Classificação de *issues* obtidas de repositórios de *software*: uma abordagem baseada em características textuais

Tarcísio Martins Ferreira

Universidade Federal de Uberlândia

Faculdade de Computação

Programa de Pós-Graduação em Ciência da Computação

Uberlândia

2015

Tarcísio Martins Ferreira

Classificação de *issues* obtidas de repositórios de *software*: uma abordagem baseada em características textuais

Dissertação de mestrado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Marcelo de Almeida Maia

Uberlândia

2015

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

F383c 2016

Ferreira, Tarcísio Martins, 1982-
Classificação de issues obtidas de repositórios de software: uma abordagem baseada em características textuais / Tarcísio Martins Ferreira. - 2016.
74 f. : il.

Orientador: Marcelo de Almeida Maia.
Dissertação (mestrado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Inclui bibliografia.

1. Computação - Teses. 2. Software - Teses. Teses. I. Maia, Marcelo de Almeida. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU: 681.3

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada "**Classificação de issues obtidas de repositórios de software: uma abordagem baseada em características textuais**" por **Tarcísio Martins Ferreira** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação.**

Uberlândia, ___ de _____ de _____

Orientador: _____
Prof. Dr. Marcelo de Almeida Maia
Universidade Federal de Uberlândia

Banca Examinadora:

_____
Prof. Dr. Heitor Augustus Xavier Costa
Universidade Federal de Lavras

_____
Prof. Dr. Rivalino Matias Júnior
Universidade Federal de Uberlândia

A minha falecida mãe, Augusta Martins Ferreira.

A minha filha Maria Clara.

Agradecimentos

Ao professor Rivalino, pelos conselhos e apoio incondicional.

Ao professor Marcelo, pelo comprometimento e grande trabalho de orientação.

Ao professor Autran, pela experiência passada em momentos importantes.

A Deus, pela iluminação que me permitiu desenvolver este estudo.

Resumo

A classificação das *issues* ou questões nos repositórios de manutenção de *software* é realizada atualmente pelos desenvolvedores de *software*. Entretanto, essa classificação manual não é livre de erros, os quais geram problemas na distribuição das *issues* para as equipes de tratamento. Isso acontece porque os desenvolvedores, geralmente os proposítores das *issues*, possuem o mal hábito de classificá-las como *bugs*. Essas classificações errôneas produzem a distribuição de *issues* para uma equipe de tratamento de outro tipo de *issue*, gerando retrabalho para as equipes entre outras desvantagens. Por isso, o principal objetivo almejado com o estudo é a melhoria dessa classificação, utilizando de uma abordagem de classificação das *issues* realizada de maneira automatizada. Essa abordagem foi implementada com técnicas de *Aprendizado de Máquina*. Estas técnicas mostram que as palavras-chave discriminantes dos tipos de *issues* podem ser utilizadas como atributos de classificadores automáticos para a predição dessas *issues*. A abordagem foi avaliada sobre 5 projetos *open source* extraídos de 2 *issue trackers* conhecidos, Jira e Bugzilla. Por se tratarem de *issue trackers* de longa data, os projetos escolhidos forneceram boa quantidade de *issues* para este estudo. Essas *issues*, cerca de 7000, foram classificadas por especialistas humanos no trabalho [Herzig, Just e Zeller 2013], produzindo um gabarito utilizado para a realização deste estudo. Este trabalho produziu um classificador automático de *issues*, com acurácia de 81%, capaz de discriminá-las nos tipos *bug*, *request for enhancement* e *improvement*. O bom resultado de acurácia sugere que o classificador concebido possa ser utilizado em sistemas de encaminhamento de *issues* para as equipes de tratamento, com a finalidade de diminuir retrabalho dessas equipes que ocorre em virtude da má classificação.

Palavras-chave: classificação automática de *issues*, repositórios de manutenção de *software*, *issue trackers*, acurácia.

Abstract

The classification of issues in software maintenance repositories is currently done by software developers. However, this classification is conducted manually and is not free of errors, which cause problems in the distribution of issues to the maintenance teams. This happen because the developers, which usually are the proponents of the issues, have the bad habit of classifying them as bugs. This erroneous rating generates rework and other disadvantages to the teams. Therefore, the main objective of this study is to improve this classification, using an issue classification approach conducted in an automated manner. In turn, this approach was implemented based on machine learning tecniques. These tecniques show that keywords discriminant of issues types can be used as attributes of automatic classifiers for prediction of these issues. The approach was evaluated on five open source projects extracted from two widely used issue trackers, Jira and Bugzilla. Because they are old issue trackers, the chosen projects provided good number of issues for this study. These issues, about 7.000, were classified by human experts at work [Herzig, Just e Zeller 2013], producing a feedback which was used for this study. This present work produced an automatic issues classifier, with 81% of accuracy, able to predict them in types of bug, request for enhancement and improvement. The result of accuracy obtained by this classifier suggests that it can be used in delivery systems to treatment teams with the purpose of reducing rework that occurs in these teams because of the poor issues rating.

Keywords: automatic issues classification, software maintenance repository, issue, accuracy.

Lista de ilustrações

Figura 1 – *Workflow* do processo de *Aprendizado de Máquina.*(Fonte: [Ho 2015]) . 28
Figura 2 – "Validação Um para Restante", acurácia medida para validação composta na Tabela 2. 55
Figura 3 – "Validação Um para Restante", acurácia medida para validação composta na Tabela 2. 57
Figura 4 – "Validação dos Projetos Separados", acurácia medida para validação composta na Tabela 4. 57
Figura 5 – Predição por classe aferida pelo algoritmo *MaxEnt* na "Validação dos Projetos Separados". 58
Figura 6 – Predição por classe aferida pelo algoritmo *Balanced Winnow* na "Validação dos Projetos Separados". 58
Figura 7 – "Validação Todos para Todos", acurácia medida para validação composta na Tabela 6. 59
Figura 8 – Predição por classe aferida pelo algoritmo *MaxEnt* na "Validação Todos para Todos". 59
Figura 9 – Predição por classe aferida pelo algoritmo *Balanced Winnow* na "Validação Todos para Todos". 60
Figura 10 – Acurácia aferida pelo algoritmo *MaxEnt* em análise da "Validação Todos para Todos" com podas dos atributos. 60
Figura 11 – "Validação Um para Restante", acurácia medida para validação composta na Tabela 2. 61
Figura 12 – "Validação dos Projetos Separados", acurácia medida para validação composta na Tabela 4. 62

Figura 13 – “Validação Todos para Todos”, acurácia medida para validação composta na Tabela 6. 62

Lista de tabelas

Tabela 1 – Detalhes dos Projetos 51
Tabela 2 – Configuração do Experimento 1 51
Tabela 3 – Tabela de Rótulos 52
Tabela 4 – Configuração do Experimento 2 52
Tabela 5 – Tabela de rotulagem 2 53
Tabela 6 – Configuração do Experimento 3 53
Tabela 7 – Top 20 *features* nos sistemas, ordenadas por *infogain*. 56

Lista de siglas

API Aplication Program Interface

KNN K Nearest Neighbour

LDA Latent Dirichlet Allocation

LSI Latent Semantic Indexing

SVD Singular Value Decomposition

SVM Support Vector Machine

Sumário

1 INTRODUÇÃO 21

1.1 Problema de Pesquisa 22

1.2 Objetivos da Pesquisa 22

1.2.1 Objetivo Geral 22

1.2.2 Objetivos Específicos 23

1.3 Desenvolvimento da Pesquisa 23

1.3.1 Revisão da Literatura 23

1.3.2 Coleta de Dados 24

1.3.3 Análise dos Dados 24

1.3.4 Estudo Experimental 24

1.3.5 Análise dos Resultados 24

1.4 Organização da Dissertação 25

2 REVISÃO DA LITERATURA 27

2.1 Fundamentação Teórica 27

2.2 Trabalhos Correlatos 33

3 CATEGORIZAÇÃO DE *ISSUES* EM REPOSITÓRIOS DE SOFTWARE 41

3.1 Construção do Classificador 41

3.1.1 Estudo Piloto #1 41

3.1.2 Estudo Piloto #2 45

3.1.3 Conclusão 47

4 ESTUDO EXPERIMENTAL 49
4.1 Planejamento do Estudo Experimental 49
4.1.1 Experimento #1 - Validação Um para Restante 50
4.1.2 Experimento #2 - Validação dos Projetos Separados 51
4.1.3 Experimento #3 - Validação Todos para Todos 52
4.1.4 Divisão dos Experimentos Classificatórios 53
4.1.5 Acurácias de Referência 54
4.2 Análise dos Resultados 55
4.2.1 Introdução 55
4.2.2 Resultados do Estudo Piloto #1 55
4.2.3 Resultados do Estudo Piloto #2 56
4.3 Conclusão 62

5 CONSIDERAÇÕES FINAIS 65
5.1 Resultados Alcançados 65
5.2 Limitações da Pesquisa 66
5.3 Dificuldades Encontradas 66
5.4 Trabalhos Futuros 67

REFERÊNCIAS 69

Capítulo 1

Introdução

Nos dias atuais, as tarefas de melhoramento de *software* têm sido abordadas com ênfase nas grandes organizações. Trata-se da realização de melhorias que garantem a estabilidade dos sistemas de *software*. Para que isso aconteça, a Engenharia de *Software* cumpre papel importante fornecendo ferramentas que apoiam tais melhorias.

Nos ambientes de desenvolvimento de *software*, ocorre a especialização das tarefas de melhoria. Existem equipes que cuidam das correções, equipes que cuidam das melhorias de requisitos existentes e equipes que cuidam do desenvolvimento de novos requisitos. Para que as tarefas ocorram com eficiência, é importante que aconteça sua distribuição entre as equipes, ou seja, a tarefa seja adequadamente direcionada a equipe adequada ao seu tratamento.

Existem sistemas *on-line* chamados *issue trackers*, geralmente integrados ao repositório de *software* onde o *software* das organizações se encontra. Como apresentado em [Zimmermann et al. 2009], *issue trackers* são sistemas que servem como um repositório central para o monitoramento das questões ou *issues* pertinentes ao desenvolvimento de *software*. Nesses sistemas, pode-se solicitar informações adicionais aos desenvolvedores do *software* e discutir as possíveis soluções.

Para apoiar a distribuição das questões(*issues*) aos desenvolvedores, a Engenharia de *Software* possui um campo de estudo denominado *Mineração de Repositórios de Software*. [Thomas, Hassan e Blostein 2014] afirmam que a mineração de repositórios de *software* é o processo de análise de dados relacionada a boas práticas de desenvolvimento de *software*, um campo emergente de pesquisa que destina-se a melhorar tarefas de evolução de *software*.

Este trabalho de pesquisa teve como alicerce a *Mineração de Repositórios de Software*

e foi realizado visando aprimorar os estudos de classificação das *issues* nos repositórios de *software* e *issue tracker systems*. No presente trabalho, as correções de *software* são representadas pelo termo *bug*, as melhorias de requisitos existentes são representadas pelo termo *improvement* e o desenvolvimento de novos requisitos é representado pelo termo *rfe* ou *request for enhancement*.

1.1 Problema de Pesquisa

Atualmente, as *issues* são propostas nos *issue tracker systems* pelos usuários de *software*. Em [Herzig, Just e Zeller 2013], observou-se que 39% das *issues* caracterizadas como defeito de *software*(*bug*), na verdade não o são. Os autores realizaram um estudo sobre milhares de *issues* e utilizaram sujeitos humanos especialistas em classificação com o objetivo de formar um gabarito decorrente da análise. As respostas do gabarito, classificadas por parte dos especialistas, foram comparadas com as respostas dadas pelos propositores das *issues*. Foi observada acurácia de 57,4% para essa classificação e notou-se que os propositores possuem o mal hábito de classificar as *issues* como *bug*.

No presente trabalho, foi construído um classificador automático de *issues* visando melhorar a classificação destas *issues* nos repositórios de software e sistemas *issue tracker systems*. Assim, a distribuição das tarefas para as equipes solucionadoras poderá ser realizada com maior acurácia, reduzindo custos de retrabalho e o tempo de manutenção do software, entre outros benefícios.

1.2 Objetivos da Pesquisa

1.2.1 Objetivo Geral

O objetivo geral planejado com esse estudo foi a construção de um classificador automático de *issues* de repositórios de manutenção de *software* ou *issue tracker systems*.

Conforme apresentado em [Herzig, Just e Zeller 2013], os desenvolvedores erram em média 42,6% de suas classificações durante a proposição das *issues* nos repositórios. Com esse resultado, obteve-se a acurácia dos desenvolvedores propositores de *issues* subtraindo 42,6% de 100%. Portanto, o objetivo geral para o presente estudo foi superar a acurácia dos desenvolvedores, 57,4%.

A superação desse valor pode revelar que a utilização de classificadores automáticos de *issues* nos *issue tracker systems* seja uma opção para direcioná-las com melhor acurácia as equipes de tratamento. Assim, pode-se diminuir o tempo de entrega da solução das *issues* e evitar retrabalho.

O classificador concebido com este estudo foi criado com a capacidade de discriminar as *issues* em *bug* ou correção de erros, *improvement* ou melhoria dos requisitos de *software* pré-existentes e *request for enhancement* ou pedidos de novos requisitos de *software*.

1.2.2 Objetivos Específicos

Os objetivos específicos planejados para esse estudo foram:

- ❑ Desenvolver um classificador que seja independente do acesso ao código fonte dos projetos de *software*, utilizando, para a classificação, as mensagens das *issues* contidas nos *issue tracker systems*;

- ❑ Desenvolver um classificador que tenha acurácia média superior a 57,4%, percentual atingido pelos propositores de *issues* nos *issue tracker systems* ao classificá-las com as classes *bug*, *improvement*, *rfe* e outras.

1.3 Desenvolvimento da Pesquisa

A construção do classificador foi realizada por meio de pesquisa, capacitação sobre o assunto *Aprendizado de Máquina* e realização de estudos experimentais baseados em trabalhos de referência apresentados na revisão da literatura. As etapas determinantes para a construção do classificador automático de *issues* são descritas a seguir.

1.3.1 Revisão da Literatura

Nessa etapa, foram estudados artigos relacionados ao tema "Mineração de Dados em Repositórios de *Software*", dando ênfase aos estudos sobre classificadores automáticos de texto. As abordagens e os resultados produzidos através dos estudos realizados nesses trabalhos foram analisados para contribuírem com o desenvolvimento desta pesquisa.

1.3.2 Coleta de Dados

Para a realização do estudo necessitou-se escolher o repositório onde os projetos seriam extraídos. Em primeiro momento, escolheu-se o repositório GitHub, pela quantidade de projetos e variadas plataformas de programação existentes nele. Em segundo momento, o trabalho foi aprofundado sobre os *issue tracker systems* Jira[1] e Bugzilla[2]. Esses repositórios de *issues* possuem projetos de *software open source* que serviram de motivação para sua utilização neste estudo. Como exemplo, pode-se citar: Lucene[3], Httpclient[4] e Tomcat5[5].

1.3.3 Análise dos Dados

Por meio de testes de busca de *palavras-chave* na base de dados coletada para a realização dos estudos, detectou-se que informações das classes *bug*, *request for enhancement* e *improvement* ocorreram nas mensagens de texto das *issues*.

1.3.4 Estudo Experimental

Foram planejadas e desenvolvidas as conversões dos dados disponibilizados em um banco de dados relacional para os formatos de entrada dos *toolkits* de apoio à aplicação dos estudos de *Aprendizado de Máquina*. Em seguida, foram realizados os primeiros experimentos de construção do classificador e aferição de sua acurácia. Em suma, as etapas do *Aprendizado de Máquina* foram aplicadas de modo iterativo. Em cada iteração, foram realizados ajustes para a melhoria do classificador.

1.3.5 Análise dos Resultados

Foram analisados os resultados do estudo experimental, produzidos pelos algoritmos de avaliação de classificadores automáticos de texto. Comparou-se resultados gerais entre os algoritmos e separadamente, comparou-se os resultados de acurácia individuais obtidos para cada classe discriminada pelo classificador. Também, foi comparada a acurácia do

[1] https://www.atlassian.com/software/jira
[2] http://www.bugzilla.org/
[3] http://lucene.apache.org/
[4] http://hc.apache.org/httpclient-3.x/
[5] https://tomcat.apache.org/tomcat-5.5-doc/

classificador de *issues* concebido neste trabalho com a acurácia dos propositores de *issues*, obtida do estudo referencial [Herzig, Just e Zeller 2013].

1.4 Organização da Dissertação

Os próximos capítulos desta dissertação estão organizados da seguinte forma:

No **Capítulo 2**, são apresentados os trabalhos correlatos e uma comparação deles com o estudo realizado nesta pesquisa. Também são apresentados os fundamentos teóricos sobre *Aprendizado de Máquina*, relevantes para a construção e avaliação do classificador automático de *issues* concebido nesta pesquisa.

O **Capítulo 3** apresenta a categorização de *issues* em repositórios de *software*, dando foco a metodologia utilizada para a concepção do classificador.

O **Capítulo 4** aborda o estudo experimental desenvolvido nesta pesquisa o qual produziu o classificador automático de *issues*. Neste capítulo, apresentam-se dados das amostras e outras informações dos experimentos como planejamento, realização e análise dos resultados.

O **Capítulo 5** apresenta as considerações finais, os resultados alcançados, as limitações da pesquisa e dificuldades encontradas, sendo finalizado com as possibilidades de realização de outros trabalhos a partir deste trabalho de pesquisa.

Capítulo 2

Revisão da Literatura

Neste capítulo, são apresentados os trabalhos de referência e os conceitos de relevância para a construção do classificador proposto neste estudo. O tema *Aprendizado de Máquina*, que constitui a teoria para a construção do classificador concebido, é descrito utilizando um *workflow*. Na apresentação da próxima seção, cada tarefa corresponde a uma etapa do processo de construção do classificador. A seção de trabalhos correlatos, que apresenta o resumo dos trabalhos de referência e uma comparação com o presente trabalho, finaliza o capítulo.

2.1 Fundamentação Teórica

O *Aprendizado de Máquina* é a parte do estudo sobre *mineração de dados* que fundamenta a construção de classificadores automáticos de informações como pode ser visto em [Bishop 2006]. Na Figura 1, as etapas desse processo de construção são apresentadas por descrições textuais dentro de retângulos de cor preta. O desenho de nuvem mostra a entrada e a saída do processo. A entrada do processo é constituída por documentos de texto e a saída tem como item o classificador automático desses documentos. Tarefas intermediárias também possuem entradas e saídas, como é o caso da tarefa *transformation* que recebe como entrada o *Corpus*, realiza seu processamento e retorna o *Corpus* transformado como saída. As etapas do processo de construção do classificador são descritas após a figura.

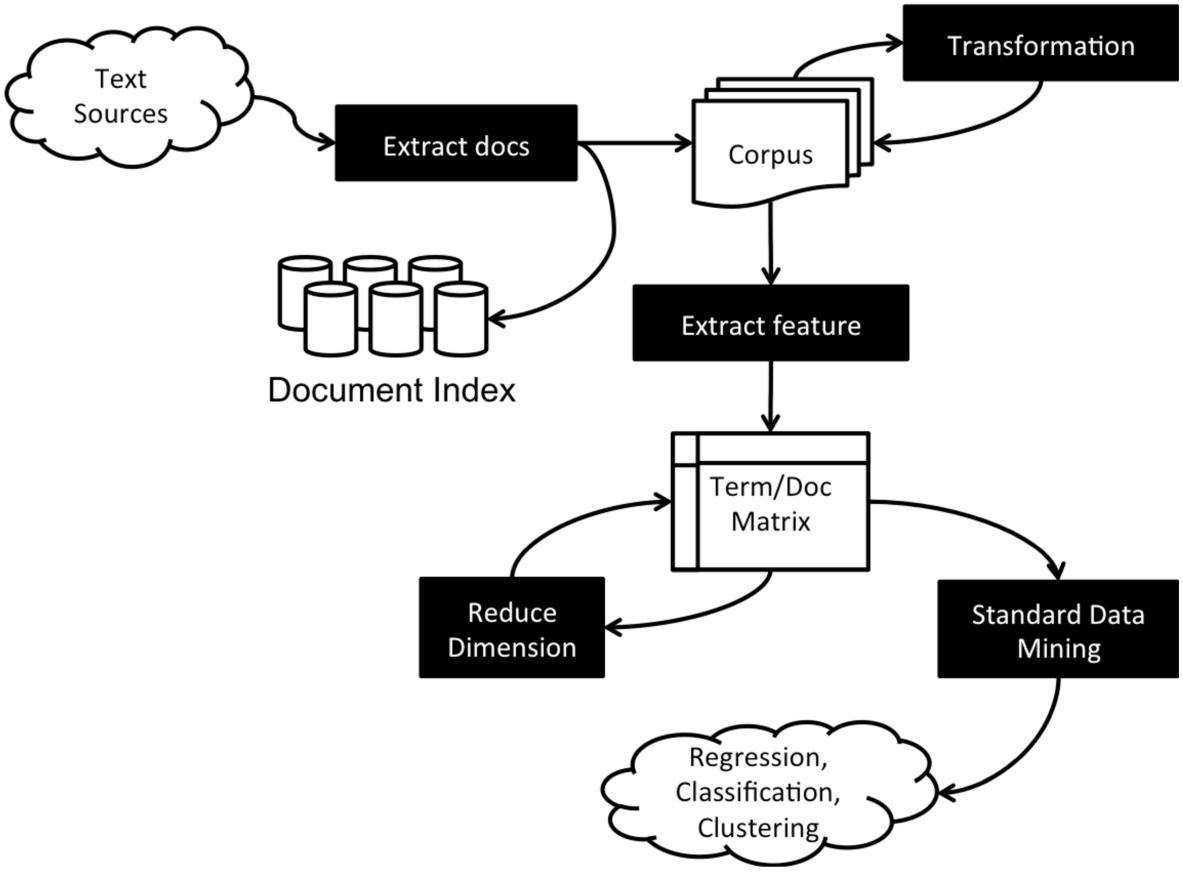

Figura 1 – *Workflow* do processo de *Aprendizado de Máquina.*(Fonte: [Ho 2015])

O primeiro passo de estudos sobre *Mineração de Dados* é a determinação do objeto que será analisado. Em alguns casos, o documento de texto é esse objeto. Em outros casos o documento de texto fornece informações sobre o objeto, por exemplo, um comentário do usuário sobre um produto. Para o presente trabalho, o objeto de análise é constituído pela descrição das *issues* nos repositórios de *software* e comentários realizados sobre elas.

Após determinar o objeto de análise para fornecê-lo como entrada para o processo, ocorre a etapa *extract docs* conforme apresentado na Figura 1. Essa etapa compreende a extração de informações para serem utilizadas na produção dos documentos de texto que representam o objeto de análise. Uma maneira de extrair as informações é, por exemplo, utilizar uma Aplication Program Interface (API) como a do repositório de *software* GitHub, a qual fornece informações das *issues* nas respostas da API retornadas com formatos de dados *json* ou *xml*. Após a etapa de extração, forma-se o *Corpus*, o qual compreende os documentos de texto armazenados em banco de dados ou arquivo e a estrutura de índice para realizar as pesquisas no *Corpus*.

Em seguida, o *Corpus* é submetido a uma etapa chamada *transformation* para a adap-

tação das mensagens de texto ao contexto de classificação, contribuindo para obtenção de melhores resultados de classificação. Por exemplo, necessita-se remover a pontuação das palavras, aplicar o padrão *lower case*, aplicar *lemmatization* (lematização), substituição das palavras por suas formas canônicas como apresentado em [Korenius et al. 2004] e *stemming*, substituição das palavras por seu radical tal como descrito em [Manning et al. 2008], ou remover números e palavras pouco significativos presentes nos textos fornecidos como entrada para o processo.

Após a transformação do *Corpus*, são extraídas as *features* (características, atributos ou termos) do classificador, na etapa *extract feature*. O modelo de extração mais comum é o *bag-of-words* conforme observado em [Wallach 2006]. Nesse modelo, cada documento é representado como um *vetor de features*. Assim, os documentos que compreendem o *Corpus* são representados como uma grande *matriz*. As *features* do classificador podem ser generalizadas como *uni-gram*, *bi-gram*, *tri-gram* ou *n-gram*.

O *uni-gram* significa uma *feature* formada por uma única palavra de texto, enquanto o *bi-gram* significa uma *feature* composta por duas palavras e o mesmo raciocínio segue para *tri-gram* e *n-gram*. Um exemplo de *uni-gram* é a *feature "change"*, exemplo de *bi-gram*, a *feature "change-name"*.

O valor contido em cada célula da matriz é a frequência de cada *feature* no documento, podendo ser apenas valores binários, sendo que 0 representa a ausência da *feature* no documento e 1 representa a presença da *feature*.

Após a realização da extração de *features*, etapa *extract feature*, é realizada a etapa *reduce dimension*. Na etapa *extract features* é produzida uma matriz de grande dimensão. Esta grande dimensionalidade fornece duas razões para sua diminuição:

- melhorar a eficiência de processamento do *Corpus* por meio de redução do gasto de memória computacional;

- transformar o vetor do espaço *terms* para o espaço *topics* para situar as *features* de ocorrência semelhante, próximas umas das outras.

Por exemplo, as palavras *"pet"* e *"cat"* são mapeadas para o mesmo *topic* baseando-se na semelhança de ocorrência delas, ambas contextualizam animal de estimação. As técnicas popularmente utilizadas nesta etapa são Singular Value Decomposition (SVD) [Wall, Rechtsteiner e Rocha 2003] que reduz o vetor de *features* em vetor de conceitos

utilizando Latent Semantic Indexing (LSI) [Dumais 2004] e *topic modeling* [Wang e Blei 2011], baseada em Latent Dirichlet Allocation (LDA) [Blei, Ng e Jordan 2003].

A última etapa *standard data mining*, compreende a aplicação de mecanismos resultantes em *Regressão Logística*, *Classificação* e *Clusterização*. Para o presente trabalho, essa etapa resultou na geração de um classificador automático de *issues*.

Os estudos sobre *Aprendizado de Máquina* são apoiados por ferramentas especializadas. Para o presente trabalho, foram utilizadas as ferramentas Weka[1] e Mallet[2].

Para a medida de desempenho do classificador foram utilizados algoritmos de *mineração de dados*. Esses algoritmos buscam padrões em um conjunto de treinamento ou *training set* aplicados em um conjunto de teste ou *test set* para avaliar a eficiência do classificador. De acordo com erros e acertos de predição decorrentes do mecanismo de reconhecimento de padrões característico de cada algoritmo, determina-se a acurácia do *classificador* por meio da aplicação de *cross validation* ou *validação cruzada*. A *cross validation* consiste na aplicação de critérios de comparação entre os resultados classificatórios do gabarito e os resultados aferidos pelo classificador automático. Esses critérios são estabelecidos com a determinação do *trainning set* e do *test set*, conforme pode-se observar em [Kohavi 1995].

Existem medidas (*measures*) de recuperação de informações que foram utilizadas neste trabalho e nos trabalhos correlatos. Trata-se de *recall*, *precision* e *f-measure*.

Conforme apresentado em [Buckland e Gey 1994], *recall* é a medida da eficácia da recuperação de informações representante do número de registros relevantes recuperados. Neste trabalho, estes registros possuem relação com os registros classificados.

Seja **A** o total de registros relevantes recuperados e **B** a quantidade de registros relevantes ignorados, apresenta-se a fórmula de cálculo de ***recall***:

$$\frac{A}{A+B}. \tag{1}$$

Em [Buckland e Gey 1994] definiu-se que *precision* é a quantidade de registros relevantes, representado como uma proporção do total de registros recuperados.

Seja **A** o total de registros relevantes recuperados e **C** o número de registros irrelevantes recuperados, apresenta-se a fórmula de cálculo de ***precision***:

$$\frac{A}{A+C}. \tag{2}$$

[1] www.cs.waikato.ac.nz/ml/weka
[2] http://mallet.cs.umass.edu/

A variável α representa um fator de peso, multiplicado pelos valores de *precision* e *recall*.

A fórmula de cálculo da medida ***f-measure***, que de acordo com [Zhang e Zhang 2009] representa a média harmônica entre *precision* e *recall* é apresentada:

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} \tag{3}$$

sendo $\alpha \in [0, 1]$, P o valor de *precision* e R o valor de *recall*.

A média harmônica [Makhoul et al. 1999] é utilizada para realização de cálculos envolvendo medidas inversamente proporcionais. Uma vez que *recall* e *precision* possuem comportamento inverso, ou seja, a medida que o valor calculado de *recall* aumenta, o valor calculado de *precision* diminui, utilizou-se a média harmônica para representar a acurácia envolvendo essas duas grandezas.

Também, conforme mostrado em [Zhang e Zhang 2009], existe uma variação da medida *f-measure* sendo que os pesos de *recall* e de *precision* são os mesmos, nesse caso, tem-se $\alpha = \frac{1}{2}$. Essa variação é denominada ***F1***. Considerando que R significa o valor calculado para *recall* e P significa o valor calculado para *precision*, sua fórmula de cálculo é apresentada:

$$F_1 = \frac{2PR}{P+R}. \tag{4}$$

Os cálculos anteriores foram apresentados devido sua utilização nos experimentos, para as medições do classificador para cada classe *bug*, *rfe* e *improvement*. Estes resultados podem ser visualizados na Figura 8 (há outras semelhantes) desta pesquisa. Tais resultados foram obtidos com o apoio de *toolkits*, que serão apresentados mais a frente. O cálculo **F1**, que depende de *precision* e *recall*, também foi utilizado para as medições do classificador automático de *issues* concebido nessa pesquisa.

Outro termo utilizado nessa pesquisa foi a *Acurácia* [Makridakis 1993], que representa a proporção entre "falsos positivos" e "falsos negativos", itens recuperados. Para esse estudo, representa uma das medidas de desempenho da classificação automática (discriminação de classes) de *issues*, além das citadas anteriormente (*recall*, *precision*, *F1*), realizada pelo classificador automático concebido.

Para a aferição da acurácia, foram utilizados algoritmos de avaliação de classificadores de informações. Cada algoritmo possui uma estratégia (*reconhecimento de padrão*) discriminante das classes (categorias) escolhidas para a classificação. Esta estratégia utiliza as *features* definidas para o classificador e estão relacionadas a estudos sobre o assunto *Aprendizado de Máquina*.

Para esta pesquisa, foram utilizados os seguintes algoritmos: *Max Entropy*, *Decision Tree*, *Naïve Bayes*, *Winnow* e *Balanced Winnow*. A seguir é apresentado o funcionamento destes algoritmos.

O modelo estatístico *Max Entropy* [McCallum, Freitag e Pereira 2000], nomeado nessa pesquisa como *Max Ent*, executa o cálculo das somas de probabilidades de cada *feature* ser igual a um, levando em consideração a construção de *modelos estocásticos* por meio de uma *função não linear*. Em [Zhang 2015] são apresentadas ferramentas para um estudo aprofundado sobre esse algoritmo.

O algoritmo *Decision Tree* [Landgrebe 1991] ou *Árvore de Decisão*, utilizado para tarefas de classificação, efetua a combinação das *features* utilizando modelos lineares e não lineares. O modelo tradicional de geração da árvore utiliza estratégias *top-down* e *divide and conquer* em sua abordagem. Os algoritmos evolutivos de *árvores de decisão* utilizam abordagens alternativas que melhoram os componentes particulares para classificadores como apresentado em [Barros et al. 2012].

O algoritmo *Naïve Bayes*, baseado na hipótese *Bayesiana* [McCallum, Nigam et al. 1998], define que, dado um documento de teste ou *test set*, a classificação desse documento é executada utilizando o cálculo da probabilidade para cada classe discriminada pelo classificador, considerando o maior valor desta probabilidade para as amostras de entrada. O processo é aplicado iterativamente para as amostras do documento de teste. Ao final desse processo, a classe predominante em cada amostra é utilizada para o cálculo de acurácia.

O algoritmo *Winnow* [Golding e Roth 1999] é um modelo linear de classificação baseado em uma *função booleana* de predição. Nesse ponto da escolha do tipo de função de predição, existe similaridade com a abordagem de classificação do algoritmo *Perceptron*, o qual não foi aplicado a esse estudo. No processo de predição do algoritmo *Winnow*, existe um limiar que divide o hiperplano na instância do espaço de classificação. A função preditiva de classificação, pode atribuir valores 0 e 1 às amostras ou contabilizar a frequência das *features* nos documentos, sendo acompanhada por um conjunto de pesos para cada *feature*. A diferença em relação ao algoritmo *Balanced Winnow* [Koster, Seutter e Beney 2003] é que esse algoritmo mantém dois conjuntos de pesos em dois hiperplanos distintos, conforme apresentado em [Littlestone 1988], enquanto o algoritmo *Winnow* mantém apenas um conjunto de pesos.

2.2 Trabalhos Correlatos

Em [Tsukada, Washio e Motoda 2001], os autores aplicaram o *Aprendizado de Máquina* para realizarem a classificação automática de *webpages*. A técnica de classificação utilizada baseou-se no algoritmo *Árvores de Decisão*. O *dataset* utilizado foi as páginas do Yahoo! JAPAN[3]. A*acurácia* da classificação foi detalhada utilizando *recall* e *precision*. Também, foram definidas 5 categorias de classificação:

- ❏ "Arts & Humanities",
- ❏ "Business & Economy",
- ❏ "Education","Goverment",
- ❏ "Health".

Os autores utilizaram para a classificação 200 *webpages* por categoria, totalizando 1000 páginas. Foram gerados diferentes conjuntos de *features* por categoria, e posteriormente, os resultados foram comparados. O melhor resultado da medida *precision* foi obtido para a categoria "Arts & Humanities". Para um conjunto de 19 *features*, alcançou-se o valor 95,3% para essa medida.

No presente trabalho, pode-se destacar a diferença do contexto de classificação em relação ao trabalho [Tsukada, Washio e Motoda 2001]: no Capítulo 4 deste trabalho, classificou-se os textos das *issues* localizadas nos *repositórios de software* e nos *issue tracker systems*, enquanto naquele trabalho, classificou-se os textos de *webpages*. Uma semelhança entre os dois trabalhos é a utilização da medida *precision* (embutida na medida *F1*) em conjunto com a aferição de acurácia do classificador.

Em [Ugurel, Krovetz e Giles 2002], os autores realizaram a classificação automática de código fonte em relação a linguagem de programação e categorias de aplicação. Eles mostraram uma técnica de classificação aderente a avaliação de classificação do algoritmo *Support Vetor Machine*, utilizando extração de *features* de código fonte, comentários e arquivos *README*. Para linguagens de programação, as *features* foram definidas como *tokens* do código fonte e palavras-chave dos comentários desses códigos fonte. Os autores aplicaram a técnica nos repositórios de código Ibiblio Linux Archive, SourceForge, Planet Source Code, Free Code e sobre páginas da Internet contendo fragmentos de código. Os

[3] http://www.yahoo.co.jp/

autores do estudo mediram a acurácia da técnica em cada um dos repositórios utilizando diferentes conjuntos de *features*. A maior acurácia, obtida aplicando *cross validation*, foi registrada no repositório Ibiblio Linux Archive; para um conjunto de *features* denominado *Top 100 features from single words and bigrams*, a técnica atingiu o valor 72% de acurácia.

No presente estudo, as *features* foram extraídas dos textos das *issues* contidas nos repositórios de *software* e *issue trackers systems*, não ocorrendo análise do código fonte dos projetos avaliados. Uma semelhança deste estudo com [Ugurel, Krovetz e Giles 2002] é a utilização de repositórios de *software* nos estudos classificatórios e *features* constituídas de *unigram* e *bigram*.

No trabalho [Li et al. 2006] foram investigados os impactos de novos fatores de erros de *software* realizando-se estudos em dois projetos *open source*, Mozilla e Apache Web Server. Os autores classificaram os erros baseando-se em causa raiz, impactos e componentes de *software* e estudaram a correlação entre as categorias de erros definidas. Além disso, eles utilizaram técnicas de classificação de texto e recuperação de informações para classificarem automaticamente milhares de erros e analisarem os resultados com o objetivo de estabelecerem tendências dos *bugs*. Durante a etapa de treinamento do classificador, os autores do estudo rotularam os *bugs* manualmente e produziram o modelo de classificação para as categorias de *bugs*. Foram utilizados os algoritmos de classificação *Support Vector Machine (SVM)*, *Winnow*, *Perceptron* e *Naïve Bayes*. Os autores mediram a acurácia do classificador com esses algoritmos e encontraram o de melhor desempenho. Os resultados do estudo foram apresentados utilizando as medidas *recall*, *precision* e *F1*. Aplicando seu modelo de classificação, os autores conseguiram identificar os *bugs* que ocorreram em tempo de execução, chamados de *runtime errors*, com 89,6% para a medida *precision*.

Nesta pesquisa, as *issues* dos repositórios de *software* e *issue trackers systems*, de um total de 5 projetos avaliados, foram discriminadas em *bug*, *rfe* e *improvement*. Como semelhança entre os dois trabalhos pode-se citar que o desempenho do classificador automático de cada um dos trabalhos, medido utilizando múltiplos algoritmos, e a sua acurácia foi provida em conjunto com as medidas *recall* e *precision*.

No trabalho [Antoniol et al. 2008], os autores investigaram se as mensagens de texto postadas nos *issue trackers systems* eram suficientes para realizarem a classificação da manutenção corretiva de *software* (*bugs*) e outros tipos de atividades. Para isso, eles construíram um classificador e utilizaram os algoritmos de classificação *Decision Tree* e *Naïve Bayes*. Os autores também realizaram um estudo de *Regressão Logística* sobre o

dataset da pesquisa. Esse *dataset* foi constituído dos projetos JBOSS, Mozilla e Eclipse. A acurácia do classificador foi obtida dentro do intervalo de 77% a 82%. A conclusão do trabalho foi as mensagens de texto sendo suficientes para distinguir de forma automatizada as manutenções corretivas de outras atividades. A principal finalidade dessa distinção foi a possibilidade de construção de sistemas de encaminhamento automático das atividades para os times de manutenção, que corrigem *bugs*, e times de liderança, que tratam dos melhoramentos e outros tipos de atividades propostas nos sistemas *issue trackers*.

O contexto de classificação desta pesquisa e o do estudo [Antoniol et al. 2008] são semelhantes, a classificação de *issues*. Neste trabalho, como em [Antoniol et al. 2008], foram escolhidos grandes projetos *open source* (Tomcat 5, Lucene, Jackrabbit, Httpclient e Rhino). Outra semelhança entre os estudos é o propósito de classificar as *issues* sem acesso ao código fonte dos projetos de *software*. Também, ambos objetivam melhorar a distribuição das *issues* as equipes de tratamento.

No trabalho [Antoniol et al. 2008], os autores realizaram a classificação das *issues* manualmente antes de utilizá-las para a construção do classificador automático. Neste trabalho, as *issues* utilizadas para a construção do classificador automático foram classificadas manualmente no estudo da referência [Herzig, Just e Zeller 2013]. Também, naquele trabalho as categorias discriminadas pelo classificador automático foram duas, *bug* e *others*. Neste trabalho, além da discriminação automática das *issues* entre *bug* e *others*, foi realizado um estudo de classificação automática dessas *issues* entre as categorias *bug*, *request for enhancement* e *improvement*, além da categoria *refactoring*, substituída durante os estudos.

Em [Katakis, Tsoumakas e Vlahavas 2008], os autores construíram um algoritmo de classificação de texto para um recomendador de *tags* de Bibsonomy[4], um gerenciador de publicações científicas. No estudo, foram utilizados 3 arquivos de treinamento contendo os dados para a classificação. Para cada arquivo, foi definido um conjunto específico de *features*. Os autores utilizaram Weka como ferramenta de apoio para a construção do classificador. Ao final do estudo, os autores apresentaram os resultados alcançados utilizando cálculos da medida *F1*. O estudo classificatório baseou-se no assunto *topics*, cuja aplicação experimental produziu a agregação *a posteriori* em uma mesma categoria, das *features* de significado semelhante. Os autores afirmaram que desejavam aplicar futuramente outros algoritmos de classificação e realizar *seleção de features* para melhorar os

[4] http://www.bibsonomy.org/

resultados do estudo.

Neste trabalho, foram utilizados dois tipos de definição das *features* do classificador:

- *a priori*, utilizado inicialmente;
- *a posteriori*, utilizado nos estudos finais.

Nos primeiros estudos, as *features* do classificador foram definidas por meio de testes de busca de palavras-chave na base de dados. Nos estudos finais, foi utilizada a técnica LDA para a definição automática de *features*. Uma distinção entre os trabalhos pode ser apontada sobre a utilização da *seleção de features*. Nesta pesquisa, este mecanismo de melhoramento do classificador foi empregado enquanto no estudo [Katakis, Tsoumakas e Vlahavas 2008] não.

Em [Herzig, Just e Zeller 2013], os autores realizaram um trabalho de predição de *bugs* em 5 projetos de *software open source*. Os autores concluíram que o processo de classificação deve ser elaborado de modo a não gerar confusão entre *bugs* e *improvements*. Segundo eles, os propositores de *issues* têm o hábito de confundir esses dois tipos de *issues*. O trabalho mostra que, em média, 39% das *issues* classificadas pelos desenvolvedores (propositores), como *bugs*, não o são. Além disso, os autores afirmaram que o processo de classificação deve passar fortemente por inspeção humana, que compensará os resultados do trabalho de predição. Os autores desejaram a longo prazo, melhores levantamentos de dados levarem a melhores recomendações, que farão os desenvolvedores mais conscientes de manter a qualidade das informações nos *issue trackers systems*.

O presente trabalho foi baseado fortemente sobre o estudo apresentado em [Herzig, Just e Zeller 2013], utilizando mesmos *dataset* e propósito de aumentar a acurácia de classificação das *issues* nos *issue trackers*. Os autores do trabalho de referência coordenaram a inspeção das *issues* por sujeitos humanos. O resultado dessa inspeção produziu um gabarito de correção da classificação realizada pelos desenvolvedores. Essas informações das *issues*, categorias aplicadas pelos humanos especialistas em classificação baseando-se no texto delas, foram utilizadas para a concepção do classificador desta pesquisa. Também, aquele trabalho se concentrou em estudos de classificação manual de *issues* utilizando as categorias *bug*, *others*, *request for enhancement*, *documentation*, *improvement* e *refactoring*. Para esta pesquisa foi realizado o estudo classificatório de *issues* automatizado, discriminado-as em categorias *bug* e *others* em um dos estudos, e *bug*, *request for enhancement* e *improvement* no outro estudo. A principal diferença entre as pesquisas foi a

classificação manual das *issues* realizada em [Herzig, Just e Zeller 2013] e a classificação automatizada das *issues* realizada nesta pesquisa.

Os autores do trabalho [Ray et al. 2014] investigaram o efeito das linguagens de programação sobre a qualidade de *software*. Como parte do trabalho, realizou-se a categorização dos *bugs* baseando-se em sua causa e impacto. Os autores definiram um conjunto de palavras-chave e aplicaram uma técnica automatizada sobre 10% do *log* de mensagens *bug-fix* ou *log* de mensagens de erros e correções sobre projetos de *software*. As mensagens foram coletadas de projetos localizados no repositório GitHub. A técnica determinava a discriminação da categoria "Erros de Conconcorrência" se, por exemplo, a mensagem continha as palavras-chave *dead lock* e *race condition*. Após processar as mensagens por meio dessa técnica automatizada, utilizaram-nas como dados de treinamento do classificador que construíram. O algoritmo de classificação utilizado para a avaliação do estudo foi o SVM. Para medir a acurácia do classificador produzido pelo estudo, os autores classificaram 180 mensagens, igualmente distribuídas entre as categorias definidas. Para essa classificação, utilizaram a análise humana de especialistas no assunto classificatório de categorias de *bugs*. No final do estudo, os resultados da classificação humana e da classificação automatizada foram comparados por meio dos cálculos de *precision* e *recall*. Após essa comparação, os autores observaram que as categorias *Performance, Security, Failure, Memory, Programming, Concurrency e Algorithm* obtiveram valor 83,71% para o melhor cálculo de *recall* e 84,34% para o melhor cálculo de *precision*.

Nesta pesquisa, foram utilizadas mais de 7000 *issues* classificadas por inspeção humana, nesse aspecto, a quantidade de registros inspecionados por sujeitos humanos foi superior ao de [Ray et al. 2014], o qual classificou manualmente 180 mensagens de texto de *logs*. Nesta pesquisa, foram definidas 3 categorias de classificação enquanto no estudo [Ray et al. 2014] foram definidas 7 categorias. Semelhantemente, os estudos classificatórios das duas pesquisas foram avaliados por múltiplos algoritmos e foram utilizadas as medidas *precision* e *recall* em conjunto com a aferição de acurácia do classificador automático de cada trabalho. O estudo de [Ray et al. 2014] utilizou a classificação automática de *bugs* como parte de sua abordagem para verificar os efeito das linguagens de programação sobre a qualidade de *software*. Por outro lado, esta pesquisa teve como abordagem a classificação automática de *issues* nos *issue trackers systems*.

O estudo realizado em [Campos Eduardo Cunha e Maia 2014] apresenta a construção de um classificador de Questões e Respostas para desenvolvedores de *software*. Foi

estudado um conjunto de questões do StackOverflow[5], um *site* popular sobre o assunto. As 120 questões selecionadas foram classificadas manualmente em três classes. Foram aplicados os algoritmos de classificação presentes na ferramenta Weka e também a *seleção de features*. O algoritmo que alcançou o maior resultado de classificação foi o *Naïve Bayes* que atingiu alta acurácia. Os autores pretenderam futuramente, melhorar a acurácia do classificador e descobrir como as categorias poderão ser usadas em novas aplicações.

Um aspecto de distinção entre esta pesquisa e o estudo realizado em [Campos Eduardo Cunha e Maia 2014] foi a quantidade de registros de texto avaliados por inspeção humana. Outra diferença entre os trabalhos foi o conteúdo de classificação, *issues* versus *questions and answers*. Uma semelhança observada é ambos definirem três categorias de classificação de texto. Também, ambos mediram a acurácia do classificador por meio de múltiplos algoritmos.

Em [Linares-Vásquez et al. 2014], os autores propuseram uma abordagem para a classificação automática de *software* de repositórios. Nessa abordagem, as categorias estão relacionadas com o domínio das aplicações. Os autores realizaram uma comparação com o estado da arte que utiliza *Aprendizado de Máquina* para categorização de *software*. O estudo foi realizado sobre dois repositórios, "um aberto (*open source*)" e "um particular". Foram empregados os algoritmos *Support Vetor Machine*, *Naïve Bayes*, *Decision Tree*, *JRIP* e *IBK* para a avaliação do classificador produzido. Nesse estudo, as chamadas de API foram utilizadas como *features* do classificador. Como resultado, foi obtida a acurácia média de 60%. Com esse resultado, sugeriram aos desenvolvedores utilizarem do trabalho para a obtenção de uma lista de possíveis categorias de domínio para os *softwares*, mesmo quando o código fonte não estiver disponível. Os autores também afirmaram que o conhecimento dessas categorias poderia ser utilizado para a antecipação de problemas e obtenção de *features* de qualidade do *software* ou para a extração de *bugs* e *features* relacionados.

No presente trabalho, o classificador automático de *issues* foi concebido sem acesso ao código fonte dos projetos, utilizou-se apenas as palavras-chave contidas nos textos das *issues*. No estudo [Linares-Vásquez et al. 2014] o classificador também foi concebido sem acesso ao código fonte, porém foram utilizadas chamadas de API e não análise de palavras-chave contidas nos textos das *issues*. Também, as categorias de classificação de cada estudo são distintas, apesar de ambos serem aplicados em repositórios de *software*.

[5] http://stackoverflow.com/

Uma semelhança da avaliação de acurácia dos classificadores foi a utilização de múltiplos algoritmos.

Capítulo 3

Categorização de *Issues* em Repositórios de Software

Nste capítulo é apresenta a metodologia qur foi utilizada para a concepção do classificador de *issues* proposto neste trabalho. É apresentada a evolução dos estudos para a construção do classificador que iniciam com a definição das *features* do classificador e continua com a aplicação das etapas da pesquisa fundamentadas pelos estudos sobre *Aprendizado de Máquina*. Os *toolkits* Weka e Mallet apoiaram a realização dessas etapas. Ao final do capítulo, é apresentada uma conclusão sobre os dois estudos, justificando a escolha daquele que originou o classificador concebido em estado final para este trabalho.

3.1 Construção do Classificador

A metodologia utilizada para a construção do classificador foi baseada em procedimentos existentes na literatura apresentados em [Linares-Vásquez et al. 2014] e [Ugurel, Krovetz e Giles 2002]. Também, a metodologia desta pesquisa foi desenvolvida com base em dois estudos piloto apresentados a seguir.

3.1.1 Estudo Piloto #1

Com base no trabalho [Ugurel, Krovetz e Giles 2002], no qual os autores apresentaram que as palavras-chave que discriminam os *bugs* podem ser utilizadas para sua predição, para o presente estudo foram definidas 31 *features*, representantes de palavras-chave discriminantes das classes *refactoring*, *bug* e *improvement*. Lista-se, a seguir, estas *features*:

{refactor, add parameter, change name, rename, move, move method, extract, extract class, extract interface, extract method, replace, hide, hide method, move field, pull up, inline class, inline method, remove parameter, replace code, bug, fix, exception, defect, fault, error, null pointer, crash, overflow, add funcionality, improve, upgrade}

Nota-se que no presente estudo a classe *refactoring* não havia sido substituída pela classe *rfe*, substituição justificada na Seção 5.3. Por isso as *features* listadas anteriormente contêm palavras-chave ainda referentes a essa classe. Essas *features* foram definidas com base em testes de busca das palavras-chave que possuem a mesma nomenclatura dessas *features*. As buscas foram realizadas nas mensagens de texto das *issues* armazenadas na base de dados relacional que constituiu o *dataset*.

Com o avanço do estudo experimental, esse conjunto de *features* sofreu mudanças com o objetivo de ser melhorado, para a produção de melhores resultados de acurácia, e possibilitar aumento da eficiência de execução dos algoritmos de classificação. Outro motivo para a mudança desse conjunto foi a necessidade de substituição da classe *refactoring* pela classe *request for enhancement*.

Em seguida, foi aplicado o algoritmo de aprendizado de máquina *Naïve Bayes*. Esse algoritmo de avaliação da classificação das *issues* foi a escolha inicial partindo da premissa que ele é adequado para situações em que as *features* do classificador possuem grau de dependência, apesar de utilizar como premissa a independência entre as *features* observando-se as categorias de classificação, conforme apresentado em [Rish 2001]. Nesta presente pesquisa assumiu-se que o primeiro conjunto de *features* definido possuía grau de dependência desconhecido, porém não era nulo ao se observar que existia *features*, como é o caso de "rename", que estava presente em *issues* classificadas pelos especialistas em classes distintas como as classes *improvement* e *refactoring*. Em seguida, aplicou-se *seleção de features* para a melhoria do conjunto de *features*.

Este primeiro estudo foi realizado com o apoio do *toolkit* Weka [Hall et al. 2009]. A utilização dessa ferramenta trouxe a contribuição de poder trabalhar com o processo de *Aprendizado de Máquina*, durante esse estudo, de modo transparente através das funcionalidades que representaram a execução das tarefas do processo. Porém, foram necessários muitos passos iterativos para se chegar aos resultados do estudo, o que estabeleceu a de-

pendência da habilidade do pesquisador durante a realização dos experimentos, para não ocorrer o manuseio inadequado da ferramenta.

Uma vez que o *dataset* utilizado no estudo experimental encontrava-se em uma base de dados relacional, foi possível exportar o vetor de *features*, representado como uma tabela da base de dados, para o formato *csv*, e depois, processá-lo no Weka. O idioma do texto submetido ao Weka para os estudos de classificação foi a língua inglesa.

A seguir, mostram-se os passos que descrevem a metodologia executada no primeiro estudo:

1. Realização de *download* do *dataset* no qual foram aplicados os estudos.
2. Realização de testes de compreensão do *dataset*, a fim de entender sua estrutura. Por exemplo, observou-se que cada esquema do banco de dados correspondia a um projeto do repositório.

    2.1 Extração de um conjunto de tabelas de banco de dados comuns aos projetos para a realização dos estudos.
    2.2 Identificação da tabela referente a *issues* e da tabela referente aos *commits*.

3. Execução de uma varredura inicial na base de dados para verificar a existência de palavras-chave nas mensagens de texto de *issues* e *commits*. Essas palavras-chave representariam as *features* do classificador, tendo em vista as classes *refactoring*, *bug* e *improvement*.
4. Criação, em cada esquema de projeto, dentre os 5 eleitos, de uma tabela com as *features* definidas para o classificador. Em seguida foi implementado um procedimento do banco de dados que verifica cada mensagem de texto e atribui 1, se a *feature* que representa a palavra chave está presente, ou 0 se não.
5. Exportação da tabela da base de dados representante do vetor de *features* para um arquivo de formato *csv*.
6. Importação do arquivo de formato *csv* produzido no passo anterior para o arquivo de entrada da ferramenta Weka, que possui extensão *arff*.
7. Refinamento do estudo experimental.

No passo 5, destaca-se que a ferramenta Weka possui um mecanismo que converte o vetor de *features* no arquivo de formato *csv* para um arquivo base da ferramenta que recebe o formato *arff*. Esse arquivo de extensão *arff* contém a definição das *features*

e os dados que podem ser carregados na ferramenta para a realização dos estudos de classificação. Assim, a *tabela de features* foi exportada para o arquivo de formato *csv*, para a condução dos estudos.

No passo 6, por meio da importação dos dados no Weka, o desenvolvimento do classificador pôde ser conduzido. Existe a interface de usuário na ferramenta que permite a escolha dos algoritmos de classificação, *K Nearest Neighbour (KNN)*, *Naïve Bayes* e outros. Outros parâmetros podem ser alterados utilizando essa interface, como a quantidade de *folds* para *cross validation*. Essa quantidade de *folds* representa as divisões do conjunto de dados para sua utilização durante *cross validation*.

Nesse passo, aplicou-se divisão semelhante em quantidade de partes divididas, com a divisão *5-fold* tradicional, utilizada para realizar *cross validation*. Nesse tipo de divisão, divide-se o conjunto de dados em 4 partes para o treinamento do classificador e 1 parte para o teste do classificador. Conforme observado na configuração do experimento #1 apresentado na Seção 4.1, foi utilizada 1 parte para o treinamento do classificador e 4 partes para o teste do classificador, seguindo idéia inversa a da divisão *5-fold*. As configurações de *cross validation* dos experimentos #1 e #2, também apresentados na Seção 4.1, não foram utilizadas nessa parte do estudo por ele ter sido descontinuado conforme justificado na Subseção 3.1.3.

No passo 7, realizando uma sequência de execuções e de avaliações de resultados, conduziu-se o refinamento do experimento aplicando *seleção de features*. Por meio desta técnica de melhoria de classificadores de dados baseados em *Aprendizado de Máquina*, as *features* escolhidas podem ser ranqueadas e pode-se reduzir o tamanho do vetor de amostras, possibilitando a diminuição do tempo de execução da classificação e, possivelmente, aumentar a acurácia do classificador.

O gabarito utilizado no trabalho de referência [Herzig, Just e Zeller 2013] foi o mesmo utilizado neste estudo. Esse gabarito foi produzido por especialistas em classificação de *issues*. Assim, devido as amostras estarem previamente classificadas, esta parte do estudo foi realizada de maneira supervisionada.

Este estudo piloto constituiu o primeiro estudo para a concepção do classificador de *issues*. Os resultados dele são apresentados na Seção 4.2.

3.1.2 Estudo Piloto #2

Neste estudo, as etapas apresentadas na Seção 2.1 foram abordadas utilizando da mesma definição do primeiro estudo piloto, no qual as palavras-chave discriminantes de *bugs* foram utilizadas para a sua predição, como foi descrito em [Linares-Vásquez et al. 2014].

Para o estudo, foi utilizado um *dump* de banco de dados com os textos das *issues*. A *estrutura de índice* do banco foi utilizada para as pesquisas sequenciais no *dataset* e o *Corpus* foi formado pelos campos das tabelas do banco de dados contendo as informações descrição, resumo e comentários das *issues*.

Para este estudo, a etapa de transformação (*transformation*) consistiu em aplicar redução das palavras ao radical (*stemming*), desconsiderar os textos dentro de *tags html* que porventura foram armazenadas no banco de dados dos *issue trackers* e remover *stopwords* ou palavras pouco significativas. O *stemming* foi aplicado aos arquivos de texto utilizando o algoritmo Porter [Porter 2006]. Podia ter sido aplicada a *lematização*, porém, foi escolhido arbitrariamente aplicar *stemming*.

Este estudo também foi realizado de maneira supervisionada, ou seja, as amostras de texto das *issues* utilizadas para o treinamento do classificador estavam previamente classificadas. A tarefa de classificação das *issues* foi realizada pelos especialistas do trabalho de referência [Herzig, Just e Zeller 2013].

Neste segundo estudo foi utilizado outro *toolkit* de suporte a experimentos de *Aprendizado de Máquina*, o Mallet. A utilização dessa ferramenta trouxe como contribuição a possibilidade de trabalhar com menor transparência em relação ao processo de *Aprendizado de Máquina*, não exigindo tanta habilidade do pesquisador com o manuseio dessa ferramenta em relação ao Weka. Foram necessárias iterações menores para se chegar aos resultados do segundo estudo.

O Mallet possui como mecanismo de importação do *dataset*, arquivos texto em formato pré-definido bastante próximo à estrutura de um arquivo *csv*. Porém, estes arquivos de entrada foram gerados por meio de um procedimento de banco de dados desenvolvido pelo humano que realizou o estudo, de modo semelhante a importação feita no estudo piloto #1.

Após os arquivos texto serem gerados, foram executados comandos da ferramenta Mallet, executados a partir do *shell sistema operacional* onde a ferramenta foi instalada,

que realizaram a importação dos arquivos texto para formato de arquivo específico da ferramenta. Este arquivo possui o *vetor de features* o qual foi especificado na Seção 2.1, populado automaticamente pela ferramenta nesta fase de importação.

Durante esta pesquisa as *features* do classificador foram formadas por *uni-gram* e *bi-gram*. Para definição dessas *features*, foi instruído à ferramenta um parâmero apropriado que possibilitou a remoção das *stopwords*, palavras pouco significativas.

As *stopwords* são palavras irrelevantes para a produção do classificador. A ferramenta Mallet possui uma lista própria destas palavras constituída por conjunções, preposições, artigos, pronomes e outros tipos de palavras determinadas por regras gramaticais oriundas do idioma dos textos que foram classificados. O idioma inglês foi utilizado como idioma padrão deste estudo.

Assim, palavras como “for”, “in”, “at”,“yes”, “yet”, “you” e “your” na maioria dos estudos de classificação de texto são consideradas pouco significativas e fazem parte da lista padrão do Mallet. Além destas palavras, algumas outras foram indicadas nesta lista, após a análise das mensagens das *issues*, com o objetivo de permitir que o classificador fosse formado por *features* genéricas, sem a influência de termos específicos dos textos avaliados. São listadas a seguir, as palavras incluídas na lista de *stopwords* além daquelas padronizadas pelo Mallet:

> “tomcat”, “rhino”, “apache”, “jackrabbit” “httpclient”, “tomcat5”, “lucene”, “classes”, “ul”, “li”, “package”, “emoticon”, “blockquote”, “api”, “interface”, “align”, “emoticons”, “absmiddle”, “icons”, “img”,“alt”, “good”, “images”, “height”, “smile”, “things”, “idea”, “gif”, “width”, “border”, “fast”, “slower”, “jira”, “interfaces”, “yeah”, “modules”, “lot”, “easier”, “great” “https”, “benchmark”, “uwe”, “usage”, “sense”, “square”, “javadocs”, “objects”, “faster”, “lang”, “backwards”, “cost”, “stuff”, “level”, “jsp”, “methods”, “patch”, “big” , “directly“, “small”, “hard”, “nice”, “people”, “memory”, “browse”, “number”, “title”, “class”

Em seguida, a ferramenta Mallet foi utilizada como apoio para a definição das *features* automaticamente, utilizando LDA [Blei, Ng e Jordan 2003]. As *top* 20 *features*, que são mais significativas no processo classificatório, foram listadas na Tabela 7.

Falando da configuração de *cross validation* planejada para este estudo piloto, foi seguido o exemplo *cross validation 10-fold* do estudo [Antoniol et al. 2008]. Esta configuração pode ser observada nos experimentos #2 e #3, que são apresentados na Seção

4.1. Também, foi aplicada a configuração para *cross validation* do experimento #1, semelhante na quantidade de divisões do conjunto de dados de entrada a *cross validation 5-fold* tradicional.

Este estudo piloto constituiu o segundo estudo para a concepção do classificador de *issues*. Os resultados dele são apresentados na Seção 4.2.

3.1.3 Conclusão

Os resultados apresentados na Sebseção 4.2.2 mostram a acurácia do classificador obtida com a aplicação do algoritmo *Naïve Bayes*. Outros algoritmos foram aplicados, como o KNN. Porém, esses algoritmos obtiveram valores baixos de acurácia em comparação ao *Naïve Bayes*.

Para melhorar os resultados de acurácia entre os algoritmos do "estudo piloto #1", que obteve acurácia média 66%, foram aplicados ajustes das *features*, como substituição, inserção e remoção dessas *features* representadas pelas palavras-chave. Para realizar esses ajustes, foram executadas pesquisas por novas palavras-chave sobre o *dataset*.

Porém, após os ajustes, não foi obtido um conjunto de *features* que produzisse o aumento da acurácia média 66%. Assim, foi realizado o "estudo piloto #2", no qual foram obtidos os resultados apresentados na Seção 4.2.3. Nesses resultados, pode-se notar valores médios de acurácia superiores a 66%, alcançando o valor aproximado 81% no experimento "Validação dos projetos separados" realizado para a classificação das *issues* em duas categorias, durante o auge da realização dos experimentos executados no estudo.

Assim, o aumento da acurácia média do classificador durante a realização dos experimentos do "estudo piloto #2", determinaram a sua continuidade e a descontinuidade do "estudo piloto #1".

Capítulo 4

Estudo Experimental

Neste capítulo é apresentado o planejamento do estudo experimental, dando foco às informações das amostras, como os projetos onde o classificador foi testado e no modo de avaliação do classificador baseada na configuração da validação cruzada. O estudo experimental foi dividido em três partes, cada uma delas refere-se a uma configuração. Também, são apresentados os algoritmos utilizados em cada parte do estudo e o resultado de comparação com aqueles obtidos nesta pesquisa.

4.1 Planejamento do Estudo Experimental

Inicialmente tentou-se produzir um *dataset* utilizando uma API codificada com a linguagem de programação Java, para o repositório de *software* GitHub. Foram realizados estudos das classes da API representantes de entidades que seriam traduzidas para um banco de dados representando o*dataset*. Porém, houve restrições de *downloads* do repositório que dificultaram o trabalho. Essas restrições consistiram na disponibilidade de acesso às informações dos *commits* e *issues* dos projetos armazenados no repositório de *software*. As pesquisas com a API foram limitadas em quantidade e velocidade, tornando-se inviável a recuperação das informações para a aplicação dos estudos desta pesquisa. Assim, passou-se a buscar um *dataset* concebido.

Foi encontrado um *dataset*[1] representado em uma base de dados relacional. Esse *dataset* foi disponibilizado em um *database dump* constituído de 86 arquivos separados, cada um de 1 *gigabytes* de tamanho, totalizando 86 *gigabytes* no total. O *dataset* foi

[1] http://macbeth.cs.ucdavis.edu/jirdatadump/

disponibilizado por Prem Devanbu[2] e referenciado no trabalho [Ray et al. 2014], o qual apresentou um estudo das linguagens de programação em repositórios de *software* utilizando a classificação de *bugs* como parte do estudo experimental. Nesse *dataset* existiam 118 projetos de *software*, onde cada *esquema de banco de dados* representava um projeto. Cada projeto continha pelo menos 8 tabelas de banco de dados em comum, provendo informações de *issues* e *commits*. Alguns projetos possuíam os dados de *issues* armazenados nos *issue trackers systems* Jira[3] e bugzilla[4]. Esse *dataset* foi explorado durante o estudo apresentado na Subseção 3.1.1 desta pesquisa.

A segunda parte do estudo experimental, apresentada na Subseção 3.1.2, foi realizada sobre os cinco projetos de *software open source* listados na Tabela 1. Nessa tabela, são mostrados o nome do projeto, o mantenedor que compreende a instituição responsável pelo projeto, o tipo de *issue tracker* de onde foram coletadas as *issues* do projeto correspondente e, na última coluna, é apresentada a quantidade de *issues* existentes no projeto em questão.

Este estudo foi divido em três experimentos que definiram a aplicação do classificador sobre as *issues*. Esses experimentos foram configurados baseando-se no processo de avaliação do classificador, com a qual foi aplicada a validação cruzada ou *cross validation*. Esse processo determina a divisão do *dataset* constituído dos projetos avaliados em duas partes:

- *training set*, utilizado para o treinamento do classificador,
- *test set*, onde o classificador foi testado.

Essa divisão se faz necessária como parte do processo de *Aprendizado de Máquina*, sendo utilizada para a aferição da acurácia da classificação. Cada avaliação do classificador, também denominada validação do classificador, refere-se a um experimento.

4.1.1 Experimento #1 - Validação Um para Restante

O primeiro experimento foi denominado "Validação Um para Restante". O nome foi dado baseando-se que o *training set* foi formado por cada projeto isolado dos outros. Os demais projetos foram concatenados para a formação do *test set*. Esse experimento

[2] http://web.cs.ucdavis.edu/ devanbu/
[3] https://www.atlassian.com/software/jira
[4] http://www.bugzilla.org/

Tabela 1 – Detalhes dos Projetos.

Nome Projeto	**Mantenedor**	**Issue Tracker**	**# Issues**	**Url Projeto**
HttpClient	APACHE	Jira	746	http://hc.apache.org/httpclient-3.x/
Jackrabbit	APACHE	Jira	2402	https://jackrabbit.apache.org/jcr/
Lucene-Java	APACHE	Jira	2443	https://jackrabbit.apache.org/jcr/
Rhino	MOZILLA	bugzilla	1226	https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino
Tomcat5	APACHE	bugzilla	584	https://tomcat.apache.org/tomcat-5.5-doc/

foi configurado de modo que cada projeto se comportou como conjunto de dados de treino ou *training set* e os demais formaram o conjunto de dados de teste ou *test set*, em uma validação. Como foram 5 projetos avaliados, no total foram realizadas 5 validações, apresentadas na Tabela 2.

Tabela 2 – Configuração do Experimento 1

# Validação	**Training Set**	**Test Set**
1	HttpClient	Jackrabbit,Lucene-Java,Rhino,Tomcat5
2	Jackrabbit	HttpClient,Lucene-Java,Rhino,Tomcat5
3	Lucene-Java	HttpClient,Jackrabbit,Rhino,Tomcat5
4	Rhino	HttpClient,Jackrabbit,Lucene-Java,Tomcat5
5	Tomcat5	HttpClient,Jackrabbit,Lucene-Java,Rhino

4.1.2 Experimento #2 - Validação dos Projetos Separados

O segundo experimento foi denominado "Validação dos Projetos Separados". Essa denominação foi dada pelo fato do *trainnig set* e do *test set* serem definidos internamente a cada projeto, gerando uma validação interna para cada projeto separado dos demais. A configuração da validação cruzada para este experimento foi realizada com *10-fold*. Isto significa que cada projeto foi dividido em 10 partes iguais, sobre as quais a validação foi executada. A Tabela 4 mostra essa configuração. Visando facilitar a apresentação dessas partes nas colunas *test set* e *training set* da tabela, foi dado um rótulo a elas que varia da letra A até a letra J, seguindo sequencialmente a ordem alfabética da língua portuguesa brasileira. É mostrada na Tabela 3 essa rotulagem em que *[Projeto]* pode ser substituído por HttpClient, Jackrabbit, Lucene-Java, Rhino ou Tomcat5. A configuração desse experimento é apresentada na Tabela 4.

Tabela 3 – Tabela de Rótulos

[Projeto]	Rótulo
1º 10%	A
2º 10%	B
3º 10%	C
4º 10%	D
5º 10%	E
6º 10%	F
7º 10%	G
8º 10%	H
9º 10%	I
10º 10%	J

Tabela 4 – Configuração do Experimento 2

# Validação	Test Set	Training Set
1	A	B,C,D,E,F,G,H,I,J
2	B	A,C,D,E,F,G,H,I,J
3	C	B,A,D,E,F,G,H,I,J
4	D	B,C,A,E,F,G,H,I,J
5	E	B,C,D,A,F,G,H,I,J
6	F	B,C,D,E,A,G,H,I,J
7	G	B,C,D,E,F,A,H,I,J
8	H	B,C,D,E,F,G,A,I,J
9	I	B,C,D,E,F,G,H,A,J
10	J	B,C,D,E,F,G,H,I,A

4.1.3 Experimento #3 - Validação Todos para Todos

O terceiro experimento foi denominado "Validação Todos para Todos". Esse nome foi dado ao experimento por ser caracterizado pela concatenação dos projetos para a formação dos *test set* e *training set* a partir do "Todo" produzido por essa concatenação. Esse experimento se assemelha ao segundo na representação das validações. A diferença entre eles é, no experimento #3 serem somados os i-ésimos 10% de cada projeto em cada validação, de um total de 10 validações. Apresenta-se na Tabela 5 a rotulagem que expressa essa soma. Deve-se considerar a utilização dos rótulos da Tabela 3 na soma. Também, deve-se observar que os rótulos representados pelo intervalo de letras de A até J possuem o mesmo significado apresentado na Tabela 3, porém aparecem seguidos por uma numeração que varia de 1 a 5. Considera-se que o número 1 representa o projeto HttpClient, o número 2 representa o projeto Jackrabbit, o número 3 representa o projeto Lucene-Java, o número 4 representa o projeto Rhino e o número 5 representa o projeto Tomcat5. O processo de validação cruzada do experimento #3 é apresentado na Tabela 6.

Tabela 5 – Tabela de rotulagem 2

Rótulo	**Soma Equivalente**
A’	A’= A1 + A2 + A3 + A4 + A5
B’	B’= B1 + B2 + B3 + B4 + B5
C’	C’= C1 + C2 + C3 + C4 + C5
D’	D’= D1 + D2 + D3 + D4 + D5
E’	E’= E1 + E2 + E3 + E4 + E5
F’	F’= F1 + F2 + F3 + F4 + F5
G’	G’= G1 + G2 + G3 + G4 + G5
H’	H’= H1 + H2 + H3 + H4 + H5
I’	I’= I1 + I2 + I3 + I4 + I5
J’	J’= J1 + J2 + J3 + J4 + J5

Tabela 6 – Configuração do Experimento 3

# Validação	**Test Set**	**Training Set**
1	A’	B’,C’,D’,E’,F’,G’,H’,I’,J’
2	B’	A’,C’,D’,E’,F’,G’,H’,I’,J’
3	C’	B’,A’,D’,E’,F’,G’,H’,I’,J’
4	D’	B’,C’,A’,E’,F’,G’,H’,I’,J’
5	E’	B’,C’,D’,A’,F’,G’,H’,I’,J’
6	F’	B’,C’,D’,E’,A’,G’,H’,I’,J’
7	G’	B’,C’,D’,E’,F’,A’,H’,I’,J’
8	H’	B’,C’,D’,E’,F’,G’,A’,I’,J’
9	I’	B’,C’,D’,E’,F’,G’,H’,A’,J’
10	J’	B’,C’,D’,E’,F’,G’,H’,I’,A’

4.1.4 Divisão dos Experimentos Classificatórios

Os experimentos classificatórios foram aplicados em duas partes: classificação das *issues* em múltiplas categorias e classificação em duas categorias. A primeira parte representou a classificação das *issues* nas classes *bug*, *rfe* e *improvement*, constituindo o objetivo final de classes discriminadas pelo classificador concebido neste trabalho.

A segunda parte foi elaborada para enriquecer as conclusões sobre o classificador de *issues* concebido neste trabalho. O mesmo estudo foi realizado no trabalho [Herzig, Just e Zeller 2013]. Este estudo consiste na classificação das *issues* em *bugs*, classe que representa os pedidos de correção de *software* e *others* que engloba qualquer outra classe que não seja *bug*. Desse modo, o estudo pode ser visto como a classificação binária entre *bugs* e *não bugs*.

Na primeira parte, o experimento de classificação em múltiplas categorias foi realizado utilizando os dois *toolkits*, Weka e Mallet. Por isso, tornou-se conveniente subdividi-lo em

duas partes sendo cada uma apresentando os resultados obtidos utilizando cada *toolkits*. A segunda parte, que aborda a classificação em duas categorias, não foi subdividida por ter sido realizada inteiramente utilizando-se o Mallet.

O estudo apresentado na Seção 3.1.1 é equivalente a essa primeira parte. Para esse estudo, foi executado apenas o algoritmo *Naïve Bayes*. Isto ocorreu devido a falta de evolução dos resultados para esse algoritmo utilizado para medir o desempenho de classificadores.

Para o outro estudo, equivalente à segunda parte e apresentado na Subseção 3.1.2, foram utilizados os algoritmos *Max Entropy*, *Decision Tree*, *Naïve Bayes*, *Winnow* e *Balanced Winnow* para a avaliação do classificador. Estes algoritmos foram escolhidos, por serem os principais algoritmos disponíveis no Mallet, além de terem sido aplicados em outros estudos [McCallum, Freitag e Pereira 2000], [Landgrebe 1991], [McCallum, Nigam et al. 1998], [Golding e Roth 1999], [Koster, Seutter e Beney 2003].

4.1.5 Acurácias de Referência

Após a análise dos resultados obtidos pela avaliação dos algoritmos, é apresentada a conclusão desses resultados por meio da comparação com acurácias de classificação das *issues* atingida pelos desenvolvedores que as propõe e as classifica nos *issue trackers*. Essas acurácias foram obtidas do trabalho [Herzig, Just e Zeller 2013].

Os autores do estudo avaliaram cinco projetos, que são os mesmos avaliados neste estudo, e encontraram que 42,6% das *issues* reportadas estavam classificadas de forma incorreta, considerando a classificação delas em múltiplas categorias. Com base nessa análise, concluímos que a taxa de acertos dos desenvolvedores foi de 57,4% sobre os projetos avaliados.

Em relação a classificação de *issues* em duas categorias, *bug* e *others*, o estudo de [Herzig, Just e Zeller 2013] apresentou que 39% das *issues* classificadas como *bug* pelos desenvolvedores, não refletiam essa categoria.

Esses dois resultados foram utilizados para as comparações com os resultados dos experimentos deste estudo, com a finalidade de produzir uma boa experiência de classificação automática de *issues* e justificar a utilização de classificadores automáticos nos *issue trackers systems*.

Na próxima seção, são apresentados os resultados do estudo planejado para esta pes-

quisa.

4.2 Análise dos Resultados

4.2.1 Introdução

Nesta subseção, são apresentados os resultados do classificador de *issues* concebido nesta pesquisa. Esses resultados são baseados no planejamento realizado na Seção 4.1 e refletem a avaliação de desempenho do classificador. São apresentados resultados medidos pelos algoritmos com as configurações apresentadas nas tabelas 2, 4 e 6. Na sequência, é apresentada a conclusão sobre os resultados alcançados.

4.2.2 Resultados do Estudo Piloto #1

4.2.2.1 Classificação de *Issues* em múltiplas categorias

Os resultados a seguir apresentam a aplicação do experimento "Validação Um para Restante", referentes à aplicação do algoritmo *Naïve Bayes* sobre os projetos avaliados.

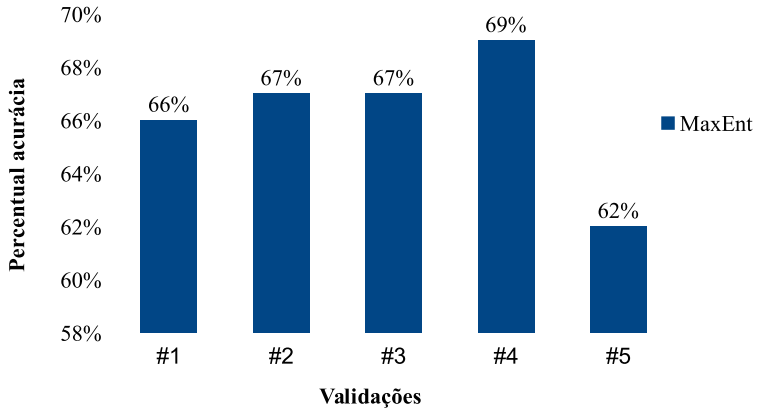

Figura 2 – "Validação Um para Restante", acurácia medida para validação composta na Tabela 2.

Pode-se observar na Figura 2 que o resultado médio de avaliação dos atributos classificatórios do estudo piloto #1, os quais discriminaram os textos das *issues* nas classes *bug*, *improvement* e *refactoring*, foi de aproximadamente 66% de acurácia. Também, pode-se notar que as validações configuradas na Tabela 2 atingiram valores de acurácia no intervalo de 61,84% a 68,75%.

4.2.3 Resultados do Estudo Piloto #2

4.2.3.1 Classificação de *Issues* em Múltiplas categorias

Na Tabela 7, são listados os 20 atributos do classificador ordenados pelo *infogain*, uma medida da importância de cada atributo no processo classificatório. Cada atributo em uma coluna dessa tabela é apresentado separado por "vírgula" em ordem crescente de importância para o processo de classificação, observando-se a tabela no sentido de cima para baixo. A frequência desses atributos é o valor apresentado após os "dois pontos". Pode ser observado que ter alta frequência não implica ter maior importância no processo classificatório, como é o caso do atributo *fix* que encontra-se na terceira posição, lendo uma coluna no sentido de cima para baixo. Sua frequência nos projetos é superior a frequência de atributos melhor ranqueados e de menores frequências como é o caso do atributo *improv* que encontra-se na primeira posição.

Tabela 7 – Top 20 *features* nos sistemas, ordenadas por *infogain*.

Httpclient	**Jackrabbit**	**Rhino**	**Lucene**	**Tomcat5**
improv:92	improv:314	improv:24	improv:1180	improv:54
add:401	add:809	add:73	add:448	add:305
fix:886	fix:2033	fix:347	fix:4748	fix:1889
implem:282	implem:535	implem:85	implem:1801	implem:189
support:322	support:531	support:67	support:1368	support:190
make:583	make:878	make:105	make:5163	make:494
featur:102	featur:244	featur:47	featur:393	featur:105
implement:302	implement:887	implement:132	implement:1665	implement:173
reproduc:106	reproduc:179	reproduc:100	reproduc:371	reproduc:280
bug:503	bug:310	bug:373	bug:1886	bug:1486
id:107	id:368	id:34	id:1149	id:128
depend:135	depend:569	depend:26	depend:926	depend:84
option:198	option:274	option:27	option:922	option:298
onward:0	onward:1	onward:0	onward:5	onward:351
creat_attach:631	creat_attach:2	creat_attach:260	creat_attach:290	creat_attach:600
report:162	report:354	report:52	report:410	report;402
commit:468	commit:758	commit:170	commit:5541	commit:193
readi:31	readi:19	readi:3	readi:448	readi:21
separ:97	separ:239	separ:26	separ:1322	separ:86
move:212	move:447	move:19	move:1515	move:98

Na Figura 3 são apresentados os resultados de avaliação da acurácia do classificador para o experimento "Validação Um para Restante", obtidos em cada sistema através da execução dos algoritmos escolhidos para a avaliação. Pode ser observado que o algoritmo que obteve o melhor desempenho para o classificador foi o *Max Ent* com acurácia média

de 59%. Uma possível justificativa para tal fato encontra-se na Seção 4.3. O algoritmo que obteve a pior medida de desempenho foi o *Winnow*, com acurácia média de 35%.

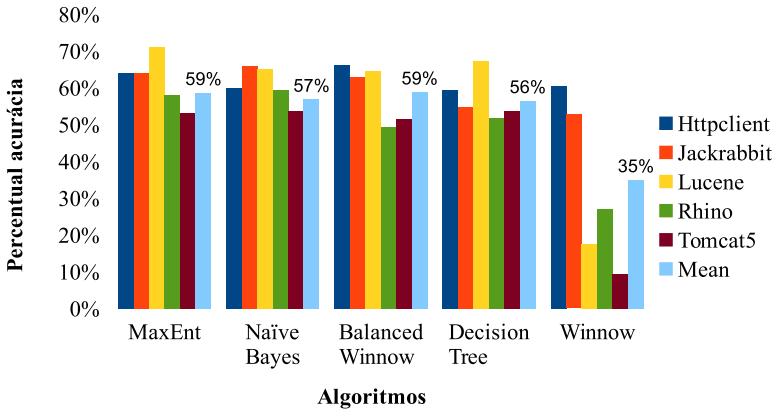

Figura 3 – "Validação Um para Restante", acurácia medida para validação composta na Tabela 2.

Na Figura 4, são apresentados os resultados de avaliação da acurácia do classificador para o experimento "Validação dos Projetos Separados", obtidos em cada sistema, por meio da execução dos algoritmos escolhidos para a avaliação. Pode ser observado que o algoritmo que obteve o melhor desempenho para o classificador foi o *Max Ent*, com acurácia média de 79%. O algoritmo com pior medida de desempenho foi o *Winnow* com acurácia média 69%.

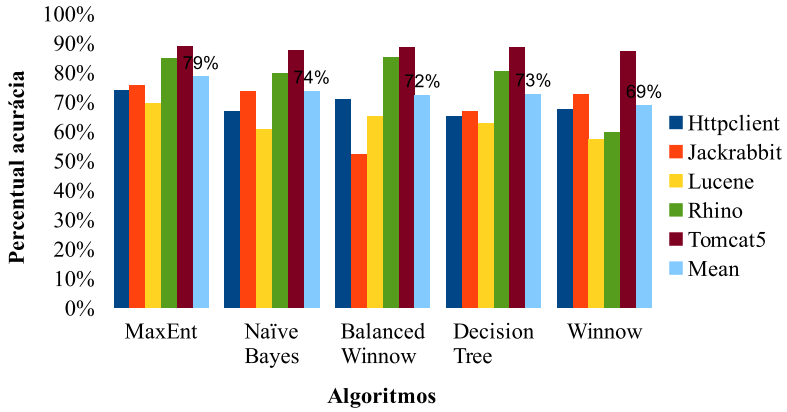

Figura 4 – "Validação dos Projetos Separados", acurácia medida para validação composta na Tabela 4.

Na Figura 5, são apresentados os valores percentuais obtidos da medida *F1* considerando as classe *bug*, *rfe* e *improvement*, aferidos utilizando-se o algoritmo *Max Ent* no

experimento #2, sobre o projeto *Tomcat 5*. Os resultados foram produzidos com o apoio da ferramenta Mallet. Pode ser observado que a classe *improvement* obteve valor 100%, seguida pela classe *bug* com 94% e em último lugar a classe *rfe* com 28%.

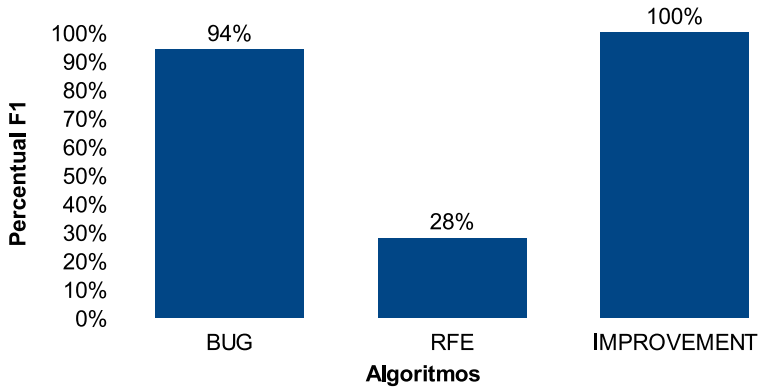

Figura 5 – Predição por classe aferida pelo algoritmo *MaxEnt* na "Validação dos Projetos Separados".

Na Figura 6, são apresentados os valores percentuais obtidos da medida *F1* considerando as classe *bug*, *rfe* e *improvement*, aferidos utilizando-se o algoritmo *Balanced Winnow* no experimento #2, sobre o projeto Tomcat 5. Os resultados de *F1* para as classes *improvement* e *bug* foram os mesmos atingidos pelo algoritmo *Max Ent*, 100% e 94% respectivamente. Para a classe *rfe* foi medido o valor 31%.

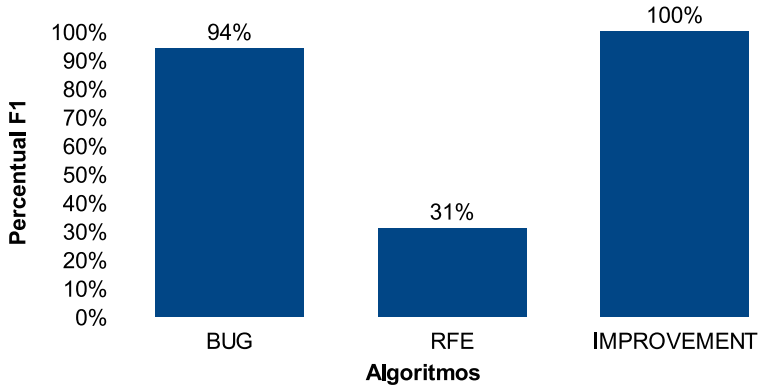

Figura 6 – Predição por classe aferida pelo algoritmo *Balanced Winnow* na "Validação dos Projetos Separados".

Na Figura 7, são apresentados os resultados de avaliação da acurácia do classificador para o experimento "Validação Todos para Todos", por meio da execução dos algoritmos

escolhidos para a avaliação. Pode ser observado o algoritmo o qual obteve o melhor desempenho para o classificador, *Max Ent*, com acurácia média 75%. O algoritmo com pior medida de desempenho foi o *Winnow* com acurácia média 63%.

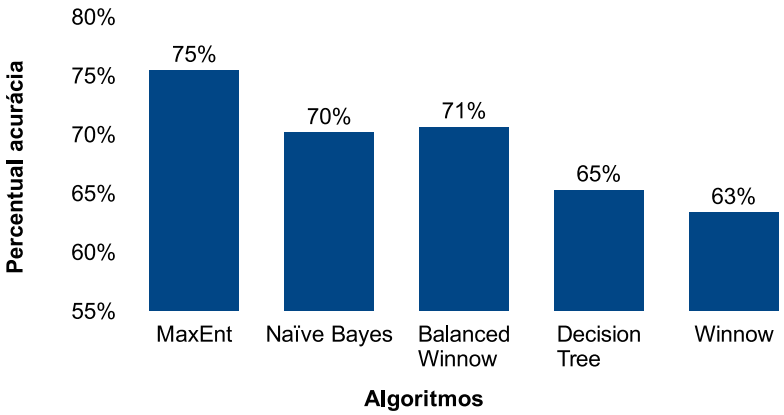

Figura 7 – "Validação Todos para Todos", acurácia medida para validação composta na Tabela 6.

Na Figura 8, são apresentados os valores percentuais obtidos da medida *F1* considerando as classe *bug*, *rfe* e *improvement*, aferidos utilizando-se o algoritmo *MaxEnt* durante execução do experimento #3. Para a classe *bug* foi obtido o valor 83,79%, seguida pela classe *improvement* com 67,15% e por último a classe *rfe* 28%.

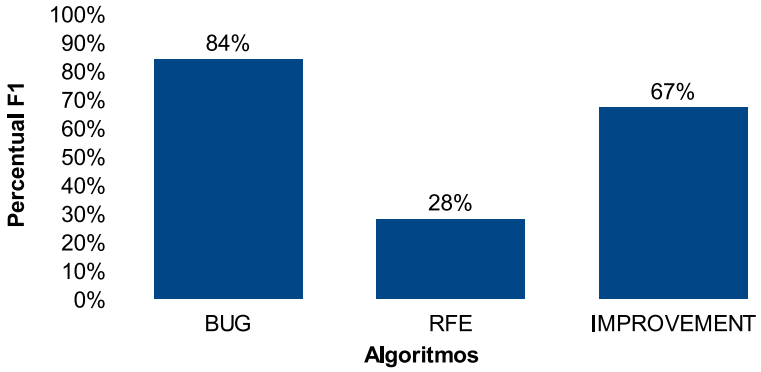

Figura 8 – Predição por classe aferida pelo algoritmo *MaxEnt* na "Validação Todos para Todos".

Na Figura 9, são apresentados os valores percentuais obtidos da medida *F1* considerando as classe *bug*, *rfe* e *improvement*, aferidos utilizando-se o algoritmo *Balanced*

Winnow no experimento #3. Para a classe *bug* foi obtido o valor 80%, seguida pela classe *improvement* com 62% e, por último, a classe *rfe* com *F1* 17%.

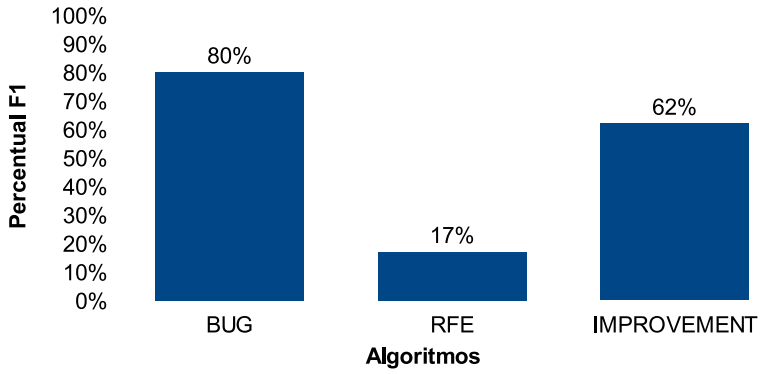

Figura 9 – Predição por classe aferida pelo algoritmo *Balanced Winnow* na "Validação Todos para Todos".

Na Figura 10, são apresentados os resultados de predição obtidos pelo algoritmo *Max Ent* no experimento #3, com podas de atributos. Pode ser notado que, para um conjunto mínimo de 10 atributos ranqueados por *infogain*, a acurácia medida foi aproximadamente 67%. Para a avaliação com 900 atributos, a acurácia alcançada foi de 72%. Para a quantidade máxima de atributos do classificador, 786.688 atributos, a medida de acurácia atinge o pico 75%.

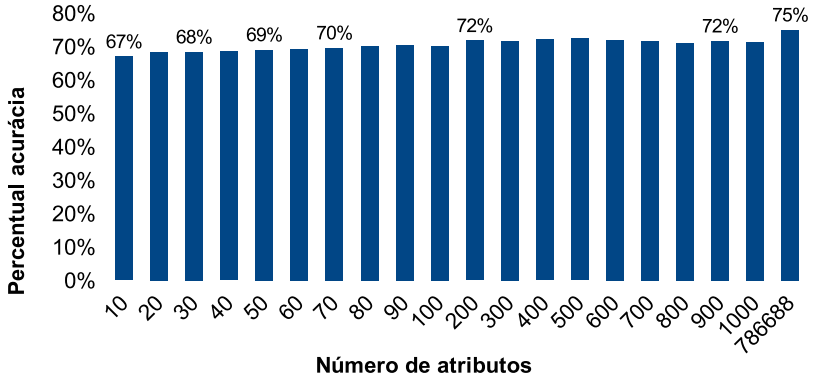

Figura 10 – Acurácia aferida pelo algoritmo *MaxEnt* em análise da "Validação Todos para Todos" com podas dos atributos.

4.2.3.2 Classificação de *Issues* em *bugs* e *Others*

Os resultados apresentados a seguir refletem a avaliação do classificador no estudo piloto #2. Nesse estudo, foi produzido o classificador em seu estado final. Para ele, foi utilizada a ferramenta de apoio Mallet, apenas.

Na Figura 11, são apresentados os resultados de avaliação da acurácia do classificador para o experimento "Validação Um para Restante", obtidos em cada sistema. Pode ser observado que o algoritmo com o melhor desempenho para o classificador foi o *Max Ent* com acurácia média de 66%. O algoritmo com pior medida de desempenho foi o *Winnow* com acurácia média de 52%.

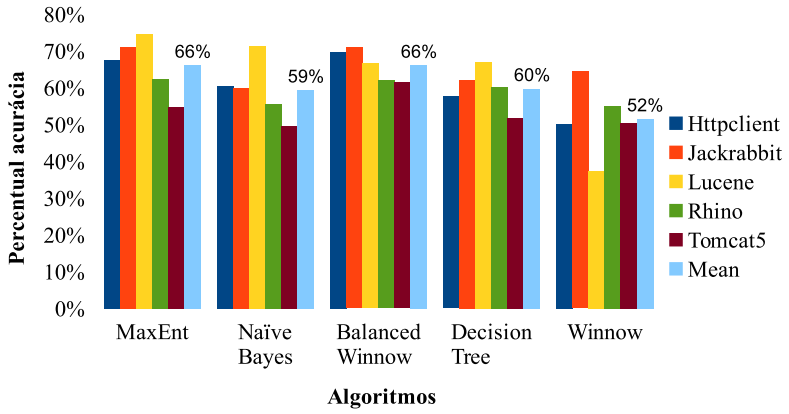

Figura 11 – "Validação Um para Restante", acurácia medida para validação composta na Tabela 2.

Na Figura 12, são apresentados os resultados de avaliação da acurácia do classificador para o experimento "Validação dos Projetos Separados", obtidos em cada sistema, por meio da execução dos algoritmos escolhidos para a avaliação. Pode ser observado que o algoritmo com o melhor desempenho para o classificador foi o *Max Ent* com acurácia média de 81%. O algoritmo com pior medida de desempenho foi o *Winnow* com acurácia média 68%.

Na Figura 13, são mostrados os resultados de avaliação da acurácia do classificador para o experimento "Validação Todos para Todos", por meio da execução dos algoritmos escolhidos para a avaliação. Pode ser observado que o algoritmo que obteve o melhor desempenho para o classificador foi o *Max Ent* com acurácia média de 80%. O algoritmo com pior medida de desempenho foi o *Decision Tree* com acurácia média de 66%.

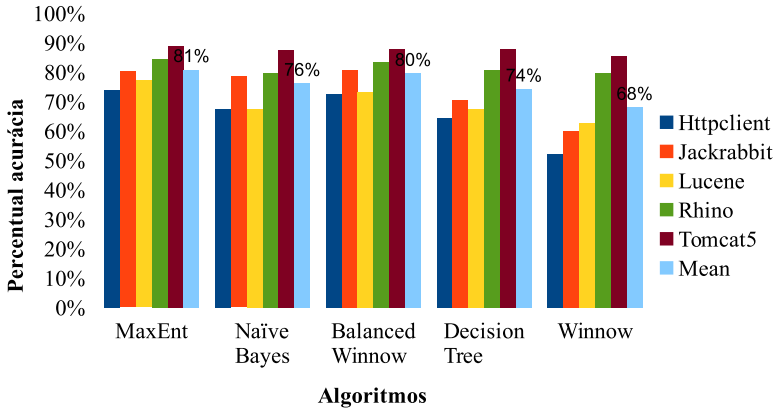

Figura 12 – "Validação dos Projetos Separados", acurácia medida para validação composta na Tabela 4.

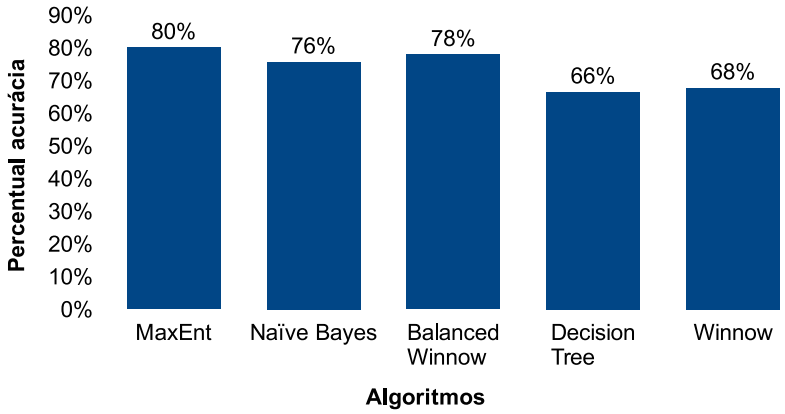

Figura 13 – "Validação Todos para Todos", acurácia medida para validação composta na Tabela 6.

4.3 Conclusão

Analisando os resultados da Seção 4.2, observa-se que o estudo experimental utilizando o *toolkit* Weka alcançou resultado de desempenho para o algoritmo *Naïve Bayes* com a média aproximada de 66%. Conclui-se que pode significar bom resultado utilizando o Weka com o conjunto de atributos do estudo piloto #1, em comparação com a acurácia atingida pelos desenvolvedores apresentada no Seção 4.1, com valor 57,4%.

Os resultados apresentados com a utilização do *toolkit* Mallet no estudo experimental principal desta pesquisa (Subseção 3.1.2), que produziu o classificador de *issues* em seu estado final, mostraram que, nos experimentos, o algoritmo que atingiu a melhor acurácia foi o *MaxEnt*.

O algoritmo *Max Ent* possui como mecanismo de funcionamento a procura por uma distribuição que maximiza a entropia entre as *features* do modelo. Esse algoritmo busca estabelecer uma distribuição mais uniforme possível. Para isto utiliza *constraints* (restrições) aplicadas a função exponencial de ajuste dos pesos. Essa função, possui regras de otimização que são potencializadas em determinados cenários, como o desse estudo em que são analisados documentos de texto. As regras estão ligadas ao fato dele não assumir hipótese de independência entre *features*. No cenário de classificação de texto, considerando-se as *features* sendo as palavras-chave que discriminam os tipos de *issues*, observa-se a existência de graú de dependência entre elas. Ainda favorecendo ao graú de dependência entre as *features*, os textos analisados pertencem ao mesmo domínio, de *issues*. Por existir graú de dependência entre as *features* sugestivamente alto (garantido em documentos de texto), permitindo alcance de otimização potencializada do algoritmo *Max Ent*, entende-se que por isso ele obteve destaque em relação aos demais, durante as aferições de acurácia do classificador.

Durante a classificação das *issues* em múltiplas categorias esse algoritmo obteve 79% de acurácia para o experimento #2. Isso significa ótimo resultado em comparação aos 57,4% de acurácia dos desenvolvedores, houve aumento de 21.6% na acurácia . A partir desse resultado, conclui-se que a classificação das *issues* nos *issue trackers systems* pode ser melhorada utilizando a classificação automática.

Os algoritmos da família *Winnow*, neste estudo representados pelo *Winnow* e *Balanced Winnow*, possuem uma função linear para o ajuste dos pesos que representa a frequência das *features* nos documentos analisados. São algoritmos dirigidos a erros (*mistake driven*), o que significa dizer que necessitam da ocorrência de boa quantidade de erros para completar seu mecanismo de aprendizado, uma vez que a função de ajuste da predição é sensibilizada quando ocorrem erros durante o treinamento do classificador.

O algoritmo *Winnow* possui apenas um vetor de pesos associado as *features*. Esses pesos podem ser valores negativos, quando a função preditiva contabiliza a ausência da *feature* em um documento analisado, ou valores positivos, quando contabiliza a presença da *feature*.

O algoritmo *Balanced Winnow* possui otimização representada pela utilização de dois vetores de pesos, o positivo e o negativo. Durante sua execução a função preditiva computa a diferença entre os valores de pesos armazenados nestes dois vetores. Esse ajuste o torna mais robusto em relação ao *Winnow*, em situações onde o *dataset* possui tamanho elevado,

como é o caso do *dataset* desse estudo. Devido a essa otimização entende-se que esse algoritmo superou o *Winnow* durante as aferições de acurácia do classificador.

A classificação em duas categorias mostrou que o algoritmo *MaxEnt* obteve a melhor medida de desempenho atingindo o maior valor de 80.85%. O estudo apresentado em [Herzig, Just e Zeller 2013] revelou que em média 39% das *issues* classificadas pelos desenvolvedores como *bug* não o são. Logo a acurácia dos desenvolvedores é de 61%. Comparando este resultado com aquele apresentado no estudo classificatório de *issues* em duas categorias, observamos o aumento de 19.85% da acurácia obtida pelo classificador automático em relação a classificação realizada pelos desenvolvedores. Esse resultado mostra que pode-se obter alta acurácia utilizando a classificação automática das *issues* para distingui-las entre *bugs* e *others*.

Uma importante conclusão sobre os resultados apresentados pode ser feita com a observação da variedade de projetos avaliados, de um total de cinco projetos, em diferentes *issues trackers systems*, num total de dois. Os bons resultados apresentados para o classificador concebido, emparelhados com as validações (*cross validation*), sugerem que o classificador desse estudo pode ser utilizado com alta acurácia em outros projetos de *software* para a classificação automática de *issues*.

De modo geral, conclui-se que a classificação automática de *issues* nos *issue trackers systems* pode ser realizada por meio de classificadores automáticos como o apresentado neste estudo. Dessa forma as *issues* podem ser endereçadas aos seus solucionadores por meio de sistemas de encaminhamento automático, com melhor qualidade de classificação do que aquela apresentada pelos desenvolvedores que as propõem. Isto contribuirá para o aumento da eficiência do tratamento das *issues* e diminuição do retrabalho das equipes de desenvolvimento de *software* que as tratam.

Capítulo 5

Considerações Finais

5.1 Resultados Alcançados

Os estudos realizados nesta pesquisa obtiveram como principal resultado a possibilidade de utilização de classificação automática de *issues* nos repositórios de manutenção de *software*. O estudo de classificação realizado com mensagens de texto das *issues*, obtidas dos *issue trackers* Jira e Bugzilla, apresentou resultados de acurácia superiores aos resultados dos seus propositores (desenvolvedores de *software*). Assim, os sistemas de encaminhamento automático de *issues* podem utilizar este estudo para obter melhor distribuição das *issues* às equipes solucionadoras. Dessa maneira, pode-se economizar o retrabalho das equipes de desenvolvimento de *software* das corporações e diminuir o tempo de encontrar a solução das tarefas de manutenção de *software*.

Um outro resultado que se destaca é a possibilidade de classificação automática das *issues* com um elevado valor de predição, como foi apresentado nos experimentos deste trabalho. O experimento "Validação dos projetos separados" apresentado na Subseção 4.1.2 obteve acurácia média de 79% como apresentado na Figura 4, para o estudo de classificação em múltiplas categorias. O mesmo experimento apresentou acurácia média de 81%, para o estudo de classificação em duas categorias, como apresentado na Figura 12.

Outro resultado importante é a possibilidade de classificação das *issues* quando o código fonte dos sistemas de *software* não estiver disponível.

Também, sugere-se que o classificador concebido nesta pesquisa possa ser utilizado em estudos empíricos para separar automaticamente, com maior acurácia, os diferentes tipos de *issues* e desta forma contribuir para a realização de análises sobre a qualidade e evolução dos sistemas, através da mineração de repositórios de software.

5.2 Limitações da Pesquisa

Esta pesquisa pode ser reproduzida em repositórios de *software* que possuem integração com *issues trackers*, como é o caso do GitHub, ou em sistemas *issue trackers* que constituem um *repositório de manutenção de software* para projetos variados, por exemplo, os *issue trackers* Jira e Bugzilla. Tratam-se de repositórios *open source*. A falta da avaliação de projetos em repositórios particulares pode representar uma ameaça ao classificador desta pesquisa em relação a sua diversidade de aplicação, uma vez que os critérios de documentação das *issues* nos repositórios particulares não foram representados neste estudo.

Este estudo foi baseado na inspeção humana das *issues* por especialistas do trabalho da principal referência [Herzig, Just e Zeller 2013]. Dessa forma, o classificador produzido reproduz os critérios de classificação desses especialistas, que podem divergir dos critérios de outros especialistas. Logo, a classificação de *issues* por meio deste classificador, ainda que acurada pode ser refutada em algum momento, por outros especialistas.

Foi apresentado neste estudo, que o objeto de análise do classificador automático de *issues* desta pesquisa é a sua documentação. Dessa maneira, é necessário que os replicadores deste estudo utilizem um *dataset* contendo as informações de *issues* bem documentadas. As descrições e os comentários delas devem refletir com qualidade, o conteúdo da abordagem da solução. Caso as *issues* sejam mal descritas e comentadas, a aplicação do classificador deste estudo será prejudicada com resultados de classificação ruins.

5.3 Dificuldades Encontradas

Durante a coleta dos dados desta pesquisa, foi realizada a tentativa de formação de um *dataset* fornecido por meio do consumo de serviços da API disponibilizada pelo repositório de *software* GitHub. Porém, esse repositório limitou a quantidade de dados que podiam ser coletados por minutos, somada a limitação de quantidade total diária de *downloads* de informações. Esse fato inviabilizou a constituição deste *dataset*, considerando o tempo que havia disponível para a realização dos estudos dessa pesquisa. Uma outra dificuldade encontrada a qual pode ser destacada foi que as classes consideradas pelos especialistas do trabalho da referência principal sofreram adaptações, por exemplo, algumas classes

com poucos registros de classificação realizada pelos desenvolvedores, foram agregadas em uma única classe denominada *others*. Diante disso, durante a etapa de importação dos dados, foi necessário mitigar o reflexo dessa adaptação na filtragem das *issues* utilizadas no *trainning set*. Outro exemplo de adaptação realizada pelos especialistas foi a desconsideração da classe *refarctoring* durante a experiência de classificação humana, pelo fato de um dos *issues trackers* não fornecer a possibilidade de classificação das *issues* pelos propositores, para essa classe. Portanto, a partir da Subseção 3.1.2, essa classe foi substituída pela classe *rfe* ou *request for enhancement*.

5.4 Trabalhos Futuros

Para a realização de trabalhos futuros, deseja-se mesclar informações de *issues* de repositórios particulares com informações de *issues* de repositórios *open source* ou públicos, para garantir maior diversidade de cenários de aplicação do estudo de classificação.

Durante a etapa *transformation* descrita na Seção 2.1, correspondente a filtragem dos dados, foi aplicado *stemming*. Futuramente pode-se aplicar *lematização* para observar o comportamento do classificador para este tipo de filtragem dos dados, uma vez que a aplicação destas técnicas produz diferentes *features*. Por sua vez, essa diferença afeta os resultados do experimento dos estudos classificatórios.

A avaliação do classificador foi realizada com configuração semelhante ao número de divisões da *cross validation 5-fold* tradicional, no experimento #1, e com a configuração *cross validation 10-fold* nos experimentos #2 e #3. Para estudos futuros sugere-se que sejam testados outros número de *folds* durante a realização da *cross validation*.

Com o objetivo de observar o comportamento do classificador desta pesquisa sendo aplicado a *issues* não rotuladas, almeja-se realizar um estudo de aplicação neste tipo de *issue*. Dessa forma, poderá ser observado o seu desempenho em um estudo de caso de *issue trackers systems* ou repositórios de *software* com integração a *issue trackers systems* que não disponham a classificação das *issues* aos seus propositores ou, até mesmo, em repositórios e *issue trackers* que disponham essa classificação, mas que o estudo de caso a desconsidere.

Para outro estudo futuro, deseja-se testar o classificador desta pesquisa em um *dataset* constituído por *commits* de repositórios de *software*, para ser observada sua capacidade

de incorporação de outros contextos de classificação de *software*, como o contexto dos *commits*.

Referências

ANTONIOL, G. et al. Is it a bug or an enhancement?: A text-based approach to classify change requests. In: **Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds**. New York, NY, USA: ACM, 2008. (CASCON '08), p. 23:304–23:318. Disponível em: <http://doi.acm.org/10.1145/1463788.1463819>.

BARROS, R. et al. A survey of evolutionary algorithms for decision-tree induction. **Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on**, v. 42, n. 3, p. 291–312, May 2012. ISSN 1094-6977.

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

BLEI, D. M.; NG, A. Y.; JORDAN, M. I. Latent dirichlet allocation. **J. Mach. Learn. Res.**, JMLR.org, v. 3, p. 993–1022, mar. 2003. ISSN 1532-4435. Disponível em: <http://dl.acm.org/citation.cfm?id=944919.944937>.

BUCKLAND, M.; GEY, F. The relationship between recall and precision. **Journal of the American Society for Information Science**, Wiley Subscription Services, Inc., A Wiley Company, v. 45, n. 1, p. 12–19, 1994. ISSN 1097-4571. Disponível em: <http://dx.doi.org/10.1002/(SICI)1097-4571(199401)45:1<12::AID-ASI2>3.0.CO;2-L>.

CAMPOS EDUARDO CUNHA E MAIA, M. Automatic categorization of questions from q&a sites. In: **Proceedings of the 29th Annual ACM Symposium on Applied Computing**. New York, NY, USA: ACM, 2014. (SAC '14), p. 641–643. ISBN 978-1-4503-2469-4. Disponível em: <http://doi.acm.org/10.1145/2554850.2555117>.

DUMAIS, S. T. Latent semantic analysis. **Annual Review of Information Science and Technology**, Wiley Subscription Services, Inc., A Wiley Company, v. 38, n. 1, p. 188–230, 2004. ISSN 1550-8382. Disponível em: <http://dx.doi.org/10.1002/aris.1440380105>.

GOLDING, A. R.; ROTH, D. A winnow-based approach to context-sensitive spelling correction. **Mach. Learn.**, Kluwer Academic Publishers, Hingham, MA, USA, v. 34, n. 1-3, p. 107–130, fev. 1999. ISSN 0885-6125. Disponível em: <http://dx.doi.org/10.1023/A:1007545901558>.

HALL, M. et al. The weka data mining software: An update. **SIGKDD Explor. Newsl.**, ACM, New York, NY, USA, v. 11, n. 1, p. 10–18, nov. 2009. ISSN 1931-0145. Disponível em: <http://doi.acm.org/10.1145/1656274.1656278>.

HERZIG, K.; JUST, S.; ZELLER, A. It's not a bug, it's a feature: How misclassification impacts bug prediction. In: **Proceedings of the 2013 International Conference on Software Engineering**. Piscataway, NJ, USA: IEEE Press, 2013. (ICSE '13), p. 392–401. ISBN 978-1-4673-3076-3. Disponível em: <http://dl.acm.org/citation.cfm?id=2486788.2486840>.

HO, R. **Pragmatic Programming Techniques**. 2015. Visited 21-May-2015. Disponível em: <http://horicky.blogspot.com.br/2014/03/common-text-mining-workflow.html>.

KATAKIS, I.; TSOUMAKAS, G.; VLAHAVAS, I. Multilabel text classification for automated tag suggestion. In: **In: Proceedings of the ECML/PKDD-08 Workshop on Discovery Challenge**. [S.l.: s.n.], 2008.

KOHAVI, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: **INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE**. [S.l.: s.n.], 1995. p. 1137–1143.

KORENIUS, T. et al. Stemming and lemmatization in the clustering of finnish text documents. In: **Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management**. New York, NY, USA: ACM, 2004. (CIKM '04), p. 625–633. ISBN 1-58113-874-1. Disponível em: <http://doi.acm.org/10.1145/1031171.1031285>.

KOSTER, C.; SEUTTER, M.; BENEY, J. Multi-classification of patent applications with winnow. In: BROY, M.; ZAMULIN, A. (Ed.). **Perspectives of System Informatics**. Springer Berlin Heidelberg, 2003, (Lecture Notes in Computer Science, v. 2890). p. 546–555. ISBN 978-3-540-20813-6. Disponível em: <http://dx.doi.org/10.1007/978-3-540-39866-0_53>.

LANDGREBE, D. A survey of decision tree classifier methodology. **Systems, Man and Cybernetics, IEEE Transactions on**, v. 21, n. 3, p. 660–674, May 1991. ISSN 0018-9472.

LI, Z. et al. Have things changed now?: An empirical study of bug characteristics in modern open source software. In: **Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability**. New York, NY, USA: ACM, 2006. (ASID '06), p. 25–33. ISBN 1-59593-576-2. Disponível em: <http://doi.acm.org/10.1145/1181309.1181314>.

LINARES-VáSQUEZ, M. et al. On using machine learning to automatically classify software applications into domain categories. **Empirical Softw. Engg.**, Kluwer Academic Publishers, Hingham, MA, USA, v. 19, n. 3, p. 582–618, jun. 2014. ISSN 1382-3256. Disponível em: <http://dx.doi.org/10.1007/s10664-012-9230-z>.

LITTLESTONE, N. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. **Machine Learning**, Kluwer Academic Publishers-Plenum Publishers, v. 2, n. 4, p. 285–318, 1988. ISSN 0885-6125. Disponível em: <http://dx.doi.org/10.1023/A%3A1022869011914>.

MAKHOUL, J. et al. Performance measures for information extraction. In: **In Proceedings of DARPA Broadcast News Workshop**. [S.l.: s.n.], 1999. p. 249–252.

MAKRIDAKIS, S. Accuracy measures: theoretical and practical concerns. **International Journal of Forecasting**, v. 9, n. 4, p. 527 – 529, 1993. ISSN 0169-2070. Disponível em: <http://www.sciencedirect.com/science/article/pii/0169207093900793>.

MANNING, C. D. et al. **Introduction to information retrieval**. [S.l.]: Cambridge university press Cambridge, 2008. v. 1.

MCCALLUM, A.; FREITAG, D.; PEREIRA, F. C. N. Maximum entropy markov models for information extraction and segmentation. In: **Proceedings of the Seventeenth International Conference on Machine Learning**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000. (ICML '00), p. 591–598. ISBN 1-55860-707-2. Disponível em: <http://dl.acm.org/citation.cfm?id=645529.658277>.

MCCALLUM, A.; NIGAM, K. et al. A comparison of event models for naive bayes text classification. In: CITESEER. **AAAI-98 workshop on learning for text categorization**. [S.l.], 1998. v. 752, p. 41–48.

PORTER, M. An algorithm for suffix stripping. **Program**, v. 40, n. 3, p. 211–218, 2006.

RAY, B. et al. 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. Disponível em: <http://doi.acm.org/10.1145/2635868.2635922>.

RISH, I. **An empirical study of the naive bayes classifier**. [S.l.], 2001.

THOMAS, S.; HASSAN, A.; BLOSTEIN, D. Mining unstructured software repositories. In: MENS, T.; SEREBRENIK, A.; CLEVE, A. (Ed.). **Evolving Software Systems**. Springer Berlin Heidelberg, 2014. p. 139–162. ISBN 978-3-642-45397-7. Disponível em: <http://dx.doi.org/10.1007/978-3-642-45398-4_5>.

TSUKADA, M.; WASHIO, T.; MOTODA, H. Automatic web-page classification by using machine learning methods. In: **Proceedings of the First Asia-Pacific Conference on Web Intelligence: Research and Development**. London, UK, UK: Springer-Verlag, 2001. (WI '01), p. 303–313. ISBN 3-540-42730-9. Disponível em: <http://dl.acm.org/citation.cfm?id=645960.673927>.

UGUREL, S.; KROVETZ, R.; GILES, C. L. What's the code?: Automatic classification of source code archives. In: **Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York, NY, USA: ACM, 2002. (KDD '02), p. 632–638. ISBN 1-58113-567-X. Disponível em: <http://doi.acm.org/10.1145/775047.775141>.

WALL, M.; RECHTSTEINER, A.; ROCHA, L. Singular value decomposition and principal component analysis. In: BERRAR, D.; DUBITZKY, W.; GRANZOW, M. (Ed.). **A Practical Approach to Microarray Data Analysis**. Springer US, 2003. p. 91–109. ISBN 978-1-4020-7260-4. Disponível em: <http://dx.doi.org/10.1007/0-306-47815-3_5>.

WALLACH, H. M. Topic modeling: Beyond bag-of-words. In: **Proceedings of the 23rd International Conference on Machine Learning**. New York, NY,

USA: ACM, 2006. (ICML '06), p. 977–984. ISBN 1-59593-383-2. Disponível em: <http://doi.acm.org/10.1145/1143844.1143967>.

WANG, C.; BLEI, D. M. Collaborative topic modeling for recommending scientific articles. In: **Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York, NY, USA: ACM, 2011. (KDD '11), p. 448–456. ISBN 978-1-4503-0813-7. Disponível em: <http://doi.acm.org/10.1145/2020408.2020480>.

ZHANG, E.; ZHANG, Y. F-measure. In: LIU, L.; ÖZSU, M. (Ed.). **Encyclopedia of Database Systems**. Springer US, 2009. p. 1147–1147. ISBN 978-0-387-35544-3. Disponível em: <http://dx.doi.org/10.1007/978-0-387-39940-9_483>.

ZHANG, L. **Maximum Entropy Modeling**. 2015. Visited 14-Jun-2015. Disponível em: <http://homepages.inf.ed.ac.uk/lzhang10/maxent.html>.

ZIMMERMANN, T. et al. Improving bug tracking systems. In: **Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on**. [S.l.: s.n.], 2009. p. 247–250.