

Na Figura 5.15 são mostrados os valores de MAP obtidos em cada uma das quatro configurações de teste, para o modelo Regressão+Apriori com consulta fixa no primeiro arquivo

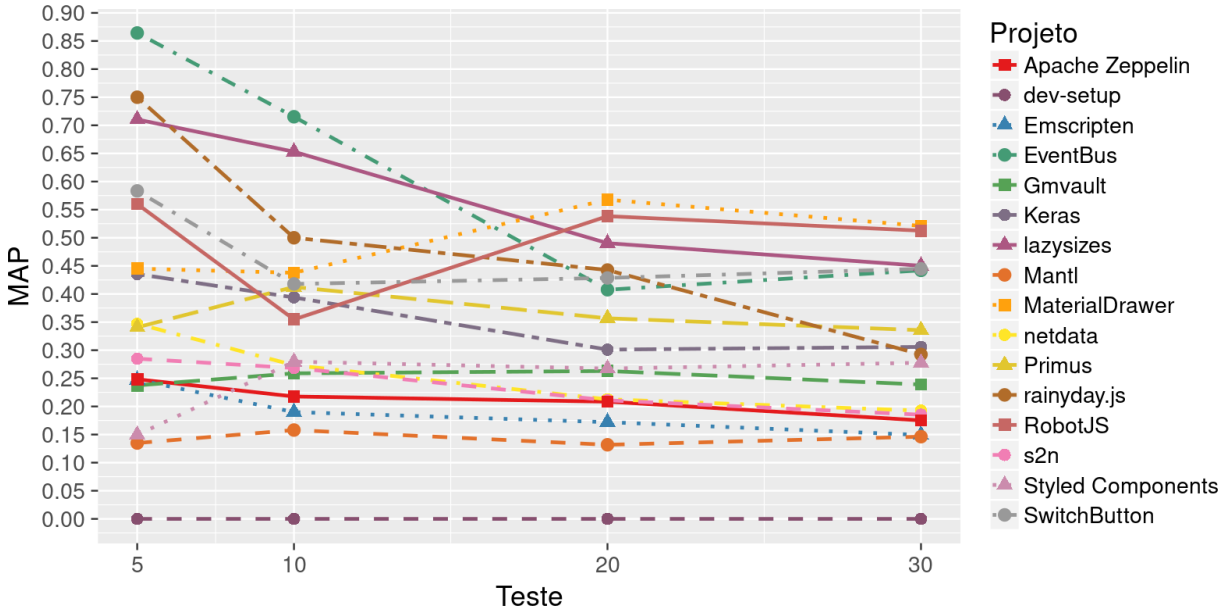

Figura 5.13. Valores de MAP com Consulta Fixa para o modelo Genético+Apriori

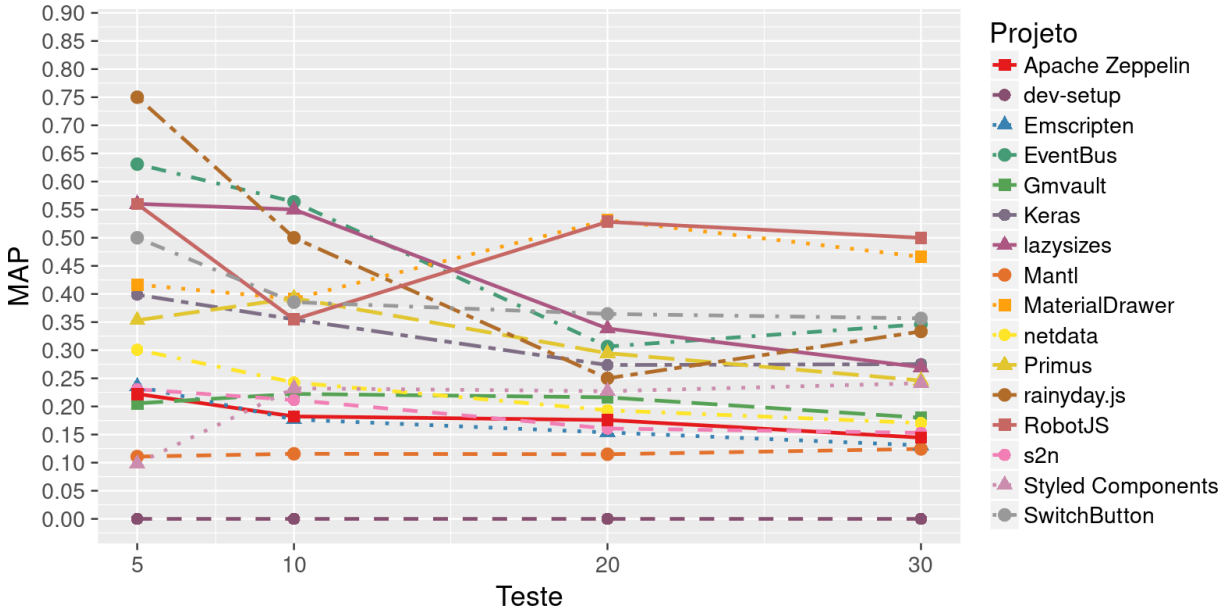

Figura 5.14. Valores de MAP com Consulta Variável ara o modelo Genético+Apriori

modificado. Cinco projetos perderam acurácia das recomendações a medida que novas transações foram inseridas no teste, são eles: Apache Zeppelin, Emscriptem, Keras, lazysizes e netdata.

Os valores de MAP obtidos em cada uma das configurações de teste, para o modelo Regressão+Apriori com consulta de tamanho variável são mostrados na Figura 5.16. Cinco projetos perderam acurácia das recomendações a medida que novas transações passam a serem testadas, são eles: Emscriptem, Keras, lazysizes, netdata e SwitchButton.

Os resultados da acurácia das recomendações obtidos em cada uma das quatro configurações de teste, para o modelo Regressão+TARMAQ com consulta fixa no primeiro arquivo modificado é mostrado na Figura 5.17. Cinco projetos perderam acurácia das recomendações a

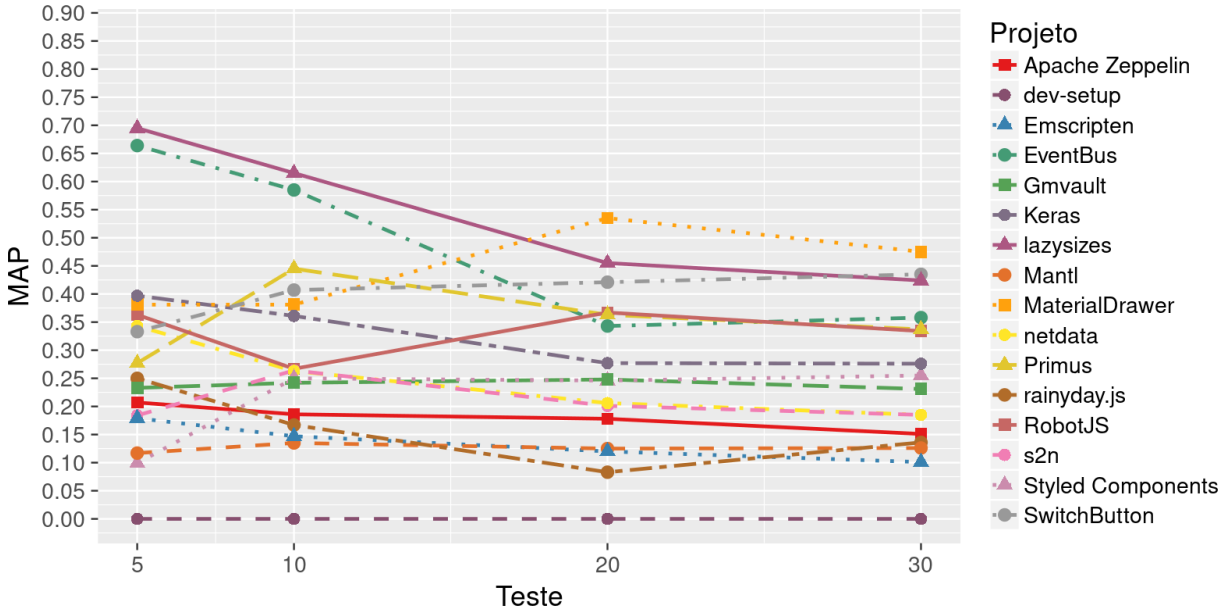

Figura 5.15. Valores de MAP com Consulta Fixa para o modelo Regressão+Apriori

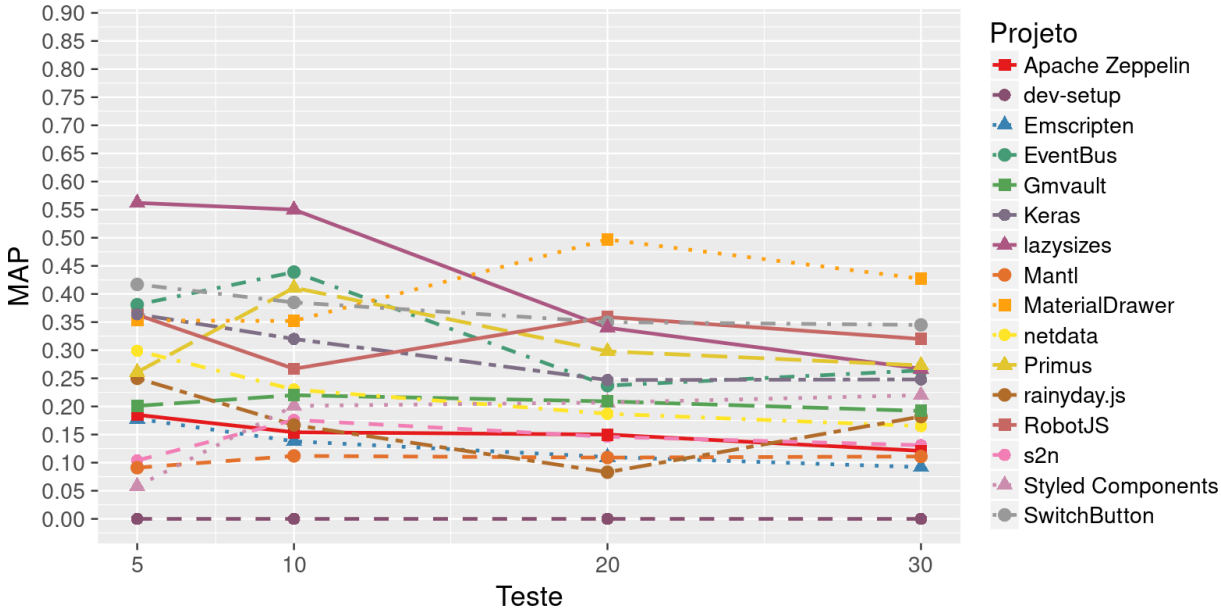

Figura 5.16. Valores de MAP com Consulta Variável para o modelo Regressão+Apriori

medida que novas transações passam a serem testadas, são eles: Emscriptem, Keras, lazysizes, netdata e rainyday.js.

Para o projeto dev-setup, apenas com o modelo Regressão+TARMAQ resultados de MAP foram obtidos. Neste caso, o que ocorreu foi o contrário de deterioração, pois a medida em que o conjunto de teste aumentou a acurácia das recomendações também ficou maior.

Os valores de MAP obtidos em cada uma das configurações de teste, para o modelo Regressão+TARMAQ com consulta de tamanho variável é mostrado na Figura 5.18. Cinco projetos perderam acurácia das recomendações a medida que novas transações passam a serem testadas, são eles: Emscriptem, Keras, lazysizes, netdata e s2n.

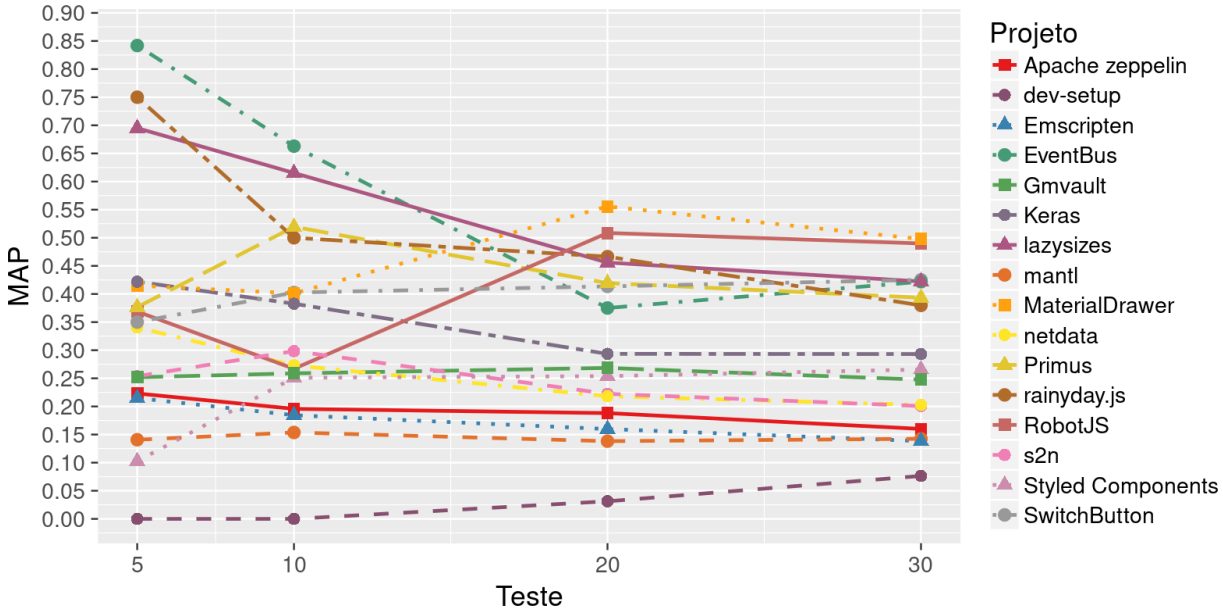

Figura 5.17. Valores de MAP com Consulta Fixa para o modelo Regressão+TARMAQ

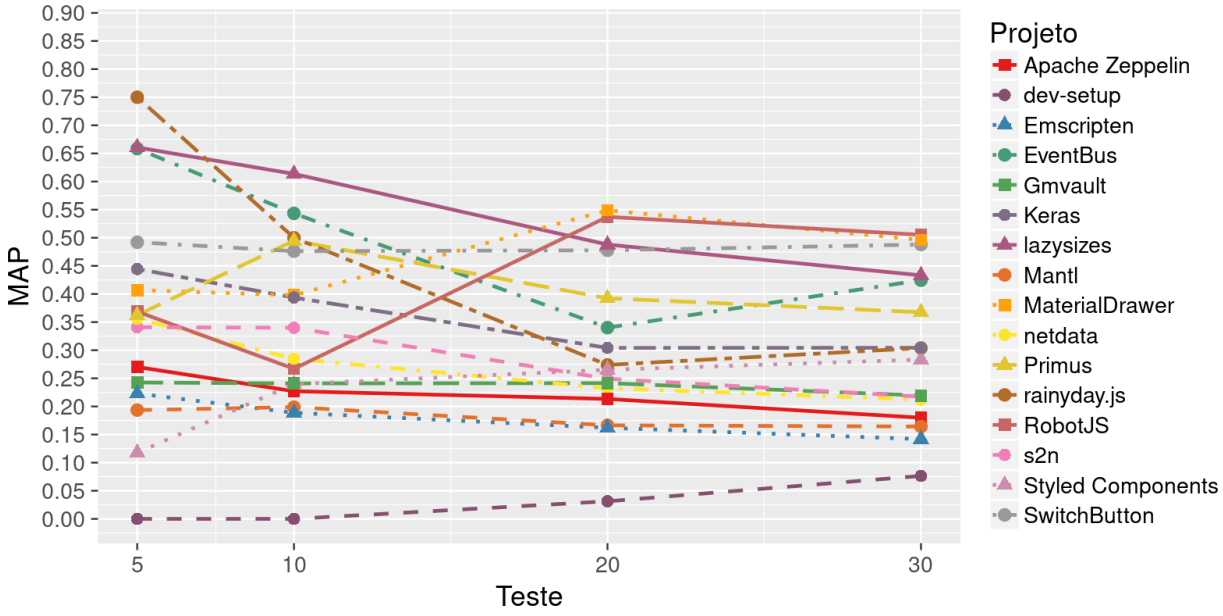

Figura 5.18. Valores de MAP com Consulta Variável para o modelo Regressão+TARMAQ

5.2.2. Análise do Impacto das Medidas de Interesse e Histórico de Mudanças na Deterioração dos Modelos

Enquanto o AG utilizado pelo modelo Genético+Apriori conseguiu otimizar o tamanho do histórico de treinamento utilizado para gerar as regras de associação, como pode ser visto na Figura 5.19, a função de regressão usada pelos modelos Regressão+Apriori e Regressão+TARMAQ não foi capaz de prever o tamanho do histórico de treinamento para os projetos, como pode ser visto na Tabela 5.9.

A quantidade de transações de treinamento recomendada pela função de regressão de Moonen et al. (2016b) é superior ao tamanho de histórico disponível em cada um dos projetos e por este motivo a coluna 70% apresenta a quantidade de transações usadas treinamento.

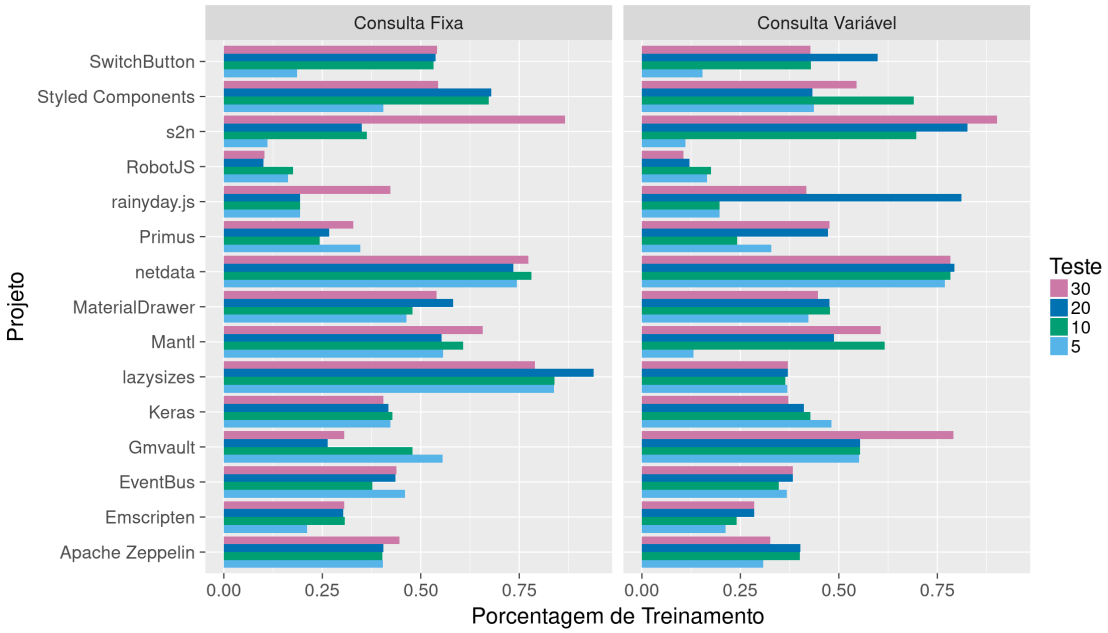

Figura 5.19. Porcentagem do histórico de mudanças selecionado pelo AG para os cenários de avaliação da $\mathbf{QP}_1$

Tabela 5.9. Resultado da Função de Regressão para os cenários de avaliação da $\mathbf{QP}_2$

Projeto	Regressão	70%
RobotJS	28085	331
s2n	22246	1041
netdata	24992	3580
Emscripten	26754	11660
SwitchButton	19468	92
EventBus	24581	291
MaterialDrawer	24480	1204
Apache Zeppelin	20839	2343
rainyday.js	28198	179
lazysizes	10718	423
Primus	27433	1043
Styled Components	24283	1033
dev-setup	29509	240
Gmvault	26084	760
Mantl	25115	2284
Keras	26076	2649

Os limiares de suporte e confiança selecionados pelo AG, que otimizam as recomendações de mudança para cada um dos projetos em cenários de consulta fixa, estão descritos na Tabela 5.10.

Em todos os cenários de avaliação, a maior acurácia das recomendações com o modelo Regressão+Apriori foi obtida ao usar um limiar de frequência das regras de 2 ocorrências e um limiar de confiança de 0, 10. Os limiares de suporte e confiança selecionados pelo AG, no cenário

Tabela 5.10. Limiares de Suporte e Confiança selecionados pelo AG em cenários de consulta fixa para a $\mathbf{QP}_2$

Projeto	5%			10%			20%			30%		
	Sup	Conf	Freq	Sup	Conf	Freq	Sup	Conf	Freq	Sup	Conf	Freq
RobotJS	0,017	0,35	2	0,0159	0,28	2	0,0191	0,22	2	0,0232	0,19	2
s2n	0,0056	0,22	2	0,0014	0,13	2	0,001	0,12	2	0,002	0,12	3
netdata	0,0007	0,12	3	0,0004	0,12	2	0,0003	0,12	2	0,0003	0,12	2
Emscripten	0,0003	0,10	2	0,0003	0,11	2	0,0003	0,11	2	0,0003	0,11	2
SwitchButton	0,0911	0,34	2	0,0796	0,26	4	0,044	0,25	3	0,0463	0,26	3
EventBus	0,0087	0,24	2	0,0094	0,14	2	0,0084	0,12	2	0,0083	0,12	2
MaterialDrawer	0,0037	0,25	3	0,003	0,11	3	0,0016	0,11	2	0,0015	0,12	2
Apache Zeppelin	0,0013	0,12	2	0,001	0,11	2	0,0007	0,12	2	0,0012	0,11	2
rainyday.js	0,0225	0,34	2	0,0225	0,34	2	0,0065	0,27	2	0,0111	0,13	2
lazysizes	0,0032	0,32	2	0,0127	0,18	3	0,0062	0,18	2	0,0062	0,18	2
Primus	0,0021	0,12	2	0,0038	0,12	2	0,0017	0,11	2	0,0019	0,1	2
Styled Components	0,004	0,2	3	0,0027	0,11	3	0,0043	0,11	3	0,0032	0,12	3
Gmvault	0,0047	0,13	3	0,0046	0,10	3	0,0044	0,1	3	0,0033	0,11	3
Mantl	0,002	0,19	2	0,0007	0,12	2	0,0009	0,12	2	0,0007	0,12	2
Keras	0,0007	0,11	2	0,0008	0,11	2	0,0009	0,11	2	0,001	0,1	2

de avaliação com consulta fixa no primeiro arquivo, são iguais ou superiores as medidas definidas para o modelo Regressão+Apriori que obtiveram a melhor acurácia das recomendações geradas.

Os limiares de suporte e confiança selecionados pelo AG, que otimizam as recomendações de mudança para cada um dos projetos em cenários de consulta fixa, estão descritos na Tabela 5.11. Os limiares de frequência ou confiança em negrito representam os casos em que o AG selecionou um limiar menor do que o limiar usado no modelo Regressão+Apriori que obteve a maior acurácia das recomendações.

Tabela 5.11. Limiares de Suporte e Confiança selecionados pelo AG em cenários de consulta variável para a $\mathbf{QP}_2$

Projeto	5%			10%			20%			30%		
	Sup	Conf	Freq	Sup	Conf	Freq	Sup	Conf	Freq	Sup	Conf	Freq
RobotJS	0,0145	0,35	2	0,011	0,27	2	0,0258	0,21	2	0,0184	0,2	2
s2n	0,0032	0,32	**1**	0,0023	0,25	2	0,0022	0,17	2	0,0011	0,11	2
netdata	0,0007	0,14	3	0,0004	0,12	2	0,0004	0,12	2	0,0003	0,14	2
Emscripten	0,0003	0,1	2	0,0003	0,12	2	0,0003	0,10	2	0,0003	0,1	2
SwitchButton	0,0559	0,67	2	0,057	0,28	4	0,0347	0,26	3	0,0318	0,28	3
EventBus	0,0071	0,24	2	0,0083	0,14	2	0,0072	0,14	2	0,0075	0,14	2
MaterialDrawer	0,0035	0,21	3	0,0034	0,10	3	0,0013	0,14	2	0,0012	0,15	2
Apache Zeppelin	0,0009	0,12	2	0,001	0,11	2	0,0009	0,12	2	0,0009	0,12	2
rainyday.js	0,0200	0,33	2	0,0200	0,33	2	0,02	0,33	2	0,0112	0,13	2
lazysizes	0,0051	0,27	3	0,0025	0,23	2	0,005	0,19	3	0,0029	0,29	2
Primus	0,0024	0,14	2	0,0035	0,12	2	0,0022	0,12	2	0,0026	0,11	2
Styled Components	0,0024	0,18	2	0,0021	0,12	2	0,0023	0,10	3	0,003	0,13	3
Gmvault	0,0042	0,13	3	0,0048	0,11	3	0,0049	0,11	2	0,0003	0,1	**1**
Mantl	0,0008	0,19	2	0,0007	0,14	2	0,0008	0,12	2	0,0007	0,14	2
Keras	0,0008	0,12	2	0,0007	0,11	2	0,0007	0,11	2	0,0007	0,11	2

Em apenas dois projetos o limiar de frequência das regras selecionado pelo AG, para otimizar as recomendações de mudanças, é menor do que o usado no modelo Regressão+Apriori. Estes casos são o projeto s2n no cenário de 5% de teste e o projeto Gmvault no cenário de 30% de teste.

5.2.3. Comparação Estatística dos Modelos

Com o objetivo de verificar se os três modelos se comportam da mesma maneira a medida que novas mudanças são inseridas conjunto de teste, foram testadas as seguintes hipóteses:
$\mathbf{H}_0$ A distribuição de precisão média gerada por cada modelo é a mesma.
$\mathbf{H}_1$ A distribuição de precisão média gerada por cada modelo é diferente.

Essas hipóteses foram testadas a cada inserção de novas mudanças, utilizamos o teste de Kruskal-Wallis. A Tabela 5.12 reporta o resultado obtido por esse teste. Assim, 82% dos diferentes cenários produzem um $p - value < 0,05$, e, portanto, rejeitamos $\mathbf{H}_0$ nestes casos e concluímos que há uma diferença significativa entre os modelos a medida em que novas mudanças são inseridas no projeto.

Não foram testados 10 cenários de avaliação, pois estes cenários possuem menos de cinco arquivos no conjunto de teste e na literatura o recomendado é que utilize distribuições com mais de cinco amostras (COHEN, 1988). Os teste não realizados foram marcados na tabela.

Tabela 5.12. Resultado do teste de Kruskal-Wallis para a $\mathbf{QP}_2$

Projeto	Consulta Fixa				Consulta Variável			
	5%	10%	20%	30%	5%	10%	20%	30%
RobotJS	-	0,670	$p < 0,05$	$p < 0,05$	-	0,670	$p < 0,05$	$p < 0,05$
s2n	0,176	$p < 0,05$	$p < 0,05$	$p < 0,05$	0,423	$p < 0,05$	$p < 0,05$	$p < 0,05$
netdata	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$
Emscripten	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$
SwitchButton	-	0,220	0,099	0,131	-	0,155	0,189	0,142
EventBus	-	0,118	$p < 0,05$	$p < 0,05$	-	0,152	$p < 0,05$	$p < 0,05$
MaterialDrawer	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$
Apache Zeppelin	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$
rainyday.js	-	-	0,415	0,283	-	-	0,171	0,084
lazysizes	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$
Primus	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	0,124	0,141	$p < 0,05$	$p < 0,05$
Styled Components	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$
Gmvault	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	0,082	$p < 0,05$	$p < 0,05$	$p < 0,05$
Mantl	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$
Keras	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$	$p < 0,05$

Uma vez que aceitamos $\mathbf{H}_1$, podemos aplicar um teste *post-hoc* para realmente identificar quais modelos, e seus respectivos cenários de avaliação, possuem uma distribuição diferente. Da mesma forma como fizemos na $\mathbf{QP}_1$, usamos o teste de Wilcoxon para comparação de duas amostras pareadas, a fim de testar as seguintes hipóteses:
$\mathbf{H}_0^{AGvRA}$ A distribuição da precisão média gerada pelo modelo Genético+Apriori é menor que a gerada por Regressão+Apriori.

$\mathbf{H}_1^{AGvRA}$ A distribuição da precisão média gerada pelo modelo Genético+Apriori é maior que a gerada por Regressão+Apriori.
$\mathbf{H}_0^{AGvRT}$ A distribuição da precisão média gerada pelo modelo Genético+Apriori é menor que a gerada por Regressão+TARMAQ.
$\mathbf{H}_1^{AGvRT}$ A distribuição da precisão média gerada pelo modelo Genético+Apriori é maior que a gerada por Regressão+TARMAQ.

Para analisar os resultados do teste de Wilcoxon, executado par a par entre o modelo Genético+Apriori e os demais modelos, além do valor de p, foram considerados o tamanho do efeito, do inglês *Effect Size*. O tamanho do efeito é avaliado utilizando os limiares previstos em Romano e Kromrey (2006) como pode ser visto na Tabela 5.6, apresentada anteriormente. Os resultados negativos do tamanho do efeito indicam um desempenho inferior para a primeira distribuição avaliada.

O resultado do teste de Wilcoxon e do tamanho do efeito, para o par Genético+Apriori e Regressão+Apriori, para cada um dos cenários de avaliação de cada projeto está descrito na Tabela 5.13. Valores em negrito na tabela representam os testes em que o $p - value < 0,05$, e para estes rejeitamos $\mathbf{H}_0$ e aceitamos que a distribuição da precisão média gerada pelo modelo Genético+Apriori é maior.

Tabela 5.13. Resultado do teste Wilcoxon e Tamanho de efeito: Genético+Apriori e Regressão+Apriori para a $\mathbf{QP}_2$

Projeto	Consulta Fixa				Consulta Variável			
	5%	10%	20%	30%	5%	10%	20%	30%
RobotJS	-	pequeno (−0, 16)	pequeno (0, 154)	**pequeno (0, 182)**	-	pequeno (−0, 16)	pequeno (0, 157)	**pequeno (0, 18)**
s2n	**insignificante (0, 086)**	insignificante (−0, 009)	insignificante (0, 012)	insignificante (0, 001)	**pequeno (0, 201)**	insignificante (0, 016)	insignificante (−0, 032)	**insignificante (0, 033)**
netdata	insignificante (−0, 004)	insignificante (0, 008)	insignificante (0, 007)	insignificante (0, 011)	insignificante (−0, 004)	**insignificante (0, 013)**	insignificante (0, 006)	insignificante (0, 005)
Emscripten	**insignificante (0, 115)**	**insignificante (0, 067)**	**insignificante (0, 081)**	**insignificante (0, 076)**	**insignificante (0, 080)**	**insignificante (0, 044)**	**pequeno (0, 056)**	**pequeno (0, 05)**
SwitchButton	-	insignificante (0, 046)	insignificante (0, 006)	insignificante (0, 031)	-	insignificante (0)	insignificante (0, 013)	insignificante (0, 011)
EventBus	-	**pequeno (0, 217)**	**insignificante (0, 094)**	**insignificante (0, 128)**	-	pequeno (0, 182)	**insignificante (0, 116)**	**insignificante (0, 134)**
MaterialDrawer	**insignificante (0, 104)**	**insignificante (0, 066)**	**insignificante (0, 031)**	**insignificante (0, 048)**	**insignificante (0, 097)**	**insignificante (0, 044)**	**insignificante (0, 029)**	**insignificante (0, 04)**
Apache Zeppelin	**insignificante (0, 131)**	**insignificante (0, 089)**	**insignificante (0, 099)**	**insignificante (0, 09)**	**insignificante (0, 075)**	**insignificante (0, 066)**	**insignificante (0, 072)**	**insignificante (0, 069)**
rainyday.js	-	-	grande (0, 583)	pequeno (0, 289)	-	-	pequeno (0, 194)	pequeno (0, 247)
lazysizes	insignificante (0, 046)	insignificante (0, 063)	**insignificante (0, 048)**	**insignificante (0, 022)**	insignificante (−0, 01)	insignificante (−0, 003)	insignificante (−0, 0006)	insignificante (0, 007)
Primus	insignificante (0, 098)	insignificante (−0, 062)	insignificante (0, 006)	insignificante (0, 013)	**insignificante (0, 126)**	insignificante (−0, 019)	insignificante (−0, 014)	insignificante (0, 034)
Styled Components	**insignificante (0, 016)**	insignificante (0, 019)	insignificante (0, 017)	**insignificante (0, 017)**	insignificante (0, 027)	**insignificante (0, 02)**	insignificante (0, 018)	insignificante (0, 016)
Gmvault	insignificante (0, 009)	insignificante (0, 053)	insignificante (0, 023)	insignificante (0, 014)	insignificante (−0, 002)	insignificante (0, 015)	insignificante (−0, 01)	insignificante (−0, 015)
Mantl	insignificante (−0, 026)	**insignificante (0, 066)**	insignificante (0, 034)	**insignificante (0, 077)**	**insignificante (0, 037)**	insignificante (0, 023)	insignificante (0, 019)	**insignificante (0, 036)**
Keras	**insignificante (0, 044)**	**insignificante (0, 042)**	**insignificante (0, 033)**	**insignificante (0, 04)**	insignificante (0, 039)	**insignificante (0, 055)**	**insignificante (0, 053)**	**insignificante (0, 05)**

Comparando estatisticamente a distribuição da precisão média gerada pelo modelo Genético+Apriori e pelo modelo Regressão+Apriori, é possível observar que em 88% dos casos em que rejeitamos $\mathbf{H}_0$, o tamanho do efeito é insignificante. Considerando cenários com consulta fixa no primeiro arquivo, para o projeto RobotJS com 30% de teste e para o projeto EventBus com 10% de teste a distribuição dos valores de precisão média gerada pelo Genético+Apriori é maior do que para o modelo Regressão+Apriori, e o tamanho do efeito é pequeno. Neste caso usar o modelo Genético+Apriori para gerar as recomendações é melhor.

Considerando cenários com consulta de tamanho variável, para os projetos RobotJS e Emscripten com 30% de teste a distribuição dos valores de precisão média gerada pelo Genético+Apriori é maior do que para o modelo Regressão+Apriori, e o tamanho do efeito é pequeno. O mesmo acontece com o projeto Emscripten no cenário de 20% de teste e para o projeto s2n com 5% de teste. Nestes casos usar o modelo Genético+Apriori para gerar as recomendações é melhor.

O resultado do teste de Wilcoxon e do tamanho do efeito, para o par Genético+Apriori e Regressão+TARMAQ, para cada um dos cenários de avaliação de cada projeto está descrito na Tabela 5.14. Valores em negrito na tabela representam os testes em que o $p - value < 0,05$, e

para estes rejeitamos $\mathbf{H}_0$ e aceitamos que a distribuição da precisão média gerada pelo modelo Genético+Apriori é maior.

Tabela 5.14. Resultado do teste Wilcoxon e Tamanho de efeito: Genético+Apriori e Regressão+TARMAQ para a $\mathbf{QP}_2$

Projeto	Consulta Fixa				Consulta Variável			
	5%	10%	20%	30%	5%	10%	20%	30%
RobotJS	-	insignificante (0, 037)	insignificante (0, 014)	**insignificante (0, 004)**	-	insignificante (0, 037)	insignificante (−0, 057)	insignificante (−0, 065)
s2n	insignificante (−0, 029)	insignificante (−0, 087)	insignificante (−0, 024)	insignificante (−0, 024)	pequeno (−0, 198)	pequeno (−0, 3)	pequeno (−0, 243)	pequeno (−0, 17)
netdata	insignificante (0, 001)	insignificante (−0, 014)	insignificante (−0, 034)	insignificante (−0, 04)	insignificante (−0, 11)	insignificante (−0, 126)	insignificante (−0, 129)	insignificante (−0, 137)
Emscripten	**insignificante (−0, 006)**	insignificante (−0, 072)	insignificante (−0, 07)	insignificante (−0, 072)	insignificante (−0, 059)	insignificante (−0, 126)	insignificante (−0, 133)	insignificante (−0, 137)
SwitchButton	-	insignificante (0, 031)	insignificante (0, 038)	insignificante (0, 045)	-	pequeno (−0, 218)	pequeno (−0, 224)	pequeno (−0, 265)
EventBus	-	insignificante (0, 1)	insignificante (0, 016)	insignificante (0, 021)	-	insignificante (0, 044)	insignificante (−0, 097)	insignificante (−0, 139)
MaterialDrawer	**insignificante (0, 052)**	**insignificante (0, 032)**	**insignificante (0, 007)**	**insignificante (0, 014)**	insignificante (0, 025)	insignificante (−0, 034)	insignificante (−0, 039)	insignificante (−0, 071)
Apache Zeppelin	insignificante (0, 064)	**insignificante (0, 039)**	**insignificante (0, 046)**	**insignificante (0, 047)**	pequeno (−0, 167)	pequeno (−0, 181)	insignificante (−0, 132)	insignificante (−0, 122)
rainyday.js	-	-	insignificante (−0, 055)	pequeno (−0, 214)	-	-	insignificante (−0, 111)	insignificante (0, 041)
lazysizes	insignificante (0, 046)	insignificante (0, 063)	**insignificante (0, 043)**	**insignificante (0, 026)**	pequeno (−0, 157)	insignificante (−0, 095)	pequeno (−0, 228)	pequeno (−0, 287)
Primus	insignificante (−0, 141)	pequeno (−0, 216)	insignificante (−0, 1)	insignificante (−0, 1)	insignificante (−0, 052)	pequeno (−0, 175)	pequeno (−0, 195)	pequeno (−0, 263)
Styled Components	insignificante (0, 007)	insignificante (0, 015)	insignificante (−0, 002)	insignificante (−0, 005)	pequeno (−0, 197)	insignificante (−0, 11)	insignificante (−0, 14)	insignificante (−0, 14)
Gmvault	insignificante (−0, 048)	insignificante (0, 014)	insignificante (−0, 02)	insignificante (−0, 034)	insignificante (−0, 152)	insignificante (−0, 073)	insignificante (−0, 097)	insignificante (−0, 114)
Mantl	insignificante (−0, 136)	insignificante (−0, 004)	insignificante (−0, 033)	insignificante (−0, 013)	pequeno (−0, 2)	pequeno (−0, 171)	insignificante (−0, 138)	insignificante (−0, 13)
Keras	insignificante (−0, 01)	insignificante (−0, 012)	insignificante (−0, 016)	insignificante (−0, 013)	insignificante (−0, 112)	insignificante (−0, 1)	insignificante (−0, 09)	insignificante (−0, 95)

Dentre os cento e dez testes estatísticos realizados, apenas 10% (onze testes) possuem o $p - value < 0,05$ e rejeitamos $\mathbf{H}_0$. Destes onze testes, todos possuem um tamanho de efeito insignificante. Como já havíamos analisado anteriormente na $\mathbf{QP}_1$, o modelo Regressão+TARMAQ possui um desempenho superior ao se tratar de consultas de tamanho variável.

O mesmo pode ser visto no resultado do tamanho de efeito. Exceto dois testes estatísticos realizados para os cenários de consulta variável não resultaram em um tamanho de efeito negativo. O que significa que, para consultas de tamanho variável, o modelo Genético+Apriori apresenta um desempenho inferior ao apresentado pelo modelo Regressão+TARMAQ.

5.3. Considerações Finais

Foram apresentamos os resultados obtidos com os experimentos realizados com o modelo proposto, Genético+Apriori, para otimização de recomendações de mudanças conjuntas. Os resultados, em termos de acurácia do modelo, foram comparados estatisticamente com os resultados dos modelos

Regressão+Apriori e Regressão+TARMAQ para 16 projetos de código aberto em diferentes cenários de avaliação com o objetivo de responder duas questões de pesquisa.

A $\mathbf{QP}_1$ avaliou o desempenho do modelo proposto neste trabalho, Genético+Apriori, em encontrar o tamanho do conjunto de treinamento e os limiares de suporte e confiança capazes de otimizar as recomendações de mudanças para um projeto de software. O modelo proposto neste trabalho, Genético+Apriori, apresentou resultados superiores em cenários de consulta fixa no primeiro arquivo modificado e variável, em relação a acurácia do modelo Regressão+Apriori.

O modelo Regressão+TARMAQ, no entanto, alcançou melhores resultados nos cenários com consulta de tamanho variável em relação ao modelo Genético+Apriori. Desta maneira, o melhor modelo para ser aplicado em cenários de prevenção de erros na prática (consultas de tamanho variável) é o modelo Regressão+TARMAQ.

Na $\mathbf{QP}_2$ foi verificado se a variação do tamanho do conjunto de teste de 5% até 30%, com o conjunto de treinamento fixo em 70% das transações mais antigas, afeta a estabilidade dos modelos fazendo com que eles percam acurácia. Essa avaliação simulou o processo de inserção de novas mudanças nos projetos.

A inserção de novas mudanças implicou, em média, na deterioração de 37, 5% dos projetos tanto nos cenários de consulta fixa, quanto em cenários de consulta de tamanho variável para cada um dos modelos de recomendação. No entanto, o modelo Genético+Apriori foi o modelo que mais apresentou projetos que se deterioram. Enquanto os modelos Regressão+Apriori e Regressão+TARMAQ apresentaram cinco projetos por cenário de avaliação, o modelo proposto apresentou seis projetos que sofreram deterioração na acurácia.

Estatisticamente, conforme novas mudanças são inseridas no conjunto de teste, o Genético+Apriori apresentou acurácia superior ao modelo Regressão+Apriori, independentemente do cenário de avaliação analisado. O modelo Regressão+TARMAQ apresentou maior acurácia em relação ao modelo proposto, em cenários com consulta de tamanho variável.

Como inserir novas mudanças deteriora a estabilidade dos modelos de recomendação, o esforço de encontrar as medidas de interesse se torna constante se não houver uma técnica que automatize essa escolha. Escolher valores arbitrários para a configuração das medidas de interesse e para o tamanho do histórico de mudanças, com o objetivo de otimizar as recomendações geradas, requer muito esforço. Assim, o modelo Genético+Apriori é a melhor escolha para cenários de consulta fixa no primeiro arquivo (em relação aos demais modelos) e a melhor escolha para cenário de consulta variável se comparado ao modelo Regressão+Apriori, mas não ao Regressão+TARMAQ.

Capítulo 6

Ameaças à validade

Neste capítulo serão discutidas as principais ameaças à validade do modelo proposto, assim como do experimento realizado. Essas ameaças se resumem a hábitos de *commits* dos desenvolvedores, amostragem aleatória, generalização dos projetos escolhidos e a escolha de parâmetros de configuração do AG.

Hábitos de Commit. O modelo Genético+Apriori proposto neste trabalho está baseado na mineração de regras de associação a partir do histórico de modificações registradas no controle de versão de um projeto de software. Entretanto, desenvolvedores podem realizar *commits* que envolvam arquivos não relacionados e que portanto não deveriam ser modificados conjuntamente (OLIVA; GEROSA, 2015). Esse conceito é conhecido na literatura como *tangled changes* (HERZIG; ZELLER, 2013) e pode influenciar a geração das regras de associação usadas pelos modelos de recomendações de mudança. Para evitar esse viés, removemos *commits* que contém mais de 30 arquivos (ZIMMERMANN et al., 2005), já que eles podem se referir a operações de mudança de ramos do controle de versão ou múltiplas mudanças.

Amostragem aleatória. A etapa de avaliação dos modelos realiza consultas a partir de um ou mais arquivos escolhidos aleatoriamente em cada uma das transações do conjunto de teste. Embora tenhamos usado a mesma amostra para comparar os três modelos, há a possibilidade de que se a consulta fosse realizada a partir de um outro arquivo selecionado aleatoriamente poderia apresentar outro valor de MAP. Essa escolha é feita porque não se sabe qual foi a ordem em que os arquivos foram modificados pelo desenvolvedor durante a realização de uma tarefa.

Generalização. Os experimentos foram realizados em dezesseis projetos de código aberto. Os projetos selecionados variam em tempo de criação, tamanho de histórico de mudanças, frequência de transações, linguagem e domínio. Apesar das diferenças entre os projetos que selecionamos podemos não capturar todas as variações possíveis, porém o uso de projetos com pequena quantidade de transações no histórico de mudanças adiciona uma perspectiva diferente de avaliação dos resultados que não foi avaliada por Moonen et al. (2016b) e Rolfsnes et al. (2016).

Configuração do Algoritmo Genético. As configurações escolhidas para o AG influenciam os resultados obtidos. Entretanto, verificou-se que todo o espaço de busca foi explorado, bem como a melhor escolha da população foi obtida, uma vez que o algoritmo genético convergiu rapidamente para a melhor solução de escolha do conjunto de treinamento e limiares de suporte e confiança.

6.1. Considerações Finais

Este capítulo apresentou as principais ameaças à validade do modelo proposto e também dos experimentos que serão realizados. As ameaças à validade descritas podem ser classificadas como ameaças internas e externas ao estudo.

As ameaças externas se referem à forma como os dados utilizados são produzidos, por se tratarem de dados produzidos durante o desenvolvimento de software, os hábitos dos desenvolvedores ao realizar um *commit* podem influenciar a validade do estudo. As ameaças internas se referem a detalhes da implementação e à escolha das configurações do AG.

Capítulo 7

Conclusões

Regra de associação é uma técnica de mineração de dados usada para analisar o histórico de mudanças de um sistema e descobrir acoplamentos de mudança entre os artefatos. Esses acoplamentos são utilizados para recomendar artefatos que são afetados por mudanças realizadas por desenvolvedores. Tais recomendações auxiliam desenvolvedores a enfrentar a crescente complexidade do sistema, causada pela constante manutenção e evolução do software. A acurácia das recomendações de mudanças é afetada pela quantidade de histórico disponível do projeto e dos valores selecionados para os parâmetros do algoritmo de mineração, como suporte e confiança.

Neste trabalho, investigamos empiricamente como determinar valores ótimos de suporte, confiança e conjunto de treinamento que geram as recomendações de mudança com maior acurácia. Para isto, definimos uma função de aptidão para um AG que seleciona o conjunto de treinamento e valores de suporte e confiança que otimizam as recomendações de mudança para um projeto de software.

Ao comparar a acurácia do modelo de recomendação de mudanças Genético+Apriori com os modelos Regressão+Apriori e Regressão+TARMAQ, os resultados indicam que, independente do cenário de avaliação analisado, o modelo proposto possui acurácia maior do que o modelo Regressão+Apriori. Para cenários com consulta fixa no primeiro arquivo modificado o modelo proposto possui resultados semelhantes ao modelo Regressão+TARMAQ, entretanto, em cenários com consulta de tamanho variável o modelo Regressão+TARMAQ obteve maior acurácia para, em média, 40% dos projetos avaliados. Apesar dos resultados significantes, o tempo para a execução do AG foi, em média, 23 vezes maior que o modelo Regressão+TARMAQ, essa diferença média representa cerca de 350 minutos a mais para executar o modelo Genético+Apriori. O tempo de execução implica na utilização do modelo Genético+Apriori em cenários reais de desenvolvimento de software.

Investigando o comportamento dos modelos de recomendação ao passo que novas mudanças são inseridas, os resultados mostraram que acurácia dos modelos se deteriora. A

inserção de novas mudanças implicou, em média, na deterioração de 37, 5% dos projetos tanto nos cenários com consultas fixas, quanto em cenários com consultas de tamanho variável para cada um dos modelos de recomendação. Em relação a diferença de comportamento entre os modelos, o modelo Regressão+TARMAQ apresentou maior acurácia em relação ao modelo proposto, em cenários com consulta de tamanho variável. Nos cenários de avaliação com consulta fixa, o modelo proposto apresentou acurácia superior, e a vantagem em se ajustar aos diferentes tamanhos de histórico de modificações, enquanto os modelos baseados na função de regressão não conseguiram identificar o tamanho do conjunto de treinamento capaz de ser usado na prática.

Como trabalho futuro, pretendemos estender o estudo para mais projetos do GitHub e avaliar o impacto de considerar um *commit* como sendo uma transação no processo de geração das regras de associação. Além disso, iremos comparar os resultados obtidos neste trabalho por meio de acoplamento de mudança com outros métodos de recomendação de mudanças como, acoplamento semântico e acoplamento estrutural.

Referências

AGRAWAL, R.; IMIELIńSKI, T.; SWAMI, A. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, ACM, New York, NY, USA, v. 22, n. 2, p. 207–216, jun. 1993. ISSN 0163-5808.

AGRAWAL, Rakesh; SHAFER, John C. Parallel mining of association rules. *IEEE Trans. on Knowl. and Data Eng.*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 8, n. 6, p. 962–969, dez. 1996. ISSN 1041-4347. Disponível em: <http://dx.doi.org/10.1109/69.553164>.

AGRAWAL, Rakesh; SRIKANT, Ramakrishnan. Fast algorithms for mining association rules in large databases. In: *Proceedings of the 20th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994. (VLDB '94), p. 487–499. ISBN 1-55860-153-8. Disponível em: <http://dl.acm.org/citation.cfm?id=645920.672836>.

ARNOLD, Robert S. *Software Change Impact Analysis*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1996. ISBN 0818673842.

BALL, Thomas; KIM, Jung min; PORTER, Adam A.; SIY, Harvey P. *If Your Version Control System Could Talk...* 1997.

BAVOTA, G.; DIT, B.; OLIVETO, R.; PENTA, M. Di; POSHYVANYK, D.; LUCIA, A. De. An empirical study on the developers perception of software coupling. In: *Proceedings of the 2013 International Conference on Software Engineering*. Piscataway, NJ, USA: IEEE Press, 2013. (ICSE '13), p. 692–701. ISBN 978-1-4673-3076-3.

CANFORA, G.; CECCARELLI, M.; CERULO, L.; ; Di Penta, M. Using multivariate time series and association rules to detect logical change coupling: An empirical study. In: *IEEE International Conference on Software Maintenance, ICSM*. [S.l.: s.n.], 2010. ISBN 9781424486298. ISSN 1063-6773.

COHEN, J. *Statistical Power Analysis for the Behavioral Sciences*. [S.l.]: Lawrence Erlbaum Associates, 1988.

COLARES, F.; SOUZA, J.; CARMO, R.; PáDUA, C.; MATEUS, G. R. A new approach to the software release planning. In: *2009 XXIII Brazilian Symposium on Software Engineering*. [S.l.: s.n.], 2009. p. 207–215.

DIT, B.; WAGNER, M.; WEN, S.; WANG, W.; LINARES-VáSQUEZ, M.; POSHYVANYK, D.; KAGDI, H. Impactminer: A tool for change impact analysis. In: *Companion Proceedings of the 36th International Conference on Software Engineering*. New York, NY, USA: ACM, 2014. (ICSE Companion 2014), p. 540–543. ISBN 978-1-4503-2768-8.

FILHO, R. A. M.; VERGILIO, S. R. A mutation and multi-objective test data generation approach for feature testing of software product lines. In: *2015 29th Brazilian Symposium on Software Engineering*. [S.l.: s.n.], 2015. p. 21–30.

GOTSHALL, S.; RYLANDER, B. Optimal population size and the genetic algorithm. *Population*, Citeseer, v. 100, n. 400, p. 900, 2002.

GRAVES, T. L.; KARR, A. F.; MARRON, J. S.; SIY, H. Predicting fault incidence using software change history. *IEEE Transactions on Software Engineering*, v. 26, n. 7, p. 653–661, Jul 2000. ISSN 0098-5589.

HAFEZ, Aladdin; DEOGUN, Jitender S.; RAGHAVAN, Vijay V. The item-set tree: A data structure for data mining. In: *Proceedings of the First International Conference on Data Warehousing and Knowledge Discovery*. London, UK, UK: Springer-Verlag, 1999. (DaWaK '99), p. 183–192. ISBN 3-540-66458-0. Disponível em: <http://dl.acm.org/citation.cfm?id=646108.679118>.

HAHSLER, Michael; CHELLUBOINA, Sudheer; HORNIK, Kurt; BUCHTA, Christian. The arules r-package ecosystem: Analyzing interesting patterns from large transaction data sets. *Journal of Machine Learning Research*, JMLR.org, v. 12, p. 2021–2025, jul. 2011. ISSN 1532-4435.

HASSAN, A. E. The road ahead for mining software repositories. In: *2008 Frontiers of Software Maintenance*. [S.l.: s.n.], 2008. p. 48–57.

HASSAN, A. E.; HOLT, R. C. Predicting change propagation in software systems. In: *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.* [S.l.: s.n.], 2004. p. 284–293. ISSN 1063-6773.

HAUPT, R. L.; HAUPT, S. E. *Practical Genetic Algorithms*. New York, NY, USA: John Wiley & Sons, Inc., 1998. ISBN 047-1188735.

HERZIG, K.; ZELLER, A. The impact of tangled code changes. In: *Proceedings of the 10th Working Conference on Mining Software Repositories*. Piscataway, NJ, USA: IEEE Press, 2013. (MSR '13), p. 121–130. ISBN 978-1-4673-2936-1.

HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. [S.l.: s.n.], 1975. Ann Arbor. 183 p. ISSN 10834419. ISBN 0262581116.

KUBAT, Miroslav; HAFEZ, Alaaeldin; RAGHAVAN, Vijay V.; LEKKALA, Jayakrishna R.; CHEN, Wei Kian. Itemset trees for targeted association querying. *IEEE Trans. on Knowl. and Data Eng.*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 15, n. 6, p. 1522–1534, nov. 2003. ISSN 1041-4347. Disponível em: <http://dx.doi.org/10.1109/TKDE.2003.1245290>.

LEHMAN, M. M. Programs, life cycles, and laws of software evolution. *Proc. IEEE*, v. 68, n. 9, p. 1060–1076, September 1980.

MAIMON, O.; ROKACH, L. *Data Mining and Knowledge Discovery Handbook*. 2nd. ed. [S.l.]: Springer Publishing Company, Incorporated, 2010. ISBN 0387098224, 9780387098227.

MALIK, H.; HASSAN, A.E. Supporting software evolution using adaptive change propagation heuristics. In: *IEEE International Conference on Software Maintenance, 2008. ICSM 2008.* [S.l.: s.n.], 2008. p. 177–186. ISSN 1063-6773.

MIRANDA, Márcio Nunes de. *Algoritmos Genéticos: Fundamentos e Aplicações*. 2007.

MOONEN, L.; ALESIO, S. Di; BINKLEY, D.; ROLFSNES, T. Practical guidelines for change recommendation using association rule mining. In: *International Conference on Automated Software Engineering (ASE)*. [S.l.]: ACM, 2016.

MOONEN, L.; ALESIO, S. Di; ROLFSNES, T.; BINKLEY, D. Exploring the effects of history length and age on mining software change impact. In: *International Working Conference on Source Code Analysis and Manipulation (SCAM)*. [S.l.]: IEEE, 2016.

OLIVA, G. A.; GEROSA, M. A. Change coupling between software artifacts: Learning from past changes. In: BIRD, C.; MENZIES, T.; ZIMMERMANN, T. (Ed.). *The Art and Science of Analyzing Software Data*. [S.l.]: Morgan Kaufmann, 2015. p. 285–324. ISBN 978-0124115194.

PINTO, G.; STEINMACHER, I.; GEROSA, M. A. More common than you think: An in-depth study of casual contributors. In: *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. [S.l.: s.n.], 2016. v. 1, p. 112–123.

RAY, B.; POSNETT, D.; FILKOV, V.; DEVANBU, P. A large scale study of programming languages and code quality in github. In: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. New York, NY, USA: ACM, 2014. (FSE 2014), p. 155–165. ISBN 978-1-4503-3056-5.

ROLFSNES, T.; ALESIO, S. Di; BEHJATI, R.; MOONEN, L.; BINKLEY, D. W. Generalizing the analysis of evolutionary coupling for software change impact analysis. In: *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. [S.l.]: IEEE, 2016. p. 201–212.

ROMANO, Jeanine; KROMREY, Jeffrey D. Appropriate statistics for ordinal level data: Should we really be using t-test and cohen's d for evaluating group differences on the nsse and other surveys? 01 2006.

SCRUCCA, L. GA: A package for genetic algorithms in R. *Journal of Statistical Software*, v. 53, n. 4, p. 1–37, 2013. Disponível em: <http://www.jstatsoft.org/v53/i04/>.

SRIKANT, Ramakrishnan; VU, Quoc; AGRAWAL, Rakesh. Mining association rules with item constraints. In: *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1997. (KDD'97), p. 67–73. Disponível em: <http://dl.acm.org/citation.cfm?id=3001392.3001404>.

STEINMACHER, I.; CONTE, T.; GEROSA, M. A.; REDMILES, D. Social barriers faced by newcomers placing their first contribution in open source software projects. In: *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work and Social Computing*. New York, NY, USA: ACM, 2015. (CSCW '15), p. 1379–1392. ISBN 978-1-4503-2922-4.

SU, W.; YUAN, Y.; ZHU, M. A relationship between the average precision and the area under the roc curve. In: *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*. New York, NY, USA: ACM, 2015. (ICTIR '15), p. 349–352. ISBN 978-1-4503-3833-2.

TANOMARU, Julio. Motivação, fundamentos e aplicações de algoritmos genéticos. In: *II Congresso Brasileiro de Redes Neurais*. [S.l.: s.n.], 1995. p. 373–403.

WESSEL, Mairieli Santos; ANICHE, Maurício Finavaro; OLIVA, Gustavo Ansaldi; GEROSA, Marco Aurélio; WIESE, Igor Scaliante. Tweaking association rules to optimize software change recommendations. In: *Proceedings of the 31st Brazilian Symposium on Software Engineering*. New York, NY, USA: ACM, 2017. (SBES'17), p. 94–103. ISBN 978-1-4503-5326-7. Disponível em: <http://doi.acm.org/10.1145/3131151.3131163>.

YING, A. T. T.; MURPHY, G. C.; NG, R.; CHU-CARROLL, M. C. Predicting source code changes by mining change history. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 30, n. 9, p. 574–586, set. 2004. ISSN 0098-5589.

ZHENG, Z.; KOHAVI, R.; MASON, L. Real world performance of association rule algorithms. In: *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2001. (KDD '01), p. 401–406. ISBN 1-58113-391-X.

ZIMMERMANN, T.; WEISSGERBER, P.; DIEHL, S.; ZELLER, A. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, v. 31, n. 6, p. 429–445, 2005. ISSN 00985589.