Predizendo a Participação de Desenvolvedores em Discussões em Projetos de Software Livre

André Luis Schwerz[1], Rafael Liberato[1], Igor Scaliante Wiese[1], Igor Steinmacher[1], Marco Aurélio Gerosa[2], João Eduardo Ferreira[2]

[1] Coordenação de Ciências da Computação – Universidade Tecnológica Federal do Paraná (UTFPR)
BR 369 - km 0,5 - Caixa Postal: 271 - Campo Mourão - PR - Brasil

[2]Departamento de Ciência da Computação – IME – USP
Rua do Matão, 1010 - São Paulo - SP - Brasil

{andreluis, liberato, igor, igorfs }@utfpr.edu.br,
{gerosa, jef}@ime.usp.br

Abstract. *Developers of distributed open source projects use management and issues tracking tool to communicate. These tools provide a large volume of unstructured information that makes the triage of issues difficult, increasing developers' overhead. This paper shows the importance of the content of comments in an open source project to build a classifier to predict the participation for a developer in an issue. To design this prediction model, we used two machine learning algorithms called Naive Bayes and J48. We used the data of three Apache Hadoop subprojects to evaluate the use of the algorithms. By applying our approach to the most active developers of these subprojects we found out that J48 presents better results than Naïve Bayes in almost all cases. We also have achieved an accuracy ranging from 79% to 96% when using J48 algorithm. The results indicate that the content of comments in issues of open source projects is a relevant factor to build a classifier of issues for developers.*

Resumo. *Desenvolvedores em projetos de software livre usam ferramentas de gestão e acompanhamento de tarefas para se comunicar. Essas ferramentas fornecem um grande volume de informações desestruturadas que dificulta a triagem de tarefas e atividades gerando uma sobrecarga de trabalho aos desenvolvedores Este trabalho mostra a importância do conteúdo dos comentários em um projeto de software livre para construção de um classificador para predizer a participação de um desenvolvedor em uma determinada tarefa. Para a elaboração desse classificador de predição, utilizamos dois algoritmos de aprendizagem de máquina Naïve Bayes e J48. Utilizamos dados de três subprojetos do Apache Hadoop para avaliar o uso desses algoritmos. Aplicando nossa abordagem aos desenvolvedores mais ativos destes subprojetos descobrimos que o algoritmo J48 apresenta resultados superiores ao algoritmo Naive Bayes. Com o algoritmo J48 obtivemos uma acurácia variando entre 79% a 96%. Os resultados indicam que o conteúdo dos comentários publicados em tarefas de projetos de software livre é um fator relevante para construir um classificador de predição de tarefas aos desenvolvedores.*

1. Introdução

Nos últimos anos os projetos de software livre despertaram o interesse de empresas e do mundo acadêmico devido à sua filosofia de desenvolvimento distribuído (Raymond, 1999). Esse interesse foi alavancado pelo esforço voluntário de milhares de desenvolvedores espalhados pelo mundo. Devido à natureza descentralizada e colaborativa dos projetos de software livre, repositórios e ferramentas de gestão e acompanhamento de tarefas e erros são utilizados para facilitar o planejamento e a comunicação entre os desenvolvedores. A disponibilização dessas ferramentas à comunidade de desenvolvedores e usuários agiliza o processo de detecção de falhas e a elaboração de possíveis soluções, melhorando a qualidade do software (Raymond, 1999; Anvik et al., 2005). Essas ferramentas mantêm diversas informações, tais como código-fonte, histórico de alterações nos arquivos, quem são os desenvolvedores, lista de tarefas pendentes, entre outras. As mensagens enviadas na lista de tarefas discutem e mantêm um histórico de importantes assuntos do projeto, como o desenvolvimento de novas funcionalidades, falhas, ajuda aos usuários, decisões de projeto etc.

As informações contidas na lista de tarefas formam uma base de conhecimento do projeto que pode ser utilizada para auxiliar os próprios desenvolvedores. Entretanto, em grandes projetos de software livre há um elevado volume de mensagens, dificultando a escolha das tarefas em que o desenvolvedor deseja participar. Por exemplo, no mês de março de 2012 no projeto Apache Hadoop foram enviadas 1.622 mensagens na lista de tarefas. Assim, os desenvolvedores estão propensos a perderem oportunidades de participar de tarefas relevantes ao seu perfil devido ao grande volume de dados e à dificuldade de seleção.

A dificuldade de sugerir tarefas relevantes aos colaboradores não é um problema exclusivo da comunidade de software livre. Cosley *et al*. [2007] propõem um software que executa o encaminhamento inteligente de tarefas à usuários na Wikipédia. Abel *et al*. [2010] recomenda informações relevantes a usuários em fóruns de discussão em sistemas *e-learning*. Este trabalho mostra a importância do conteúdo dos comentários em um projeto de software livre para construção de um classificador para predizer a participação de um desenvolvedor em uma determinada tarefa. O classificador é baseado na análise de conteúdo do vocabulário utilizado pelo desenvolvedor nas mensagens enviadas à lista de tarefas. Na análise de conteúdo, cada mensagem é transformada em um conjunto de palavras (*tokens*). Esse formato possibilita definir um vocabulário para um determinado autor, tarefa ou para todo o projeto.

Para realizar a análise foram coletados dados do projeto Apache Hadoop[1] da Apache Software Foundation[2] que está dividido em três subprojetos: Hadoop Commons[3], Hadoop Distributed File System (HDFS)[4] e Hadoop MapReduce[5]. Essa análise foi realizada com objetivo de responder à seguinte questão de pesquisa:

[1] http://hadoop.apache.org

[2] http://www.apache.org

[3] http://hadoop.apache.org/common

[4] http://hadoop.apache.org/hdfs

[5] http://hadoop.apache.org/mapreduce

Q1 - É possível predizer a participação dos desenvolvedores mais ativos na lista de tarefas (*issue tracking*) com base na sua história de participação?

Para cada desenvolvedor foi construído um classificador que prediz sua participação em uma determinada tarefa baseado no seu vocabulário utilizado em outras participações.

Na composição do classificador proposto comparamos os resultados obtidos pelo algoritmo J48 e pelo classificador bayesiano, detalhados em Quinlan [1993] e Mitchell [1997]. Para cada desenvolvedor, as tarefas foram divididas em duas classes: "*Participar*" ou "*Não participar*". Em seguida, para a análise do desenvolvedor, definimos aleatoriamente um conjunto de dados para treino e outro para teste. A proporção utilizada para a definição do conjunto de treino e de teste foi 80-20, respectivamente.

O restante do artigo está organizado como segue. Na Seção 2 são apresentados os trabalhos relacionados. Na Seção 3 é apresentado o método de pesquisa utilizado. Os resultados são apresentados na Seção 4. Por fim, na Seção 5 são apresentadas as discussões, conclusões e trabalhos futuros.

2. Trabalhos Relacionados

As comunidades de desenvolvedores em projetos de software livre utilizam ferramentas web para manter diversos tipos de informações relacionadas ao andamento do projeto. Uma desvantagem do uso de tais ferramentas está diretamente relacionada à sobrecarga de trabalho atribuída aos desenvolvedores mais experientes na triagem das informações fornecidas pela comunidade (Anvik et al., 2005). Neste sentido, o levantamento descrito nesta seção preconiza trabalhos que abordam a redução da carga de trabalho por meio da indicação de quais tarefas o desenvolvedor deve participar ou em qual lista de e-mail deve participar.

Matter *et al*. [2009] apresentam uma abordagem automática para realizar a triagem de relatórios de falhas atribuindo a tarefa ao desenvolvedor com a melhor expertise para lidar com a falha. Os autores construíram um modelo de expertise para cada desenvolvedor baseado no seu código-fonte produzido. Nesse trabalho foi utilizado o cálculo do cosseno entre o modelo de expertise e um vetor de palavras construído pela mineração do texto do relato da falha para classificar os desenvolvedores. Usando essa abordagem, reportaram um *precision* de 33.6% e *recall* 71.0% no projeto Eclipse. Seguindo a mesma linha, os trabalhos de Mockus e Herbsleb [2002] e Fritz et al. [2007] constroem modelos de expertise baseados nas alterações realizadas no software pelo desenvolvedor. Siy *et al.* [2008] e Alonso *et al.* [2008] propõem um modelo de expertise baseado nos diretórios em que o código-fonte está localizado. Em ambos os trabalhos os autores consideram que um desenvolvedor tende a trabalhar em arquivos que se localizam em diretórios próximos, e dessa forma são mais capacitados para responder a questões relacionadas a essa parte do software.

O nosso trabalho considera o histórico de mensagens publicadas pelo desenvolvedor como o único fator para predizer a sua participação em uma determinada lista de tarefas. Uma abordagem semelhante é apresenta por Cubranic e Murphy [2004], que propõem a aplicação do classificador bayesiano para auxiliar a triagem de falhas. No processo de classificação, os autores realizaram uma caracterização textual considerando a descrição

das falhas e obtiveram 30% de acurácia na aplicação dessa abordagem em um grande projeto de software livre. Canfora e Cerulo [2005] traçaram um esboço de uma técnica de recuperação de informação que alcança *recall* de 20% aplicada no projeto Mozilla.

Anvik *et al.* [2006] apresentam uma abordagem que automatiza parte do processo de triagem dos relatórios de falhas. Baseado no conjunto de desenvolvedores recomendado pelo algoritmo de aprendizagem de máquina SVN, o responsável pela triagem indica um desenvolvedor mais qualificado para resolução de falhas. O algoritmo é treinado com dados textuais sobre o relatório de falhas anteriores rotulados com o desenvolvedor que resolveu a falha. O *precision* alcançado com essa abordagem foi de 57% e 64% para os projetos Eclipse e Firefox, respectivamente.

Ibrahim *et al.* [2010] apresentam um trabalho com uma abordagem semelhante à nossa, no entanto, aplicado para predições em lista de e-mails. Os autores desenvolveram um modelo que prediz a participação do desenvolvedor em uma discussão por e-mail. Esse modelo é desenvolvido baseado no comportamento histórico do desenvolvedor. Esse trabalho também mostrou que a quantidade de mensagens de uma discussão, o conteúdo, o remetente e a época em que a mensagem é submetida influenciam o comportamento do desenvolvedor. O classificador bayesiano e a árvore de decisão foram usadas para aplicar a abordagem em listas de e-mails dos projetos da Apache, PostGreSQL e Python. Bird [2012] construiu um modelo único para todos os desenvolvedores baseado em uma rede neural para predizer quais e-mails um determinado desenvolvedor teria interesse.

As abordagens apresentadas nesta seção buscam fornecer meios automatizados para reduzir a carga de trabalho dos desenvolvedores em projetos de software livre e possuem características similares às apresentadas neste trabalho. No entanto, nossa abordagem procura recomendar ao desenvolvedor sua participação em lista de tarefas baseada no seu envolvimento com o projeto em listas anteriores. O desenvolvedor pode colaborar na lista de tarefas indicada pela abordagem de diversas formas, não se limitando a realizar *commits* ou encaminhar novos *patches*.

3. Método de Pesquisa

O método é composto de três passos: coleta, preparação e classificação dos dados.

3.1. Coleta de Dados

Para conduzir a análise e construção do classificador, primeiramente realizamos a coleta dos dados da lista de tarefas dos projetos mencionados. Esses dados estão disponíveis no próprio website dos projetos. Para extrair os dados foi desenvolvido um minerador que realiza a coleta dos dados por meio da URL "*https://issues.apache.org/jira/browse/*<NOME_PROJETO>-<NÚMERO_ISSUE>", com o qual extraímos as seguintes informações para cada tarefa (*issue*):

- descrição;
- usuário relator (reporter);
- data de criação;
- data da fechamento;
- prioridade;
- status atual;
- comentários (com autor, data e mensagem).

No entanto, utilizamos somente os dados contidos nos comentários publicados na tarefa. A Tabela 1 mostra informações sobre os dados coletados.

Neste artigo utilizamos uma amostra dos dez desenvolvedores que mais publicaram comentários em tarefas de cada projeto. Essa amostra representa os desenvolvedores que enviaram mais de um terço do total de comentários de cada projeto. É possível notar que a concentração dos comentários é maior no subprojeto HDFS. Esses dez desenvolvedores publicaram 57,2% de todos os comentários existentes no projeto.

Tabela 1. Resumo dos dados coletados.

Subprojetos	***Hadoop Commons***	***HDFS***	***Hadoop MapReduce***
Tarefas *(issues)*	7.720	2.980	3.591
Comentários	76.065	34.051	36.987
Desenvolvedores	1.035	512	614
Inicio da coleta	01-2006	03-2006	05-2006
Fim da coleta	04-2012	04-2012	04-2012
% de comentários dos desenvolvedores mais ativos (TOP10)	37,3%	57,2%	39%

A Figura 1 ilustra a evolução da quantidade de comentários publicados em cada projeto. As linhas verticais representam *releases* no período. É possível notar que existe uma concentração de *releases* entre 2007 e 2009 e a partir de 2011.

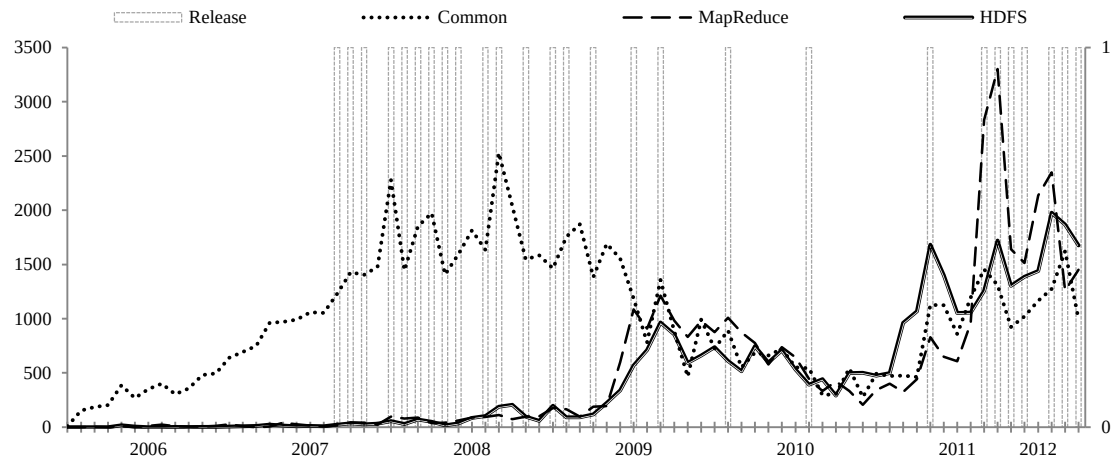

Figura 1. Distribuição temporal das mensagens.

A **Erro! Autoreferência de indicador não válida.** mostra a taxa de participação em tarefas de cada um dos desenvolvedores em cada projeto. Essa taxa é a proporção entre as tarefas em que o desenvolvedor colaborou e as tarefas em que não colaborou. Se a taxa de participação é próxima a 1 então a quantidade de tarefas em que o desenvolvedor participou é próxima à quantidade de tarefas em que não participou. Caso contrário, a classe "*Não participar*" é maior que a classe "*Participar*".

Tabela 2. Taxa de participação.

Desenvolvedor	*Hadoop Common*	*HDFS*	*Hadoop MapReduce*
Top-1	0,28	0,45	0,23
Top-2	0,19	0,33	0,25
Top-3	0,21	0,24	0,13
Top-4	0,13	0,28	0,13
Top-5	0,12	0,15	0,13
Top-6	0,11	0,11	0,09
Top-7	0,09	0,22	0,15
Top-8	0,13	0,14	0,11
Top-9	0,12	0,10	0,10
Top-10	0,07	0,14	0,06

Para evitar ambiguidade definimos formalmente os dados coletados. Uma lista de tarefas de um determinado projeto p é definida pelo conjunto:

$$ISSUES_p = \{i_1, i_2, i_3, \ldots, i_n\},$$

em que n é a quantidade de tarefas do projeto p. Podemos ainda representar o conjunto de todos os desenvolvedores de um determinado projeto p como:

$$AUTHORS_p = \{a_1, a_2, a_3, \ldots, a_m\},$$

em que m é a quantidade de desenvolvedores que comentaram em alguma tarefa $i_k \in ISSUE_p$ para qualquer $1 \leq k \leq n$. Também podemos representar todos os comentários de um projeto p como:

$$C_p = \{c_1, c_2, c_3, \ldots, c_q\},$$

em que q é a quantidade de comentários publicados no projeto p. Uma tarefa $i_k \in ISSUE_p$ com $1 \leq k \leq n$ é representada pela tripla:

$$i_k = <C_{ik}, d_{in}, d_{out}>,$$

em que $C_{ik} \subseteq C_p$ é o subconjunto de todos os comentários publicados na tarefa i_k e d_{in} e d_{out} são as datas da publicação do primeiro e último comentário, respectivamente.

Um desenvolvedor $a_r \in AUTHORS_p$ sendo $1 \leq r \leq m$ é representado pela tripla:

$$a_r = <C_{ar}, d_{in}, d_{out}>,$$

em que $C_{ar} \subseteq C_p$ é o subconjunto de todos comentários que esse desenvolvedor publicou no projeto e d_{in} e d_{out} são as datas da publicação do primeiro e último comentário desse desenvolvedor. Dessa forma, podemos notar que o conjunto de todos os comentários do projeto p é igual à união de todos os comentários das tarefas, ou ainda, é igual à união de todos os comentários dos desenvolvedores, conforme:

$$C_p = C_{i1} \cup C_{i2} \cup \ldots \cup C_{in} = C_{a1} \cup C_{a\,2} \cup \ldots \cup C_{am},$$

Um comentário $c \in C_p$ é representado pela tripla:

$$c = < T_c, d, a >,$$

em que T_c é conjunto de *tokens* que compõem o comentário c, d é a data de publicação e $a \in AUTHORS_p$ é o desenvolvedor que publicou esse comentário.

Podemos representar o conjunto de todos os *tokens* do projeto *p* como:

$$WORLD_p = T_1 \cup T_2 \cup ... \cup T_q \mid \forall c \in C_p ,$$

Analogamente, podemos representar todos os *tokens* de uma tarefa $i_k \in ISSUE_p$ como:

$$TOKENS_k = T_1 \cup T_2 \cup ... \cup T_w \mid \forall c \in C_{ik} ,$$

em que *w* é a quantidade de elementos do conjunto C_{ik}.

3.2. Preparação dos dados

O processo de preparação de dados consistiu em organizar e criar o conjunto $WORLD_p$ dos subprojetos da Apache Hadoop. O primeiro passo foi identificar e remover da nossa amostra os comentários que foram inseridos por sistemas de software de integração contínua, tal como o Hudson[6]. Com esse simples passo foi possível remover uma grande quantidade de comentários irrelevantes a este trabalho como apresentado na Tabela 3.

Tabela 3. Resumo do processo de preparação dos dados.

Comentário	*Hadoop Commons*	*HDFS*	*Hadoop MapRaduce*
% de comentários extraídos	18,8	24,6	29,9
#comentários restantes	61.769	25.688	25.932
#tokens	59.733	27.774	31.294

Em seguida, utilizamos o Apache Lucene™[7] (AL) na versão 3.5 para retirar as marcações HTML dos comentários, realizar o processo de *tokenização,* extração dos *stopwords* e lematização (*stemming*). Usamos o *tokenizer* e a lista de *stopwords* padrão do AL e aplicamos o algoritmo de lematização amplamente utilizado de Martin Porter [1980]. Consideramos também *urls* e nomes de pacotes de código-fonte como *tokens* únicos. A Tabela 3 mostra o tamanho do conjunto $WORLD_p$ de cada projeto.

Entendemos que o processo de preparação dos dados ainda pode ser mais bem explorado a fim de melhorar os resultados apresentados neste trabalho. Por exemplo, o trabalho Cubranic e Murphy [2004] indica que há pouco ganho com a lematização. Outro aspecto de estudo, é o fato de existir trechos de comentários que são compostos por código-fonte e podem ser tratados de uma melhor maneira.

3.3. Classificação

Após os dados terem sido preparados, para cada desenvolvedor do projeto, dividimos aleatoriamente o conjunto $ISSUES_p$ em dois subconjuntos, um para treino com 80% das tarefas e outro para teste com os 20% restantes. Os dois subconjuntos foram utilizados para classificar se o desenvolvedor participa ou não em uma determinada tarefa baseado em seu conteúdo. Para a predição utilizamos dois algoritmos: o algoritmo de árvore de decisão J48 e o algoritmo de classificação bayesiana Naïve Bayes, ambos disponibilizados pelo WEKA[8], um ambiente para análise de conhecimento que implementa vários algoritmos de aprendizado de máquina.

[6] http://hudson-ci.org/

[7] http://lucene.apache.org/

[8] http://www.cs.waikato.ac.nz/ml/weka

A árvore de decisão é um modelo de aprendizado de máquina que prediz o valor do atributo alvo para uma nova instância baseado nos valores de vários outros atributos disponíveis. O atributo alvo que será predito é conhecido como variável dependente, pois seu valor depende ou é decidido pelos valores de todos os outros atributos (variáveis independentes). Para classificar uma nova instância, é necessário criar uma árvore de decisão baseada nos valores dos atributos do conjunto de treinamento. Com a árvore de decisão gerada, basta verificar os valores dos atributos da nova instância para predizer o valor do atributo alvo.

O algoritmo de classificação bayesiana é baseado na regra de Bayes da probabilidade condicional. O algoritmo utiliza todos os atributos contidos nos dados do conjunto de treinamento, e os analisa de forma individual como se fossem igualmente importantes e independentes uns dos outros. O classificador bayesiano considerará separadamente cada um desses atributos quando classificar uma nova instância.

Para a execução dos algoritmos no ambiente WEKA utilizamos um computador com processador de quatro núcleos com 2.9 GHz, 8GB de memória RAM e um sistema operacional de 64 bits. Devido à grande quantidade de *tokens* do projeto Hadoop Commons e esse limite de memória RAM, não foi possível executar os algoritmos com todos os dados desse projeto. Então, para viabilizar a análise reduzimos o conjunto $WORLD_{common}$ para 70% do original por meio de um algoritmo que calcula a relevância de cada $t \in WORLD_{common}$ de acordo com a equação definida em Jone [1972]:

$$score\ (t) = tf(t, d) * idf(t, D)$$

Vale ressaltar que essa redução foi realizada somente para o projeto Hadoop Common. Para os demais projetos mantivemos o conjunto $WORLD_p$ inalterado. Os resultados são apresentados na Seção 4.

3.3.1. Formato de entrada dos dados

Para execução dos algoritmos os subconjuntos de dados de treino e teste foram formatados em uma matriz de entrada como ilustra a Figura 2.

$ISSUES_p$	$WORLD_p$: t_1	t_2	t_3	⋯	$t_{\lvert WORLD_p \rvert}$	Atributo Alvo (AA)
i_1	0	1	1	⋯	1	1
i_2	1	0	0	⋯	0	0
i_3	0	1	1	⋯	0	0
⋮						
i_n	1	1	1	⋯	1	0

Figura 2. Matriz de entrada dos dados.

As linhas da matriz são representadas pelos elementos do conjunto $ISSUES_p$ e as colunas, salvo a última, são representadas pelos elementos do conjunto $WORLD_p$. A matriz foi construída seguindo duas regras. A Regra 1 estabelece o preenchimento das células da matriz, menos as da última coluna.

Regra 1: Seja uma tarefa $i_k \in ISSUES_p$ e um *token* $t_j \in WORLD_p$

$$M[i_k, t_j] = \begin{cases} 1 & se \;\; t_j \in TOKENS_k \\ 0 & se \;\; t_j \notin TOKENS_k \end{cases}$$

Com isso, todos os *tokens* pertencentes à tarefa são assinalados com 1, definindo seu vocabulário.

As células da coluna *AA*, que representa o atributo alvo a ser predito, são preenchidas respeitando a Regra 2.

Regra 2: Seja uma tarefa $i_k \in ISSUES_p$ e um desenvolvedor $a_r \in AUTHORS_p$

$$M[i_k, AA] \begin{cases} 1 & se \;\; C_{ik} \cap C_{ar} \neq \emptyset \\ 0 & se \;\; C_{ik} \cap C_{ar} = \emptyset \end{cases}$$

Aplicamos dois filtros adicionais nos dados de entrada. O primeiro filtro foi aplicado tanto no subconjunto de treino quanto no de teste, descartando as tarefas em que o desenvolvedor não teve a oportunidade de participar. Para isso, definimos o escopo para a tarefa como o intervalo entre a publicação da primeira e da última mensagem publicada. Também definimos o escopo para o desenvolvedor como o intervalo entre a sua primeira e a última mensagem publicada, independente da tarefa. Esse filtro respeita o foi estabelecido na Regra 3.

Regra 3: Dado uma tarefa ik ∈ ISSUESp e um desenvolvedor ar ∈ AUTHORSp

$$\begin{cases} descartar(i_k), & se \; i_k[d_{out}] < a_r[d_{in}] \; OR \; i_k[d_{in}] > a_r[d_{out}] \\ manter(i_k), & caso \; contrário \end{cases}$$

Nessa abordagem identificamos seis tipos de tarefas como ilustra a Figura 3.

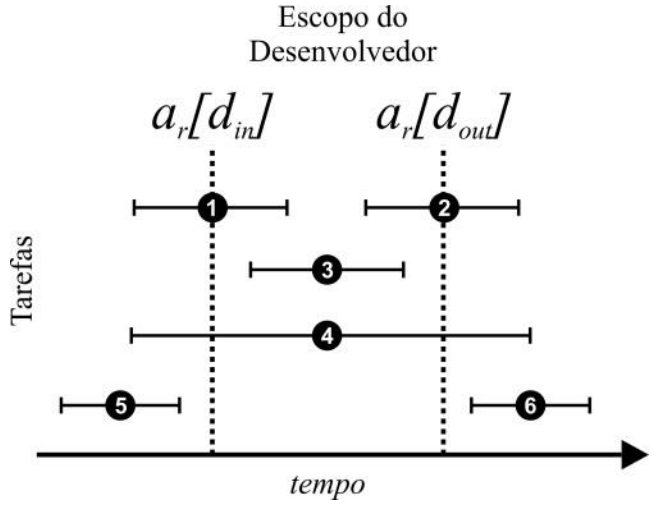

Figura 3. Tipos de Tarefas.

As tarefas do **tipo 1** são aquelas que começam antes do início do escopo do desenvolvedor, mas terminam dentro do escopo. As tarefas do **tipo 2** são aquelas que iniciam dentro do escopo do desenvolvedor, mas terminam fora. As tarefas do **tipo 3** estão totalmente contidas no escopo do desenvolvedor. As tarefas do **tipo 4** permeiam por todo escopo do desenvolvedor, mas iniciam e terminam fora. As tarefas do **tipo 5** começam e terminam antes do início do escopo do desenvolvedor. E por último, as tarefas do **tipo 6** iniciam e terminam após o término do escopo do desenvolvedor. Com

esses seis tipos de tarefas, entendemos que o desenvolvedor teve, em algum momento, a oportunidade de colaborar com as tarefas dos tipos 1, 2, 3 e 4. As tarefas dos tipos 5 e 6 foram descartadas por estarem totalmente fora do escopo do desenvolvedor, não oferecendo a oportunidade de participação. Portanto, para a predição de cada desenvolvedor, o universo de tarefas do projeto foi reduzido de acordo com seu escopo.

O segundo filtro, aplicado somente no subconjunto de teste, descarta todos os *tokens* proferidos pelo desenvolvedor em cada tarefa. Utilizamos essa abordagem baseado na premissa de que os *tokens* proferidos pelo desenvolvedor darão vantagem à tarefa na predição, por fazerem parte de seu vocabulário e, provavelmente, terem sido utilizados no treinamento. Dessa forma, esse filtro tem como objetivo evitar um viés na predição.

3.3.2. Avaliação dos Resultados

Por se tratar de um aprendizado de máquina típico, utilizamos a matriz de confusão para avaliar os valores de *precision* (*p*), *recall* (*r*) e da acurácia (*a*) dos resultados. A Tabela 4 mostra a matriz de confusão, na qual temos quatro tipos de resultados: verdadeiro positivo (*vp*), falso positivo (*fp*), falso negativo (*fn*) e verdadeiro negativo (*vn*).

Tabela 4. Matriz de confusão.

Classes previamente conhecidas	***Classificado como***	
	Participar	Não participar
Participar	*vp*	*fn*
Não participar	*fp*	*vn*

No contexto de classificação, a medida de *precision* é a fração dos valores classificados que são relevantes e a medida de *recall* é a fração dos valores relevantes que foram classificados com sucesso.

$$p_{contribuir} = \frac{vp}{vp + fp} \qquad p_{\text{não}Contribuir} = \frac{vn}{vn + fn}$$

$$r_{contribuir} = \frac{vp}{vp + fn} \qquad r_{\text{não}Contribuir} = \frac{vn}{vn + fp}$$

A acurácia é medida por meio dos resultados verdadeiros (*vp* e *vn*). Uma acurácia próxima ao valor 1 indica que os valores obtidos na classificação são próximos aos que eram conhecidos previamente.

$$a = \frac{vp + vn}{vp + vn + fp + fn}$$

Podemos interpretar a medida de $p_{participar}$ como a probabilidade de que uma tarefa classificada como "*Participar*" seja realmente verdadeira e a medida de $r_{participar}$ como a probabilidade de que uma tarefa em que o desenvolvedor participou seja classificada como "*Participar*". As medidas de $p_{não Participar}$ e $r_{não Participar}$ seguem o mesmo raciocínio. A acurácia pode ser interpretada como a qualidade de classificação dos valores verdadeiros independente da classe a qual pertencem.

4. Resultados

As Tabelas 5, 6 e 7 mostram as medidas de recall, precision e acurácia obtidas pelo algoritmo de classificação Naïve Bayes e J48 nos subprojetos Hadoop Common, MapReduce e HDFS, respectivamente. Os resultados alcançados pelo algoritmo J48 foram superiores ao algoritmo Naïve Bayes atingindo uma melhoria da taxa média da acurácia de 11,2%, 9,6% e 13,3% para os subprojetos Hadoop Common, MapReduce e HDFS respectivamente.

Mesmo possuindo as informações sobre o nome e dados de acesso dos desenvolvedores analisados, optamos por preservar suas identidades e referenciá-los apenas por Top-1 a Top-10. Desta forma, o desenvolvedor Top-1 é aquele que mais postou comentários em listas de tarefas do projeto.

A Tabela 5 apresenta o resultado da predição dos desenvolvedores do subprojeto Hadoop Common, utilizando o algoritmo de Naive Bayes e o J-48. Na figura as células sombreadas com valores em negrito apresentam o melhor resultado quando comparamos os dois algoritmos. A escala de cinza representa a diferença proporcional dos acertos, sendo que quanto mais escuro o fundo da célula, maior a diferença (o fundo mais claro representa diferenças pouco significativas, e o fundo mais escuro representa diferenças muito significativas). Pode-se perceber que o algoritmo J-48 supera o Naive Bayes em todos os casos, ressaltando as taxas obtidas para a classe "*Participar*", em que o J48 supera muito o algoritmo Naive Bayes.

A Figura 5 mostra ainda que as taxas de *recall* e *precision* para a classe "*Não participar*" são mais satisfatórias que as taxas da classe *"Participar*" para os dois algoritmos. Esse comportamento é similar ao apresentado nas Tabelas 6 e 7. Para o algoritmo J48, a taxa de *recall* para a classe "*Participar*" varia entre 0,2 e 0,54 com *precision* de 0,53 e 0,91. A classe "*Não participar*" apresenta *recall* entre 0,96 e 1 com taxa de *precision* variando entre 0,83 e 0,95.

Tabela 5. Taxas de *recall*, *precision* e acurácia – Hadoop Common.

Desenvolvedor	Naïve Bayes					J48				
	Não Participar		Participar		Acurácia	Não Participar		Participar		Acurácia
	Recall	Precision	Recall	Precision		Recall	Precision	Recall	Precision	
Top - 1	0,85	0,81	0,27	0,34	0,73	**0,97**	**0,83**	**0,30**	**0,76**	**0,83**
Top - 2	0,87	0,86	0,21	0,23	0,77	**0,98**	**0,90**	**0,37**	**0,73**	**0,88**
Top - 3	0,86	0,84	0,26	0,28	0,75	**0,96**	**0,86**	**0,30**	**0,62**	**0,84**
Top - 4	0,88	0,89	0,20	0,19	0,80	**0,99**	**0,92**	**0,39**	**0,84**	**0,92**
Top - 5	0,91	0,91	0,27	0,28	0,84	**0,99**	**0,92**	**0,34**	**0,75**	**0,91**
Top - 6	0,90	0,92	0,28	0,24	0,83	**0,98**	**0,92**	**0,30**	**0,61**	**0,91**
Top - 7	0,90	0,93	0,23	0,16	0,84	**0,98**	**0,95**	**0,40**	**0,68**	**0,94**
Top - 8	0,92	0,89	0,13	0,17	0,83	**0,98**	**0,90**	**0,20**	**0,53**	**0,89**
Top - 9	0,89	0,91	0,27	0,22	0,82	**0,98**	**0,95**	**0,54**	**0,76**	**0,93**
Top - 10	0,93	0,93	0,09	0,09	0,87	**1,00**	**0,95**	**0,39**	**0,91**	**0,95**

A Tabela 6 apresenta o resultado da predição para os desenvolvedores do subprojeto MapReduce. Percebemos que em poucos casos o algoritmo Naive Bayes superou o algoritmo J48. Entretanto, o J48 continua apresentando-se mais eficiente que o Naive

Bayes no geral. Destacamos aqui o desenvolvedor Top-9, que apresentou menor taxa de *recall* de toda nossa amostra para o algoritmo J48. Apesar disso, a média da taxa de *recall* e *precision* mantiveram-se semelhantes ao subprojeto Hadoop Common.

Tabela 6. Taxas de *recall*, *precision* e acurácia – Hadoop MapReduce.

Desenvolvedor	Naïve Bayes					J48				
	Não Contribuir		Contribuir		Acurácia	Não Contribuir		Contribuir		Acurácia
	Recall	Precision	Recall	Precision		Recall	Precision	Recall	Precision	
Top - 1	0,86	0,81	0,24	0,30	0,73	**0,97**	**0,88**	**0,47**	**0,83**	**0,87**
Top - 2	0,88	**0,86**	**0,36**	0,41	0,79	**0,97**	0,86	0,29	**0,66**	**0,84**
Top - 3	0,93	0,91	0,18	0,24	0,85	**0,99**	**0,93**	**0,38**	**0,76**	**0,92**
Top - 4	0,92	0,89	0,22	0,28	0,83	**0,98**	**0,91**	**0,34**	**0,76**	**0,91**
Top - 5	0,89	0,88	0,12	0,14	0,80	**1,00**	**0,91**	**0,32**	**0,93**	**0,91**
Top - 6	0,95	0,91	0,06	0,11	0,87	**0,99**	**0,94**	**0,38**	**0,83**	**0,94**
Top - 7	0,87	0,88	0,14	0,12	0,78	**0,99**	**0,92**	**0,41**	**0,86**	**0,92**
Top - 8	0,92	0,91	0,20	0,21	0,85	**0,98**	**0,93**	**0,36**	**0,71**	**0,92**
Top - 9	0,93	**0,91**	**0,16**	0,21	0,85	**1,00**	0,90	0,07	**0,63**	**0,90**
Top - 10	0,95	0,94	0,11	0,15	0,90	**1,00**	**0,95**	**0,25**	**0,79**	**0,95**

A Tabela 7 apresenta os resultados para o projeto HDFS. Mais uma vez o algoritmo J48 superou o algoritmo Naive Bayes para todos os casos. As taxas obtidas no projeto HDFS representam os melhores resultados com os maiores valores de *recall* e de *precision*. Entre os desenvolvedores do subprojeto HDFS, destacamos o Top-10 que apresentou a maior taxa de recall na classe "Participar" da amostra.

Tabela 7. Taxas de *recall*, *precision* e acurácia – J48 - HDFS.

Desenvolvedor	Naïve Bayes					J48				
	Não Contribuir		Contribuir		Acurácia	Não Contribuir		Contribuir		Acurácia
	Recall	Precision	Recall	Precision		Recall	Precision	Recall	Precision	
Top - 1	0,86	0,72	0,26	0,47	0,67	**0,94**	**0,79**	**0,46**	**0,78**	**0,79**
Top - 2	0,86	0,75	0,18	0,31	0,69	**0,97**	**0,81**	**0,33**	**0,77**	**0,80**
Top - 3	0,83	0,83	0,27	0,28	0,72	**0,97**	**0,88**	**0,45**	**0,77**	**0,87**
Top - 4	0,87	0,79	0,23	0,36	0,72	**0,95**	**0,84**	**0,41**	**0,73**	**0,83**
Top - 5	0,91	0,89	0,29	0,33	0,82	**0,97**	**0,90**	**0,31**	**0,63**	**0,88**
Top - 6	0,92	0,90	0,06	0,07	0,84	**0,99**	**0,93**	**0,31**	**0,73**	**0,93**
Top - 7	0,89	0,81	0,21	0,34	0,75	**0,99**	**0,85**	**0,35**	**0,87**	**0,85**
Top - 8	0,89	0,91	0,29	0,23	0,82	**0,97**	**0,93**	**0,43**	**0,63**	**0,91**
Top - 9	0,92	0,95	0,30	0,20	0,88	**0,99**	**0,97**	**0,56**	**0,71**	**0,96**
Top - 10	0,91	0,88	0,17	0,21	0,81	**1,00**	**0,95**	**0,62**	**0,97**	**0,95**

A Tabela 8 apresenta um resumo com as médias das taxas de *recall*, *precision* e acurácia para os três projetos, aplicando-se os dois algoritmos de classificação. Pode-se ver que na média, como era esperado, o algoritmo J48 supera o algoritmo Naive Bayes. Para o algoritmo J48, as médias das acurácias por projeto foram 90%, 91% e 87%,

mostrando que o algoritmo tem eficácia alta. O ponto negativo fica por conta do *recall* e da classe "*Participar*", que não ultrapassa 41%. O *precision* para essa mesma classe, apresenta valor máximo de 78%. Apesar de baixo frente à classe "*Não participar"*, o valor é alto se compararmos, por exemplo, com o trabalho de Matter et al. [2009].

Tabela 8. Taxas médias de *recall*, *precision* e acurácia - J48.

Projeto	Naïve Bayes					J48				
	Não Contribuir		Contribuir		Acurácia	Não Contribuir		Contribuir		Acurácia
	Recall	Precision	Recall	Precision		Recall	Precision	Recall	Precision	
Hadoop Common	0,89	0,89	0,23	0,24	0,81	**0,98**	**0,91**	**0,34**	**0,71**	**0,90**
Map Reduce	0,91	0,89	0,20	0,25	0,83	**0,99**	**0,91**	**0,33**	**0,78**	**0,91**
HDFS	0,89	0,84	0,23	0,30	0,77	**0,97**	**0,89**	**0,41**	**0,76**	**0,87**

A Figura 4 apresenta três gráficos que mostram uma comparação entre os resultados das acurácias alcançadas com os subprojetos em estudo para o Naïve Bayes e o J48. Nota-se que o algoritmo J48 é melhor que o algoritmo Naïve Bayes para todos os casos.

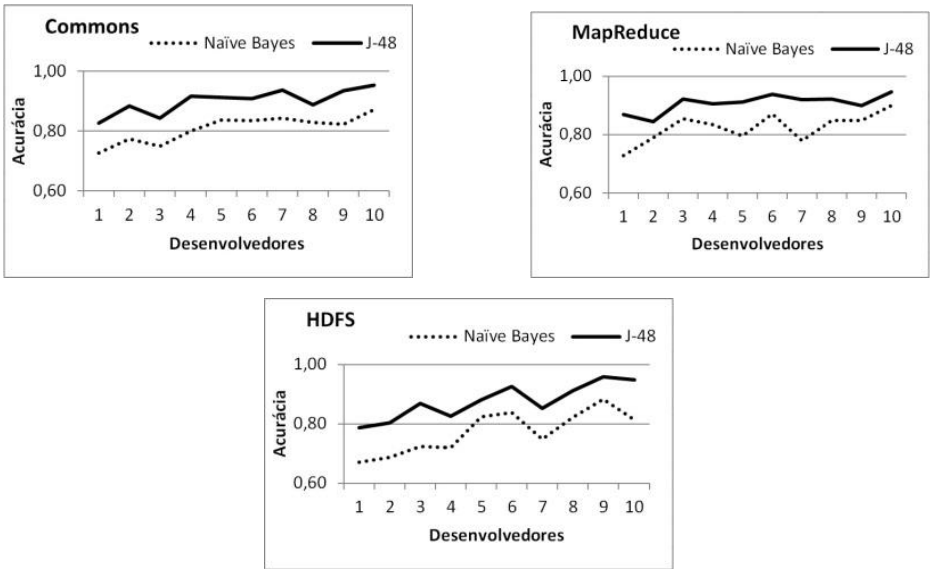

Figura 4. Naïve Bayes versus J48.

A Figura 5 apresenta o gráfico que mostra a relação entre a acurácia obtida pelo algoritmo J48 e os desenvolvedores que estão ordenados de acordo com a taxa de participação apresentada na A Erro**! Autoreferência de indicador não válida.** mostra a taxa de participação em tarefas de cada um dos desenvolvedores em cada projeto. Essa taxa é a proporção entre as tarefas em que o desenvolvedor colaborou e as tarefas em que não colaborou. Se a taxa de participação é próxima a 1 então a quantidade de tarefas em que o desenvolvedor participou é próxima à quantidade de tarefas em que não participou. Caso contrário, a classe "*Não participar*" é maior que a classe "*Participar*".

Tabela 2.

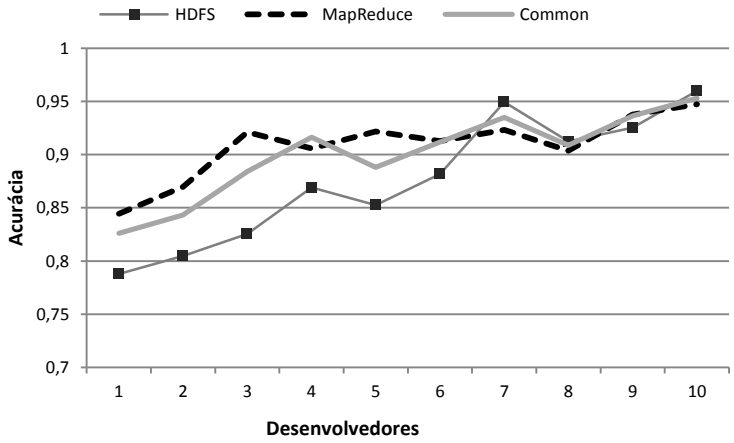

Figura 5. Relação da acurácia dos dez desenvolvedores entre os três projetos.

5. Discussão

Em nossa análise utilizamos os comentários publicados nas discussões das listas de tarefas em três subprojetos de um software livre. A amostra foi limitada aos dez desenvolvedores que mais publicaram comentários. Não podemos afirmar que os mesmos resultados serão obtidos para os desenvolvedores com menos participações, pois o vocabulário desses desenvolvedores seria menor. Um possível trabalho futuro é aplicar o algoritmo para uma amostra maior de desenvolvedores para verificar a generalização da abordagem. Os resultados obtidos pelo classificador elaborado usando os métodos descritos na Seção 2 visam responder à seguinte questão de pesquisa:

Q1 - É possível predizer a participação dos desenvolvedores mais ativos na lista de tarefas (issue tracking) com base na sua história de participação?

As Tabelas 5, 6, 7 e 8 mostraram o resultado da predição com uma alta acurácia, variando de 79% a 96%. No entanto, esse classificador apresenta uma baixa taxa de *recall* para a classe "*Participar*". Entendemos que isso é uma deficiência do nosso classificador, visto que ao não atribuir ao desenvolvedor uma tarefa em que poderia participar, deixamos de aproveitar o seu potencial e conhecimento. Entretanto, os resultados mostram que, dentre as tarefas indicadas ao desenvolvedor, a taxa de *precision* apresentou uma média de 75%variando entre 53% e 97%. Suspeitamos que melhores resultados para caracterizar um desenvolvedor possam ser obtidos quando outros fatores, tais como os apresentados em (Matter et al., 2009), são combinados à nossa abordagem.

A alta acurácia do nosso classificador deve-se aos resultados obtidos com a classe "*Não participar*". Essa participação é significativa, visto que nosso classificador foi satisfatório ao identificar tarefas que não são de interesse ou do conhecimento do desenvolvedor. Comportamento semelhante das classes "*Participar*" e "*Não participar*" pode ser visto em Ibrahim *et al.* [2010] que apresentaram resultados de um classificador bayesiano para predizer a participação de um desenvolvedor em lista de e-mails.

Ao comparar nossa abordagem com outros trabalhos relacionados, vimos que apresentamos melhores resultados do que aqueles obtidos por Cubranic e Murphy [2004] e por Canfora e Cerulo [2005]. No entanto, esses trabalhos procuram identificar qual desenvolvedor é o mais adequado para resolver uma falha específica no projeto. Como nosso objetivo é predizer a participação de um desenvolvedor em uma determinada tarefa, não é possível comparar os resultados de maneira tão simplificada.

Neste trabalho ainda exploramos o desempenho do classificador construído pelo algoritmo Naïve Bayes. No entanto, como demostrado na Seção 4, o algoritmo J48 apresentou melhores valores de acurácia sobre o algoritmo Naïve Bayes para os dez desenvolvedores em todos os subprojetos. A maior diferença (14,5%) foi apresentada pelo desenvolvedor Top-3 do projeto HDFS e a menor diferença (4,7%) pelo desenvolvedor Top-10 do subprojeto MapReduce. No entanto, obtivemos um aumento da acurácia de 11,2%, 9,6% e 13,3% em média para o Hadoop Common, MapReduce e HDFS, respectivamente ao aplicarmos o algoritmo J48.

A Figura 5 apresenta um gráfico que faz uma relação entre a participação dos desenvolvedores (veja A **Erro! Autoreferência de indicador não válida.** mostra a taxa de participação em tarefas de cada um dos desenvolvedores em cada projeto. Essa taxa é a proporção entre as tarefas em que o desenvolvedor colaborou e as tarefas em que não colaborou. Se a taxa de participação é próxima a 1 então a quantidade de tarefas em que o desenvolvedor participou é próxima à quantidade de tarefas em que não participou. Caso contrário, a classe "*Não participar*" é maior que a classe "*Participar*".

Tabela 2) e a taxa da acurácia do classificador construído pelo algoritmo J48. Podemos notar que o comportamento entre a taxa de acurácia dos dez desenvolvedores mais ativos é semelhante nos três projetos. Um resultado não esperado pode ser visto no crescimento dos valores de acurácia. Esse crescimento indica que entre os TOP10 existe uma tendência de quanto menor é a participação melhor é a acurácia na classificação de sua participação ou não em tarefas. Vale ressaltar que mesmo sendo o último entre os dez, sua participação é muito significativa no projeto. Esse comportamento de crescimento foi encontrado nos três subprojetos analisados, no entanto, é necessário mais estudos para investigar se também ocorre para outros conjuntos de desenvolvedores. Pode-se atribuir tal comportamento ao corte realizado na lista de termos dos desenvolvedores (30% o conjunto $WORLD_{common}$), que pode ter retirado do vocabulário termos que seriam relevantes para melhorar a classificação dos desenvolvedores mais ativos. Para eliminar tal ameaça seria necessário utilizar o vocabulário completo dos desenvolvedores, o que requer alto processamento e grande quantidade de memória.

6. Ameaças à validade

Os resultados da análise são específicos para os projetos analisados, não podendo ser generalizados. Os projetos podem possuir diferentes características decorrentes da estrutura organizacional ou do domínio da aplicação. A análise com uma maior quantidade de projetos, ou com o ecossistema do Apache Software Foundation tornaria os resultados mais genéricos.

A análise foi realizada com base em dados coletados desde a criação do projeto e no início não havia uma separação formal dos subprojetos. Assim, suspeitamos que algumas das tarefas que eram destinadas ao HDFS ou ao MapReduce possam ter sido publicadas na lista do Hadoop Common.

Devido a uma restrição imposta pela limitação física do equipamento utilizado para executar os algoritmos, tivemos que reduzir em 30% o conjunto $WORLD_{common}$. O uso de todo o vocabulário em máquinas com maior poder de processamento e memória, podem gerar resultados melhores para os desenvolvedores mais ativos. Existe ainda a possibilidade de avaliar a eficiência do uso da frequência de termos (TF) ao invés do TF-IDF, para amenizar tal viés.

A abordagem para a extração dos *tokens* que compõem o conjunto $WORLD_p$ pode ser melhor explorada, principalmente com relação aos códigos-fonte publicados nas tarefas.

Não é possível também afirmar que a abordagem tem os mesmos resultados para os desenvolvedores com menor quantidade de comentários enviados. Como dito anteriormente, uma análise com uma amostra maior e com desenvolvedores escolhidos aleatoriamente deve ser conduzida futuramente para verificar a efetividade da abordagem em outros casos. Adicionalmente, podemos dizer que os classificadores utilizados tem como base a análise do vocabulário do desenvolvedor utilizado em comentários anteriores, e pode não proporcionar os mesmos resultados para novos desenvolvedores (problema de *cold start*).

7. Conclusão

Neste trabalho apresentamos resultados que indicam que o conteúdo dos comentários publicados em tarefas (*issues*) de projetos de software livre é um importante fator para construir um classificador de predição de tarefas aos desenvolvedores. Baseado neste fator, mostramos que é possível construir um classificador com alta acurácia que pode diminuir a carga de trabalho do desenvolvedor indicando a sua participação em tarefas de seu interesse. Entretanto, reconhecemos que a taxa de *recall* obtida para a classe "*Participar*" não foi satisfatória. Por último, mostramos que nossa abordagem é mais efetiva se utilizarmos o algoritmo de classificação J48 em comparação ao Naïve Bayes. Este trabalho é uma versão mais detalhada de um trabalho anterior [Schwerz et al., 2012]

A respeito dos trabalhos futuros, além daqueles já citados na análise das ameaças à validade, entendemos que outros fatores que auxiliem a caracterização dos desenvolvedores devem ser explorados e combinados para melhorar o desempenho da predição. Estão sendo conduzidas análises para verificar se o uso de *Term Frequencies* (TF) ao invés do uso do *Term Frequency – Inverse Document Frequency* (TF-IDF), para tentar melhorar as taxas de acerto para os desenvolvedores mais ativos. Além disso, estão sendo realizados experimentos para verificar se o uso do algoritmo PageRank sobre a rede social de desenvolvedores pode trazer melhores resultados que o uso de métricas puramente aplicadas sobre texto (no caso, o TF-IDF).

Referências

Abel, Fabia; Ig Ibert Bittencourt; Evandro Costa; Nicola Henze; Daniel Krause; Julita Vassileva. 2010. Recommendations in Online Discussion Forums for E-Learning Systems. IEEE Trans. Learn. Technol. 3, 2 (April 2010), 165-176. http://dx.doi.org/10.1109/TLT.2009.40

Alonso, Omar; Premkumar T. Devanbu; Michael Gertz. 2008. Expertise identification and visualization from CVS. In MSR ’08: Proceedings of the 2008 Intl. working conference on Mining software repositories, pages 125–128, New York, NY, USA.

Anvik, John; Lyndon Hiew; Gail C. Murphy. 2005. Coping with an open bug repository. In Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange (eclipse '05). ACM, New York, NY, USA, 35-39. DOI= http://doi.acm.org/10.1145/1117696.1117704

Anvik, John; Lyndon Hiew; Gail C. Murphy. 2006. Who should fix this bug? In Proceedings of the 28th international conference on Software engineering (ICSE '06). ACM, New York, NY, USA, 361-370. DOI=10.1145/1134285.1134336 http://doi.acm.org/10.1145/1134285.1134336

Bird, Cristian. Predicting email response using mined data, Disponível em: http://www.cabird.com/papers/mlpaper.pdf, last accessed, May 2012.

Canfora, Gerardo; Luigi Cerulo. 2005. How software repositories can help in resolving a new change request. In Proceedings of the Workshop on Empirical Studies in Reverse Engineering, September 2005.

Cosley, Dan; Dan Frankowski; Loren Terveen,; John Riedl. 2007. SuggestBot: using intelligent task routing to help people find work in wikipedia. In Proceedings of the 12th international conference on Intelligent user interfaces (IUI '07). ACM, New York, NY, USA, 32-41. http://doi.acm.org/10.1145/1216295.1216309

Cubranic, Davor; Gail C. Murphy. 2004. Automatic bug triage using text classification. In Proceedings of Software Engineering and Knowledge Engineering, pages 92-97.

Ibrahim, Walid M.; Nicolas Bettenburg, Emad Shihab, B. Adams, and A. E. Hassan. 2010. Should I contribute to this discussion? In Proceedings of MSR 2010, pages 181–190. IEEE.

Jone, Karen Spärck. 1972. A statistical interpretation of term specificity and its application in retrieval. Journal of Documentation 28 (1): 11–21. doi:10.1108/eb026526.

Matter, Dominique; Adrian Kuhn; Oscar Nierstrasz. 2009. Assigning bug reports using a vocabulary-based expertise model of developers. In Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories (MSR '09). IEEE Computer Society, Washington, DC, USA, 131-140. http://dx.doi.org/10.1109/MSR.2009.5069491

Mockus, Audris; James D. Herbsleb. 2002. Expertise browser: a quantitative approach to identifying expertise. In ICSE ’02: Proceedings of the 24th Intl. Conference on Software Engineering, pages 503–512, New York, NY, USA.

Porter; Martin F. 1980. An algorithm for suffix stripping. Program, 14(3):130–137.

Quinlan, Ross. 1993. Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo, CA.

Raymond, Eric S.. 1999. The Cathedral and the Bazaar (1st ed.). Tim O'Reilly (Ed.). O'Reilly & Associates, Inc., Sebastopol, CA, USA.

Siy, Harvey; Parvathi Chundi; Mahadevan Subramaniam. 2008. Summarizing developer work history using time series segmentation: challenge report. In MSR ’08: Proceedings of the 2008 Intl. working conference on Mining software repositories, pages137–140, New York, NY, USA.

Thomas Fritz, Gail C. Murphy, and Emily Hill. 2007. Does a programmer’s activity indicate knowledge of code? In ESEC-FSE ’07: Proceedings of the 6th European software engineering conference, pages 341–350, New York, NY, USA.

Tom M. Mitchell. 1997. Machine Learning. McGraw-Hill, New York, NY-USA.